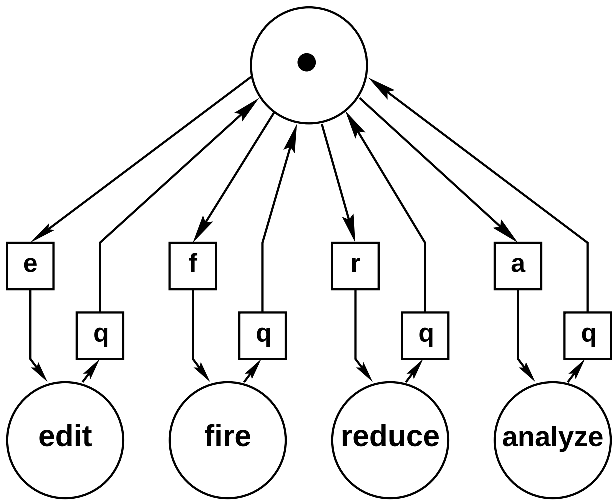
INA
Integrated Net Analyzer
Version 2.2



Manual

Stephan Roch

Prof. Peter H. Starke

Humboldt–Universität zu Berlin
Institut für Informatik
Lehrstuhl für Automaten– und Systemtheorie
Unter den Linden 6
10099 Berlin

Preface

The Integrated Net Analyzer INA is a program with whose help Petri nets of very different kinds can be investigated under different firing rules (in particular timing) with regard to their general properties. Typical properties which can be verified through the analysis are boundedness of places, liveness of transitions, and reachability of markings or states.

When modelling an existing or projected system in the language of Petri nets, the underlying abstraction process assigns certain properties of the system to properties of the net, e.g. the absence of deadlocks in the system can correspond to the non-reachability of a dead marking.

The life cycle of a net model can be roughly sketched out as follows:

- Design
- Analysis
- Interpretation
- Re-design

Therein the interpretation of the analytical results is not to be underestimated.

INA was developed by Prof. Dr. Peter H. Starke. With this interactive, menu-driven program you can edit, reduce, execute and analyze nets. It comprises the analytical procedures offered in the earlier toolkits “Petri-Net-Machine”, “PAN”, “CPNA”, and “ATNA” and was essentially expanded upon by Dr. Karsten Schmidt, Dipl.-Inf. Lutz Pogrell, Dipl.-Inf. Karsten Lüttke and Dipl.-Inf. Stephan Roch.

This manual describes the functionality of version 2.2 of INA. It was written by Stephan Roch in collaboration with Prof. Peter H. Starke, and translated by Dirk Schlatter with the help of Stefan Haar and Mathew O’Breigh (partially supported by the Deutsche Forschungsgemeinschaft in the “PEP”-project).

A certain knowledge of net theory — at least of its fundamental definitions — is necessary to understand this program. For that purpose we refer to the book *“Analyse von Petri–Netz–Modellen”* published 1990 in the series “Leitfäden und Monographien der Informatik” by B. G. Teubner, Stuttgart [Sta90]. Many of the algorithms implemented in INA are described there in greater detail.

No responsibility is assumed for any mistakes in this manual or in the program.

Berlin, April 1999

Peter H. Starke and Stephan Roch

Contents

Further Tables

Summaries and Illustrations

Example Nets

1. Introduction

1.1. Overview

This manual is divided into 6 chapters and an appendix. The chapters contain information which acquaints the user with the Integrated Net Analyzer INA and presents the functions it offers.

Chapter 2 describes the installation, operation and dialogue with the program, entering of data, files, extensions and data formats used, the structure of an INA–session and possible options.

Recommendation: Please read this chapter first, so that you are spared many inconveniences in using the program.

Chapter 3 describes the editor, chapter 4 the firing. Among other things, the latter contains a description of the representation of states and of firing with time allocation.

Chapters 5 and 6 introduce into the reduction and analysis of nets.

In the appendix, you find references to literature containing definitions and details of the implementation, as well as a list of key words. There you will also find further information about this manual and the program.

1.2. Form of representation

Commands will be desribed in the following way:

`<key>     example command`

The explanation of the command starts here, somewhat indented.

`<key>` here indicates the key which is to be entered at the terminal in order to choose the `example command` from the menu.

Sometimes menu items are marked as marginal notes. `Example menu`

Special keys and symbols

The key <cr> denotes the input of return, or enter, <esc> stands for escape and <space> for the space bar (about the keys <delete> and <backspace> see remarks on page 4 in chapter 2.2).

The symbol oo means ∞ and is to be entered by a single <o> (see e.g. under infinite capacity in chapter 2.5.3 on page 20).

Further representation forms

All example inputs, program and file extracts are shown in the same format as the original:

```
The text is indented and written in

a slightly smaller typewriter font.
```

References to the literature have the following form, where a more exact description of the reference point is not always possible: [book signature, reference point (pages)].

A description of the frequently used EBNF-syntax can be found on page 13 of chapter 2.3.4.

2. Foundations

In this chapter, you will find all important information concerning INA, from its installation and the general dialogue rules to the structure of a so-called INA–session, as well as an overview of files, extensions and data formats used, and the options which can be selected. Please read through this chapter carefully before working with the program for the first time.

2.1. Getting started

As a rule you should create a directory on your hard drive and copy the INA–executable into this directory. For the necessary operations, please refer to your operating system's manual.

For any file it needs, INA searches the directory from which it was started; it also saves its files into this directory. Should you be working with several different types of nets at the same time, it is recommendable to create sub-directories for each net or net type. Since in this way, the current options are saved separately, it will not be necessary to select them again each time, because it is sufficient to change the directory and restart the program.

The directory in which the executable file of INA was placed should be recorded in the search path.

2.2. General rules for the dialogue with INA

Before the functions offered by INA are illustrated in detail, important considerations about the dialogue and work with the program can be found here.

Interacting with INA

Since INA is an interactive program, at many different points you have to answer questions with `<Y>` or `<N>`. When you press any key other than `<Y>`, the default answer `<N>` will always be assumed. The answer `yes` can be entered with `<y>` or `<Y>`. INA does not differentiate between upper and lowercase letters here.

Requesting files and entering names

When, for example, you are requested to enter a name of a file or a name of a place, INA suggests a default value, and the cursor rests on the first symbol. You can accept it by pressing `<cr>`. If, however, you wish to enter a different name, you can simply overwrite the given one.

Unfortunately it is not possible to use the `<delete>` and cursor keys to edit. INA offers other editing possibilities in their place:

With the `<space>` key you can move the cursor one position to the right, without overwriting any symbols. Entering `<backspace>` or `<?>` moves the cursor one position to the left. You can therefore use `<?>` for deleting, since you can always overwrite the symbol beneath the cursor. As long as INA is not requesting any file names, you can enter a blank space with `<#>`.

If you have entered a file name and then press `<cr>`, INA will assume everything to the left of the current cursor position as the entry. `<esc>`, on the other hand, assumes the entire row as the entry, regardless of the current cursor position.In either case, the appropriate extension is automatically appended if the entry contains no period.

Please use neither paths nor special symbols as part of a data name, as this can lead to the abortion of the program.

The length of a name for nets, places, or transitions is restricted to a maximum of 16 characters.

Should INA request data, for example a file name or a number of a place, and you enter non-existent data, you receive the message `not found!` and can enter the requested data again.

Switching to the terminal mode

When INA requests files, it is possible to switch to the terminal mode by entering `<esc>`. Afterwards, the required data is entered directly via the keyboard. This is necessary in particular if you do not have data of the requested type available (e.g. during the first sessions with INA).

Saving data

When saving data in an `output file`, entering `<esc>` also causes INA to switch to the terminal mode. When saving data in an already existing file, the question `overwrite?` appears. By entering `<Y>` the file will be overwritten, and upon entering `<N>` a new name will be requested.

2.3. Files, extensions, formats and the entering of nets

In this section you find an overview of the files used and created by INA, a list of the file extensions, a presentation of some file formats and a description of the entering procedure for nets.

2.3.1. Important files

`OPTIONS.ina` In this file, INA saves (unless it terminates irregularly) all options that you have set up during your session, for example the net type, the output formats, the depth of the reachability graphs to be constructed, etc. At the beginning of the next session, these options are automatically set up again. On this matter, see also chapter 2.5 starting on page 18, where the options which can be changed in the options menu are described. All other settings can only be modified at specific points in the program; you find descriptions in the corresponding chapters and sections of this manual.

`COMMAND.ina` This file is created at the end of a session upon request. All commands entered during this session are saved in it. If, upon re-starting INA, the question `Same procedure as last time?` is answered with `<Y>`, the stored commands are executed in the given sequence, i.e. the session is repeated. If meanwhile you enter `<H>` for `halt` INA switches to the terminal mode.

In chapter 2.4.3, it will be illustrated how this may be used to analyze several nets successively according to the same scheme.

`SESSION.ina` INA writes all results (and deductions) obtained during a session into this file. `SESSION.ina` is overwritten at the beginning of the next session. You should therefore rename this file beforehand, if it is still of importance.

Further notes on this file can be found in chapters 2.4.1, 2.4.2 and 2.4.3.

Further files which may be in the current directory after an INA session:

`INVARI.hlp` Auxiliary file for calculation of invariants (in calculating minimal semi-positive invariants)

`ININET.pnt` Auxiliary file which contains the initial net (is saved before the execution of the reduction steps — see page 44 in chapter 5.2)

2.3.2. File extensions

By default, INA uses the following extensions:

`.pnt` Net description of a place/transition net (see chapter 2.3.4 starting on page 13)

`.cnt` Net description of a coloured net (see chapter 2.3.4 starting on page 14)

`.mar` Marking of a net

`.tim` Time intervals of transitions or transition colours

`.tmd` Firing durations of transitions or transition colours

`.atm` Time intervals of arcs

`.icp` Initial clock positions for places

`.ict` Initial clock positions for transitions

`.pri` Priorities

`.cap` Capacities

`.red` Reduction results

.exc	Reduction exceptions
.val	Values
.inv	Invariants
.hlp	Auxiliary file with unformatted invariants
.res	General results
.tra	Trace (path) to a state
.sta	Set of states
.cir	Circuits
.pdc	Predicates
.ctl	CTL–formulae
.prf	Proofs

2.3.3. Entering nets

In the editor nets can be entered with the command <R><T> (see page 24). Besides, if a net is requested as a file, you can switch to the terminal mode with <esc> and enter the net as described in this subsection.

Uncoloured nets

The uncoloured net of Example 1 models two terminals being shared by three programmers. Each programmer is either working at a terminal or pausing. Programmer 1 needs two terminals at the same time in order to work. The transitions model the logging in and out of the programmers.

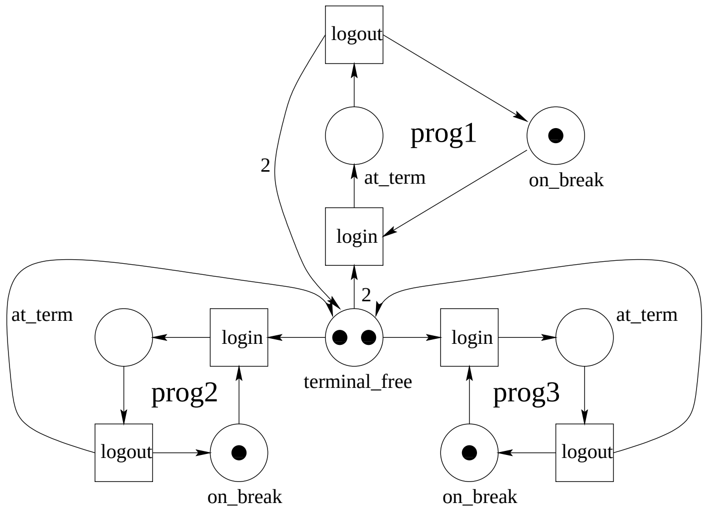
In the following, the entering of this net will be described:

First of all, the net number (default value: 0) and the net name (16 characters maximum) are requested. It is recommendable to fill out both, because the net number and name appear at many points in the protocol and in the saved files, and may therefore help to prevent confusion.

Afterwards, the syntax for place-oriented entries is displayed:

```
Input syntax:
<nr>" " <tokens>" "[ <prelist> ] [ "," <postlist> ] "<cr>"
 <prelist> ::= { <nr> [ ":"<mult> ]" " }
<postlist> ::= { <nr> [ ":"<mult> ]" " }
P>
```

INA expects one line to describe each place. Each such line starts with the number of the place, followed by one blank and the number of tokens in the initial marking. If the place has pre-transitions, these are entered as elements of a list, each of the form <number>:<multiplicity>, and separated by blanks. The input of a simple multiplicity :<1> can be omitted, since this



Example 1: Three programmers sharing two terminals

is set as default value. Of course, the list of pre-transitions can also be empty. If the place is to have post-transitions, then you have to enter a similar list, which is preceded by `<,>`. The end of a line describing a place is given by `<cr>`.

For Net 1 of the three programers sharing two terminals, you have to enter the following lines:

```
Input of Place/Transition Net nr: 1    Netname = 3_Prog_2Term

0 2 4:2 5 6, 1:2 2 3
1 0 1, 4
2 0 2, 5
3 0 3, 6
4 1 4, 1
5 1 5, 2
6 1 6, 3

Q
```

The line `0 2 4:2 5 6, 1:2 2 3` describes place 0, which initially contains two tokens. Its pre-transitions are the transitions 4, 5, and 6, where the arc from transition 4 to place 0 has multiplicity two. The place 0 has the post-transitions 1, 2, and 3, where the arc from place 0 to transition 1 also has multiplicity two.

While entering a net, please keep in mind the general rules described in chapter 2.2 on page 4: You can delete a symbol with `<backspace>` or `<?>`. For entering larger transition lists, it is recommendable to type in a “smaller”-symbol `< < >` at the end of a list element followed by `<cr>`, in order to continue your entry on a new line.

If you have entered all places by their number, press `<Q>` at the prompt `P>` in order to quit this part of the input process.

Next, INA requests a name for every place you have entered (16 characters maximum). When entering `<cr>`, the name will be filled up with blanks if necessary. If the capacity firing rule is selected (see chapter 2.5.3 on page 20), INA additionally requires the specification of a capacity for each place. The default value is `<oo>`, a hardware-dependent constant which signifies an infinite capacity.

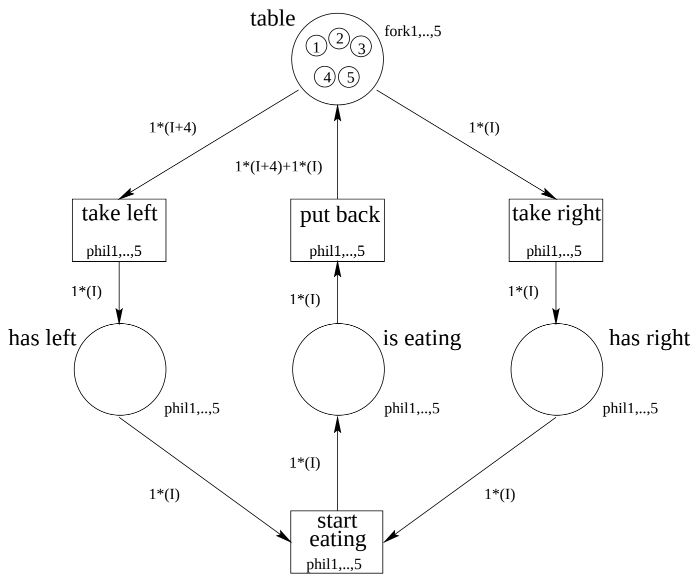
Subsequently, the transitions are named in the same way. Should you be working with priorities (see chapter 2.5.4 on page 21), these are requested thereafter. Their default value is zero.

If you have selected a time option (see chapter 2.5.2 on page 20), an initial clock position is requested, according to the selected option and the net element to be evaluated.

Thereupon the input of the net is completed, and with `<W><F>`, you can save the net into a file.

Coloured nets

The coloured net in Example 2 models the well-known problem of the dining philosophers: Five philosophers are sitting at a round table, on which five forks are placed. To change from the "thinking" state into the "eating" state, a philosopher needs to have the forks at both his right and left sides. After eating, both forks are laid down again, and the philosopher switches back into the "thinking" state.



Example 2: Dining philosophers

On the arcs of the net you can find the corresponding colour information.

The net is entered as follows:

After entering the number and name of the entire net, all places are defined by entering their number, name and colour set. Firstly, INA suggests a number which can either be confirmed with `<cr>` or overwritten. Subsequently, the name of the place (16 characters maximum), as well as the amount of colours are requested. Afterwards, the names of the colours and the initial marking of the place are asked for; under the capacity firing rule, the capacity is to be given as well.

Since the names of the colours of a place are frequently only slightly different, the preceding colour name is presented for editing, starting with the second colour. You can overwrite this suggestion: `<space>` moves the cursor to the right, `<backspace>` or `<?>` to the left.

For entering initial markings, two special values are provided: If you enter `<z>` (as `zero`) or `<s>` (as `standard`) on the first colour, then for all colours the number of tokens on the current place will be automatically set to zero or one, respectively.

In many cases, different nodes have the same colour sets. Therefore, starting with the second place, the entering of the node name is succeeded by the following menu:

```
Colour set to be
read from the terminal................................R
copied from an existing node..........................C
constructed as square CxC of a known colour set.......S
constructed as product CxC1 of known colour sets......P
```

The colour set can be copied from an already constructed node with `<C>`, constructed as the cartesian square $C \times C$ of an existing colour set C with `<S>`, constructed as the cartesian product $C \times C_1$ of two existing colour sets with `<P>`, or entered by hand with `<R>`, as already described.

The entering of places can be quit with `<Q>` upon request of a place number.

With net 2 of the five dining philosophers, the entry dialogue for the number, name, and places of the net looks like the following (condensed presentation):

```
Input of Coloured Petri Net nr: 2    Netname = Dinner
Please, give first the places and the initial marking.

place nr.     1 name =     table
   number of colours: 5
 colour    1: name = fork1             m0: 1
 colour    2: name = fork2             m0: 1
 colour    3: name = fork3             m0: 1
 colour    4: name = fork4             m0: 1
 colour    5: name = fork5             m0: 1

place nr.     2 name =     has_left
Colour set to be
read from the terminal................................R
copied from an existing node..........................C
constructed as square CxC of a known colour set.......S
constructed as product CxC1 of known colour sets......P > R
   number of colours: 5
 colour    1: name = phil1             m0: 0
 colour    2: name = phil2             m0: 0
 colour    3: name = phil3             m0: 0
 colour    4: name = phil4             m0: 0
 colour    5: name = phil5             m0: 0

place nr.     3 name =     has_right
Colour set to be
```

```
....................................................
copied from an existing node.........................C > C
place nr. 2.has_left
 colour    1: phil1             m0: 0
 colour    2: phil2             m0: 0
 colour    3: phil3             m0: 0
 colour    4: phil4             m0: 0
 colour    5: phil5             m0: 0

place nr.     4 name =     is_eating
Colour set to be
....................................................
copied from an existing node.........................C > C
place nr. 2.has_left
 colour    1: phil1             m0: 0
 colour    2: phil2             m0: 0
 colour    3: phil3             m0: 0
 colour    4: phil4             m0: 0
 colour    5: phil5             m0: 0

place nr.     Q
```

Thereafter, you can enter the transitions and arcs. For each transition, the number, name, colour set, and, if applicable, also the priority are requested, followed by its input- and output-arcs, respectively.

There are three modes for defining the arcs: `(C)opy,(E)xpression, by (H)and`. With `<H>` you can enter the arcs by hand. For every transition colour, the multiset of place colour numbers, which is given by the function next to the arc, is to be entered as a formal sum. The copy mode `<C>` is to be used, if there is already an arc which has been inscribed as desired. In this case, you only enter the nodes which are linked by that arc. In expression mode `<E>`, pre-defined functions can be used to inscribe the arc. The choice depends on the the number of colours of the transition and the place. The identity function is always offered.

In the following illustrations, an arc between transition t with f_t colours and a place p with f_p colours is considered. The arc inscription $3*(I+5)+1*(I+7)$ causes 4 tokens to flow through this arc during the firing of the transition t in the colour j (with $1 \leq j \leq f_t$), namely three tokens with the place colour $(I+5)(j)$ and one token with the place colour $(I+7)(j)$, where $(I+n)(j)$ is defined as follows: if $f_p = 1$, then $(I+n)(j) = 1$ for every n, whereas for $f_p > 1$, $(I+n)(j) = ((j+n-1) \bmod f_p) + 1$. In the case $n = 0$ and $f_p = f_t$, this just gives the identity function.

If $f_t > f_p$ and f_t is divisible by f_p without a remainder, then the functions `<F>` (for `first component`) and `<S>` (for `second component`) are offered. INA assumes the colour set of t to be a cartesian product of colour sets, where the colour set of p appears as the first (with `<F>`) or the second (with `<S>`) factor. For a transition colour number j, $F(j)$ or $S(j)$ is the colour of the first or second component of the colour j, respectively: $F(j) = ((j-1) \text{ div } (f_t \text{ div } f_p)) + 1)$ and $S(j) = ((j-1) \bmod f_p) + 1$

If $f_t < f_p$ and f_p is divisible by f_t without a remainder, then the functions `<M>` (for `mine`) and `<H>` (for `his`) are offered. INA assumes the colour set of p to be a cartesian product of colour sets, where the colour set of t appears as the first (with `<M>`) or the second (with `<H>`) factor. For a transition colour number j, $M(j)$ or $H(j)$ is the formal sum of the place colours, whose first or second component equals j and whose second or first component does not equal j, respectively.

In the case $f_p = f_t \cdot f_t$, the function <D> (for `diagonal`) is offered in addition. INA assumes the colour set of p to be the cartesian square of the colour set of t. For a transition colour number j, $D(j)$ is the number of the corresponding diagonal element.

Further functions are not provided.

The entering of pre- or post-arcs is quit by answering the request for a place number with <Q>.

If you work with time allocation (see chapter 2.5.2 on page 20), an initial clock position is requested, according to the selected option and the net element to be evaluated.

