

# TINX 13.0 reference cards

Application structure, Basic Temporal Logic syntax,  
user interface features and file formats

Version 1.1.0

## Table of contents

Application structure.....	3
Introduction.....	3
Modules.....	3
Files.....	4
Paths.....	4
Input and output channels.....	4
Default binary input and output symbols.....	5
Signals managed by the inference engine.....	5
Extension server channels.....	6
Main information flows.....	7
Temporal inference networks.....	8
Basic Temporal Logic.....	10
General instruction syntax.....	10
Multidimensional arrays.....	11
Intervals.....	11
Logical operators.....	12
Mathematical operators.....	13
Temporal operators.....	14
Iterations and selections.....	15
Declarations.....	16
Other constructs.....	17
Input and output options.....	18
String control sequences.....	19
Examples of BTL source file.....	20
User interface.....	21
Main controls.....	21
Configuration controls.....	24
Display format.....	29
Examples of sample display.....	29
File formats.....	30
Inference engine trace and log file format.....	30
Temporal inference network file format.....	31
Input and output codes.....	32
Symbol table file format.....	32
Some suggestion to run.....	33
General use.....	33
Logical coding.....	33

Logical debug.....	34
Estimate of quantum.....	35
Benchmarks.....	38
Credits and references.....	39
Credits.....	39
External references.....	39

# Application structure

## Introduction

TINX is a real time inference engine for system specification in an executable temporal logic. It is able to acquire, process and generate any binary,  $n$ -ary or real signal through POSIX IPC, files or UNIX sockets. Specifications of signals and dynamic systems are represented as special graphs named *temporal inference networks* and executed in real time, with a predictable sampling time which varies from few microseconds to some milliseconds, depending on the complexity of the specification.

Real time signal processing, dynamic system control, modeling of state machines, logical and mathematical property verification, realization of reactive systems are some fields of application of this inference engine, which is deterministic but fully relational. It adopts driven forward reasoning in a three-valued logic, the clauses of which are assumed unknown as default, to satisfy relations (2SAT) and it is able to run on an unlimited temporal horizon.

The accepted language is named *Basic Temporal Logic* and provides logical and mathematical operators, temporal operators on instants and intervals, parametrization of signals by multidimensional arrays and bounded quantifiers on them.

This software runs on Linux operating system and it is distributed under GNU Public License.

## Modules

Module	Description
tinx	Temporal inference network executor, single core version of the inference engine
tinx_mt <sup>1</sup>	Temporal inference network executor, multiple core version of the inference engine
tinx_dt	Temporal inference network executor, optimized dual core version of the inference engine for high throughput applications
tinx_zt	Temporal inference network executor, multiple core version of the inference engine for high throughput applications, <i>experimental and provided for study purposes only</i>
ting	Temporal inference network generator, the Basic Temporal Logic compiler
gtinxsh	Graphical TINX shell, the interactive development environment
extserv	Simple data stream server

Rows highlighted in yellow have to be considered first, in this table and the following ones.

---

<sup>1</sup> Currently, the multiple core versions of the inference engine do not support dynamic mathematics, so logic programs containing real functions and relational operators will not be executed by them.

## Files

File	Description
*.btl	Basic Temporal Logic source files, editable text
*.tin	Temporal inference network object files, text generated by module “ting”
*.evl	Event list files, text generated by module “ting” to describe the initial conditions of execution and by module “tinx” as an inference process log
*.io	Input and output signal files, text containing binary symbols, 8-bit characters or real numbers and generated by modules “tinx” and “gtinxsh”
*.sym	Symbol table files, text generated by module “ting”
.gtinxshrc	Configuration file of the graphical shell, binary

## Paths

Path	Description
/usr/bin/	Executable modules
/usr/share/doc/tinx/	Documentation in PDF and ASCII formats
/usr/share/tinx/examples/	Examples of binary dynamical system specifications in Basic Temporal Logic source files and temporal inference network object files
/usr/share/tinx/sources/	C language and metacompiler complete source code

## Input and output channels

Channel	Description
/TINX:C	Default IPC communication port associated to clause or real function $C$
$D/C.io$	File with default path $D$ associated to clause or real function $C$
$W1.W2.W3.W4:N$	Socket at network address $W1.W2.W3.W4$ and port $N$ associated to clause $W1.W2.W3.W4(N)$ or real function $W1.W2.W3.W4(N)$