For net 2 of the five dining philosphers the entry dialogue for the transitions and arcs looks like the following (condensed presentation):

```
Give next the transitions and the arcs.

transition nr.    1  name =    take_left
Colour set to be
read from the terminal................................R
copied from an existing node..........................C
constructed as square CxC of a known colour set.......S
constructed as product CxC1 of known colour sets......P > C
place nr. 2.has_left
 colour    1: phil1
 colour    2: phil2
 colour    3: phil3
 colour    4: phil4
 colour    5: phil5
Pre-arc input mode: (C)opy , (E)xpression, by (H)and > E
Firing the transition 1.take_left        REMOVES from
place nr.     1.table
functions:  (I)dentity,
expression: 1*(I+4)
place nr.     Q

Post-arc input mode: (C)opy , (E)xpression, by (H)and > E
Firing the transition 1.take_left        SENDS to
place nr.     2.has_left
functions:  (I)dentity,
expression: 1*(I)
place nr.     Q


transition nr.     2 name =    take_right
Colour set to be
......................................................
copied from an existing node..........................C > C
............Colour set C to be taken from a place? Y/N N
transition nr. 1.take_left
 colour    1: phil1
 colour    2: phil2
 colour    3: phil3
 colour    4: phil4
 colour    5: phil5
Pre-arc input mode: (C)opy , (E)xpression, by (H)and > E
Firing the transition 2.take_right       REMOVES from
place nr.     1.table
functions:  (I)dentity,
expression: 1*(I)
place nr.     Q
```

```
Post-arc input mode: (C)opy , (E)xpression, by (H)and > E
Firing the transition 2.take_right       SENDS to
place nr.     3.has_right
functions:  (I)dentity,
expression: 1*(I)
place nr.     Q


transition nr.     3 name =    start_eat
Colour set to be
....................................................
copied from an existing node.........................C > C
............Colour set C to be taken from a place? Y/N N
transition nr. 1.take_left
 colour    1: phil1
 colour    2: phil2
 colour    3: phil3
 colour    4: phil4
 colour    5: phil5
Pre-arc input mode: (C)opy , (E)xpression, by (H)and > E
Firing the transition 3.start_eat        REMOVES from
place nr.     2.has_left
functions:  (I)dentity,
expression: 1*(I)
place nr.     3.has_right
functions:  (I)dentity,
expression: 1*(I)
place nr.     Q

Post-arc input mode: (C)opy , (E)xpression, by (H)and > E
Firing the transition 3.start_eat        SENDS to
place nr.     4.is_eating
functions:  (I)dentity,
expression: 1*(I)
place nr.     Q


transition nr.     4 name =    put_back
Colour set to be
....................................................
copied from an existing node.........................C > C
............Colour set C to be taken from a place? Y/N N
transition nr. 1.take_left
 colour    1: phil1
 colour    2: phil2
 colour    3: phil3
 colour    4: phil4
 colour    5: phil5
Pre-arc input mode: (C)opy , (E)xpression, by (H)and > E
Firing the transition 4.put_back         REMOVES from
place nr.     4.is_eating
functions:  (I)dentity,
expression: 1*(I)
place nr.     Q

Post-arc input mode: (C)opy , (E)xpression, by (H)and > E
Firing the transition 4.put_back         SENDS to
place nr.     1.table
functions:  (I)dentity,
expression: 1*(I)+ 1*(I+4)
place nr.     Q
```

```
transition nr.     Q
```

2.3.4. File formats

In this section, you find descriptions of file formats in extended Backus-Nauer-Form (EBNF). The symbols: `[ ]` denotes an option, `{ }` a repetition, | an alternative, `'' ''` a string, `< >` a syntactic unit, and `::=` is the substitution symbol. The only particularity is the string ``<cr>'', which stands for the end-of-line symbol. Line breaks in the presentation are only used for formatting purposes, and have no meaning.

These descriptions serve as an interface, which makes it possible for you to attach any (graphical) editor, as long as you know the formats well enough so that you can write a program for translation into the INA–file format.

Syntax of a .pnt–file

The `.pnt`–net file consists of three sections separated by `@` symbols. The sections contain information about the net structure, the places, and the transitions, respectively. As a rule, names consist of 16 characters, filled up with blanks if necessary. Some special symbols like ``_'' are also allowed. The upper bound for integers is system-dependent.

```
 <pnt-file> ::= <netheader> "<cr>"
                <netstruct> "<cr>"
                "@<cr>"
                <placedata> "<cr>"
                "@<cr>"
                <transdata> "<cr>"
                "@<cr>"

<netheader> ::= "P   M   PRE,POST  NETZ " <netid>

    <netid> ::= <netnr> [ ":" <netname> ]

    <netnr> ::= <number>

  <netname> ::= <name>

<netstruct> ::= <placedef>
                { "<cr>" <placedef> }

 <placedef> ::= { "  " <placenr> " " <tokens> "    " [ <prelist> ] [ ", " <postlist> ]

  <placenr> ::= <number>

   <tokens> ::= <number>

  <prelist> ::= { <transnr> [ ": " <arcmult> ] " " }

 <postlist> ::= { <transnr> [ ": " <arcmult> ] " " }

  <transnr> ::= <number>

  <arcmult> ::= <number>
```

```
<placedata> ::= "place nr.             name capacity time"
                { "<cr>       " <placenr> ": " <placename> "      " <capacity> <time> }

<placename> ::= <name>

 <capacity> ::= " " <nr> | "oo"

     <time> ::= <number>

<transdata> ::= "trans nr.             name priority time"
                { "<cr>       " <transnr> ": " <transname> "      " <priority>
                <time> }

<transname> ::= <name>

 <priority> ::= <number>
```

Example for a .pnt–file

Here is the file of net 1 "Three programmers sharing two terminals":

```
P   M   PRE,POST  NETZ 1:3_prog_2_term
  0 2     4: 2 5 6, 1: 2 2 3
  1 0     1, 4
  2 0     2, 5
  3 0     3, 6
  4 1     4, 1
  5 1     5, 2
  6 1     6, 3
@
place nr.             name capacity time
       0: terminal_free         oo    0
       1: prog1_at_term         oo    0
       2: prog2_at_term         oo    0
       3: prog3_at_term         oo    0
       4: prog1_on_break        oo    0
       5: prog2_on_break        oo    0
       6: prog3_on_break        oo    0
@
trans nr.             name priority time
       1: login_prog1            0    0
       2: login_prog2            0    0
       3: login_prog3            0    0
       4: logout_prog1           0    0
       5: logout_prog2           0    0
       6: logout_prog3           0    0
@
```

Syntax of the .cnt–file

The `.cnt`–net file consists of the unfolded net structure (which describes the place colours, their initial markings, and the relations with the transition colours), and the place and transition colours. So far, there is no difference in the syntax compared to a `.pnt`–file. But in an additional fourth section, the folding information is included, i.e., the information which coloured places and transitions exist, and what their colours are.

```
 <cnt-file> ::= <pnt-file>
                <aggregat> "<cr>"
                "@<cr>"

 <aggregat> ::= "AGGREGATION:<cr>"
                <placeaggr> "<cr>"
                "@<cr>"
                <transaggr>

<placeaggr> ::= "places:<cr>"
                <placecol>
                { "<cr>" <placecol> }

 <placecol> ::= "    " <placenr>:<placename> " " <placenr> { " " <placenr> }

<transaggr> ::= "transitions:<cr>"
                <transcol>
                { "<cr> <transcol> }

 <transcol> ::= "    " <transnr>:<transname> " " <transnr> { " " <transnr> }
```

This syntax allows compatibility of the net files; so it is possible to fold `.pnt`–files, and to unfold `.cnt`–files, i.e., to ignore the folding information (see also chapter 2.5.1 on page 19).

Example for a .cnt–file

Here is the file of the coloured net 2: “Five dining philosophers”

```
P   M   PRE,POST  NETZ 2:Dinner
  1 1     16 17, 2 6
  2 1     17 18, 3 7
  3 1     18 19, 4 8
  4 1     19 20, 5 9
  5 1     16 20, 1 10
  6 0     1, 11
  7 0     2, 12
  8 0     3, 13
  9 0     4, 14
 10 0     5, 15
 11 0     6, 11
 12 0     7, 12
 13 0     8, 13
 14 0     9, 14
 15 0     10, 15
 16 0     11, 16
 17 0     12, 17
 18 0     13, 18
 19 0     14, 19
 20 0     15, 20
@
place nr.            name capacity time
       1: fork1                 oo    0
       2: fork2                 oo    0
       3: fork3                 oo    0
       4: fork4                 oo    0
       5: fork5                 oo    0
       6: phil1                 oo    0
       7: phil2                 oo    0
       8: phil3                 oo    0
```

```
       9: phil4                 oo    0
      10: phil5                 oo    0
      11: phil1                 oo    0
      12: phil2                 oo    0
      13: phil3                 oo    0
      14: phil4                 oo    0
      15: phil5                 oo    0
      16: phil1                 oo    0
      17: phil2                 oo    0
      18: phil3                 oo    0
      19: phil4                 oo    0
      20: phil5                 oo    0
@
trans nr.              name priority time
       1: phil1                  0    0
       2: phil2                  0    0
       3: phil3                  0    0
       4: phil4                  0    0
       5: phil5                  0    0
       6: phil1                  0    0
       7: phil2                  0    0
       8: phil3                  0    0
       9: phil4                  0    0
      10: phil5                  0    0
      11: phil1                  0    0
      12: phil2                  0    0
      13: phil3                  0    0
      14: phil4                  0    0
      15: phil5                  0    0
      16: phil1                  0    0
      17: phil2                  0    0
      18: phil3                  0    0
      19: phil4                  0    0
      20: phil5                  0    0
@
AGGREGATION:
places:
    1:table               1     2     3     4     5
    2:has_left            6     7     8     9    10
    3:has_right          11    12    13    14    15
    4:is_eating          16    17    18    19    20
@
transitions:
    1:take_left           1     2     3     4     5
    2:take_right          6     7     8     9    10
    3:start_eat          11    12    13    14    15
    4:put_back           16    17    18    19    20
@
```

Note

In net files, you can insert any comments after the last `@`. These are not read, and will therefore be lost when the net is saved into a different file.

2.4. The structure of an INA–session

2.4.1. Starting and exiting INA

To start the program, change to the directory which contains the net to be analyzed, and into which the files to be constructed should be stored. Then enter `ina` at the prompt.

If the file `OPTIONS.ina` is in the directory, it will be read. The file `COMMAND.ina`, on the other hand, will only be executed if you answer the question `same procedure as last time?` (which then appears at the beginning) with `<Y>`. In this case, the stored commands of the last session are repeated until `<H>` (for `halt`) is entered, or all commands have been executed.

If there is no `COMMAND.ina`, or the question above has been answered with `<N>`, the main menu appears on the screen and the selected options are displayed:

```
>>>>>>>>>>>>  Welcome to the Integrated Net Analyzer! <<<<<<<<<<<<
Version 2.2;                             Peter Starke, Berlin 1999

Current net options are:
    token  type: black        (for Place/Transition nets)
    time option: no times
    firing rule: normal
    priorities : not to be used
    strategy   : single transitions
    line length: 80

Do You want to
    edit ? ......................................E
    fire ? ......................................F
    analyse ? ...................................A
    reduce ? ....................................R
    read the session report ? ...................S
    delete the session report ? .................D
    change options ? ............................O
    quit ? ......................................Q
 choice >
```

By pressing the indicated letters the corresponding functions are selected. To stop a command, or quit a menu, `<Q>` can be entered in most cases. At many points this will also cancel running computations prematurely.

In the main menu, the program is quit by entering `<Q>`. If there is still a net in the internal memory, INA will first ask whether you would like to save it, or whether it shall be discarded: `Forget the net?`. Moreover, you will be asked whether the commands to be entered should be saved: `Save the commands to COMMANDS.ina?`. In the affirmative case, they may be used in a future session. INA will also ask you whether you want to rename the current session–report in order to protect it from overwriting: `Rename SESSION.ina?` If so, INA requests a name for a `Session output file` , which is not allowed to already exist.

2.4.2. The session report

The session report is an ASCII file, which contains all analysis results and deductions. In the main menu, two menu items are offered, one to display the session report, and the other to erase it.

`<S>      read the session report`

The file `SESSION.ina` is displayed pagewise on the screen; it can be scrolled down by pressing any key. With `<Q>`, the page mode can be switched off (the remaining data of the file is then displayed without breaks), a further `<Q>` quits reading. Afterwards, you are back in the main menu again.

`<D>      delete the session report`

INA deletes the file `SESSION.ina`, and returns to the main menu. If you then enter `<S>` again, INA informs you that the initial part was deleted: `INA session report (first part deleted)`.

To read the session report during an analysis, you may have go back to the main menu (except for graph analysis, see page 82 in chapter 6.5.4).

2.4.3. Analysis of several nets following the same pattern

By using the file `COMMAND.ina`, it is possible to analyze several nets in the same way. This will be demonstrated by an example. Assuming the files `net01.pnt`, `net02.pnt` and `net03.pnt` are to be analyzed, you may proceed as follows:

1. Rename the first net `net01.pnt` to `current.pnt`.
2. Start INA and analyze `current.pnt` with the appropriate commands.
3. Save `COMMAND.ina` while quitting the session. Perhaps rename `SESSION.ina` at this point.
4. Undo the renaming of `current.pnt` (for the first net `net01.pnt`) and rename the next net (`net02.pnt`) to `current.pnt`.
5. Start INA and answer the question `Same procedure as last time?` with `<Y>`. The net `current.pnt` is now analyzed exactly like the previous net, without any further command. Afterwards, `SESSION.ina` may have to be renamed again.

The last two steps can be repeated for all nets that you would like to investigate (`net03.pnt` in the example).

2.5. Options

After the start of INA, the current options are displayed in the main menu. By entering `<O>`, these can be changed. This way you inform INA which kind of nets you would like to investigate.

There, the token type, time option, firing rule, use of priorities, firing strategy, and line length are requested subsequently. Options which you want to retain can be confirmed by entering `<cr>`. If for some inquiry only one choice exists, the dialogue continues immediately with the next one. After the first start of the program, the default options are:

```
token  type: black       (for Place/Transition nets)
time option: no times
firing rule: normal
priorities : not to be used
strategy   : single transitions
line length: 80
```

The selected options are saved in the file `OPTION.ina` (see chapter 2.3.1 on page 4).

In the following, all possible selections are individually presented.

2.5.1. Token types

With this option, you can determine whether the tokens are coloured. `token type`

`<B>     black (indistinguishable) tokens`

Hereby, you choose to work with (uncoloured) place/transition nets, whose markings consists of black indistinguishable tokens. [Sta90, Definition 2.2 (23)].

`<C>     coloured tokens`

With this selection, you will work with coloured nets, which allows to work with coloured, i.e. distinguishable tokens [Sta90, Kapitel 20 (222–235)]. Please note: Coloured nets cannot be reduced with INA.

If you have set the token type of a coloured net to `black`, INA will ask: `Forget the colour structure?`; with `<Y>`, the colours are deleted [Sta90, Definition 20.3 (226)]. Warning: Information about the net can get lost this way!

If, on the other hand, you change the token type in the opposite way, the question appears: `Fold the Net?` If your answer is `<Y>`, you have to indicate how INA should fold it. The possible choices are [Sta90, Definition 20.2 and Bemerkung (225)]:

folding type	necessary input	effect
`1-place;1-transition CPN`	`<A>`	maximal folding
`by hand`	`<B>`	user-defined folding
`trivially coloured CPN`	`<C>`	no folding

2.5.2. Time option

time option

Here you can choose between different clocked net types [Sta90, Kapitel 17 (192–196)]:

`<N>     no times`

With this selection, no clocks will be used. The dialogue continues with the next option.

`<D>     durations`

To each transition a duration can be assigned. [Sta90, Kapitel 18 (197–208)].

`<I>     intervals`

A time interval can be assigned to transitions `<T>` (as earliest and latest firing time after being enabled), or arcs `<A>` (as permeability of arcs) [Sta90, Kapitel 19 (209–221)].

clocked net elements

If you have decided in favour of interval-clocking (with `<I>`), you will be required to determine which net elements are to be clocked (`<T>` for transitions, `<A>` for arcs). However, in the duration-clocking mode (with `<D>`), only transitions can be clocked, which is therefore automatically set.

The representation of states depends on the selected time option. You can get an overview, together with some examples, in chapter 4.2 on page 31ff. Notes on the firing behaviour of clocked nets can be found in chapter 4.4.2 on page 34ff.

2.5.3. Firing rule

firing rule

The firing rule indicates under which conditions a transition may fire, and what the effect of the firing is on the marking.

Depending on the net type and the time option, up to three different firing rules can be chosen:

`<N>     normal firing rule`

According to the normal firing rule, a transition is enabled if the number of tokens on each pre-place is not less than the multiplicity of the arc of this pre-place to the transition. Firing a transition causes as many tokens to be subtracted from each pre-place as the multiplicity of the corresponding arc to the transition, and as many tokens to be added to each post-place as the multiplicity of the corresponding arc from the transition [Sta90, Definition 2.4 (25)].

Please note: Reduction is possible only under this firing rule!

`<C>     normal firing rule with capacities`

Compared with the normal firing rule, the firability of the transitions is further restricted by upper bounds (capacities) for the number of tokens on each place. The capacity of a place may also be infinite. [Sta90, Definition 10.2 (102)].

The symbol oo signifies an infinite capacity, and is to be entered with <o>. It corresponds to a system-dependent constant.

A place which already contains more tokens at the initial marking than its capacity allows is considered overloaded (`overloaded`).

`<S>     safe firing rule`

This rule can only be useded with ordinary nets (i.e., where all arcs have multiplicity 1). Applying it, a transition is enabled if all of its pre-places and none of its post-places are marked. Places which contain more than one token at the initial marking are considered overloaded.

2.5.4. Priorities

Here you decide whether priorities should be obeyed (entering <Y>) or ignored (entering <N>) while firing. By default, priorities are ignored. `obey priorities`

Priorites are non-negative integers assigned to the transitions. To obey priorities means that, in a given state, only those enabled transitions may fire which possess the highest (i.e. largest) priority among all enabled transitions [Sta90, (103)]. The use of priorities restricts the analysis possibilities.

2.5.5. Firing strategy

INA offers two firing strategies. However, depending on the selection of the other options, a restriction on the available strategies might be necessary. In case only one strategy is possible within the current context, it is automatically selected, and INA continues with the inquiry for the next option. `firing strategy`

Here are the alternatives:

`<T>     fire single transitions`

This is the common firing strategy, which fires transitions subsequently.

`<S>     firing maximal steps`

This strategy always fires maximal (w.r.t. set inclusion) sets of concurrently enabled transitions. However, the set of reachable markings is thereby reduced.

2.5.6. Line length

This information is necessary for the formatting of the output file `SESSION.ina`, and is thus to some degree also effective for the screen. If your screen or your printer is capable of more than 80 columns, you can change the value accordingly, so that the output is more readable (maximal value: 255 columns). `line length`

3. Editor

The editor is an alphanumeric editor which contains all functions necessary to enter and edit a net, as well as to merge nets. The nets constructed this way can be saved into files, and re-read from there. A description of the formats can be found in chapter 2.3.4 on page 13ff.

When you select the `edit` function by entering `<E>` in the main menu, an editor menu appears according to the current options. The illustration below shows the menu according to the default options. If a different kind of net has been selected, additional deleting and changing functions are offered. If there is not already a net in the internal memory of INA, the range of functions is further reduced, as only the applicable commands will be displayed.

```
Editor Menue:
Quit the editing process........................Q
Read a net from a File.........................RF
Read a net from the Terminal...................RT
Read a Node from the terminal..................RN
Add a loop.....................................RL
Read an Arc from the terminal..................RA
Insert a run-place.............................RP
Write the net to a File........................WF
Write the net to the Terminal..................WT
Delete the net Totally.........................DT
Delete a Node..................................DN
Delete all Isolated nodes......................DI
Change nUmbers.................................CU
Change Names...................................CN
Change Multiplicities..........................CM
Change the Token load (marking)................CT
Node Doubling..................................ND
Merge two nOdes................................MO
Merge two Nets ................................MN
Test Connectedness.............................TC
edit>
```

The commands can be split into three categories: Reading and writing commands (beginning with `<R>` or `<W>`), deleting and changing commands (beginning with `<D>` or `<C>`), and merging commands (beginning with `<M>`).

In the following sections, you will find a description of all commands which can appear in the editor menu. Important to note: the availability of some functions depends on the current net options!

Some functions only make sense if INA has a net in the internal memory. Otherwise you receive the message `no net present`. Should the net consist of places or transitions only, you receive corresponding messages, because the net is incomplete: `only places present` or `only transitions present`.

The following command belongs to none of the above mentioned categories, but is, of course, nonetheless available:

`<Q>` Quit the editing process

By entering `<Q>` you leave the editor. In case you have entered an incomplete net, you will be requested to delete: `Delete the fragmented net?`

3.1. Reading and writing

INA offers commands for reading and saving nets, places, transitions, and arcs:

`<R><F>` Read a net from a File

If INA already has a net in the internal memory, the question `Delete the old net?` appears. Since only one net can be kept in the memory, entering `<N>` prevents reading the new net. Entering `<Y>` causes the old net to be deleted. All net files in the current directory are then displayed, and INA requests a file name. In chapter 2.2 on page 3, a description of how to enter a file name can be found. Afterwards, INA reads the specified file into the internal memory and returns to the editor menu. While the file is requested, it is possible to switch to the terminal mode by entering `<esc>`.

`<R><T>` Read a net from the Terminal

With this command, it is possible to enter nets directly at the terminal. If INA already has a net in the internal memory, it reacts in a similar way to the command described above.

A description of the entering process can be found in chapter 2.3.3 on page 6ff.

`<R><N>` Read a Node from the terminal

With this command, you can enter nodes in a (possibly empty) net. First you are asked whether you would like to add a place or a transition. Then the entry syntax is displayed, which, in the case of a transition to be inserted, reads as follows:

```
Input syntax:
<nr>" "[ <prelist> ] [ "," <postlist> ] "<cr>"
 <prelist> ::= { <nr> [ ":"<mult> ]" " }
<postlist> ::= { <nr> [ ":"<mult> ]" " }
T>
```

For a place to be inserted, the syntax appears, which you can find in the description of a place input (see page 6 in chapter 2.3.3). There you find further notes on entering places.

A transition is entered in an analogous way; however, no initial marking is to be given, and instead of transitions lists, lists of places are requested.

If the specified node number already exists, the old definition is replaced. If, on the other hand, you enter a non-existent node number, a corresponding node will be inserted into the net without a name. The same applies to new nodes occuring in the definition of this node, which are automatically inserted by INA.

By entering `<Y>` upon the inquiry `next`, you can insert further transitions or places. In case you prefer a transition-oriented input, rather than selecting `<R><T>`, you can use this function to enter the net from the view of transitions, and then later name the places and determine an initial marking, as well as possible capacities and initial clock settings, using the changing functions described below.

When inserting a node into a coloured net, you may proceed similar to the description in chapter 2.3.3 on page 8ff. When inserting nodes into coloured nets please note the following: for a transition to be inserted, all adjacent places must exist. INA issues the following warning: `Please, make sure that all places exist to which the new transition will be connected`.

`<R><L>     Add a loop`

With this function, a loop can be added. A loop consists of a place which is marked with a token, and a transition which has this place as both pre- and post-place. The purpose of this construction is to indicate transient states (see page 34 in chapter 4.4.2) under the option of time allocation to transitions (see chapter 2.5.2 on page 20).

INA assigns to ech of the inserted nodes the number 0, or other available numbers. The place is named `loop-place` with the initial clock setting 1, and the transition is named `loop-transition` with the initial clock setting 0.

`<R><A>     Read an Arc from the terminal`

With this command, you can enter a new arc, delete an old one, or change its multiplicity. Firstly, the adjacent transition and place are requested, then the multiplicity. Finally, you have to indicate the direction of the arc: `Place to Transition?` The answer `<Y>` produces an arc from the place to the transition, whereas the answer `<N>` produces an arc from the transition to the place. An existing arc will be overwritten; it can be deleted by setting its multiplicity to 0. In a coloured net, an arc is entered as described in the entering of coloured nets (see page 10 in chapter 2.3.3).

`<R><P>     Insert a run-place`

With this function, a run-place can be inserted into an existing net. A run-place is marked with a token and has loops around all transitions. The purpose of this construction is to guarantee single element steps, if transitions are clocked with a firing duration, and the firing strategy of maximal steps is used (see page 20ff in chapter 2.5.2).

The inserted place receives the number following the highest existing place number, and the name `run-place`.

Please note, that after adding transitions, the original run-place is to be removed, and replaced by a new one.

Using the safe firing rule, the net is dead; INA issues the following warning during the pre-analysis: `inconsistency:  firing mode=safe, but net is not pure` (see also page 54 in chapter 6.2.1).

`<W><F>     Write the net to a File`

With this command, you can save the net in the internal memory to a file. By entering `<Y>` to the question `Select Places`, you can select specific places which span a sub-net, that is to be saved. After entering a file name, the (sub-)net is written into this file.

By selecting a set of places, you later have the opportunity to analyze the corresponding sub-net separately.

`<W><T>     Write the net to the Terminal`

With this command, you can display various information about the net in the internal memory on the screen:

```
Information on the net ('N'), the places ('P'),
       the transitions ('T')  or quit    ('Q')?
```

Using net 1, the different outputs of these commands will be demonstrated.

The sub-command `<N>` displays the net number and name, and provides an overview of the amount of places and transitions used, as well as the intervals containing the node numbers:

```
Netnr:   1    Netname: 3_prog_2_term
Places:                7   [0,6]
Transitions:           6   [1,6]
```

With the sub-command `<P>`, the places of the current net displayed. You have the option to select certain places. The following information is given: place number, place name, marking, name and number of pre-transitions with multiplicities, and likewise for post-transitions:

```
place nr.      0.terminal_free      load:  2
------------------------------------------------------------------------
PREname                  nr: mlt                nr: mlt   POSTname
logout_prog1              4:   2                 1:   2   login_prog1
logout_prog2              5                      2        login_prog2
logout_prog3              6                      3        login_prog3

place nr.      1.prog1_at_term      load:  0
------------------------------------------------------------------------
PREname                  nr: mlt                nr: mlt   POSTname
login_prog1               1                      4        logout_prog1

place nr.      2.prog2_at_term      load:  0
------------------------------------------------------------------------
PREname                  nr: mlt                nr: mlt   POSTname
login_prog2               2                      5        logout_prog2

place nr.      3.prog3_at_term      load:  0
------------------------------------------------------------------------
PREname                  nr: mlt                nr: mlt   POSTname
login_prog3               3                      6        logout_prog3

place nr.      4.prog1_on_break     load:  1
------------------------------------------------------------------------
PREname                  nr: mlt                nr: mlt   POSTname
logout_prog1              4                      1        login_prog1
```

```
place nr.      5.prog2_on_break     load:  1
------------------------------------------------------------------------
PREname                 nr: mlt               nr: mlt   POSTname
logout_prog2             5                     2        login_prog2

place nr.      6.prog3_on_break     load:  1
------------------------------------------------------------------------
PREname                 nr: mlt               nr: mlt   POSTname
logout_prog3             6                     3        login_prog3
```

With the subcommand <T>, the transitions are displayed in a way which is similar to the previous description:

```
transition nr.      1.login_prog1
------------------------------------------------------------------------
PREname                 nr: mlt               nr: mlt   POSTname
terminal_free            0:   2                1        prog1_at_term
prog1_on_break           4

transition nr.      2.login_prog2
------------------------------------------------------------------------
PREname                 nr: mlt               nr: mlt   POSTname
terminal_free            0                     2        prog2_at_term
prog2_on_break           5

transition nr.      3.login_prog3
------------------------------------------------------------------------
PREname                 nr: mlt               nr: mlt   POSTname
terminal_free            0                     3        prog3_at_term
prog3_on_break           6

transition nr.      4.logout_prog1
------------------------------------------------------------------------
PREname                 nr: mlt               nr: mlt   POSTname
prog1_at_term            1                     0:   2   terminal_free
                                               4        prog1_on_break

transition nr.      5.logout_prog2
------------------------------------------------------------------------
PREname                 nr: mlt               nr: mlt   POSTname
prog2_at_term            2                     0        terminal_free
                                               5        prog2_on_break

transition nr.      6.logout_prog3
------------------------------------------------------------------------
PREname                 nr: mlt               nr: mlt   POSTname
prog3_at_term            3                     0        terminal_free
                                               6        prog3_on_break
```

With <Q> you can quit viewing the net.

3.2. Deleting and changing

INA offers commands to delete nets and net elements. Some of these commands are only available for coloured nets.

`<D><T>    Delete the net Totally`

The entire net is removed from the internal memory, without any further queries.

`<D><N>    Delete a Node`

With this command, you can delete a place or a transition of the current net. If the question `delete a place` is answered with `<Y>`, you can enter the number of a place (otherwise, the number of a transition) to be deleted. The selected node and all incident arcs are removed from the internal memory, without any further queries.

`<D><C>    Delete a Colour`

In a coloured net, you can delete a colour of a net node with this command, and possibly even the diagonal (see page 11).

`<D><I>    Delete all Isolated nodes/colours`

With this command, you can delete unused (isolated) nodes or colours, which have emerged while editing the net. Since such nets cannot be analyzed by INA, it is recommendable to execute this command before saving the net or leaving the editor.

`<D><A>    Delete all input- or output-Arcs`

In a coloured net, you can delete all input- or output-arcs of a net node with this command.

`<C><P>    Change Priorities`

This command can only be used if you have opted to `work with Priorities` (see chapter 2.5.4 on page 21). The current priorities are displayed. You can either read a `.pri`–file which contains the new priorities, or overwrite them individually by entering `<esc>`. `<cr>` restores old values, which have been inadvertently selected to be changed.

`<C><C>    Change Capacities`

This command can only be used, if you have opted to use the `normal firing rule with capacities` (see chapter 2.5.3 on page 20). The current capacities are displayed. You can either read a `.cap`–file which contains the new capacities, or overwrite them individually by entering `<esc>`. An infinite capacity can be entered with `<o>`. `<cr>` restores old values, which have been inadvertently selected to be changed.

`<C><I>    Change Initial times`

This command can only be used, if you have opted to clock transitions or arcs (see chapter 2.5.2 on page 20). The current initial times are displayed. You can either read a `.icp`–file, or `.ict`–file which contains the new initial times, or overwrite them individually by entering `<esc>`. `<cr>` restores old values, which have been inadvertently selected to be changed.

`<C><U>      Change numbers`

With this command, you can change the numbers of the net elements and the net itself. The following menu is offered:

```
Do you want to
change the net number.............................N
change a place number.............................P
add a span to all place numbers...................A
renumber the places starting with 1...............R
change a transition number........................T
add a span to all transition numbers..............S
renumber the transitions starting with 1..........F
```

You can either change the numbers of individual places or transitions, or add a constant to all place or transition numbers. This operation has the purpose to make the set of place or transition numbers disjoint from the set of place or transition numbers (respectively) of another net, with which the current net is to be merged. Also, a consecutive (dense) numbering is offered, which makes some algorithms more efficient.

`<C><N>      Change Names`

This command is used to enter and change names:

```
Do You want to change the name of
the net...................................N
a place...................................P
a transition..............................T
a place colour............................C
a transition colour.......................F
```

`<C><M>      Change Multiplicities`

This command works exactly like the command `<R><A>` (see chapter 3.1 on page 25).

`<C><T>      Change the Token load (marking)`

With this command, you can change the initial marking. The current initial marking is displayed. You can either read a `.mar`–file which contains the new initial marking, or overwrite it individually by entering `<esc>`. `<cr>` restores old values, which have been inadvertently selected to be changed.

3.3. Merging

`<N><D>      Node Doubling`

This command doubles a node, i.e., for a given node, it inserts an identical one into the net. This can be useful, e.g., for a refinement of the net. Firstly, you have to enter the type of node to be doubled: `Double a place (P), a transition (T) or quit (Q)`. After entering `<P>` or `<T>`, you can enter the number of the node to be doubled. Then, a new node number is requested: `Duplicate node number`. If you enter the number of an existing node, INA rejects it (`is not free`), and you can select another number. The name of the new node is composed of `duplicate` followed by the original node number. The command is quitted with `<Q>`.

`<M><O>     Merge two nOdes`

This command merges two nodes. Firstly, you have to enter the type of nodes to be merged: `Merge two places?` After entering `<Y>`, you can enter the numbers of the places to be merged. The place with the higher number is deleted, the tokens are added, and the arcs reset. You can carry out a similar operation for transitions upon entering `<N>` above.

`<M><N>     Merge two Nets`

With this command, you can merge the current net in the internal memory of INA with a net which is stored in a file. Through the inquiry `Merge with a second net at the common places`, you can determine the type of common nodes at which the nets are supposed to be merged.

`<T><C>     Test Connectedness`

The current net is tested for connectedness (see chapter 6.2.1 on page 55). If necessary, you can write the different components in separate files, and later analyze them individually. For some analysis algorithms, connectedness is a necessary condition [Sta90, Definition 14.1 and Bemerkung (148)].

4. Firing

A user-directed execution of the net, i.e., traversing the reachability graph, provides insight into the behaviour of the net, and allows to investigate those nodes in the reachability graphs more closely at which something unexpected occur (e.g., reachability of dead states).

4.1. Fundamentals

The execution of a net is offered in the main menu under `<F>`. In case no net was read yet, INA requests one, together with the time information, if necessary. Next, you can change the writing options, in order to have transition or colour names displayed. Afterwards, a menu is offered which lists the available functions:

```
<R>: Reset to the initial state           <B>: Back to the previous state
<W>: Write the actual marking to a file
<S>: Show the stubborn set used by the stubborn reduced reachability graph
<A>: All minimal stubborn sets to be shown           <Q>: Quit computation
```

Before each firing procedure, this menu is displayed again in a condensed form:

```
<r>eset | <b>ack | <w>rite | <s>tubborn set | <a>ll stubborn sets | <q>uit
```

4.2. State representations

In the following two sections, you find a description of the state representation in INA.

4.2.1. State representation without time allocation

If you work without time allocation (see chapter 2.5.2 on page 20), information on the state only consists of the current marking, which is presented in the following vector form:

```
PL:  1  1  1  1  1  2  2  2  2  2  3  3  3  3  3  4  4  4  4  4
MA:  1  0  1  1  1  0  0  1  0  0  0  0  0  0  0  0  0  0  0  0
```

The place numbers are in the headline (for a coloured net, each number appears as often as that place has colours); the corresponding number of tokens are located below. The example shows a marking of net 2, in which there is exactly one token, namely of colour 3, on place 2.

In some situations, you also find the abbreviations `P.nr` and `toks` for place numbers and token amounts.

4.2.2. State representation under time allocation

In case you are working with time allocation, information on the state consists of the marking together with a vector, which describes the current clock position.

In the following examples, these vectors are given for a place/transition net with six places $(0, \ldots, 5)$ and five transitions $(1, \ldots, 5)$.

Time option duration for transitions — firing duration

Clocks are assigned to the transitions, and indicate the remaining time before the transition finishes firing, i.e., delivers tokens to its post-places.

```
PL: 0 1 2 3 4 5
MA: 1 1 0 0 1 1
TR: 1 2 3 4 5
TC: 0 1 0 3 0
```

In this state, the clock of transition 2 indicates, that firing this transition will last another cycle. Similarly, the firing of transition 4 will last another three time units.

In some outputs, you also find the abbreviations `P.nr` and `toks`, or `T.nr` and `time`, for place numbers and token amounts, or transition numbers and clock positions, respectively.

Time option intervals for transitions — eft/lft

Clocks are assigned to the transitions, and indicate the time that has passed since each one has been enabled.

```
PL: 0 1 2 3 4 5
MA: 0 1 1 0 0 1
TR: 1 2 3 4 5
TC: - 2 - 1 -
```

In this state, the clocks of transitions 1, 3, and 5 are switched off, because there are not enough tokens available to fire any of these transitions. The clocks of transitions 2 and 4 display 2 and 1, respectively, i.e. these transitions are enabled as far as tokens are concerned, and have been so for two and one cycle(s), respectively. Whether or not they can fire, depends on whether their earliest firing time eft has already been reached.

In some outputs, you also find the abbreviations `P.nr` and `toks`, or `T.nr` and `time`, for place numbers and token amounts, or transition numbers and clock positions, respectively.

Time option intervals for arcs — permeability

Clocks are assigned to the places and indicate the time that the most recently arrived token has already waited on the place.

```
PL: 0 1 2 3 4 5
MA: 1 0 1 1 0 1
PC: 0 0 5 6 0 0
```

In this state, the tokens on places 0 and 5 are “fresh“, i.e. originated with this state, while the tokens on places 2 and 3 have already waited 5 and 6 cycles, respectively, to be permitted by an arc.

In some outputs, you also find the abbreviations `P.nr` and `toks`, or `T.nr` and `time`, for place numbers and token amounts, or transition numbers and clock positions, respectively.

4.3. Auxiliary functions

This section describes some help functions.

`<Q>     Quit`

With this command, you can quit firing and return to the main menu, whereby the original marking of the net is automatically reset. Long calculations can be aborted similarly.

`<W>     Write the actual marking to a file`

This command offers the possibility to save the marking just reached in a file, e.g. in order to calculate a shortest path later on. After entering the name of a file, the current marking is written into it, whereby the default extension is `.mar`.

Both of the following functions are only offered if a stubborn reduction (see page 65 in chapter 6.5.1) is possible. For details of implementation see [Pog94, Berechnung sturer Mengen].

`<S>     Show the stubborn set used by the stubborn red.  reachab.  graph`

With this command, the stubborn sets are displayed, which were chosen by INA during the stubborn reduction of the reachability graph.

`<A>     All minimal stubborn sets to be shown`

With this command, all minimal stubborn sets are outputted.

4.4. Firing Functions

This section describes the actual functionality of firing.

4.4.1. General remarks

In each state, the current marking is shown first, followed by a list of enabled transitions or steps, depending on the firing strategy:

```
 PL: 0 1 2 3 4 5 6
 MA: 2 0 0 0 1 1 1
enabled: 1, 2, 3,
<r>eset | <b>ack | <w>rite | <s>tubborn set | <a>ll stubborn sets | <q>uit
Transition to fire >
```

A transition or step in the given list is selected by entering its number. The selected transition or step is then executed, and the resulting marking is displayed. For coloured nets, in addition to the transition number you also have to specify the firing colour, set apart by a semi-colon. If no transition is enabled INA returns the message: no transition has concession. In case you have entered the number of a disabled transition or step, firing is refused with the message no concession.

Under time allocation, the system time is also shown: transitional period from previous state=2 system time=10.

<cr> fire the first element

By entering <cr>, the option to always select and fire the first element of the given firing list is set.

<B> Back to the previous state

You can repeatedly reverse a firing procedure with <B>, and thus make alternative selections.

<R> Reset to the initial state

With this command, the initial state is restored, which allows to investigate a different path through the reachability graph. At the end of firing, you do not need to use this function, since it is automatically applied.

4.4.2. Firing under time constraints

In the following subsections, you find a description of firing under time allocation in INA.

Time option Duration for transitions — Firing duration

To each transitions, a non-negative integer is assigned. A transition has to fire as soon as enough tokens are available. If such a transition t with duration d fires at the moment τ, then the input tokens are removed from its pre-places at the time τ, and the output tokens are available at the time $\tau + d$ on its post-places.

A transition that is being fired, i.e. when the input tokens have been removed, but the output tokens have not been created yet, cannot start a second firing procedure during this period, even if sufficiently many tokens are available at its pre-places. Such nets are executed accordingly when firing maximal steps.

In nets with firing durations, transient states can arise, i.e., states in which no transitions can fire, but out of which, as time elapses, a state results, in which actions are possible. Even the initial state can be transient, if the initial clock position is not identically zero, i.e., some transitions are firing at present.

When the initial state is transient, INA indicates the next stable state: `Initial state is transient. Stabilization time = 3` is an example for the stabilization of the initial state. In general, transient states are stabilized by incrementing the clock positions, and the according change of the state: e.g., `Transitional period from previous state = 2` indicates, that, since the last state, two cycles have passed in which no transitions could fire.

In case you would like to obtain an overview of all (including transient) states, use the command `<R><A>` in the editor in order to insert a loop (see page 25 in chapter 3.1). The firing duration of the new transition (`loop-transition` – normally numbered with the transition number 0) is to be set to 1. In each reachable state, this transition is then ready to fire; the reachability graph cannot contain dead or transient states any more.

Time option intervals for transitions — eft/lft

To each transition, an integer interval eft, lft is assigned. The clock of the transition is started at the moment its pre-places are sufficiently marked. The transition may fire as soon as the time indicated by its clock is at least equal to eft, and it has to fire when the time is equal to lft, provided it has not been disabled by the firing of other transitions. If two not necessarily different transitions share a pre-place, i.e., they are in a static conflict, and one of them fires, then the clock of the second transition is either switched off, if this transition is disabled by the firing of the first one, or reset to zero, if it remains enabled afterwards. Firing a transition does not take time.

There are states in which transitions can fire, but no transition has to fire, because their latest firing times have not been attained yet. The enabled transitions are displayed together with their possible delay, and you have to choose not only the transition to be fired but also its delay, i.e. the number of time ticks of the system clock to elapse until its firing.

Time option intervals for arcs — permeability

To each arc, which is directed from a pre-place to a transition, a time interval eft, lft is assigned . At each place, there is a clock, which indicates the time that the most recently arrived token has already waited on the place. The clock is thus set to zero at the moment a token arrives at the place, and also when a token is removed from the place. A transition can fire if all of its pre-places are sufficiently marked and the clock of each pre-place indicates a time that lies within the interval eft, lft assigned to the arc directed from this pre-place to the transition. If a transition can fire, it has to; hence such nets are executed in steps. Firing a transition does not take time. INA stabilizes transient states automatically.

Before the first firing, an overview of the time spans is displayed, after which a clock of a place is stopped: Since a token can leave a place only by an arc, if the clock of this place indicates a time between eft and lft of this arc, the clock of a place can be switched off, if the time elapsed is greater than the maximum of all lft of the arcs leaving this place. Afterwards, no more tokens can be removed from this place, so all clock positions of the place which are greater than this maximum are equivalent. INA displays these times as `clock stop time`.

With permeability intervals, an overview of all (including transient) states is not possible, but you can obtain some transient states by using the command `<R><A>` in the editor in order to insert a loop (see page 25 in chapter 3.1). For the interval of the arc from the new place (`loop-place` – normally numbered with the transition number 0) to the new transition (`loop-transition` – normally numbered with the transition number 0), eft is to be set to one, and lft to `oo`. However, the reachability graph may still contain transient states in certain cases.

4.5. Example

Using the unfolded version of the coloured net 2 of the dining philosophers, an example for firing will now be given. Screen displays and excerpts from the session file are shown.

At the start, the initial state and the list of its enabled transitions are outputted:

```
Fire
Current name options are:
     transition names to be written

 PL:  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20
 MA:  1  1  1  1  1  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
enabled:  1.phil1_take_left,  2.phil2_take_left,  3.phil3_take_left,
          4.phil4_take_left,  5.phil5_take_left,  6.phil1_take_right,
          7.phil2_take_right,  8.phil3_take_right,  9.phil4_take_right,
         10.phil5_take_right,
Transition to fire >
```

By entering `<1>`, `<6>`, `<11>`, and `<16>` subsequently, you can execute a run-through in which the first philosopher picks up the left and right forks, eats, and puts the forks back. Afterwards, the net is reset to its initial state.

```
Transition to fire > 1

fired transition:  1.phil1_take_left


 PL:  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20
 MA:  1  1  1  1  0  1  0  0  0  0  0  0  0  0  0  0  0  0  0  0
enabled:  2.phil2_take_left,  3.phil3_take_left,  4.phil4_take_left,
          5.phil5_take_left,  6.phil1_take_right,  7.phil2_take_right,
          8.phil3_take_right,  9.phil4_take_right,
Transition to fire > 6

fired transition:  6.phil1_take_right


 PL:  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20
 MA:  0  1  1  1  0  1  0  0  0  0  1  0  0  0  0  0  0  0  0  0
```

```
enabled:  3.phil3_take_left,  4.phil4_take_left,  5.phil5_take_left,
          7.phil2_take_right,  8.phil3_take_right,  9.phil4_take_right,
         11.phil1_start_eat,
Transition to fire > 11

fired transition: 11.phil1_start_eat


 PL:  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20
 MA:  0  1  1  1  0  0  0  0  0  0  0  0  0  0  0  1  0  0  0  0
enabled:  3.phil3_take_left,  4.phil4_take_left,  5.phil5_take_left,
          7.phil2_take_right,  8.phil3_take_right,  9.phil4_take_right,
         16.phil1_put_back,
Transition to fire > 16

fired transition: 16.phil1_put_back


 PL:  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20
 MA:  1  1  1  1  1  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
```

Starting in this situation, by entering `<cr>` repeatedly, the first transition is always fired, and all the philosophers pick up their left fork.

```
 PL:  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20
 MA:  1  1  1  1  1  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
enabled:  1.phil1_take_left,  2.phil2_take_left,  3.phil3_take_left,
          4.phil4_take_left,  5.phil5_take_left,  6.phil1_take_right,
          7.phil2_take_right,  8.phil3_take_right,  9.phil4_take_right,
         10.phil5_take_right,
Transition to fire >

fired transition:  1.phil1_take_left


 PL:  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20
 MA:  1  1  1  1  0  1  0  0  0  0  0  0  0  0  0  0  0  0  0  0
enabled:  2.phil2_take_left,  3.phil3_take_left,  4.phil4_take_left,
          5.phil5_take_left,  6.phil1_take_right,  7.phil2_take_right,
          8.phil3_take_right,  9.phil4_take_right,
Transition to fire >

fired transition:  2.phil2_take_left


 PL:  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20
 MA:  0  1  1  1  0  1  1  0  0  0  0  0  0  0  0  0  0  0  0  0
enabled:  3.phil3_take_left,  4.phil4_take_left,  5.phil5_take_left,
          7.phil2_take_right,  8.phil3_take_right,  9.phil4_take_right,
Transition to fire >

fired transition:  3.phil3_take_left


 PL:  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20
 MA:  0  0  1  1  0  1  1  1  0  0  0  0  0  0  0  0  0  0  0  0
enabled:  4.phil4_take_left,  5.phil5_take_left,  8.phil3_take_right,
          9.phil4_take_right,
```

```
Transition to fire >

fired transition:  4.phil4_take_left


 PL:  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20
 MA:  0  0  0  1  0  1  1  1  1  0  0  0  0  0  0  0  0  0  0  0
enabled:  5.phil5_take_left,  9.phil4_take_right,
Transition to fire >

fired transition:  5.phil5_take_left


 PL:  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20
 MA:  0  0  0  0  0  1  1  1  1  1  0  0  0  0  0  0  0  0  0  0
 no transition has concession!

Transition to fire >
```

As indicated by the message above, no transition is enabled any more; the whole net is in a dead state.

In this situation, you can, for example, save the current marking with `<W>` in a file, and then go back one step with `<B>`.

```
Transition to fire >   W

Marking output file > dinner.mar

Marking written to dinner.mar


Transition to fire >   B

previous state


 PL:  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20
 MA:  0  0  0  1  0  1  1  1  1  0  0  0  0  0  0  0  0  0  0  0
enabled:  5.phil5_take_left,  9.phil4_take_right,
```

ow you can select the other transition and thereby let the fourth philosopher take the missing fork. By entering `<cr>`, the eating process can start, and afterwards, both forks may be put back again.

```
Transition to fire > 9

fired transition:  9.phil4_take_right


 PL:  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20
 MA:  0  0  0  0  0  1  1  1  1  0  0  0  0  1  0  0  0  0  0  0
enabled: 14.phil4_start_eat,
Transition to fire >

fired transition: 14.phil4_start_eat
```

```
 PL:  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20
 MA:  0  0  0  0  0  1  1  1  0  0  0  0  0  0  0  0  0  0  1  0
enabled: 19.phil4_put_back,
Transition to fire >

fired transition: 19.phil4_put_back

 PL:  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20
 MA:  0  0  1  1  0  1  1  1  0  0  0  0  0  0  0  0  0  0  0  0
enabled:  4.phil4_take_left,  5.phil5_take_left,  8.phil3_take_right,
          9.phil4_take_right,
Transition to fire >
```

Now the philosophers three and five may take another fork. Of course, the fourth philosopher could immediately get hungry again, and unfairly take the forks again, so that the other philosophers would eventually starve.