## Default binary input and output symbols

Symbol	Description
0	False
1	True
?	Unknown
.	The signal reached its end and can be dropped as a source of information, when all the input signals are dropped the inference and outputs proceed anyway until possible
Escape	An immediate termination of the inference process and outputs is requested (the same result can be obtained by sending an interrupt signal or Ctrl-C to the inference engine)

## Signals managed by the inference engine

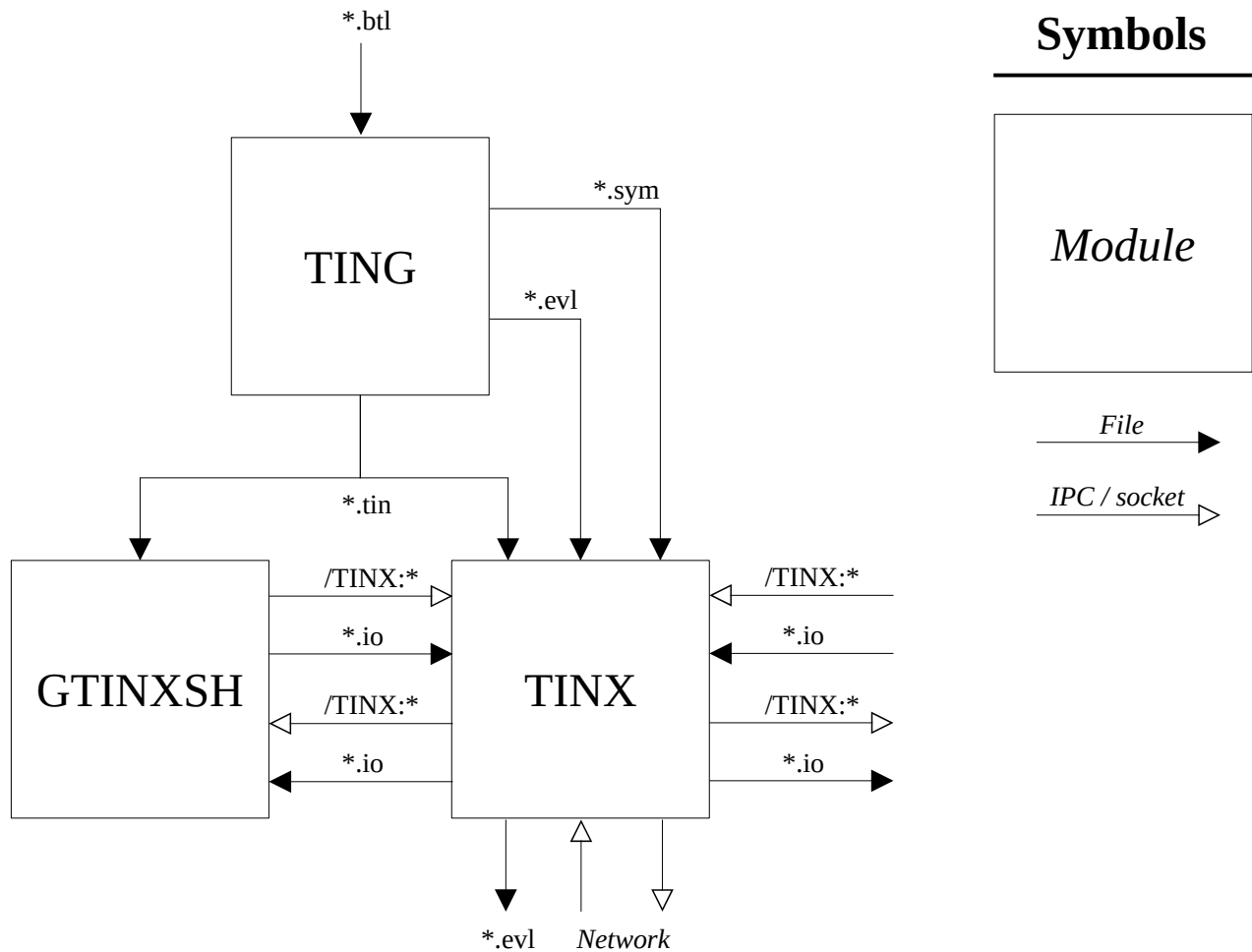
Signal	Description
SIGINT	Causes the immediate termination of the inference process and outputs
SIGUSR1	Resets the inference engine and causes a reload of its configuration files
SIGUSR2	Pauses or resumes the inference process and outputs

## Extension server channels

Channel	Category	Description
/TINX:command	<i>n-ary input</i>	Bit 0 for system clock Bits 1-4 for random number generators Bits 5-8 for user interface buttons Bits 9-12 for user interface sliders Bit 13 for user interface messages Bits 14-17 for user interface lights Bits 18-21 for user interface numeric displays Bit 22 to provide sleep time to the server Bit 23 to quit the server
/TINX:Clock	<i>Real output</i>	System clock expressed in seconds
/TINX:Random( <i>n</i> )	<i>Real outputs</i>	Random number generators from 0 to 1, <i>n</i> is between 0 and 3
/TINX:button( <i>n</i> )	<i>Binary outputs</i>	True on button press and false otherwise, <i>n</i> is between 0 and 3
/TINX:Slider( <i>n</i> )	<i>Real outputs</i>	User selected number from 0 to 1, <i>n</i> is between 0 and 3
/TINX:message	<i>n-ary input</i>	Status/error string
/TINX:light( <i>n</i> )	<i>Binary inputs</i>	Binary light, <i>n</i> is between 0 and 3
/TINX:Display( <i>n</i> )	<i>Real inputs</i>	Numeric display, <i>n</i> is between 0 and 3
/TINX:Sleep	<i>Real input</i>	Duration of sleep time for each cycle expressed in seconds