With <R>, you can quit the run and return to the initial state.

```
Transition to fire >  R

reset to the initial state


 PL:  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20
 MA:  1  1  1  1  1  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
enabled:  1.phil1_take_left,  2.phil2_take_left,  3.phil3_take_left,
          4.phil4_take_left,  5.phil5_take_left,  6.phil1_take_right,
          7.phil2_take_right,  8.phil3_take_right,  9.phil4_take_right,
         10.phil5_take_right,
Transition to fire >
```

Moreover, with <S> you can at any time inspect the stubborn sets which are used by INA for the stubborn reduction. At the initial marking the stubborn set looks as follows:

```
Transition to fire >  S

Stubborn set used by stubborn set method is:
       1.phil1_take_left, 10.phil5_take_right,
```

With <A>, all minimal stubborn sets are displayed; a counter indicates the progression of the calculation:

```
Transition to fire >  A
                      Candidate nr. 32
All minimal stubborn sets:
   1:  5.phil5_take_left,  9.phil4_take_right,
   2:  4.phil4_take_left,  8.phil3_take_right,
   3:  3.phil3_take_left,  7.phil2_take_right,
   4:  2.phil2_take_left,  6.phil1_take_right,
   5:  1.phil1_take_left, 10.phil5_take_right,
```

With <Q>, you quit the firing mode and return to the main menu. The initial marking is automatically reset.

5. Reduction

For large nets, it may become impossible to calculate the entire state graph. To make it smaller, you can reduce the net step by step. In a reduction step a sub-net is replaced by another sub-net (generally with fewer places) in such a way that the net characteristics of liveness, non-liveness, boundedness and non-boundedness are preserved.

5.1. Fundamentals

The reduction of nets is offered in the main menu under `<R>`, though only for uncoloured place/transition-nets without time allocation and without use of priorities, because it is only in this case that the reduction steps are appropriate, i.e., that characteristics of the original net can be deduced from the characteristics of the reduced one.

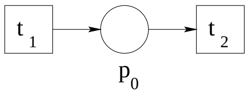
First, the reduction menu, together with the following note, is displayed:

```
Reduction:
Transitions without pre-places are live and will be deleted together with their
post-places. If such transition has excepted post-places, it will be excepted.
Isolated transitions will be deleted even if they are excepted from reduction!
Reduction menue:
For:                            please, type:
    Quit ......................................Q
    Program a sequence of reduction steps.......P
    change eXceptions...........................X
    connectedness Test..........................T
    save the actual net.........................N
    Raise node numbers with exceptions..........R
    terminal Output of the current net..........O
    reset to the Initial net....................I
    Squeeze the node numbers....................S
    Fusion of congruent nodes...................F
    Merging of equivalent places................M
    ELIMINATION of F(pF)={p}-places.............A
    ELIMINATION of (Fp)F={p}-places.............B
    ELIMINATION of looping places...............C
    DELETION of looping transitions.............U
    DELETION of single output transitions.......V
    DELETION of PTP-sequences...................W
 choice>
```

For the reduction, you can define exceptions by using the function `<X>` (see page 43), i.e., sets of places and transitions that must not be deleted during reduction.

As INA indicates in the reduction menu, transitions without pre-places are deleted together with their post-places, unless they themselves or one of their post-places are exempted from the reduction. Transitions without pre-places are live, and their post-places are unbounded [Sta90, Regel 1 (85)].

If you reduce a net that contains transitions without pre-places, then these pre-places are not deleted until you use the commands <A>, <B>, <V> or <W>. The search for transitions without pre-places is only performed, if the net was newly read, just set back, or successfully reduced. Please note: Before you start any other reduction procedure, execute one of the specified commands, for the following reason. Should the application of the other procedure be impossible, INA would assume that the net was not changed, and therefore (incorrectly), no search for transitions without pre-places would be performed during the next run-through.



Example 3: A net containing a transition without pre-places

The simple net of example 3 is completely deleted by a single application of one of the preceding commands:

```
There are transitions without pre-place, the net is not bounded.
 1 live transition(s) deleted
 1 unbounded place(s) deleted
There are transitions without pre-place, the net is not bounded.
 1 live transition(s) deleted
The net is live and totally unbounded.
```

In the session report, the same situation is described by the following protocol:

```
deleted (live) transitions:
       1,
deleted (unbounded) places:
       0,
deleted (live) transitions:
       2,
deleted (unbounded) places:
      empty
The net is live and totally unbounded.
```

In case the current net does not satisfy the conditions of the reduction procedure selected, INA returns to the reduction menu with the message `No application possible`. With <Q>, you can abort running reductions at any time; steps already begun are finished, and the message `Stopped` is displayed.

INA records the reduction process on the screen and in the session report. Places and transitions are displayed not with their names, but with their numbers preceded by `p` or `t`, respectively.

Isolated nodes which have been created during reduction are always deleted, even if they were exceptions before.

Please note: in the reduced net, the names of places and transitions are often deleted, and values such as capacities and priorities have sometimes not been copied. In case a reduced net is to be analyzed further, it should therefore be checked in the editor beforehand.

5.2. Auxiliary functions

During reduction you are supported by the following functions:

<P> Program a sequence of reduction steps

With this command, you can program a sequence of reduction steps, which may be executed automatically. You enter a sequence of letters ending with <Q>: Give a sequence of letters finished by Q. To each letter, a reduction step is assigned. Possible entries are therefore <I>, <S>, <F>, <M>, <A>, <B>, <C>, <U>, <V>, <W>, and <Q>. After you have quit programming, INA displays the program and awaits confirmation of its execution: Execute? If you answer the question with <Y>, all entered procedures are executed in the specified order. Afterwards, INA returns to the reduction menu.

If this command is used again (without returning to the main menu meanwhile), INA displays the most recently entered program, and allows another execution.

<X> change eXceptions

With this command, you can specify places and transitions of the current net which are to be excluded from the reduction. The exceptions can be saved in a file with the extension .exc. INA first lists all such files. You can either select a file to be read, or, by pressing <esc>, enter the exceptions directly.

If the command is used again (INA has exempted places and transitions in the memory), you first have to answer the question Delete exceptions? (P)laces, (T)ransitions, (N)o . When entering <P> or <T>, you see the numbers of the places or transitions, which have been exempted so far. Now you can either determine the exceptions to be deleted one after the other, or, by entering <A>, delete them all. However, should you answer with <N>, you can read another file or directly enter further exceptions. Please note: the existing exceptions are maintained in this case.

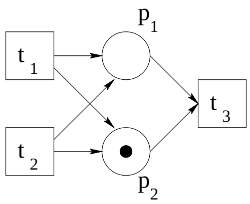
<T> connectedness Test

With this command, the current net can be tested for connectedness. It works exactly like the command <T><C> in the editor (see page 30 in chapter 3.3).

<N> save the actual net

With this command, the current net can be saved with a new net number. You are requested to enter a new net number and the name of the output file. The default extension is .red, since you are saving a reduction result.

<R> Raise node numbers with exceptions



Example 4: A net with two parallel places and transitions

With this command, the place or transition numbers can be incremented by a user-defined constant. The exempted nodes will not be considered here. See also the description of the editor command <C><U> (page 29 in chapter 3.2).

`<O>` terminal Output of the current net

This command displays information about the current net on the screen. It is identical to the command <W><T> in the editor (see page 26 in chapter 3.1).

`<I>` reset to the Initial net

With this command, the original net (which was stored at the beginning in the file `ININET.pnt`) can be read again, in order to try out other reduction steps, or a different order of operations.

The exceptions entered so far remain unaltered, and thus, if necessary, have to be checked: `Exceptions ok after reset?`.

`<S>` Squeeze the node numbers

This command works exactly like the sub-commands <R> and <F> in the editor command <C><U> (see page 29 in chapter 3.2), and is used to compact the node numbers. The places and/or transitions are re-numbered, starting with 1. It is recommendable to apply this function at the end of reduction, as it is advantageous for many analysis procedures, with regard to running time and efficient memory usage, when the node numbers are "dense".

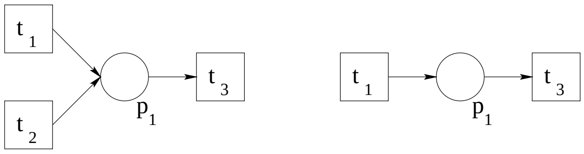
5.3. Reduction procedures

`<F>` Fusion of congruent nodes

INA searches for either two places or two transitions which are connected with all other nodes of the net in exactly the same way, so-called parallel nodes [Sta90, Definition 9.3 (87)].

Illustration 4 shows a net with two parallel places (p1 and p2) and two parallel transitions (t1 and t2).

You have to indicate directly the type of nodes to be investigated: `Check places (P), transitions (T) or exit (E) ?` After entering <P> or <T> to search for places or transitions, respectively, INA either states that no parallel nodes were found (`No application possible!`), or informs about how many nodes were deleted. Normally, always the nodes with the higher number are deleted. For parallel places, however, the place with the smaller number is deleted, if it contains more tokens [Sta90, Regel 3 (88)].



Example 5: Reductions with respect to parallel nodes of the example net 4

When entering `<P>`, the net of example 4 becomes the net on the left of example 5. Since the place p_2 had a higher number as p_1, and p_1 did not contain more tokens than p_2, place p_2 has been deleted. If further reduced with `<T>`, the right net of example 5 results. The transition t_2 has been deleted, since it had a higher number than transition t_1.

In the session report, you find the following protocol:

```
Fuse:
p2 deleted
 1 place(s) deleted
t2 deleted
 1 transition(s) deleted
```

`<M>` Merging of equivalent places

This command merges equivalent places of the net. Two places p_1 and p_2 are equivalent, if

- each place possesses exactly one post-transition, t_1 and t_2, respectively,
- the multiplicity of the arcs from p_1 to t_1 and from p_2 to t_2 equals 1, and
- the transitions t_1 and t_2 are connected to all places different from p_1 and p_2 in exactly the same way [Sta90, Definition 9.4 (88)].

In the net of example 6, the places p_1 and p_2 are equivalent.

In case INA finds two equivalent places p_1 and p_2, place p_2 (with a higher number as p_1), as well as its post-transition t_2, are deleted, the tokens on p_2 are added onto p_1, and the arcs leading to p_2 are directed towards p_1 [Sta90, Regel 4 (89)].

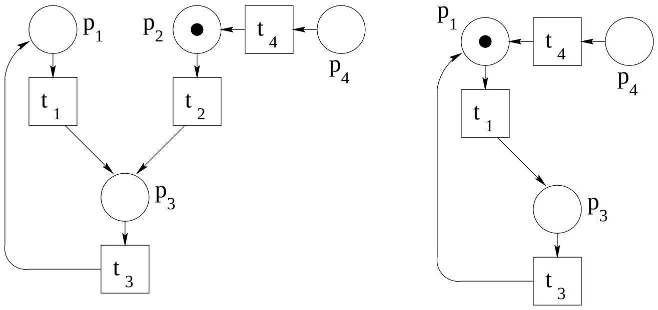
The net on the right of example 6 is the result of the reduction of the net on the left. Place p_2 and its post-transition t_2 have been deleted, the token from p_2 has been placed on p_1, and the arc from t_4 to p_2 has been directed towards p_1.

In the session report, you find the following protocol:

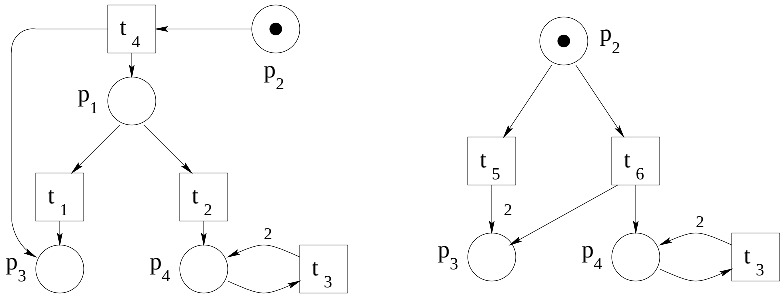
```
Merge equivalent places:
Transition  2 deleted, Place 2 deleted
```

`<A>` ELIMINATION of $F(pF) = \{p\}$-places

This command merges pre- and post-transitions. Here, pF denotes all post-transitions



Example 6: A net with equivalent places and their reduction



Example 7: A net which is reducible with `<A>` and the result

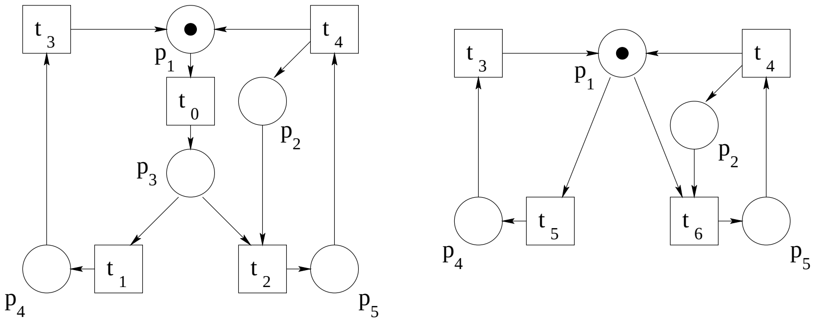
of the place p, and $F(pF)$ all pre-places of the post-transitions of p. $F(pF) = \{p\}$ are the places p which are the only pre-places of their post-transitions.

A place with the following characteristics is sought:

- p has $k > 0$ pre-transitions $t_1, \dots, t_k$
- p has $n > 0$ post-transitions $t'_1, \dots, t'_n$
- no transition is both pre- and post-transition of p
- at least one post-transition of p has a post-place
- p is the only pre-place of its post-transitions
- all arcs starting at p have the same multiplicity v
-
  - if p has exactly one post-transition ($n = 1$), then all arcs ending at p have multiplicity divisible by v
  - if p has more than one post-transition ($n > 1$), then all arcs ending at p have multiplicity v

The place p_1 in the net on the left of example 7 satisfies the above conditions.

In case INA finds a place p that satisfies all conditions, appropriate transitions are fired, if necessary, to ensure that fewer than v tokens are on p. Then the pre-transitions $t_1, \dots, t_k$, the place p itself, and the post-transitions $t'_1, \dots, t'_n$ are deleted; instead, $k \cdot n$ new transitions $t_{i,j}$ are inserted in such a way that firing $t_{i,j}$ has the same effect as firing



Example 8: A net which is reducible with <B> and the result

t_i followed by firing t'_j x times — where x is the multiplicity of the arc from t_i to p divided by v[Sta90, Regel 5 (90)]. INA states only the amount of deleted places.

In example 7, the net on the right results. Place p_1 and the transitions t_1,t_2, and t_4 have been deleted and replaced by the transitions t_5 and t_6. Firing t_5 in the reduced net has the same effect as firing t_4 and t_1 in the old net, and similarly, t_6 is fired instead of t_4 and t_2.

In the session report, you find the following protocol:

```
place elimination A:
New transition 5 replaces transitions 4 * 1
New transition 6 replaces transitions 4 * 2
Place 1 deleted
```

<B> ELIMINATION of $(Fp)F = \{p\}$-places

This command merges a pre-transition with post-transitions. Here, Fp denotes all pre-transitions of the place p, and $Fp(F)$ all post-places of the pre-transitions of p. $Fp(F) = \{p\}$ are the places p which are the only post-places of their pre-transitions.

A place with the following characteristics is sought:

- p has exactly one pre-transition t_0, and p is the only post-place of this transition
- p has $n > 0$ post-transitions $t'_1, \ldots, t'_n$
- no transition is both pre- and post transition of p
- t_0 is the only post-transition of its pre-places
- all arcs starting and ending at p have the same multiplicity v, and there are less than v tokens on p

The place p_3 in the net on the left of example 8 satisfies the above conditions.

In case INA finds a place which satisfies all conditions, the pre-transition t_0, the place p itself, and the post-transitions $t'_1, \ldots, t'_n$ are deleted; instead, n new transitions $t_1, \ldots, t_n$ are inserted in such a way that firing t_j has the same effect as firing t_0 followed by firing t'_j[Sta90, Regel 6 (92)]. INA states only the amount of deleted places.

In example 8, the net on the right results. Place p_3 and the transitions t_0, t_1, and t_2 have been deleted and replaced by the transitions t_5 and t_6. Firing t_5 in the reduced net has the same effect as firing t_0 and t_1 in the old net, and similarly, t_6 is fired instead of t_0 and t_2.

In the session report, you find the following protocol:

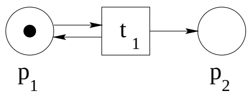
```
place elimination B:
Transition 5 replaces transitions 1 * 0
Transition 6 replaces transitions 2 * 0
Place 3 deleted
```

`<C>` ELIMINATION of looping places

This command searches for bounded and unbounded places that satisfy the following conditions:

- each transition t subtracts at most as tokens from p as it adds to p
- p is sufficiently marked.

Place p will be deleted, if there are no isolated transitions as a result. INA states the amount of deleted places.



Example 9: A net with a reducible looping place

In the net of example 9, the place p_1 satisfies the above conditions, and is therefore deleted.

When you start to reduce with this command, INA asks the question `Shall we delete only bounded places?`. If you answer `<Y>`, only bounded places will be deleted, otherwise also unbounded places. A place p is unbounded, if firing a transition t adds more tokens to p than are subtracted from p, and the transition t is live, i.e., p is adequately marked.

If, in the net of example 9, the multiplicity of the arc from t_1 to p_1 is increased by one to two, then the place p_1 is unbounded, and therefore it will not be deleted, if you have answered the question with `<N>`.

In the session report, you find the following protocol:

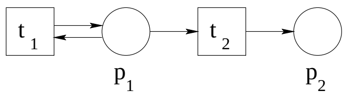
```
place elimination C:
only bounded places to be deleted
place elimination C:
bounded and unbounded places to be deleted
p1 deleted, was unbounded, if the net is live.
```

`<U>` DELETION of looping transitions

This command deletes all transitions t that satisfy the following conditions:

- firing t does not alter the marking
- a different transition t' exists which — in order to be enabled — requires at least as many tokens on each pre-place of t as t does.

This situation is described as a loop. In case INA finds such a transition, it is deleted. INA states the amount of deleted transitions.



Example 10: A net with a reducible loop

A simple example for a loop is net 10. Firing transition t_1 does not alter the marking of the net. Since the transition t_2 subtracts at least as many tokens as t_1 from p_1, t_1 is deleted.

In the session report, you find the following protocol:

```
transition elimination U:
transition 1 deleted
```

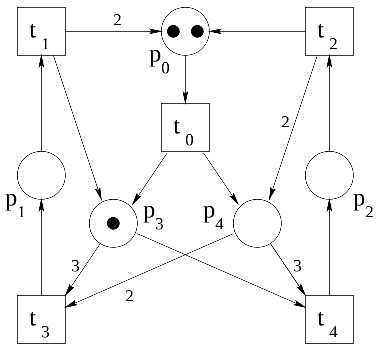
<V> DELETION of single output transitions

This command merges a pre-place with post-places. A transition with the following characteristics is sought:

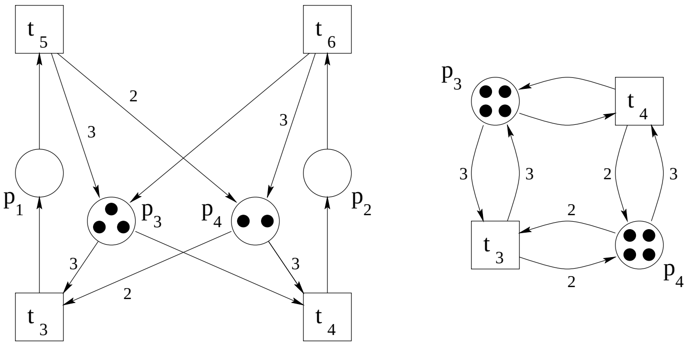
- t has exactly one pre-place p
- the multiplicity of the arc from p to t is one
- p has $k > 0$ pre-transitions $t_1, \ldots, t_k$
- p is not a post-place of t, but t has post-places

The transition t_0 of the net in example 11 satisfies the above conditions.

In case INA finds a transition which satisfies all conditions, t is fired repeatedly until p contains no more tokens. Then the transition t, the place p, and the pre-transitions $t_1, \ldots, t_k$ are deleted; instead, k new transitions $t_j^\star$ are inserted in such a way that firing $t_j^\star$ has the same effect as firing t_j followed by firing $t_j^\star$ x times — where x is the multiplicity of the arc from t_j to p. INA states only the amount of deleted places.



Example 11: A net which is reducible with <V>



Example 12: Reduction of Example net 11

In example 11, a single application of this rule yields the net on the left of example 12. First, the transition t_0 is fired until the two tokens on p_0 have left. This implies that there are three tokens on p_3 and two on p_4 then. Next, the transition p_0, the place p_0, and the transitions t_1 and t_2 are deleted and replaced by transitions t_5 and t_6. Firing t_5 in the reduced net has the same effect as firing t_1 and firing twice t_0 in the old net, and similarly, t_6 is fired instead of t_2 and t_0.

Altogether, the rule is applied three times, yielding the net on the right in example 12.

In the session report, you find the following protocol:

```
transition elimination V:
t0,p0 deleted.
t1,p1 deleted.
t2,p2 deleted.
```

<W> DELETION of PTP-sequences

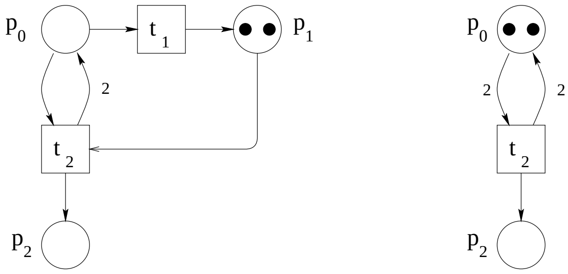
INA searches for a transition t and two places p and q, which satisfy the following conditions:

- p is the only pre-place, q the only post-place of t, where $p \neq q$
- t is the only pre-transition of q, and q possesses post-transitions
- the multiplicity of the arcs from p to t and from t to q equals one.

In such a situation, the places p and q are merged, and the transition t is deleted.

This transformation can transform bounded or non-live nets into unbounded or live nets, respectively; its application is therefore limited. However, you can deduce from the boundedness or non-liveness of the reduction result the boundedness or non-liveness of the original net. INA warns `Non-liveness and boundedness are not preserved under this rule`, and requests you to confirm the execution: `Execute?` If you answer this question with `<N>`, the command will not be executed.

The net on the left in example 13 is dead, since no transition can fire. After reducing with `<W>`, the deletion of the transition t_1 and the merging of the place p_1 into p_0 yield the net on the right. This net is now live, since the transition t_2 can always fire. Hence, a dead net has been transformed into a live net!



Example 13: A net which is reducible with `<W>` and the result

INA states the amount of deleted transitions, and writes another warning in the session report:

```
transition elimination W:
Boundedness and non-liveness are not preserved!

Nodes deleted:
p1, t1;
```

6. Analysis

Three different ways of proceeding with the analysis of a net can be distinguished: Reachability analysis, invariant analysis, and structural analysis.

Under reachability analysis, we understand every procedure which relies on a (at least partial) construction of the state or reachability graph of the net. The computation of paths belongs to this category, as well as the computation and analysis of the state graph, where the latter may be computed using symmetric or stubborn set reductions. For unbounded nets, the coverability graph can be computed.

The invariant analysis comprises all processes which aim at the solution of systems of linear equations derived from the incidence matrix of the net. In addition to the classical invariant computation, the simplex-oriented methods, separately described below, also belong to this category.

Structural analysis attempts to obtain results about dynamic properties of the net, such as liveness, based on structural properties, like the deadlock-trap-property.

6.1. Fundamentals

There is an expert system integrated into the analysis section. Considering the relations between net characteristics proved theoretically, it draws conclusions from the results already obtained. Thus, in certain cases, an elementary property — such as that the total number of tokens is not altered by firing any transitions (the net is then conservative, respectively subconservative, if the number of tokens is decreased at most) — can be used to deduce a dynamic property like boundedness. On the other hand, superfluous analyses can be avoided by means of the expert system, because, for example, it is useless to investigate whether an inhomogenous net has the deadlock-trap-property.

Please note that INA returns used memory at the end of a computation. This is announced with the message `returning heap` and may last several seconds after memory-intensive procedures. Therefore, please wait until the memory is freed and the analysis can be continued.

6.2. Analysis status

The progress of the analysis is indicated in two status lines.

```
ORD HOM NBM PUR CSV SCF CON SC  Ft0 tF0 Fp0 pF0 MG  SM  FC  EFC ES
 Y   Y   Y   N   N   N   Y   N   Y   N   N   N   N   N   N   N   Y
DTP SMC SMD SMA CPI CTI  B  SB  REV DSt BSt DTr DCF  L   LV L&S WL  CL
 ?   N   ?   ?   N   ?   N   N   ?   ?   N   ?   ?   ?   ?   N   ?   ?
```

The possible properties of the considered net are listed. The symbol below each property indicates whether the property is fulfilled (Y), not fulfilled (N), or no decision could have been made yet (?). The first line contains elementary net characteristics, which are found during the pre-analysis by considering the structure of the net. The second line, on the other hand, consists of further dynamic properties, i.e. those depending on the initial marking, which may be clarified after further analysis. Of course, the status entries also depend on the selected net options and the results already found, as some properties only make sense under certain conditions. The example above displays all possibilities.

6.2.1. Elementary net properties

ORD **ordinary**

A net is said to be ordinary, if the multiplicity of every arc equals one [Sta90, Definition 2.3 (24)]. A coloured net is ordinary, if its unfolded net is ordinary.

HOM **homogenous**

A net is said to be homogenous, if for any place p, all arcs starting at p have the same multiplicity [Sta90, Definition 14.4 (157)].

NBM **non-blocking multiplicity**

A net has non-blocking multiplicity, if for each place p, the minimum of multiplicities of the arcs ending at p is not less than the maximum of multiplicities of the arcs starting at p [Sta90, Definition 15.3 (164)].

PUR **pure**

INA checks, whether there is a transition in the current net, for which a pre-place is also a post-place. In this case, the net is not pure, i.e. not loop-free [Sta90]. Such nets are not live under the safe firing rule.

CSV **conservative**

A net is said to be conservative, if all transitions add exactly as many tokens to their post-places as they subtract from their pre-places. In a conservative net, the total number of tokens is thus not altered by firing any transition. Therefore, the amount of tokens is an invariant. A net is sub-conservative, if all transitions add at most as many tokens to their post-places as they subtract from their pre-places. The total number of tokens can therefore not increase.

SCF **static conflict free**

If two transitions have a common pre-place, they are in a static conflict about the tokens on this pre-place. Then, the net is not static conflict free [Sta90, Definition 3.4 (35)].

CON **connected**

A net is said to be connected, if for each node in the net, there exists an undirected path to every other node [Sta90, Definition 14.1 (148)]. The direction of the arcs is thus ignored in the corresponding investigation.

SC **strongly connected**

If a net is connected, it is checked whether each node also has a directed path to every other node, i.e. the direction of the arcs is also considered [Sta90, Definition 14.1 (148)].

One-sided nodes

Ft0 **transition without pre-place**

A net has $Ft0$–transitions, if there are transitions without a pre-place; $Ft = \emptyset$.

tF0 **transition without post-place**

A net has $tF0$–transitions, if there are transitions without a post-place; $tF = \emptyset$.

Fp0 **place without pre-transition**

A net has $Fp0$–places, if there are places without pre-transitions; $Fp = \emptyset$.

pF0 **place without post-transition**

A net has $pF0$–places, if there are places without post-transitions; $pF = \emptyset$.

Structures

MG **marked graph**

A net is a synchronisation (or marked) graph, if every place has exactly one pre-transition and one post-transition. Here, arc multiplicities are ignored [Sta90, Definition 14.3.1 (152)].

SM **state machine**

A net is a state machine, if every transition has exactly one pre-place and one post-place. Here, arc multiplicities are ignored [Sta90, Definition 14.3.2 (152)].

FC **free choice**

A net has free choice, if every shared place is the only pre-place of its post-transitions [Sta90, Definition 14.3.3 (152)].

EFC **extended free choice**

A net has extended free choice, if the post-transitions of shared places have the same pre-places [Sta90, Definition 14.3.4 (152)].

ES **extended simple**

A net is extended simple, if the following holds: If two places have a common post-transition, then for one of them, its post-transitions are also post-transitions of the other place; i.e., one of the two places can also have other post-transitions [Sta90, Definition 14.3.5 (153)].

6.2.2. Further properties

DTP **deadlock-trap-property**

A net satisfies the deadlock-trap-property , if the maximal trap in each minimal deadlock is sufficiently marked [Sta90, Kapitel 15 (162–175)]. A trap is a set of places that, if it contains tokens, cannot become clean, because every transition which subtracts tokens from one place in this set has a post-place in this set, and thus returns tokens to the set. Hence, the empty set is a trap. A trap is maximal, if it is not a proper subset of a trap. A deadlock is a non-empty set of places that cannot be marked again once it is clean, because every transition which would fire tokens onto a place in this set has a pre-place in this set (and so cannot fire). A deadlock is minimal, if it does not properly contain a deadlock. A set of places is sufficiently marked, if it contains a place which contains sufficiently many tokens to enable all its post-transitions.

Further information can be found in chapter 6.7 on page 93ff.

SMC **state machine coverable**

A component of a net is a set of places which is both a deadlock and a trap. It defines a sub-net comprising these places and the adjoining transitions. If a set of components is given such that the corresponding sub-nets define state machines, and their union equals the set of all places in the net, then this net is said to be state machine coverable [Sta90, Definition 16.2.2 (180)].

Further information can be found in chapter 6.7 on page 93ff.

SMD **state machine decomposable**

If a set of components is given sucht that the corresponding sub-nets define strongly connected state machines, and their union equals the set of all places in the net, then this net is said to be state machine decomposable [Sta90, Definition 16.2.3. (180)].

Further information can be found in chapter 6.7 on page 93ff.

SMA **state machine allocatable**

In a state machine decomposable net, every transition has a pre-place; a pre-place allocation is a mapping which assigns to each transition one of its pre-places. A component is said to be selected in a pre-place allocation, if to all transitions that have a pre-place in the component, a pre-place in the component is assigned. The net is said to be state machine allocatable, if every pre-place allocation selects a strongly connected state machine component [Sta90, Definition 16.3 (182)].

Further information can be found in chapter 6.7 on page 93ff.

CPI — **covered by place invariants**

A net is covered by place invariants, if there exists a P–invariant which assigns a positive value to each place [Sta90, Definition 11.4 (114)].

CTI — **covered by transition invariants**

A net is covered by transition invariants, if there exists a T–invariant which assigns a positive value to each transition [Sta90, Definition 11.3 (112)].

B — **bounded**

A net is bounded, if there is a number k such that, in any reachable marking, there are never more than k tokens on a place [Sta90, Definition 14.1 (38)].

SB — **structurally bounded**

A net is structurally bounded, if it is bounded in every initial marking [Sta90, Bemerkung (116)].

REV — **reversible**

A net is reversible, if the initial state can be reached from every reachable state [Sta90, Definition 6.2 (59)].

A description of the reversibility test can be found in chapter 6.5.4 on page 80.

DSt — **dead state reachable**

A dead state is reachable, if a state is reachable in which no transition can fire any more [Sta90, Definition 6.1.1 (54)].

BSt — **bad state reachable**

If a state satisfies a so-called “bad” predicate (see chapter 6.5.4 on page 73), it is not further developed when computing a state graph. In this case, the attribute `Bst` is set. However, after leaving the graph analysis, this attribute is reset to `?`.

DTr — **dead transition at initial marking**

This attribute indicates whether the net has dead transitions in the initial marking, i.e. facts.

DCF — **dynamically conflict free**

A net is said to be dynamically conflict free, if no state is reachable in which two transitions are enabled and one of them can be disabled by firing the other. [Sta90, Definition 3.4.3 (35)].

L — **live**

A net is live, if all its transitions are live in the initial marking, i.e., no state is reachable in which a transition is dead. Here, facts are ignored [Sta90, Definition 6.1.5 (55)].

A description of the liveness test can be found in chapter 6.5.4 on page 79.

`LV` **live when ignoring dead transitions**

A net is live when ignoring dead transitions, if all its transitions, which are not already dead in the initial marking, are live. The transitions thereby ignored can be considered as unspecified facts.

`L&S` **live and safe**

A net is live and safe, if it is live and, in any reachable marking, not more than one token is on a place [Sta90, Definition 4.1 (39)].

Liveness for coloured nets

`WL` **weakly live**

A coloured net is weakly live, if all its transitions are weakly live, i.e. for each transition, there is a colour in which the transition is live in the initial marking [Sta90, Definition 20.6.1 (228)] (see also page 79 in chapter 6.5.4).

`CL` **collectively live**

A coloured net is collectively live, if all its transitions are collectively live. A transition is collectively live, if for every reachable state a colour exists, such that in a state reachable from this marking, the transition can fire in this colour [Sta90, Definition 20.6.2 (228)] (see also page 79 in chapter 6.5.4). In particular, every weakly live transition is also collectively live [Sta90, Folgerung 20.3.2 (229)].

6.3. Analysis menu and pre-analysis

In the analysis menu, different analysis procedures are offered depending on the net type. Not for every type of net, all of the procedures are already theoretically justified, or even reasonable. Clocked nets, for example, are equivalent to the TURING MACHINE, which excludes the existence of algorithms to decide e.g. boundedness. For each type of net, only those procedures are offered which can lead to statements about the important dynamic properties. To return to the main menu, enter `<Q>`.

```
Analysis menu:
Decide structural boundedness...............................................B
Non-reachability test of a partial marking using the state equation.........N
Compute the symmetries of the net...........................................Y

Compute a shortest path from the initial state to a target marking..........P
Compute a minimal path from the initial state to satisfy a predicate........O
Compute a reachability graph................................................R
Compute a coverability graph to decide boundedness and coverability.........G

Compute a basis for all P/T-invariants [non-reachability test]..............I
Compute a basis for all semipositive P/T-[sub/sur]-invariants...............S
```

```
Format lines written to INVARI.HLP earlier..................................F
Test place- or transition-vectors for invariant properties..................T

Check the deadlock-trap-, SMD-, SMA-properties [boundedness, liveness]......D
Compute all components [check SMD-,SMA-properties]..........................C
Decide state machine coverability [for boundedness].........................M

choice >
```

Pre-analysis

Before the analysis menu is displayed, INA executes a pre-analysis of the net, which investigates structural properties (see page 54 to 56), and also checks which functions are available for the given net.

At the beginning, you can set writing options. These include, for example, the output option for static conflicts: `Print the static conflicts?` With `<Y>`, they will be displayed.

Example

For the net 1 of the three programmers sharing two terminals, the pre-analysis provides the following results:

```
Information on elementary structural properties:
The net is not statically conflict-free.
The net is pure.
The net is not ordinary.
The net is not conservative.
The net is not subconservative.
The net is not a state machine.
The net is not free choice.
The net is not extended free choice.
The net is extended simple.
The net is not safe.
The net is not live and safe.
The net is marked.
The net is not marked with exactly one token.
The net is not a marked graph.
The net is not homogenous.
The net has not a non-blocking multiplicity.
The net has no nonempty clean trap.
The net has no transitions without pre-place.
The net has no transitions without post-place.
The net has no places without pre-transition.
The net has no places without post-transition.
Maximal in/out-degree:  3
The net is connected.
The net is strongly connected.
```

From that, INA can already conclude some properties:

```
ORD HOM NBM PUR CSV SCF CON SC  Ft0 tF0 Fp0 pF0 MG   SM  FC  EFC ES
 N   N   N   Y   N   N   Y   Y   N   N   N   N   N    N   N   N   Y
DTP CPI CTI  B   SB  REV DSt BSt DTr DCF  L   LV L&S
 ?   ?   ?   ?   ?   ?   ?   N   ?   ?   ?   ?   N
```

The static conflicts of the net are:

```
Static conflicts:
transition 1.login_Prog1 is in conflict with:
       2.login_Prog2,  3.login_Prog3,
transition 2.login_Prog2 is in conflict with:
       1.login_Prog1,  3.login_Prog3,
transition 3.login_Prog3 is in conflict with:
       1.login_Prog1,  2.login_Prog2,
```

6.4. Simplex oriented methods

In addition to the classical invariant analysis, further simplex oriented methods exist, which use systems of linear equations derived from the incidence matrix.

6.4.1. Fast boundedness test

In the analysis menu, a fast boundedness test can be executed. This can be useful in order to avoid the search for covering markings during a subsequent graph computation.

`<B>      Decide structural boundedness`

This function decides whether the net is covered by P–subinvariants. If this is the case, it is structurally bounded, i.e. bounded in every initial marking (see also the explanation in chapter 6.6 on page 82ff, and under `SB` in chapter 6.2.2 on page 57).

INA determines whether the system of equations $C \cdot x \leq 0$, $x \geq 1$ is solvable, where C is the incidence matrix of the net. For this purpose, the solvability of an associated auxiliary problem is considered. If it is solvable, the net is covered with P–subinvariants. During the computation, a counter named `eliminated columns` is running.

For the net 2 of the dining philosophers, the computation yield the following results:

```
Deciding structural boundedness
The net is structurally bounded.
The net is bounded.
There are no proper semipositive T-surinvariants.
The net is covered by semipositive place sub-invariants.
  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20
  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  2  2  2  2  2
```

In the last line, information on a covering P–subinvariant is displayed.

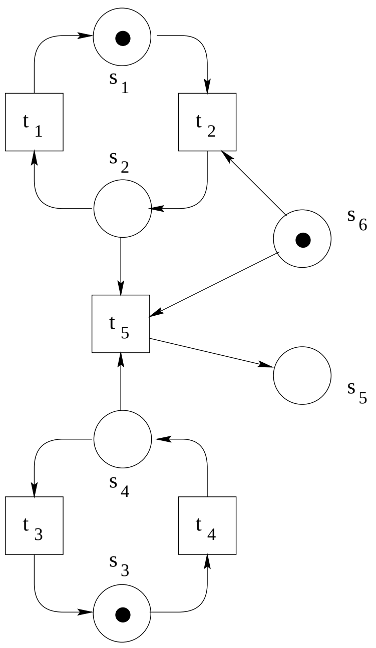
6.4.2. Fast non-reachability test

In the analysis menu, a fast non-reachability test for partial markings can be executed. This test exists alongside the non-reachability test when computing invariants (see on page 86 in chapter 6.6.1), and the reachability tests, which are based on a (partial) construction of the reachability graph (see on page 62ff in chapter 6.5).

`<N>`	`Non-reachability test of a partial marking using the state equation`

This function can decide the non-reachability of a marking using the state equation. A partial marking is sufficient here: the marking of the remaining places is considered to be not defined, i.e. dealt with by means of `don't care` conditions (see also page 70 in chapter 6.5.3).

INA determines whether the state equation $C \cdot x = m - m_0$, is solvable, where C is the transposed incidence matrix of the net, m_0 the initial marking, and m the parametrized partial marking. For this purpose, the solvability of an associated auxiliary problem is considered. If it is not solvable, the non-reachability can be deduced. During the computation, a counter named `eliminated columns` is running. If, however, the state equation is solvable, a solution is displayed — as far as non-reachability is concerned, no decision can be made.



Example 14: A net in which one place is not markable

For the net 14, it is clear that the place five is not markable, but with the non-reachability test by invariants, this decision cannot be made:

```
place invariants basis
```

```
=================
   1 |     1: 1,  2: 1,  3: -1,  4: -1
   2 |     1: 1,  2: 1,  5: 1 +

At least the following places are covered by semipositive invariants:
       1,  2,  5

Non-reachability test of the state:
  p :  1  2  3  4  5  6
m(p):  0  0  0  0  1  0

No conclusions possible.
```

Using the state equation, on the other hand, the non-reachability can be proved:

```
Testing non-reachability of a partial marking

target submarking:
   5:1;

The marking is not reachable.
```

For the net above, no statement about the (non-)markability of place two can be made by means of the state equation, since the latter is solvable:

```
Testing non-reachability of a partial marking
target submarking:
   2:1;
The state equation is solvable:
t2: 1    t4: 1
```

6.5. Reachability analysis

The computation of the state graph of a net is one of the most powerful analysis methods, limited only by the fact that even a small bounded net can have very many states. INA's methodology promises very efficient memory usage whilst allowing fast access.

6.5.1. Reductions of the state graph

In addition to an efficient use of memory, INA optionally offers possibilities to reduce the state graph already during the computation. The aim of the reduction is to find appropriate subgraphs which represent certain properties of the reachability graph. It is thereby possible, that nodes in these reduced graphs are representatives for whole classes of nodes of the reachability graph, or that only one part of the possible state transitions is considered, so that some arcs and nodes of the reachability graph will not be computed. Furthermore, representations exist for the infinite state graphs of unbounded nets.

The net 2 of the dining philosophers will serve as an example for state graph reductions. Its state graph has 393 nodes.

Symmetrical reduction

Many net models have symmetries, which can be exploited for the analysis.

When computing a state (or coverability) graph, all states which are mapped to each other by a symmetry can be identified. In some circumstances, the number of states to be computed is thereby considerably reduced, so that the overall computation (including the determination of the symmetries) can be faster than the one of the complete state graph. The disadvantage of the symmetrically reduced state graph is that, in general, it only allows to make weaker statements about liveness of transitions.

The symmetries are computed either separately, with the function `<Y>`, or automatically (in case they had not already been), prior to the construction of a symmetrically reduced graph. Only the generators are computed, from which the symmetries can be combined during the construction. In the worst case, a net has an exponential number of symmetries; the number of generators, however, is polynomial in net size. By generating symmetries from the formerly computed generators, an enormous amount of memory can be saved. The procedure implemented here, generally has a good run-time behaviour.

`<Y>`	`Compute the symmetries of the net`

When computing the symmetry group of the current net, INA considers possible time allocations.

First, you have to state whether fixed points should be set for places and transitions, or whether the initial state should be considered as symmetric: `Do you want to set fixpoints` or `initial state to be symmetric?`

Sometimes, other symmetries or none at all are found in this way, and the computation is cancelled: `Trivial transition partition` or `identity is the only symmetry`.

By answering the question `Write the symmetries to the session file` with `<Y>`, the generators of the symmetry group are written into the session-report. With `<N>`, only the decompositions of the place and transition sets into equivalence classes are written.

During the computation, a counter records the generators found; the running computation can be aborted with `<Q>`.

At the end of the computation, the number of generators and the number of symmetries, which can be obtained by combining them, are displayed.

The function `<Y>` can also be used to have a (already once computed) symmetry group be re-computed. In order to do this, you only have to answer the question `compute the symmetries once again?` with `<Y>`. For example, you can set new fixed points, or consider the initial marking.

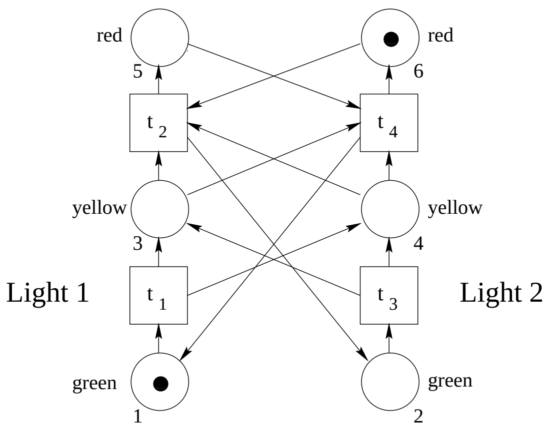
For the net 1 without multiplicities, the computation looks like the following:

```
......................Do You want to set fixpoints? Y/N N
.....................Initial state to be symmetric? Y/N Y
.........Write the symmetries to the session file? Y/N Y

Generators:          3
```

```
Generating set of the symmetries:

* Places:      (2 3)(5 6)
  Transitions: (2 3)(5 6)

* Places:      (1 2 3)(4 5 6)
  Transitions: (1 2 3)(4 5 6)

* Places:      (1 3 2)(4 6 5)
  Transitions: (1 3 2)(4 6 5)

3 generators for 6 symmetries found!

Classes of equivalent places:
   1:  4.prog1_on_break,  5.prog2_on_break,  6.prog3_on_break,
   2:  1.prog1_at_term,  2.prog2_at_term,  3.prog3_at_term,
   3:  0.terminal_free,

Classes of equivalent transitions:
   1:  4.logout_prog1,  5.logout_prog2,  6.logout_prog3,
   2:  1.login_prog1,  2.login_prog2,  3.login_prog3,
```

First, the set of generators is displayed in cyclic notation: the first generator means that place 2 is mapped to place 3 and place 5 to place 6 (and vice versa); similarly for the transitions. The first part of the second generator means that place 1 is mapped to place 2, place 2 to place 3, and place 3 to place 1.

Afterwards, the equivalence classes are outputted: the places 1, 2, and 3 are equivalent, as are the places 4, 5, and 6. Place 0 is a singleton set with respect to the decomposition. As for the transitions, 1, 2, and 3 are eqivalent, and so are 4, 5, and 6.



Example 15: Two traffic lights

The net 15 is (in contrast to net 1) symmetric only without consideration of the initial marking. It models two traffic lights: the places correspond to the colours of the traffic lights (note that this is, nevertheless, not a coloured net). The transitions provide for the necessary traffic security.

A run of the symmetry computation of this net can look like the following:

```
Computing the symmetries
Do You want to set fixpoints? N
Initial state to be symmetric? Y
Identity is the only symmetry!

Compute the symmetries once again? Y
Do You want to set fixpoints? N
Initial state to be symmetric? N

Generating set of the symmetries:

* Places:      (3 4)
  Transitions:

* Places:      (1 2)(3 4)(5 6)
  Transitions: (1 3)(2 4)

2 generators for 4 symmetries found!

Classes of equivalent places:
   1:  5.red_one,  6.rot_two,
   2:  3.yellow_one,  4.yellow_two,
   3:  1.green_one,  2.green_two,

Classes of equivalent transitions:
   1:  2,  4,
   2:  1,  3,
```

Since the philosophers of net 2 have a symmetric behaviour, the computation of the symmetry group is worthwhile: the symmetrically reduced reachability graph has only 47 nodes.

For details, see [Sta90, Kapitel 7 (62–73)], [Sta91], and [SS91]. An up-to-date description of the implementation can be found in [Sch97].

Stubborn reduction

Stubborn reduction is another possibility to decrease the number of nodes of the reachability graph. It can be combined with symmetric reduction; however, it is not available for all net types.

A freedom in the order of firing the transitions (as implied by concurrency) leads to an increase in the number of reachable states. When investigating two different paths leading to the same state, one may find that these paths differ only with respect to the order in which independent, i.e. concurrent transitions are fired. The state graph contains all intermediate states, although some of them are redundant. The aim of stubborn reduction is to construct a smallest possible excerpt from the state graph. In a given marking, only a subset of transitions is fired. This subset is chosen in such a way, that it is influenced as little as possible from other transitions, or only in a specific way already understood. This procedure makes substantial use of the locality principle as well as of concurrency. In this computation, it is in general not sufficient to consider only the enabled transitions, but also some of the transitons which are not enabled. Because it is hard to influence the resulting set, it received the name *stubborn set*.

The most important properties which can still be deduced are the existence of dead markings and infinite paths.

In net 2, the philosophers can act concurrently. The stubbornly reduced reachability graph has 183 nodes. Taking into consideration the previously mentioned symmetries of the net, a reachability graph results from stubborn and symmetric reduction, which is composed of only 28 nodes. In some sense, symmetric reduction is orthogonal to stubborn reduction. Altogether, a possibly drastic reduction of the state graph results — in the example net 2, by a factor of ten.