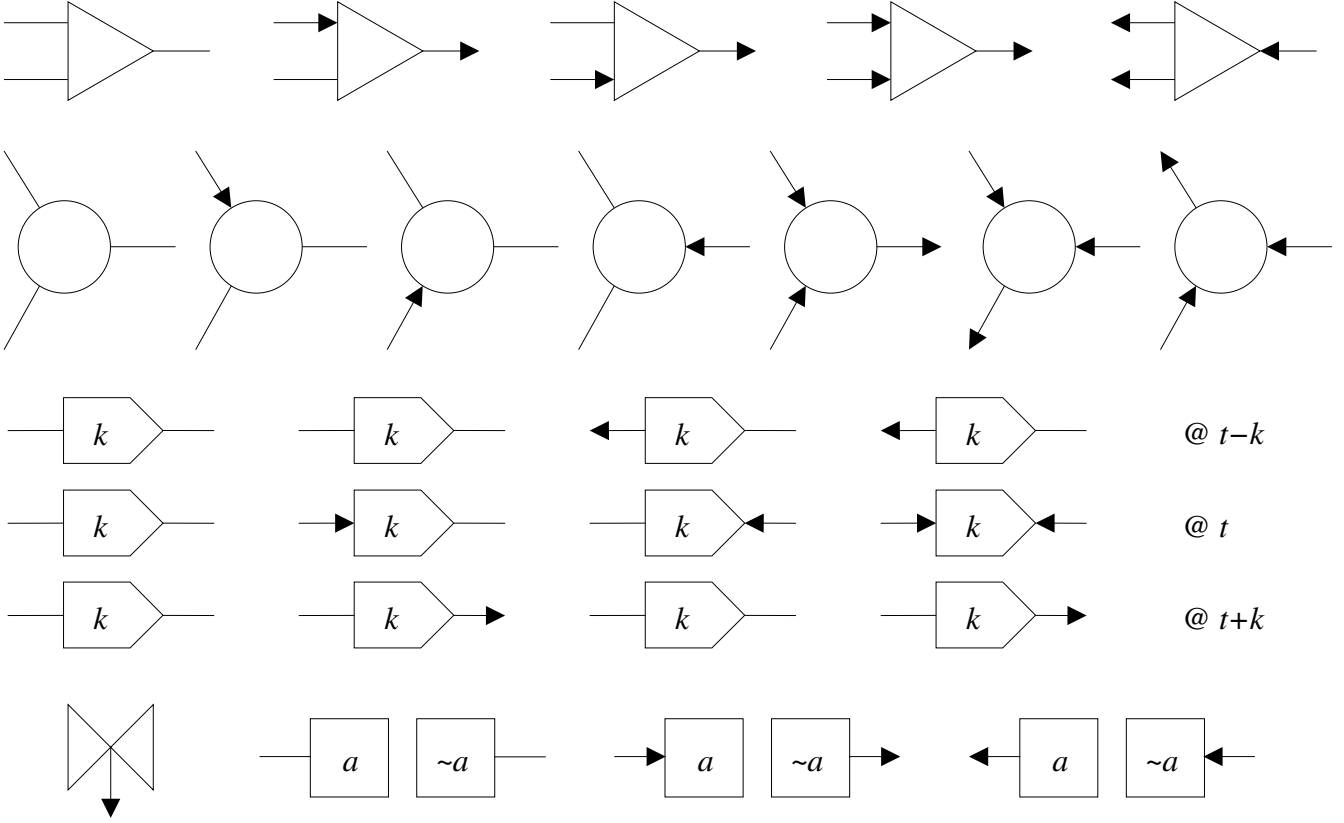
## Main information flows

Beyond the information flows shown below, module GTINXSH coordinates the other modules by launching them with a proper command line on user request, by redirecting their textual output in a specific frame of the module user interface during the execution and by terminating or pausing/resuming inference process of module TINX through a system signal if required. Among the involved files, only “\*.io” files are dynamically checked for addition of new data in tail. Configuration files of module GTINXSH are private of this module and not shown below. Text editor used to create “\*.btl” files is not included in this system and no debugger is provided.



## Temporal inference networks

In order of presentation components *joint*, *gate*, *delay*, *forever* and *logical literal* with their inference rules. Edges without arrows have an unknown logical value, forward or backward arrows represent respectively a false or true logical value associated to their edge. Propagation of these implicit values happens only in the directions of the arrows. A logical contradiction is detected when two opposite arrows meet on the same edge. Each arrow incident on a *logical literal* means that the corresponding literal is true if incoming, false if outgoing. Component *forever* is simplified together with its neighbors while *logical literals* are substituted by connections of *joints*, so that only the first three components are present in real networks.