Coverability graphs

For unbounded nets, coverability graphs provide a complete overview of the boundedness of places and the coverability of markings. Their computation is possible only for nets without time allocation and priorities, using the normal firing rule, and therefore is not offered in any other context. INA computes the graph using the algorithm of Karp and Miller. In case the net is bounded, the coverability graph corresponds to the usual state graph.

For details, see [Sta90, Kapitel 5 (43–53)].

Finkel's Method of computing the minimal coverability graph

In [Fin89], Finkel considers reachability graphs and coverability graphs as special cases of a more general structure, so-called ω–state graphs. Among all these state graphs, there exists a unique one which is minimal with respect to the number of nodes. For nets without time allocation and priorities, and under the normal firing rule, INA offers the possibility to compute the coverability graph according to Finkel's method, and thus reduce the number of nodes and arcs during construction. For unbounded nets, a graph results which has fewer nodes than the classical coverability graph. For bounded nets, the additional time expense lies within acceptable limits, but no essential reduction is achieved. The knowledge of the minimal ω–state graph is sufficient to decide properties such as boundedness of the net, coverability of markings, or the existence of infinite paths in the reachability graph.

The net 16 is usually considered using the step firing rule, and is then bounded. If this net is analyzed without the step firing rule, it no longer models the `alternating bit protocol` correctly. Since it is unbounded in this case, it is nonetheless introduced here as an example for a drastic reduction of the state graph. The computation of the classical coverability graph requires several days; the graph has 435316 nodes. Using Finkel's method, a graph results with 72 nodes, the computation only lasts a few seconds.

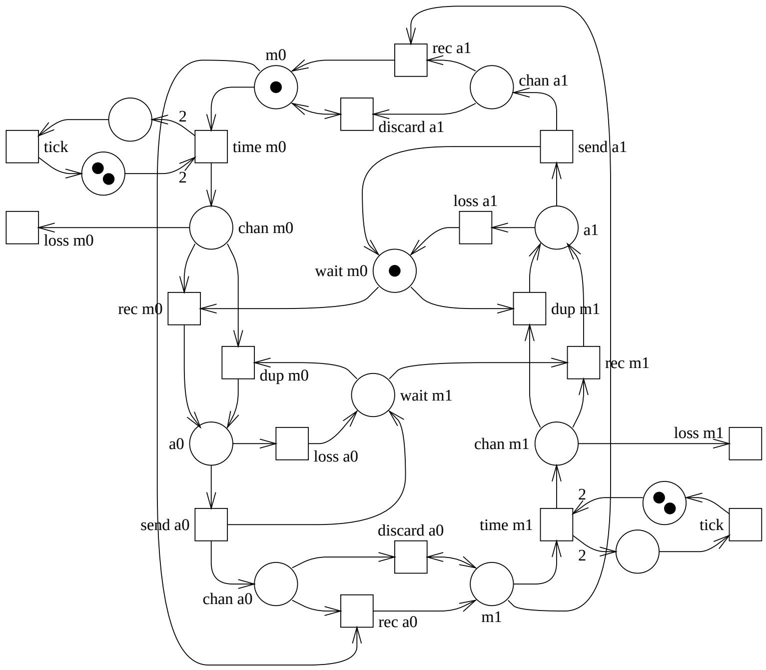
For definitions and details of the implementation, see the diploma thesis [Lü95].

6.5.2. Predicates and CTL model checking

In this section, you will find a short introduction into the syntax and semantics of CTL (Computational Tree Logic), and a description of the state predicates used in INA.

State predicates

During the computation of minimal paths (see chapter 6.5.3 on page 71ff), the reachability analysis (see under "bad" predicate on page 71 in chapter 6.5.4), and the model checking (see on page 75ff in chapter 6.5.4), INA works with state predicates.



Example 16: The alternating bit protocol

A predicate is a disjunction of conjunctions of possibly negated atoms (disjunctive normal form). An atom consists of a statement of the form

$$low \leq value \leq upp \qquad\qquad (\text{where } 0 \leq low \leq upp \leq \infty)$$

As an example for the state predicates used in INA, $value$ =number of tokens on a place; an atom p_1 : $low = 1$, $upp = 2$; is then satisfied by all states in which place p_1 contains one or two tokens. For clocked transitions, restrictions of the clock positioning can be formulated in the same way.

When a predicate is requested at the `predicate input file` prompt, you can switch to the terminal mode by pressing `<esc>`. If so, enter the first conjunction of atoms: After entering a place number, the lower and upper bounds are asked for (`lower bound`, `upper bound`). With `<o>`, you can enter *unbounded above.* You can negate the atom or not (`negate the atom?`), and then enter another state number, or quit the current conjunction with `<Q>`. When working with time allocation, further state atoms can be defined with respect to time restrictions.

If you answer the question `next conjunction` with `<Y>`, you get to the definition of the next conjunction of (possibly negated) atoms.

Predicates are saved in files with the extension `.pdc`.

In chapter 6.5.4 on page 75ff, you find a set of functions with which certain predicates can be generated automatically. In the atomic mode described there, simpler atoms, which represent the alternative whether a place is marked or not, can also be used as predicates.

Model checking

Testing the validity of CTL–formulae in the initial state is called model checking. Thereby, witnesses for existence-quantified sub-formulae, and counter examples for all-quantified sub-formulae can be determined and displayed. On the computation procedure and the entering of formulae, see chapter 6.5.4 on page 75; on the implementation, see [Hel96].

Model checking is based on the structure $M = [S, R, P]$ of a labelled state transition graph where

- S is an finite state set
- R is a binary relation on S ($R \subseteq S \times S$), which describes the possible state transitions.

  This relation is assumed to be total: for every state s, there exists a state $s' \in S$ with $(s, s') \in R$.
- P is a mapping from the state set onto the power set of the predicates.

  For a given state, this mapping yields the set of predicates which are satisfied in it.

Syntax of CTL

The syntax of CTL–formulae is defined inductively:

- Each predicate p is a CTL–formula.
- If f_1 and f_2 are formulae, then so are
  - $\neg f_1$, $(f_1 \wedge f_2)$ and $(f_1 \vee f_2)$ (logical connectors)
  - $EX\,f_1$, $EF\,f_1$, $EG\,f_1$, $AX\,f_1$, $AF\,f_1$, $AG\,f_1$, $E[f_1\,U\,f_2]$, $A[f_1\,U\,f2]$, $E[f_1\,B\,f_2]$, $A[f_1\,B\,f2]$ (temporal-logical quantors)

Semantics of CTL

A path is an infinite sequence of states $(s_0, s_1, \ldots)$, so that $(s_i, s_{i+1}) \in R$ holds for all i.

The relation $M, s_0 \models f$ means that the CTL–formula f is satisfied in the state s_0 within the given structure M.

We use the following inductive definition of $\models$:

- $M, s_0 \models p$ iff $p \in P(s_0)$
- $M, s_0 \models \neg f$ iff not $M, s_0 \models f$
- $M, s_0 \models (f_1 \wedge f_2)$ iff $M, s_0 \models f_1$ and $M, s_0 \models f_2$
- $M, s_0 \models EX\,f$ iff for a state s' with $(s_0, s') \in R$, $M, s' \models f$

  There exists an immediate post-state in which f is satisfied.

- $M, s_0 \models A[f_1\, U\, f_2]$ iff for all paths $(s_0, s_1, \ldots)$, there exists an $i \geq 0$ with $M, s_i \models f_2$, and $M, s_j \models f_1$ holds for all $0 \leq j < i$

  Along all paths, f_1 is satisfied until a state is reached which satisfies f_2.

- $M, s_0 \models E[f_1\, U\, f_2]$ iff for a path $(s_0, s_1, \ldots)$, there exists an $i \geq 0$ with $M, s_i \models f_2$, and $M, s_j \models f_1$ holds for all $0 \leq j < i$

  There exists a path, along which f_1 is satisfied until a state is reached which satisfies f_2

- $M, s_0 \models A[f_1\, B\, f_2]$ iff for all paths $(s_0, s_1, \ldots)$, there exists an $i \geq 0$ with $M, s_i \models f_2$, and $M, s_j \models f_1$ holds for a $0 \leq j < i$

  On all paths, f_1 is satisfied at least once, before a state is reached, which satisfies f_2

- $M, s_0 \models E[f_1\, B\, f_2]$ iff for at least one path $(s_0, s_1, \ldots)$, there exists an $i \geq 0$ with $M, s_i \models f_2$, and $M, s_j \models f_1$ holds for a $0 \leq j < i$

  There exists a path along which f_1 is satisfied at least once, before a state is reached which satisfies f_2

The entering of CTL–formulae is described in chapter 6.5.4 on page 75ff.

Equivalences and further notes

Among others, the following equivalences hold:

- $AX(f) \equiv \neg EX(\neg f)$

  f is satisfied in all post-states

- $AF(f) \equiv A[True\, U\, f]$

  f is satisfied on every path in the future

- $EF(f) \equiv E[True\, U\, f]$

  There exists a path which leads to a state in which f is satisfied

- $AG(f) \equiv \neg EF(\neg f)$

  f is satisfied in every state on all paths

- $EG(f) \equiv \neg AF(\neg f)$

  There exists a path where f is satisfied in every state

The semi-formal significance of the individual quantors is underlined by the symbols used: `E` stands for "exists", `A` for "always", `X` for "next", `U` for "until", `B` for "before", `F` for "forward", and `G` for "globally".

6.5.3. Computation of paths

The computation of paths is based on the partial computation of the state graph, and therefore belongs to the reachability analysis.

For nets with time allocation, note that transient states are not considered. If your marking appears only in a transient state, INA will not find it and will regard it as non-reachable, provided the entire state graph can be constructed.

`<P>` **Compute a shortest path from the initial state to a target marking**

A marking is tested for reachability, and, if possible, a path is displayed. The path is shortest with respect to the number of firing procedures. Here, a partial marking is sufficient. The marking of the remaining places is considered as not defined, i.e., treated as `don't care` conditions.

The marking to be examined can be read from a file or, by pressing `<esc>`, entered directly.

For a net with time allocation, in addition to the computation of a shortest path, it is also possible to compute a path which is minimal with respect to time (i.e. a fastest one) (`compute a minimal time path`). Time allocation will be inquired anyway.

After entering (or reading) the target marking, INA starts to construct the state graph of the net breadth first. The number of states so far computed is displayed: `States generated`.

If INA encounters a marking that agrees with the target marking on the places defined, the examination is cancelled. The target marking is reported on the screen as reachable. The path can be written into a separate file with the extension `.tra` or in the session report.

If the target marking is not reachable, there are two cases: if the entire state graph can be constructed and saved, the marking will be regarded as unreachable (`The marking is not reachable`); otherwise (always in unbounded nets), INA cancels with the message `Node overflow` and states that no decision can be made: `No decision possible`.

By pressing `<Q>`, the computation process can be cancelled at any time.

Any dead states which may have been found are taken into account in the further analysis.

If the method of stubborn reduction can be used (see page 65 in chapter 6.5.1), a net construction is used to construct the state graph stubbornly reduced in such a way that a minimal path to the partial target marking will be nonetheless found.

Display of results

Using the example of net 1, the computation of a path is shown. The partial marking thereby used here means that the programmers two and three are working at the terminal.

```
Computation of a shortest path.
target submarking:
   2:1;   3:1;
```

```
The marking is reachable.

Trace of length 2
State nr.    1
P.nr: 0 1 2 3 4 5 6
toks: 2 0 0 0 1 1 1
   n1====2.login_prog2     ==> n3
State nr.    3
P.nr: 0 1 2 3 4 5 6
toks: 1 0 1 0 1 0 1
   n3====3.login_prog3     ==> n5
State nr.    5
P.nr: 0 1 2 3 4 5 6
toks: 0 0 1 1 1 0 0

States generated: 6
```

In this example, two firing procedures are necessary in order to get from the initial marking to the target marking. In the initial marking n1, the state n3 is reached by firing transition 2, and in that state, the target marking in state n5 is reached by firing transition 3. Or simpler: if, in the initial situation, both programmers two and three log in, the given target marking is reached.

Further path computations

In addition to the path computation described above, there are further possibilities:

`<O>` Compute a minimal path from the initial state to satisfy a predicate

This function computes a minimal path to a state which satisfies a previously specified state predicate (on the definition and entering see page 66ff in chapter 6.5.2).

In contrast to the path computation with `<P>`, the following minimality criteria are available:

```
Computation of a minimal path to a state satisfying a predicate:
'minimal' to be understood as minimal length........L
'minimal' to be understood as minimal time..........T
'minimal' to be understood as minimal value.........V
```

Miminal length and minimal time are defined in an analogous way to the functionality described for `<P>`. When computing a path with minimal value, to each transition a value is assigned, first. These values can be read from a file with the extensions `.val`, or, upon pressing `<esc>`, entered directly at the `values input file` prompt. The value of a path is the sum of the values of the fired transitions (on the use of evaluations see also page 78ff in chapter 6.5.4).

After reading a predicate, you can restrict the search by specifying a so-called “bad” predicate: if the predicate is satisfied by a state, the state will not be further developed (see also page 73 in chapter 6.5.4).

Otherwise, the running and result display of this function is similar to the one with `<P>`.

6.5.4. Computation and analysis of graphs

In this subsection, you will find a description of the computation and analysis of a state (or coverability) graph, starting with a description of the options and the course of a session.

For the graph computation, INA offers the following functions:

<R>	Compute a reachability graph

This command constructs the state graph, which can be computed using symmetrical and/or stubborn reduction. Subsequently, different graph analyses can be executed, CTL–formulae and predicates can be created or checked, and certain results can be written either into separate files or in the session report.

<G>	Compute a coverability graph

This command constructs the coverability graph, which can be computed using symmetrical reduction and/or according to FINKEL'S method. A further analysis of the coverability graph is not possible.

Options

At the beginning of the computation, INA displays the possible options:

```
Current analysis options are:
     no symmetrical reduction ........................................Y
     no stubborn reduction ...........................................U
     no depth restriction ............................................D
     do not use a 'bad' predicate ....................................B
```

The following settings can be made:

Calculation mode

symmetrical/stubborn reduction

With the calculation mode, you can select the previously presented graph reductions. If possible for the current net, you can select a symmetrical or stubborn reduction of the graph with <Y> or <U>, respectively. In the case of a reachability graph, these two possibilities (if offered) can be combined in any way. For coverability graphs, only symmetrical reduction is available. If the symmetry group has not already been calculated, that will be done following the setting of the options. On this, see the description of the command <Y> on page 63 in chapter 6.5.1.

reduction by Finkel's method

For a coverability graph, you can additionally choose a reduction by FINKEL'S method with <R>.

Depth restriction

depth restriction

With the option <D>, you can set the depth level to which the state graph is to be computed.

By entering depth 0, the depth restriction is cancelled.

Bad predicate

`bad predicate`

Using a bad predicate, it is possible to restrict the computation of the state graph: If the predicate is satisfied by a given state, this state will not be developed further, and the property `BSt` is set (see chapter 6.2.2 on page 57). On entering predicates, see the description on page 66 in chapter 6.5.2. For coverability graphs, the use of such a predicate is not possible.

Reachability or coverability test

`reachability/co-verability test`

For coverability graphs, you can test for the reachability or coverability of a marking with this option (`<T>`), following the graph computation. For reachability graphs, however, this option is available as a function in the graph analysis menu (with `<R>`). For a description see also page 75.

Course of the computation

At the beginning of the computation, INA checks whether the boundedness of the net is known so far. If not, you will be asked whether testing for covering markings should be omitted during the graph computation: `omit boundedness test?` If you answer with `<Y>`, or if INA disables the test automatically, the computation can be executed substantially faster than with the test activated. Therefore, consider executing a test for structural boundedness before computing a graph.

The progress of the graph computation is visualized by the number of states created.

The computation can be cancelled at any time by entering `<Q>`. In case the construction of the graph cannot be completed because of memory restrictions, INA cancels with the message `Nodeoverflow`. The computation of a state graph is interrupted, if INA discovers that the current net is unbounded (and no depth restriction was set). In both cases, the computed graph is not complete: `computed graph is not complete`.

At the end of the graph computation, an overview of the states and arcs generated, as well as the capacities needed is written into the session report: for each place, it is shown how many tokens where maximally contained in it during the computation. For an incomplete state graph computation, only the computed markings will thus be considered. Thereafter, any dead or “bad” states which were found can be outputted (e.g. `write the state numbers of the dead states?`).

For the net 2 of the dining philosophers, the first results look like the following:

```
States generated: 393
Arcs generated: 1420

Capacities needed:
Place  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20
 Cap:  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1
Dead states:
       6, 322,
Number of dead states found: 2
```

The computation of the coverability graph is not cancelled if the net is unbounded; rather, the marking is set to `oo` (ω) at the corresponding places. At the beginning, INA states the largest amount of tokens allowed on a bounded place: `It is assumed that all bounded places are 4294836224-bounded`. This constant is dependent on the implementation, and is in any case smaller than the internal value of `oo` (ω — the given value — corresponds to the one in the UNIX implementation).

If, on the other hand, the net is bounded, the computed graphs are identical. The algorithm of Karp and Miller then computes the state graph (which, however, cannot be further analyzed; therefore, please use the function `<R>` in this case).

If you have selected Finkel's method for the computation of the coverability graph, then the screen output during the computation is different: INA counts the number of arcs computed, and displays the number of nodes generated only at the end.

For a coverability graph, a test for reachability or coverability of a marking can be executed (see the description below). No further analysis of the coverability graph is possible.

Further graph analysis

After constructing the state graph, first deductions are drawn. Then, the graph analysis menu is displayed (not for coverability graphs):

```
Graph analysis menu
Do You want to
     quit analysis of the computed graph ................ Q

     test the reachability/coverability of a marking ..... R
     convert a set of states to a predicate ............. C
     define an enabledness predicate .................... E
     check a CTL-formula ................................ F
     compute distances .................................. A
     compute circuits ................................... K
     check liveness properties .......................... L
     compute strongly connected components .............. V
     check dynamic conflicts ............................ Y

     write the computed graph (states and arcs) .......... W
     write all arcs ..................................... X
     write all states ................................... M
     write all states satisfying a predicate ............ P
     write the dead states .............................. D
     write the bad states ............................... B
     write a trace to a state ........................... T
     write the list of executed steps ................... S
     inspect a result file ...............................I
choice>
```

The individual functions will be explained in the following. With `<Q>`, you can quit the graph analysis and return to the main menu.

Reachability/coverability test

`<R>`	`test the reachability/coverability of a marking`

With this function, you can execute reachability or coverability tests, and find (not necessarily minimal) paths from the initial marking to a target marking.

For coverability graphs, this function is selectable as a graph computation option (by entering <T>, see page 73).

INA first requests the number that the marking obtained during the construction of the state graph. By entering <N>, you can, of course, read a marking from a file, or enter it directly. Please note: under time allocation, transient states are not considered.

If the marking is not reachable, INA states this, and offers a coverability test. If this test also fails, the marking is reported as not coverable. Otherwise, a covering marking is displayed. By answering the question `next?` with <Y>, you can display further covering markings , until INA reports that no more exist: `no more covering marking found`. Under time allocation, the marking is presented as a component of a state when displayed.

If symmetrical reduction of the graph is selected as an option, you have to take into consideration that the results obtained may be less conclusive; it may happen that only an equivalent marking is found: `by symmetry the marking is reachable`. If the initial marking is not symmetrical, reachability is, in general, not decidable: `no conclusion on reachability possible`. For the coverability test, similar considerations apply; then only an equivalent covering marking can be found: `no conclusion, but equivalent covering marking found`.

Should a marking be reachable or coverable, you can calculate a path to this marking, (`backtracking`) and, if desired, write it into a separate file with the extension `.tra`. If you answer the question `exit?` negatively, you can display further paths.

If you want to test another marking, answer the question `once more?` with <Y>, otherwise the graph analysis menu appears on the screen.

CTL model checking and predicate definition

Besides entering predicates directly (see page 66ff in chapter 6.5.2), the following functions for defining predicates are available:

`<C>`	`convert a set of states to a predicate`

With this command, a predicate for a set of states is defined which is satisfied exactly by these given states. You can construct the predicate with respect to either the dead states (answer the question `convert the set of all dead states?` with <Y>), or a set to be specified (answer with <N>). In the latter case, states are requested at the `state nr`–prompt, until you enter <Q>. The predicate defined can be saved in a file with the extension `.pdc`.

`<E>`	`define an enabledness predicate`

With this function, a predicate is defined based on a transition set to be specified; this predicate is satisfied exactly by those states in which at least one transition of the set is enabled:

```
We define a predicate satisfied by all states which enable at least one
of the transitions to be given.
```

Transitions are requested at the transition nr prompt, until you enter <Q>. The predicate defined can be saved in a file with the extension .pdc.

Formulae of the Computational Tree Logic (CTL) can be checked based on (previoulsy defined) predicates (see page 66ff in chapter 6.5.2):

<F> check a CTL--formula

With this function, you can check CTL–formulae, i.e. determine their validity and generate proofs.

First you have to decide, if you want to work in the atomic mode: Use the atomic mode? At the formula input file prompt, you can then read a .ctl file, or, by pressing <esc>, enter a formula directly. Please only use formulae files created by INA; the syntax is displayed when entering manually:

```
<formula> ::= 'T' | 'P'<number> | '-'<formula> |
              '('<formula>'&'<formula>')' | '('<formula>'V'<formula>')' |
              'E''X'<formula> | 'E''F'<formula> | 'E''G'<formula> |
              'A''X'<formula> | 'A''F'<formula> | 'A''G'<formula> |
              'E''['<formula>'U'<formula>']' | 'A''['<formula>'U'<formula>']'
              'E''['<formula>'B'<formula>']' | 'A''['<formula>'B'<formula>']'
 <number> ::= <nonempty sequence of digits followed by a blank>
In the atomic mode, 'P'<number> holds iff the place <number> is marked.
```

T stands for the truth value “true”, predicates are referred to with P<number>, in the atomic mode P<number> denotes the criterion whether the place with the internal number <number> is marked or not, - stands for the logical “not”, & and V stand for the logical connectors “and” and “or”, respectively, and EX, EF, EG, AX, AF, AG, E U, E B, A U, and A B stand for the corresponding temporal-logical quantors (see chapter 6.5.2 on page 66ff). Please note the different bracketings with ( ) or [ ], as well as the entering of a blank at the end of a predicate number. For faulty entries, a list of expected symbols appears: expected. INA inserts any missing closing bracket automatically. In some cases, corrections may be impossible; in case of mistakes, still enter all required symbols, and answer the question ok with <N>. Larger formulae can be formatted by using <cr> and <space>.

After entering <Y>, you can write the formula into a file with the extension .ctl. Thereafter, the predicates whose numbers were stated in the formula are requested. You can either read previously defined predicates from a file, or, by pressing <esc>, enter predicates directly. Predicates are not considered in the atomic mode.

If the reachability graph contains dead or “bad” states, or if it was not completely computed due to a depth restriction or insufficient memory, you can insert a loop at each leaf of the graph tree by answering the question consider the structure as total with <Y>. In this way, statements about infinite paths are obtained, even if the graph contains end vertices. Should INA discover end vertices without the graph structure declared total, then no statements can be made: No decision because dead graph nodes exist (see also page 66 in chapter 6.5.2).

You can have a proof constructed by INA which contains either only witnesses and counter examples, or yet additional information. You can write this proof into a file with the extension

`.prf`. In proofs you find information on predicates which are satisfied (`sat`) or not, and on possible states which are already on the path (`is on path`), or have already been `visited`.

At the end of a test, INA gives an answer which, for symmetrical and/or stubborn reduction as well as for depth restrictions, is obviously restricted to the subgraph computed: `The formula is TRUE/FALSE (in the computed subgraph)`.

Computing distances

`<A>`	`compute distances`

With this command, minimal and maximal distances between nodes of the state graph can be computed. The results are written into the session report.

If you answered the inquiry `compute minimal distances` with `<Y>`, the minimal distances of the graph node entered at the `source node` prompt are computed. The maximal distance found in this way is displayed, as well as a list of non-reachable nodes. All distances from the given node to any other node can be listed following `print all the distances`. E.g., for the net 2, the minimal distance from the initial state to the state 322 (all philosophers have their right fork) is shown to be `distance(1,322)=5`. The maximal distance from the initial state which has been found is seven: `maximal computed distance = 7 to node nr. 33`. It is possible that only upper bounds are found.

If you answer the inquiry mentioned above with `N`, INA computes the maximal distances. At the beginning, the following warning is issued:

```
Computation of a maximal (loop-free) distance is very hard, if the graph is
not circuit-free. If a graph with circuits is assumed to be circuit-free, then
only a lower bound for the maximal distance is (faster) computed.
```

If, in a graph which is not circuit-free, you are only interested in a lower bound, then answer the question `assume the graph to be circuit-free` with `<Y>`. The computation can otherwise be very expensive, and may not come to a result within a reasonable time. You can enter the numbers of two nodes of the state graph between which the maximal (ciruit-free) distance is to be computed. During the search, the amount of paths currently found is displayed; at any time, the running computation can be aborted with `<Q>`. At the end, either a lower bound or the maximal distance is displayed as the result, depending on whether the latter was found.

Please note: without time allocation, in both procedures the number of fired transitions (or steps) is counted. With time allocation, you have the possibility, by answering the question `count times` with `<Y>`, to consider the time passed meanwhile, instead.

Computing circuits

`<K>`	`compute circuits`

With this command, circuits in the state graph can be computed and evaluated.

At the beginning of the circuit computation, the following options appear:

```
Work with transition values ..................................V
Print only selected circuits ................................A
Print circuits with minimal length ..........................B
Print circuits with maximal length ..........................C
Print circuits with minimal value ...........................D
Print circuits with maximal value ...........................E
Print circuits with best efficiency .........................F
Print circuits with least efficiency ........................G
Print only circuits containing state nr.     0 ..............H
Print not only the first circuit of the selected type .......I
```

With <A> you can determine whether all circuits should be displayed. If you select this option, the only other option available is <V>. Otherwise, you can display the circuits with minimal or maximal lengths with <B> or <C>, respectively. The length is given by the number of fired transitions (or steps).

With <V> you can request to work with evaluation of transitions. Afterwards, there are further options available: with <D> or <E> you can display circuits with minimal or maximal values, respectively. The value of a circuit is the sum of the values of the transitions in the circuit. The efficiency is defined as the quotient of the value and the length of the circuit. You can select circuits with minimal or maximal efficiency with <F> or <G>, respectively.

With <H> you can select circuits which contain a certain state. By entering the number of a state of the graph, only those circuits which contain this state are displayed. The state `0` is never contained in the graph, and serves to display all circuits while considering the remaining options.

By selecting <I>, only the first circuit of each selected type is displayed.

After the options have been set, the computation starts. If you have chosen to work with evaluation of transitions, a file with the extension `.val` is requested at the `values input file` prompt. By pressing <esc>, you can also enter the transition values directly at the terminal: after entering a transition number, its name is indicated, and the corresponding value (cost) is requested. When you have quit the entering of a number with <Q>, the values of the transitions not yet specified are set to zero. You have the possibility to write the entered values into a file.

The progress of the computation is indicated by a counter of computed circuits. You can, of course, abort the running computation at any time with <Q>.

The circuits are written into the session report, or, upon request, into a file with the extension `.cir`. Every circuit is outputted in reversed orientation. `Nx<-ty--Nz` means that the transition with the number y is part of the circuit, and that firing transition y leads from the state z to the state x. The number of the states are referred to the previously computed list of states.

The following is an example for the net 15:

```
Circuit nr. 1:
N1<-t4--N4<-t3--N3<-t2--N2<-t1--N1
Corresponding T-invariant:
TR: 1 2 3 4
  : 1 1 1 1
Livelocked transitions: none
Circuit length = 4
```

The circuit consists of the transitions $t1$, $t2$, $t3$, and $t4$, and leads from the initial state back to itself.

In addition, the respective transition invariants are displayed. These invariants correspond to realizable T–invariants (see page 82 in chapter 6.6).

Furthermore, transitions are displayed which are in a livelock. A transition t is in livelock, if there exists a circuit in the state graph in which t does not fire [Sta90, chapter 12 (123–133)]. The corresponding net can then fire continuously, without the live transition t being fired; t "starves". This becomes particularly clear examining the circuits in the reachability graph of the net 2 of the dining philosphers, in which there are a lot of transitions which can be in a livelock.

Liveness test

`<L>`	`check liveness properties`

With this command, a liveness analysis can be executed. This works only for completely computed state graphs without stubborn reduction.

For nets executed using the maximal step firing rule, the liveness of steps can be investigated: `Test liveness of steps?`.

If the net contains transitions which are dead in the initial state (see page 57 in chapter 6.2.2), then the liveness analysis is restricted to the transitions that are not dead: `Liveness analysis refers to the net where all dead transitions are removed (considered as facts).` The net can be live if all dead transitions are considered as facts; the property `LV` (Liveness when ignoring dead transitions) is then set accordingly (see also page 58 in chapter 6.2.2).

Further weaker notions of liveness are explained on the screen, if necessary: on the one hand, weak liveness

```
A transition is weakly live IFF it has a live colour.
```

and on the other hand, collective liveness

```
A net is collectively live IFF every transition is collectively live at the
initial state.
A transition t is collectively live IFF for every reachable state m there
exists a colour c of t and a state m' reachable from m such that t has
concession in colour c at m'.
```

On this see also the explanations for `WL` and `CL` on page 58 in chapter 6.2.2.

Possibly, transition sets are displayed which are not weakly live, or which are collectively live:

```
the following transitions are not weakly live
```

or

`the set of collectively live transitions`,

respectively.

Strongly connected components and reversibility

`<V>     compute strongly connected components`

With this function, the strongly connected components are computed, and, upon request, written into a file with the extension `.res`.

An overview of the strongly connected components in the state graph is displayed. All nodes of the graph which are in the same component are reachable from each other along directed paths. Following the component number, the contained states are listed. Afterwards, the reachable components are shown (`reachable scc's`).

Terminal connected components, i.e. components from which no arcs are starting, are indicated with `term`. Dead states always form a terminal component with exactly one state.

For the net 2, three components are displayed. The first one consists of a total of 391 states. Two other components are reachable from this first component (`Reachable scc's: 2, 3`). They are both terminal and singleton components: `2: 322, term` and `3: 6, term`. These are exactly the components of the dead states.

If the entire graph is strongly connected, the computation has not been aborted beforehand, and you have not requested stubborn reduction, then INA is able to deduce that the graph is reversible (see in chapter 6.2.2 on page 57).

For nets under time allocation, a list of time deadlocks is displayed, if necessary.

Computation of dynamic conflicts

`<Y>     check dynamic conflicts`

With this function, you can search the set of reachable states for transitions in dynamic conflict (see in chapter 6.2.2 on page 57).

If, in a given state i, two transitions j and k are in a dynamic conflict, the following entry is created in the session report: i : j # k.

For the net 1, only two lines are outputted: `1: 1.login_prog1 # 2.login_prog2` and `1: 1.login_prog1 # 3.login_prog3`, i.e. in the initial state, the transitions 1 and 2, and 1 and 3, repectively, are in a dynamic conflict.

Please note: when using state graph reduction, a computation of dynamic conflicts is not possible. For nets which are analyzed using the step firing rule, each two steps enabled in a state are in conflict with each other. Hence, the search is also impossible in this case. However, examining the graph matrix, you can discover which steps are in conflict with each other.

Writing functions

`<W>     write the computed graph (states and arcs)`

With this command, the computed graph, i.e. all states and arcs, is written into the session report, or, upon request, into a file with the default extension `.gra`. By entering `<esc>`, you can redirect the output to the screen. Furthermore, a selection of states can be defined.

See `<X>` and `<M>` for an example of the output.

`<X>     write all arcs`

With this command, all computed arcs are written into the session report, or, upon request, into a file with the default extension `.res`.

For the net 1, the complete arc list looks like the following:

```
State nr.    1
==t1=> s2
==t2=> s3
==t3=> s5
State nr.    2
==t4=> s1
State nr.    3
==t3=> s4
==t5=> s1
State nr.    4
==t5=> s5
==t6=> s3
State nr.    5
```

`<M>     write all states`

With this command, all computed states are written into the session report, or, upon request, into a file with the default extension `.sta`.

For the net 1, the complete state list looks like the following:

```
State nr.    1
P.nr: 0 1 2 3 4 5 6
toks: 2 0 0 0 1 1 1
State nr.    2
P.nr: 0 1 2 3 4 5 6
toks: 0 1 0 0 0 1 1
State nr.    3
P.nr: 0 1 2 3 4 5 6
toks: 1 0 1 0 1 0 1
State nr.    4
P.nr: 0 1 2 3 4 5 6
toks: 0 0 1 1 1 0 0
State nr.    5
P.nr: 0 1 2 3 4 5 6
toks: 1 0 0 1 1 1 0
```

`<P>     write all states satisfying a predicate`

Following the entering of a predicate, only those states are outputted which satisfy this predicate.

`<D>     write the dead states`

Only the dead states are outputted.

`<B>     write the bad states`

Only those states are outputted in which a “bad” predicate (which was entered before the computation) is satisfied.

`<T>     write a trace to a state`

Similar to backtracking during a search for a reachable marking with the command <R>, a path to a computed state can be outputted with this function.

`<S>     write the list of executed steps`

For the maximal steps firing strategy, a list of steps can be outputted. All transitional numbers displayed by other functions are related to steps instead of transitions with that firing strategy; the transitions contained in any step are visible in the list displayed.

Inspect files

`<I>     inspect a result file`

With this command, you can inspect different files created during the analysis.

The following submenu appears:

```
See the session report ......R
See a proof .................P
See a circuits list .........C
See a states list ...........S
See a trace .................T
See a result file ...........F
```

Thus, you can examine SESSION.ina with <R>, a proof file (.prf) with <P>, a file with computed circuits (.cir) with <C>, a states list (.sta) with <S>, a trace (path) to a state (.tra) with <T>, and a general result file (.res) with <F>.

6.6. Invariant analysis

The place or transition invariants of a net are the integer solutions of the homogenous system of linear equations $x \cdot C = 0$ or $C \cdot y = 0$, respectively, where C is the incidence matrix of the net; they are also called P– or T–invariants [Sta90, chapter 11 (110–122)]. The components of P–invariants are interpreted as weights of the respective place. The weighted amount of tokens is invariant with respect to firing. The components of T–invariants can be interpreted as firing numbers of the respective transition (negative values correspond to the reverse firing). Firing all transitions as many times as their firing number indicates leads to the same marking as before. Each bounded and live net has a T–invariant that is positive in all components — such a net is coverable with T–invariants.

6.6.1. Computing invariants

In this subsection, you find a description of the command <I> of the analysis menu, which computes a basis of the invariants space, and offers a fast non-reachability test. Furthermore, the outputting of the incidence matrix and the invariants is described.

In all examples, reference is made to the net 15, which models the controlling of two traffic lights.

Incidence matrix

When computing P–invariants, the transposed incidence matrix, and for T–invariants, the incidence matrix itself can be displayed. The matrix describes how transitions and places are linked [Sta90, Definition 11.1 (110)]. Places are listed in the rows, transitions in the columns (vice versa for the transposed matrix). `incidence matrix`

For the net 15, the transposed incidence matrix is as follows:

```
Input matrix (transposed incidence matrix):

          1  2  3  4  5  6
~~~~~~~~~~~~~~~~~~~~~~~~~~
   1 |   -1  0  1  1  0  0
   2 |    0  1 -1 -1  1 -1
   3 |    0 -1  1  1  0  0
   4 |    1  0 -1 -1 -1  1
```

The entry -1 in the first column of the first row means that the transition t_1 subtracts a token from place $\text{green}_{\text{one}}$. The entry 1 in the sixth column of the fourth row means, on the other hand, that firing the transition t_4 adds a token onto place red_{two}.

If entries in the matrix are written as `00` instead of 0, these entries result from the impurity of the net (see page 54 in chapter 6.2.1).

Output formats

Invariants can be displayed in different forms; you can select a format with the option <F> of the invariant commands. `output format`

As an example, you see P–invariants of the net 15.

`<T>      complete table`

The basis is written in the form of a table where the rows are the numbers of the invariants, and the columns are the numbers of the places or transitions.

```
Nr.        1  2  3  4  5  6
~~~~~~~~~~~~~~~~~~~~~~~~~~~
   1 |     1  1  1  0  0  0 +
   2 |     1  1  0  1  0  0 +
   3 |     0  0  0  0  1  1 +
```

Invariants which you have written into a separate file using this format can be tested later with the command `<T>` (see page 89). They are therefore saved in a file with the extension `.inv`. All other formats do not allow such a test, and are therefore written into a file with the extension `.res`

`<Z>      complete table, zero's not printed`

The basis is likewise written in the form of a table; however, instead of zero's, blanks are printed.

```
Nr.        1  2  3  4  5  6
~~~~~~~~~~~~~~~~~~~~~~~~~~~
   1 |     1  1  1         +
   2 |     1  1     1      +
   3 |                 1  1 +
```

`<E>      print non-zero entries only`

The basis vectors are displayed as a list of their non-zero components in the form `<node number>:<value>`.

```
   1 |     1: 1,  2: 1,  3: 1 +
   2 |     1: 1,  2: 1,  4: 1 +
   3 |     5: 1,  6: 1 +
```

`<#>      print non-zero entries with names`

The basis vectors are likewise displayed as a list of their non-zero components, but with the names of the components included.

```
   1 |     1.gruen_eins      : 1,
     |     2.gruen_zwei      : 1,
     |     3.gelb_eins       : 1 +
   2 |     1.gruen_eins      : 1,
     |     2.gruen_zwei      : 1,
     |     4.gelb_zwei       : 1 +
   3 |     5.rot_eins        : 1,
     |     6.rot_zwei        : 1 +
```

Please note: semipositive invariants are denoted by a `+` when displayed (see also page 86).

Computation of a basis

The P–(or T–)invariants form a linear space, closed under (componentwise) addition, scalar multiplication by integers, and division by the greatest commom divisor of all components. This space is spanned by a finite number of basis vectors, yet in general contains an infinite number of invariants. With the command `<I>`, a basis is computed.

`<I>      Compute a basis for all P/T--invariants`

First, you can change the displayed options:

```
computation of place-invariants..................................P
output (transposed) incidence matrix.............................O
output format = complete table...................................F
```

With the option `<P>`, you choose between computing place or transition invariants. With `<O>`, you can determine whether the (transposed) incidence matrix should be displayed, and with `<F>`, you determine the form in which the invariants are to be displayed.

Afterwards, this command computes a basis for the space of all invariants of the selected type. INA states whether invariants were found, and possibly derives further deductions from it. The program decides whether the net is covered by invariants of the selected type, i.e., whether an invariant exists which is positive in all components. If such an invariant was actually computed, it will be displayed as well.

If P–invariants were found, a fast non-reachability test is offered: `Test a marking for non-reachability?` A description of this test can be found on page 86.

At the end, the following lines are displayed on the screen: `You may display the results by typing 'F <cr> Y <esc> '` and `results by other format (f), exit (e)`.

With `<E>`, you return to the analysis menu, and with `<F>`, the four possible display formats are shown again. If you would like to have the invariants displayed on the screen, simply press `<cr>` now. If you answer the question `Open a separate output file` with `<Y>` and then press `<esc>`, the computed invariants are displayed on the screen, as desired. If, however, you want to direct the results into a separate file, then enter a file name; the extension `.inv` or `.res` is automatically appended. If desired, the display format can also be changed beforehand, of course. The question about a non-reachability test appears again, as well as the lines described above. If you wrote the results in a separate file and then enter `<F>` again, INA first asks `Close separate output file ?`. If you answer `<Y>`, the file is closed and you can enter another output file. Otherwise, the results are appended to the file already open.

Example of an invariant interpretation

The invariant basis listed above for the net 15 and its initial marking states that:

1. the sum of the number of tokens on the places $\text{green}_{\text{one}}$, $\text{green}_{\text{two}}$, and $\text{yellow}_{\text{one}}$ is always equal to one;
2. the sum of the number of tokens on the places $\text{green}_{\text{one}}$, $\text{green}_{\text{two}}$, and $\text{yellow}_{\text{two}}$ is also always equal to one;
3. the sum of the number of tokens on the places red_{one} and red_{two} is also always equal to one.

and therefore

- exactly one traffic light indicates red,
- at most one traffic light indicates green,
- one traffic light is yellow, if and only if the other one is red–yellow.

For the T–invariant basis, we obtain $1 \cdot t_1, 1 \cdot t_2, 1 \cdot t_3, 1 \cdot t_4$, i.e., when all four transitions have fired, the initial marking is reset.

For the net 15, the properties `CPI`, `CTI`, `SB`, and `B` can be deduced from the invariant analysis.

Fast non-reachability test

`non-reachability test`

If the computation of P–invariants has led to a result, a fast non-reachability test is offered. With the computed invariants, INA can easily check this, because of the following equation: For the initial marking m_0, an invariant x, and a marking m reachable from m_0, it always holds that $m_0 \cdot x = m \cdot x$. For a specified marking, this equation is tested with all basis invariants computed, and, if existent, the numbers of the P–invariants which do not satisfy the equation are displayed.

Within the context of net 15, such a non-reachability is shown in the following, based on the previously computed P–invariants:

```
Non-reachability test of the marking m:
  p :  1  2  3  4  5  6
-----------------------
m(p):  1  1  .  .  .  .

The marking is not reachable from m0.
invariant nr.  3
invariant nr.  2
invariant nr.  1

Non-reachability test of the marking m:
  p :  1  2  3  4  5  6
-----------------------
m(p):  1  .  .  .  1  .

No conclusions possible.

Non-reachability test of the marking m:
  p :  1  2  3  4  5  6
-----------------------
m(p):  .  .  1  1  .  .

The marking is not reachable from m0.
invariant nr.  3
```

As you can see, the state “both traffic lights are green” is not reachable. It violates all three P–invariants computed. Furthermore, it can not be decided whether the marking “traffic light 1 is simultaneously green and red” is reachable. The marking “both traffic lights are yellow” violates the invariant number three, which states that exactly one traffic light must indicate red; that marking is therefore not reachable.

6.6.2. Semipositive invariants

In this subsection, you find a description of the computation of a basis of semipositive invariants using the analysis command `<S>`, and the formatting of the computed basis using `<F>`. Please note the remarks on the complexity, and the output formats additionally available.

Semipositive (sub/sur)–invariants

For many applications of invariants, only the semipositive (real) ones [Sta90, Definition 11.3/4 (112,114)] are of interest, i.e., those which are ≥ 0 in all components. For example, a place is bounded iff its weight is positive (i.e. > 0) in some semipositive P–invariant [Sta90, Satz 11.9.1 (116)].

In addition to semipositive invariants, semipositive sub– or sur–invariants can also be computed: For example, a vector y is a semipositive sur–T–invariant if $C \cdot y \geq 0$. A net which possesses a non-trivial (properly semipositive) invariant cannot be bounded for every initial marking (structurally bounded — see chapter 6.2.2), because y is the counting vector of a word which, when being fired, creates a marking $m' = m + C \cdot y$ (out of a marking m), which is greater than m.

Notes on complexity

The problem of computing a basis or a generating system for the semi-space of all semipositive invariants is substantially more complex as the problem of determining a basis for all invariants. There exist nets which possess an exponential number of minimal invariants. For that reason, it is recommendable — even if you are only interested in semipositive invariants — to compute a basis for all invariants with `<I>`, first. When outputted, the semipositive invariants in the basis are marked with `+`. Under certain circumstances, the semipositive invariants found in this way are already sufficient to draw the intended conclusions.

Interactive computation

Because of the complexity of the problem, it is in many cases impossible for INA to compute a complete generating system in a reasonable time and with the memory available. In the "interactive" mode, you can direct the the computation process by entering the next column to be worked on, and thus obtain partial solutions. Should you be interested in specific places in computing P–invariants, you can achieve this by preferring the numbers of the pre- and post- transitions of these places. In this way, you may obtain at least a part of the place invariants related to these places, before you cancel the computation with `<Q>` for reasons of time.

`interactive/ automatic mode`

Output formats

Also for the output of semipositive invariants, different formats are available. In addition to the formats already introduced in 6.6.1, these are the following (as example, P–invariants of the net 15 again):

`output format`

`<S>     print supports`

This format displays the invariants in the form of their support, i.e. by the numbers of the positive components of the vector [Sta90, Definition 11.5 (119)].

```
1 |    1,  2,  3
2 |    1,  2,  4
3 |    5,  6
```

`<N>     print supports with names`

The invariants are likewise displayed by their support, but with the names of the components included.

```
1 |     1.gruen_eins      ,
  |     2.gruen_zwei      ,
  |     3.gelb_eins
2 |     1.gruen_eins      ,
  |     2.gruen_zwei      ,
  |     4.gelb_zwei
3 |     5.rot_eins        ,
  |     6.rot_zwei
```

Basis computation for semipositive (sub/sur)–invariants

This command computes the unformatted invariant basis.