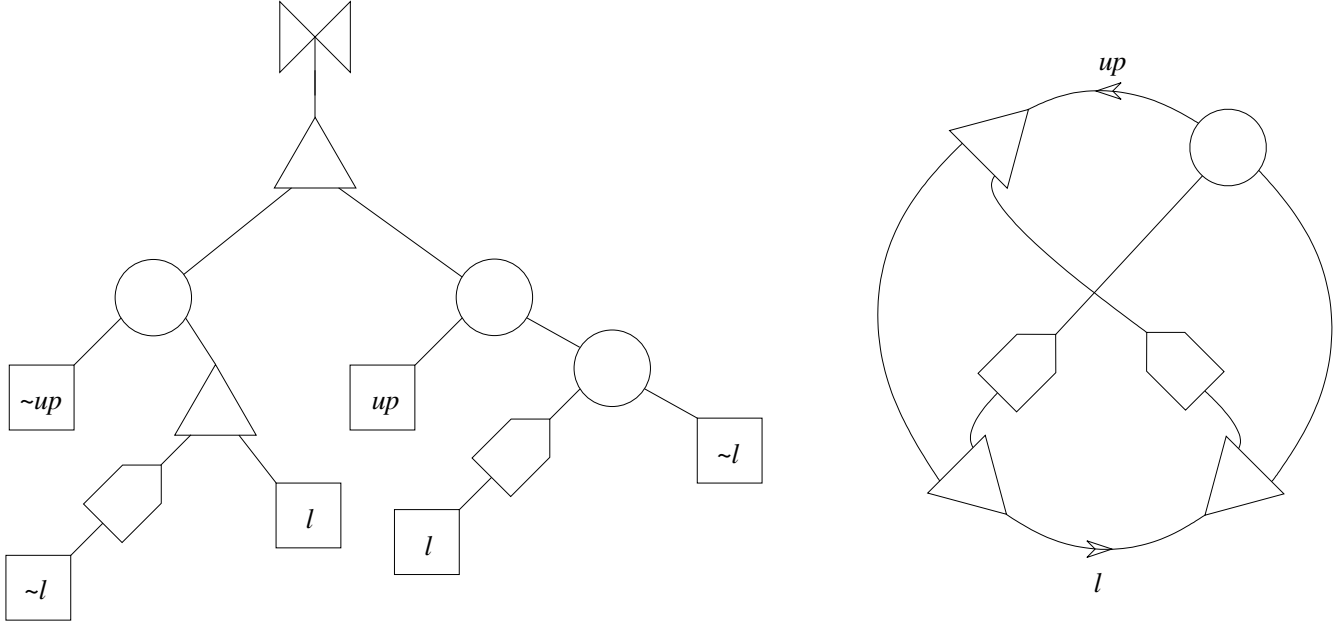


In a logical formula always true, if the negations are distributed down to literals and only some of them are known to be false or true in some instant of time, previous inference rules can be proven to be equivalent to truth tables of ordinary logical operators in determining the logical value of literals unknown in some instant of time.

In this frame, to each conjunction corresponds a *joint*, to each disjunction a *gate* and to each temporal displacement a *delay*. Components of type *gate* perform binary inference steps while *joint* and *delay* unary steps, represented by arrows which are thus superimposable. Components of type *delay* keep the state of the network while *gate* and *joint* are without memory.



As a basic example, it is possible to define a signal *up* which is false in every instant of time except when another signal *l* becomes true (changes from false to true). The two signals are linked by BTL clause “*up* ==  $\sim l @ -1 \& l$ ”, with the operators which appear inside the clause being described in next section. This clause can be rewritten as “ $(\sim up \mid (\sim l @ -1 \& l)) \& (up \mid (l @ -1 \mid \sim l))$ ” and is thus represented by both following networks. Network at left is built during an early compile phase by following the syntactic tree of last clause, network at right is the result of a later phase of automatic completion and pruning and is preferred for execution. In this network, every logical literal representing a I/O signal is associated to an edge with a label and a blank arrow which indicates the verse to follow to state that the associated logical literal is true.



# Basic Temporal Logic

## General instruction syntax

Basic Temporal Logic is a pragmatic and quantitative discrete-time temporal logic language used to declare logical relations among clauses, the truth value of which varies in time. It is able to bind also real numeric functions of time in mathematical relations which can appear inside any clause for piecewise definition of the functions or logical evaluation of their results. BTL logical quantifiers are then used to generalize statements on multidimensional arrays of clauses or functions. Generalization in a relative time is expressed by quantifiers on discrete intervals. Since in BTL not explicitly declared logical values are assumed unknown, explicit or implicit negation is required for a complete specification of any system. Current implementation put limits on recursion in space but allows infinite recursion in time.

This language does not impose a specific style of coding, but it is anyway suggested to avoid representing systems of relations which are satisfiable by multiple solutions and to prefer representations admitting a single solution. If this is not feasible, the addition of hints is requested to rule out all the extra solutions.

Syntax	Description
$P1; P2; \dots Pn;$	Set of temporal logic clauses or declarations $P1, P2, \dots Pn$ . If ground clauses, the first character of their name is alphabetic <i>and in lower case</i>
$F1 = G1; F2 = G2; \dots Fn = Gn;$	Set of mathematical identities between real functions of time $F1, F2, \dots Fn$ and $G1, G2, \dots Gn$ . If ground clauses, the first character of their name is alphabetic <i>and in upper case</i>
$N$	Integer or real parameter $N$ , depending on its definition (see “define” construct). The first character of its name is alphabetic <i>and in lower case</i>
<code>`P`</code>	Clause $P$ , if its name does not follow the case convention or contains special characters. This can be necessary since IPC port names, file names and socket addresses used to stream binary symbols or 8-bit characters are built from clause names
<code>``F``</code>	Function $F$ , if its name does not follow the case convention or contains special characters. This can be necessary since IPC port names, file names and socket addresses used to stream real numbers are built from function names
<code>// Text</code>	C++ language style comment
<code>/* Text */</code>	C language style comment

## Multidimensional arrays

Syntax	Description
$P(K)$ $F(K)$ $N(K)$	Parametric vector ground clause $P$ , function $F$ or parameter $N$ with index $K$
$P(H, K)$ $F(H, K)$ $N(H, K)$	Parametric matrix ground clause $P$ , function $F$ or parameter $N$ with indexes $H$ and $K$
$P(J, H, K)$ $F(J, H, K)$ $N(J, H, K)$	Parametric three-dimensional array ground clause $P$ , function $F$ or parameter $N$ with indexes $J$ , $H$ and $K$
$P(I, J, H, K)$ $F(I, J, H, K)$ $N(I, J, H, K)$	Parametric four-dimensional array ground clause $P$ , function $F$ or parameter $N$ with indexes $I$ , $J$ , $H$ and $K$

## Intervals

Syntax	Description
$[A, B]$	Interval between integer constant time $A$ and time $B$ , included
$(A, B]$	Interval between integer constant time $A + 1$ and time $B$ , included
$[A, B)$	Interval between integer constant time $A$ and time $B - 1$ , included
$(A, B)$	Interval between integer constant time $A + 1$ and time $B - 1$ , included

## Logical operators

Syntax	Description
$\sim P$	Not clause $P$
$P \& Q$	Clause $P$ and clause $Q$
$P   Q$	Clause $P$ or clause $Q$
$P \setminus Q$	Clause $P$ exclusive or clause $Q$
$P \rightarrow Q$	Clause $P$ implies clause $Q$
$P \leftarrow Q$	Clause $P$ is implied by clause $Q$
$P == Q$	Clause $P$ double implies clause $Q$
$\text{forall}(P(K), K \text{ on } N)$	For all parametric clauses $P(K)$ between 0 and $N - 1$
$\text{forall}(P(K), K \text{ in } A : B)$	For all parametric clauses $P(K)$ on a range from $A$ to $B$
$\text{exists}(P(K), K \text{ on } N)$	Exists a parametric clause $P(K)$ between 0 and $N - 1$
$\text{exists}(P(K), K \text{ in } A : B)$	Exists a parametric clause $P(K)$ on a range from $A$ to $B$
$\text{unique}(P(K), K \text{ on } N)$	Exists an unique parametric clause $P(K)$ between 0 and $N - 1$
$\text{unique}(P(K), K \text{ in } A : B)$	Exists an unique parametric clause $P(K)$ on a range from $A$ to $B$
$\text{one}(P(K), K \text{ is } A, K \text{ on } N)$ $\text{one}(P(K), \text{not } K \text{ is } A, K \text{ on } N)$	Asserts only one parametric clause $P(A)$ while negating the other clauses $P(K)$ between 0 and $N - 1$ or otherwise
$\text{one}(P(K), K \text{ in } A : B, K \text{ on } N)$ $\text{one}(P(K), \text{not } K \text{ in } A : B, K \text{ on } N)$	Asserts a subset from $A$ to $B$ of parametric clauses $P(K)$ while negating the other clauses $P(K)$ between 0 and $N - 1$ or otherwise, an underscore “_” can be used to replace $A$ or $B$ with 0 or $N - 1$
$\text{one}(P(K), K \text{ is } A, K \text{ in } C : D)$ $\text{one}(P(K), \text{not } K \text{ is } A, K \text{ in } C : D)$	Asserts only one parametric clause $P(A)$ while negating the other clauses $P(K)$ on a range from $C$ to $D$ or otherwise
$\text{one}(P(K), K \text{ in } A : B, K \text{ in } C : D)$ $\text{one}(P(K), \text{not } K \text{ in } A : B, K \text{ in } C : D)$	Asserts a subset from $A$ to $B$ of parametric clauses $P(K)$ while negating the other clauses $P(K)$ on a range from $C$ to $D$ or otherwise, an underscore “_” can be used to replace $A$ or $B$ with $C$ or $D$
$\text{combine}(P(K), Q(\#), K \text{ on } N)$	Builds all the combinations of negated and asserted clauses $P(K)$ between 0 and $N - 1$ and makes each one of them equivalent to the corresponding clause $Q(H)$
$\text{combine}(P(K), Q(\#), K \text{ in } A : B)$	Builds all the combinations of negated and asserted clauses $P(K)$ on a range from $A$ to $B$ and makes each one of them equivalent to the corresponding clause $Q(H)$