```
<S>     Compute a basis for all semipositive P/T-[sub/sur]-invariants
```

First, you can change the displayed options:

```
computation of place-invariants..................................P
no computation of subinvariants..................................B
no computation of surinvariants..................................R
covered sub/surinvariants not to be deleted......................D
run automatically................................................I
output format = complete table...................................F
```

With the option <P>, you choose between the computation of place or transition invariants; with <I>, you opt for the interactive or the automatic mode, and with <F>, you again determine the form in which invariants should be displayed. Additionally, you can determine whether sub– or sur–invariants should be computed by entering <B> or <R>, respectively. They cannot both be selected at the same time. With the option <D>, you can further decide whether covered sur–/sub–invariants should be deleted.

After setting the options, the computation starts. In the interactive mode, you obtain an overview of the positive and negative entries in the columns of the current matrix and of the alteration to be expected. When computing T–invariants, the columns correspond to the places, whereas for P–invariants, they correspond to the transitions. INA awaits the entering of a column number. With <P> or <cr>, the proposed column can be (`proposal for next to process`) selected, with <A>, INA can be switched into the automatic mode, and with <Q>, the computation can be interrupted. In the automatic mode, you can return to the interactive mode at any time by pressing <I>.

The computed invariants, which in general do not form a minimal system, are automatically written into the file `INVARI.hlp`. It is assumed that sufficient memory space is available on the output medium.

The computation of a minimal generating system and the formatting of the invariants is performed with the function <F> which is executed automatically after finishing the computation. The description of the invariants computation is therefore continued in the next part.

Formatting invariants

This procedure is used to format the output, in particular, it reduces the system of vectors previously computed with <S>.

<F>   Format lines written to INVARI.hlp earlier

This procedure has to be executed using the same options as in the creation of the file INVARI.hlp. INA automatically opens and reads this file; initial reduction options are automatically set according to the type of vectors.

At the beginning, INA asks if a certain number of vectors should be skipped: skip a certain number of lines?. This can be useful, if the file contains more vectors than can be loaded into the internal memory. In this case, the formatting can be performed sectionwise. After entering <Y>, you can determine the number of lines to be skipped.

If the file INVARI.hlp does not exist, INA returns not found and requests a file with the extension .hlp: lines input file. This file can be a INVARI.hlp file that you have renamed, and which contains the unformatted invariants of the current net. You cannot enter this file directly at the terminal.

Should the opened file not correspond to the current net, INA states: INVARI.hlp and current net do not match. In this case, INA returns to the analysis menu. The same happens, if the opened file is not an invariant file at all: wrong file.

You can set the following options for formatting:

```
Delete repeated occurrences of the same support.................O
Do not delete covered sub/sur/invariants........................C
Delete strict invariants covering sub/sur/invariants............I
Reset to default reductions.....................................R
```

When opening the file, options are preset in such a way as to ensure that no information gets lost. INA therefore warns NOTE that information may be lost by non-default reductions. With <R>, you can restore the default settings. With <O>, you can determine whether or not repeated occurences of the same support should be deleted. With <C>, you decide on the deletion of covered sur–/sub–/invariants, and with <I>, you decide whether to delete strict invariants which cover sur–/sub–/invariants.

After setting the options, INA demonstrates the continuation of the formatting by counting the lines read: reading line. If a certain number of lines should be skipped, then these are not counted. By pressing <Q>, you can cancel the reading process at any time.

After possible warnings such as results may be incomplete or covered solutions may be lost (option "c"), first deductions are made, and the results are written into the session report: writing solutions. Again, you can cancel writing with <Q>. Thereafter, the following line reappears: results by other format (f) or exit (e). If you enter <E>, INA returns to the analysis menu; with <F>, you can format again using other options or change the output format. You can also determine an output file or, by pressing <esc>, redirect the output onto the screen. Vectors written into an .inv file using the complete table format, can later be tested with <T> (see page 83 and 89).

6.6.3. Testing invariants

The command <T> is used to test vectors and comprises several subcommands.

<T>   Test place- or transition-vectors for invariant properties

After inquiring whether P– or T–invariants should be tested, INA computes the (transposed) incidence matrix of the net. The following menu subsequently appears:

```
Please, select a sequence of functions
(files:      o, o1, o2, o3, o4, c, c1, c2)
(read/write: r, r1, w, k)
(results:    p, b, b1, q, q1)
(special:    f, m, s)
(exit:       e )
       --- h  for help  ---   <del> = <?>   :
```

You can now enter a sequence of operations ending with `<cr>`, which INA will then perform. Of course, only a single command can also be executed; you just have to press `<cr>` after entering the corresponding letters. After succesfully completing its task, INA returns to the menu above. A description of the individual subcommands can be found in the following parts.

Within a sequence of operations, you can repeat the operations so far performed using the subcommand `<S>` (see page 93), and, e.g., thus test several vectors subsequently in the same way.

Opening files

`<O>`	`open files (input and output)`

With this subcommand, you determine the input and output files.

The files with the extension `.inv`, in which invariants, computed with the analysis menu commands `<I>` or `<S>`, have been written in the format `complete table` (see page 83), can be selected as an input file. Please note: do not attempt to later reread invariant files which you wrote with the subcommands described here; INA cancels in this case.

When opening a file, a consistency test is executed: INA checks whether the vectors saved in the opened file are consistent with the net in the internal memory. This is achieved by use of the headlines which were written automatically when saving vector files in the format `complete table` (see page 83) with the analysis menu commands `<I>` or `<S>`. Should the consistency test yield negative results, INA informs about it and closes the file.

All information computed is normally written into the session report. By specifying a separate file with the extension `.res`, you can redirect this output. Of course, you can also direct the output onto the screen by pressing `<esc>`.

`<O><1>`	`open input file only`

Like `<O>`, but only the input file is opened.

`<O><2>`	`open output file only`

Like `<O>`, but only the output file is opened.

`<O><3>`	`open input file with restricted consistency test`

This subcommand opens only the input file, and the consistency test is thereby restricted so that only the number of places or transitions is compared with the length of the headlines.

<O><4>   open input file without consistency test

With this subcommand, you can open a file without headlines. Use it, if INA produces related warnings when attempting to open a file with one of the other commands.

Closing files

The input file is closed automatically, if errors occur, if the end of the file is reached, or (together with the output file) if the invariant test is finished.

<C>   close files (input and output)

With this subcommand, you can close the input and output files by hand.

<C><1>   close input file only

Like <C>, but only the input file is closed.

<C><2>   close output file only

Like <C>, but only the output file is closed.

Reading operations

<R>   read vector

If you opened an input file with a <O>–subcommand before, this subcommand will read a vector from that file; otherwise, you can enter one directly: Please, give a vector. For the terminal entering, two modes exist: jump to next non-zero component with <J>, and sequential mode with <S>. In the sequential mode, INA requests the components of the vector sequentially. In the jump mode, you can first enter a node number, and then the value of the correpsonding component. All components skipped are set to zero; a reset is not possible. The dialogue starts in the sequential mode, and can at any time be continued in the repective other mode by entering <J> or <S> instead of the expected number. With <N>, you can set all remaining components to zero at the end: set remaining components to zero. With <Q>, you can cancel the entering; the vector may then be incomplete.

<R><1>   read vector without terminal protocol

When reading files with <R>, the current vector is recorded on the screen. With the subcommmand <R><1>, this protocol remains inactive. When entering components at the terminal, there are no changes.

Writing operations

<W>   write vector

If you opened an output file with a <O>–subcommand, this subcommand writes a vector. Please note: do not attempt to later reread a vector saved; INA cancels in this case.

`<K>      write headlines for vector`

With this subcommand, you can write a headline into an opened output file. Use this command once, before you start writing vectors.

`<F>      change output format`

With this subcommand, you change the output format of the invariants: the same formats are available that were already described in the sections 6.6.1 and 6.6.2 on pages 83 and 87, respectively.

Testing functions

As an example for the testing of an invariant, the invariant "a traffic light is yellow, if and only if the other traffic light is red–yellow", already introduced in section 6.6.1 on page 85, will be referred to. A corresponding vector is `0 0 1 -1 1 1`; it is tested in the following.

`<P>      give properties of the vector`

This subcommand checks the just entered (or most recently read) vector, whether it is semipositive, and whether it is invariant, sub–invariant, or sur–invariant. The symbol `+` stands for yes, `-` for no.

The vector `0 0 1 -1 1 1` has the following properties:

```
The place vector under test is not semipositive.
It has the following properties:

place invariant:        +
place subinvariant:     +
place surinvariant:     +
```

`<B>      give results of equations`

With this subcommand, the equation results for the vector are displayed.

```
equation for transition  1.               gives result:      0
equation for transition  2.               gives result:      0
equation for transition  3.               gives result:      0
equation for transition  4.               gives result:      0
```

`<B><1>     give equations with non-zero results`

This subcommand gives an overview of the transitions (in the case of P–vectors) or places (in the case of T–vectors), whose respective equations are not satisfied by the current vector, i.e., do not equal zero.

`<Q>      show equations`

This subcommand works exactly like the subcommand `<B>`, except that the details of the computation are displayed in addition.

The following example shows part of the output:

```
equation for transition  4.                   :
 place              equation * vector              sum
  1    |         1   *    0 =     0   ==>      0
  2    |         0   *    0 =     0   ==>      0
  3    |        -1   *    1 =    -1   ==>     -1
  4    |        -1   *   -1 =     1   ==>      0
  5    |        -1   *    1 =    -1   ==>     -1
  6    |         1   *    1 =     1   ==>      0
```

`<Q><1>     show single equation`

With this subcommand, you can display an individual equation. You will be asked for a transition (in the case of P–vectors) or a place (in the case of T–vectors). The corresponding equation is displayed in a similar way to the previous command.

Further functions

`<M>     show (transposed) incidence matrix`

This subcommand displays the (transposed) incidence matrix.

`<S>     repeat function sequence`

With this subcommand, you can repeat the operations so far performed within a sequence of operations. If `<S>` is reached in the current sequence, INA asks: `Repeat the function sequence?`. If you answer with `<Y>`, the sequence is executed again from the beginning.

In this way, you can test several vectors subsequently in the same way: First, open the input and output files with `<O>`, thereafter enter, as a sequence, `<R>`, followed by the corresponding test commands, then the repetition command `<S>`, and finally `<C>` to close the files. The testing commands are executed on the vectors, which are read one after the other, until you enter `<N>`. Afterwards, the files are closed.

`<H>     help`

With this subcommand, you obtain a short overview of all available functions.

`<E>     exit`

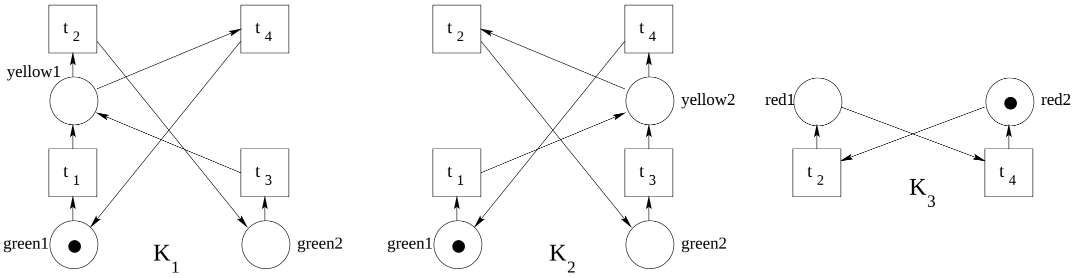
This subcommand ends the invariant test. INA returns to the analysis menu. Any open files are closed first.

6.7. Structural analysis

The functions for structural analysis are only offered in cases where the Petri net theory allows to draw conclusions about dynamic properties of the net from structures discovered. As a fundamental prerequisite, the options have to be set as follows: no time allocation, normal firing mode, no priorities, and firing of single transitions. Further conditions for the availability of the commands result from the previously executed pre-analysis of the current net.

The net 15, which models the control of two traffic lights, is the reference for all examples in this section.

It has the following strongly connected components, which are state machines:



Example 17: Three strongly connected state machines components of the net 15

The places of the components also correspond to the minimal deadlocks and the related maximal traps. Hence, the deadlock–trap–property is satisfied.

The net is not state machine allocatable, since a pre-place allocation exists [Sta90, Definition 16.3 (182)] which does not select a strongly connected state machine component. The components illustrated above, $K_1 = \{\text{green}_{\text{one}}, \text{green}_{\text{two}}, \text{yellow}_{\text{one}}\}$, $K_2 = \{\text{green}_{\text{one}}, \text{green}_{\text{two}}, \text{yellow}_{\text{two}}\}$, and $K_3 = \{\text{red}_{\text{one}}, \text{red}_{\text{two}}\}$ have the post-transitions $T_1 = \{t_1, t_2, t_3, t_4\}$, $T_2 = \{t_1, t_2, t_3, t_4\}$, and $T_3 = \{t_2, t_4\}$, respectively. The pre-place allocation α with $\alpha(t_1) = \text{green}_{\text{one}}$, $\alpha(t_2) = \text{yellow}_{\text{two}}$, $\alpha(t_3) = \text{green}_{\text{two}}$, and $\alpha(t_4) = \text{yellow}_{\text{one}}$ does not select any components, since $t_2 \in T_1$, but $\alpha(t_2) = \text{yellow}_{\text{two}} \notin K_1$; also, $t_4 \in T_2$, but $\alpha(t_4) = \text{yellow}_{\text{one}} \notin K_2$, and $t_2 \in T_3$, but $\alpha(t_2) = \text{yellow}_{\text{two}} \notin K_3$.

The notions and abbreviations used here, can be found in the index or in the chapter 6.2 on page 54ff.

In the following excerpts from the session report, you can also view, besides the actual computation, the net properties thereby deduced.

<D>	Check the deadlock-trap-, SMD-, SMA-properties

This command decides the deadlock-trap-property (`DTP`), and also, for ordinary nets, the state machine decomposability (`SMD`), and the state machine allocatability (`SMA`).

After the writing options have been set, INA computes the list of all deadlocks which are minimal with respect to set inclusion. Deadlocks which are not sufficiently marked in the initial marking are reported as `clean` or `not sufficiently marked`. The maximal trap in such a deadlock can not be sufficiently marked; transitions which have pre-places in that deadlock are dead.

Afterwards, the related maximal traps are computed, where the trap with the number i is the maximal trap in the deadlock number i. Here as well, traps, in particular empty sets, which have no or not enough tokens, are reported as clean or not sufficiently marked, respectively.

If this is not true for all traps, the deadlock-trap-property is satisfied, and therefore, no dead marking is reachable. [Sta90, Satz 15.5 (165)].

When the net has non-blocking multiplicity (NBM), is homogeneous (HOM), and extended simple (ES), then it is live [Sta90, Satz 15.6 (165)].

If one of the traps is not sufficiently marked, then the deadlock-trap-property is not satisfied. For an extended free choice net (EFC), the non-liveness can be deduced [Sta90, Satz 15.10 (173)].

For an ordinary net (ORD), INA subsequently computes all minimal strongly connected state machine components, as well as other minimal components, in order to test for state machine decomposability (SMD). In this case, the net would be structurally bounded (SB). Components which are clean in the initial marking are reported.

If the net is SMD and connected (CON), then INA tests for state machine allocatability (SMA). If this property is not fulfilled, then you find a pre-place allocation in the session report as a list of the form `<transnr> ==> <preplacenr>`, which does not select any SCSM-components.

For the net of the example 15, the following computation results:

```
Structural liveness properties of net nr. 14.Two_Traffic_Lights     :
processed candidates: 13
There are  3 minimal deadlocks:
-------------------------------
   1: 5.rot_eins, 6.rot_zwei,
   2: 1.gruen_eins, 2.gruen_zwei, 4.gelb_zwei,
   3: 1.gruen_eins, 2.gruen_zwei, 3.gelb_eins,

Corresponding maximal traps:
----------------------------
   1: 5.rot_eins, 6.rot_zwei,
   2: 1.gruen_eins, 2.gruen_zwei, 4.gelb_zwei,
   3: 1.gruen_eins, 2.gruen_zwei, 3.gelb_eins,
The deadlock-trap-property is valid.
The net has no dead reachable states.
The net is live.
The net has no dead transitions at the initial marking.
The net is live, if dead transitions are ignored.

SM-allocatability of net nr.  0:
********************************
Some minimal place sets Q with QF=FQ, but all SCSM's:

   1: 5.rot_eins, 6.rot_zwei, SCSM
   2: 1.gruen_eins, 2.gruen_zwei, 4.gelb_zwei, SCSM
   3: 1.gruen_eins, 2.gruen_zwei, 3.gelb_eins, SCSM
 3 minimal component(s) found.

The net is state machine decomposable (SMD).
The net is coverable by state machines (SMC).
The net is covered by semipositive P-invariants.
The net is structurally bounded.
The net is bounded.
The net is covered by semipositive T-invariants.
There are no proper semipositive T-surinvariants.

The net is not state machine allocatable (SMA).
A pre-place allocation allocating no SCSM:
   1==>   1,   2==>   4,   3==>   2,   4==>   3,
The net is covered by 1-token SCSM's.

ORD HOM NBM PUR CSV SCF CON SC  Ft0 tF0 Fp0 pF0 MG  SM  FC  EFC ES
 Y   Y   Y   Y   N   N   Y   Y   N   N   N   N   N   N   N   N   Y
```

```
DTP SMC SMD SMA CPI CTI  B  SB  REV DSt BSt DTr DCF  L   LV L&S
 Y   Y   Y   N   Y   Y   Y   Y   ?   N   N   N   ?   Y   Y   ?
```

The line `1==> 1, 2==> 4, 3==> 2, 4==> 3` describes the pre-place allocation α defined above.

Since the search for minimal deadlocks requires a considerable amount of computation, you can cancel it at any time by pressing `<Q>`.

Possible symmetries computed beforehand help to reduce the search: for the net 15, only 7 instead of 13 candidates have to be examined, due to its symmetry group computed in 6.5.1.

`<C>      Compute all components`

With this command, all components of the current net are computed. After you have set the writing options, the computation starts. The number of components so far computed is thereby displayed. Each component is tested whether it is a state machine (`SM`) and strongly connected (`SC`). At the end, the properties `SMD` and `SMA` are checked, and appropriate deductions are made.

For the net of the example 15, you find the following examination results in the session report:

```
candidates: 117
 All components of net nr. 14.Zwei_Ampeln     :

   1:  1.gruen_eins,  2.gruen_zwei,  3.gelb_eins,  4.gelb_zwei,  5.rot_eins,
       6.rot_zwei,
   2:  5.rot_eins,  6.rot_zwei, SM SC
   3:  1.gruen_eins,  2.gruen_zwei,  4.gelb_zwei,  5.rot_eins,  6.rot_zwei,
   4:  1.gruen_eins,  2.gruen_zwei,  4.gelb_zwei,  6.rot_zwei,
   5:  1.gruen_eins,  2.gruen_zwei,  4.gelb_zwei, SM SC
   6:  1.gruen_eins,  2.gruen_zwei,  4.gelb_zwei,  5.rot_eins,
   7:  1.gruen_eins,  2.gruen_zwei,  3.gelb_eins,  5.rot_eins,  6.rot_zwei,
   8:  1.gruen_eins,  2.gruen_zwei,  3.gelb_eins,  6.rot_zwei,
   9:  1.gruen_eins,  2.gruen_zwei,  3.gelb_eins, SM SC
  10:  1.gruen_eins,  2.gruen_zwei,  3.gelb_eins,  5.rot_eins,
  11:  1.gruen_eins,  2.gruen_zwei,  3.gelb_eins,  4.gelb_zwei,  6.rot_zwei,
  12:  1.gruen_eins,  2.gruen_zwei,  3.gelb_eins,  4.gelb_zwei,
  13:  1.gruen_eins,  2.gruen_zwei,  3.gelb_eins,  4.gelb_zwei,  5.rot_eins,

The net is state machine decomposable (SMD).
The net is coverable by state machines (SMC).
The net is covered by semipositive P-invariants.
The net is structurally bounded.
The net is bounded.
There are no proper semipositive T-surinvariants.

The net is not state machine allocatable (SMA).
A pre-place allocation allocating no SCSM:
   1==>   1,   2==>   4,   3==>   2,   4==>   3,

ORD HOM NBM PUR CSV SCF CON SC  Ft0 tF0 Fp0 pF0 MG  SM  FC  EFC ES
 Y   Y   Y   Y   N   N   Y   Y   N   N   N   N   N   N   N   N   Y
DTP SMC SMD SMA CPI CTI  B  SB  REV DSt BSt DTr DCF  L   LV L&S
 ?   Y   Y   N   Y   ?   Y   Y   ?   ?   N   ?   ?   ?   ?   ?
```

Since the computation is very time-consuming, it can be cancelled at any time by pressing `<Q>`.

`<M>     Decide state machine coverability`

This command decides for ordinary nets whether they can be covered by (not necessarily strongly connected) components which are state machines.

First, you can set the writing options. Thereafter, INA computes all minimal components [Sta90, Definition 16.1 (177)]; the number of components so far computed is thereby displayed. You can cancel the examination at any time by pressing `<Q>`.

All computed components are examined in order to determine whether they are state machines [Sta90, Definition 14.3 (152)]. In the session report, these are marked as `SM`. Components which are not marked in the initial marking are reported as `clean`.

If the place set of the net is the union of place sets of minimal state machine components, then the net is `SMC` and hence bounded [Sta90, Folgerung 16.4].

For the net of the example 15, the output looks like the following:

```
SM-coverability of net nr. 14.Zwei_Ampeln     :

Minimal place sets Q with QF=FQ:

   1:  5.rot_eins,  6.rot_zwei, SM
   2:  1.gruen_eins,  2.gruen_zwei,  3.gelb_eins, SM
   3:  1.gruen_eins,  2.gruen_zwei,  4.gelb_zwei, SM
 3 minimal component(s) found.

candidates created: 29
The net is coverable by state machines (SMC).
The net is covered by semipositive P-invariants.
The net is structurally bounded.
The net is bounded.
There are no proper semipositive T-surinvariants.

ORD HOM NBM PUR CSV SCF CON SC  Ft0 tF0 Fp0 pF0 MG  SM  FC  EFC ES
 Y   Y   Y   Y   N   N   Y   Y   N   N   N   N   N   N   N   N   Y
DTP SMC SMD SMA CPI CTI  B  SB  REV DSt BSt DTr DCF  L   LV L&S
 ?   Y   ?   ?   Y   ?   Y   Y   ?   ?   N   ?   ?   ?   ?   ?
```

If the net cannot be covered with state machines, then you find an overview of the places in the session report which lie in state machine components, and which are therefore bounded.

If INA already knows that the net is coverable by state machines (`the net is known to be state machine coverable`), the first question is: `Compute the SM-components?` If you answer this question with `<Y>`, INA executes the computation regardless of the previous knowledge.

This computation can also be cancelled at any time by pressing `<Q>`.

Appendix

Bibliography

[Sta90] STARKE, Peter H.: *Analyse von Petri–Netz–Modellen*. Stuttgart : B. G. Teubner, 1990 (Leitfäden und Monographien der Informatik)

[Sta91] STARKE, Peter H.: Reachability Analysis of Petri Nets Using Symmetries. In: *J. of Math. Modelling & Simulation in Systems Analysis* 8 (1991), S. 293–304

[SS91] SCHMIDT, Karsten ; STARKE, Peter H.: An Algorithm to Compute the Symmetries of Petri Nets. In: *Petri Net Newsletter* 40 (1991), S. 25–30

[Sch97] SCHMIDT, Karsten: *How to Calculate Symmetries of Petri nets*. Technische Universität Dresden, Preprint (MATH-AL-8-1997), 1997

[Pog94] POGRELL, Lutz: *Theoretische Grundlagen und Implementation der Methode der sturen Mengen*. Humboldt–Universität zu Berlin, Institut für Informatik, Diplomarbeit, 1994

[Lü95] LÜTTGE, Karsten: *Zustandsgraphen von Petri–Netzen*. Humboldt–Universität zu Berlin, Institut für Informatik, Diplomarbeit, 1995

[Fin89] FINKEL, Alain: *A Minimal Coverability Graph for Petri Nets*. Université Paris 11, Forschungsbericht, 1989

[Hel96] HELJANKO, Keijo: Implementing a CTL Model checker. In: *Workshop Concurrency, Specification & Programming*. Humboldt–Universität zu Berlin, Informatik–Bericht Nr. 69, S. 75–84, 1996

Index

O

P

W

About this Manual

This manual descibes the functionality of version 2.2 of INA. It was written by Stephan Roch in collaboration with Prof. Peter H. Starke with the text setting system LaTeX2e the KOMA–Script–Package (from Frank Neukamm and Markus Kohm) and the program LaTeX2HTML (from Nikos Drakos nikos@cbl.leeds.ac.uk) and translated by Dirk Schlatter with the help of Stefan Haar and Mathew O'Breigh (partially supported by the Deutsche Forschungsgemeinschaft in the "PEP"-project). It is based on extensive testing, detailed source code analysis, and personal consultations with Prof. Starke.

The www–page

`http://www.informatik.hu-berlin.de/lehrstuehle/automaten/ina/`

allows access to this manual in tex–dvi and postscript formats via the internet. A html–version, which can be viewed with common browsers, is also available there, as well as the older version 1.7 of INA. You can also download all the example nets which appear in this manual.

A full version of the program is available free of charge on `ftp.informatik.hu-berlin.de` in the directory `/pub/local/petritool` via anonymous ftp (this manual can be downloaded there, as well).

For information on further versions of the program, please refer to Prof. Starke.

Conditions for source codes are announced upon request.

No responsibility is assumed for any mistakes in this manual or in the program.

Contact address:

Humboldt–Universität zu Berlin
Institut für Informatik
Lehrstuhl für Automaten– und Systemtheorie
Prof. Peter. H. Starke
Unter den Linden 6
10099 Berlin
Telefon: (+49) (30) 2093 3078 /
Fax: (+49) (30) 2093 3081
Email: starke@informatik.hu-berlin.de