## Mathematical operators

Syntax	Description
$A + B$	Sum of number $A$ and number $B$
$A - B$	Difference between number $A$ and number $B$
$A * B$	Product of number $A$ for number $B$
$A / B$	Quotient of number $A$ for number $B$ , truncated for integer parameters
$A \% B$	Reminder of the division between number $A$ and number $B$ , available only on integer parameters
$A \wedge B$	Power of number $A$ to number $B$
$A = B$	Number $A$ equal to number $B$
$A \neq B$	Number $A$ not equal to number $B$
$A \leq B$	Number $A$ less than or equal to number $B$
$A \geq B$	Number $A$ greater than or equal to number $B$
$A < B$	Number $A$ less than number $B$
$A > B$	Number $A$ greater than number $B$
$\text{root}(A, B)$	Root of number $A$ with index $B$
$\log(A, B)$	Logarithm in base $A$ of number $B$
$\sin(A)$	Sine of number $A$ expressed in radians
$\cos(A)$	Cosine of number $A$ expressed in radians
$\tan(A)$	Tangent of number $A$ expressed in radians
$\text{asin}(A)$	Arcsine expressed in radians of number $A$
$\text{acos}(A)$	Arccosine expressed in radians of number $A$
$\text{atan}(A)$	Arctangent expressed in radians of number $A$
$\text{sum}(A(K), K \text{ on } N)$	Summation of succession $A(K)$ between 0 and $N - 1$
$\text{sum}(A(K), K \text{ in } A : B)$	Summation of succession $A(K)$ on a range from $A$ to $B$
$\text{prod}(A(K), K \text{ on } N)$	Production of succession $A(K)$ between 0 and $N - 1$
$\text{prod}(A(K), K \text{ in } A : B)$	Production of succession $A(K)$ on a range from $A$ to $B$
$\text{Ke}$	Constant “e”
$\text{Kpi}$	Constant “pi”

## Temporal operators

Syntax	Description
$P @ T$ $F @ T$	Clause $P$ or function $F$ at relative time $T$
$P @ R$	For all clauses $P$ in relative interval $R$
$P ? R$	Exists a clause $P$ in relative interval $R$
$P ! R$	Exists an unique clause $P$ in relative interval $R$
$\text{since}(P, Q)$	Since clause $P$ is true until clause $Q$ keeps true
$\text{until}(P, Q)$	Until clause $P$ is true since clause $Q$ keeps true
$F ++ R$	Summation of function $F$ on relative interval $R$
$F ** R$	Production of function $F$ on relative interval $R$

## Iterations and selections

Syntax	Description
<code>iter(K on N) P(K)</code>	Repeats a parametric clause or declaration $P(K)$ between 0 and $N - 1$
<code>iter(K on N) { P1(K); P2(K); ... Pn(K); }</code>	Repeats a set of parametric clauses or declarations $P1(K), P2(K), \dots Pn(K)$ between 0 and $N - 1$
<code>iter(K in A : B) P(K)</code>	Repeats a parametric clause or declaration $P(K)$ on a range from $A$ to $B$
<code>iter(K in A : B) { P1(K); P2(K); ... Pn(K); }</code>	Repeats a set of parametric clauses or declarations $P1(K), P2(K), \dots Pn(K)$ on a range from $A$ to $B$
<code>when(K is A) P(K)</code> <code>when(not K is A) P(K)</code> <code>when(K is A) P(K) else Q(K)</code> <code>when(not K is A) P(K) else Q(K)</code>	Selects or excludes only one parametric clause or declaration $P(A)$ within a loop, eventually while substituting it with clause or declaration $Q(K)$ in the other iterations
<code>when(K is A) { P1(K); P2(K); ... Pn(K); }</code> <code>when(not K is A) { P1(K); P2(K); ... Pn(K); }</code> <code>when(K is A) { P1(K); P2(K); ... Pn(K); } else</code> <code>    { Q1(K); Q2(K); ... Qn(K); }</code> <code>when(not K is A) { P1(K); P2(K); ... Pn(K); } else</code> <code>    { Q1(K); Q2(K); ... Qn(K); }</code>	Selects or excludes only one set of parametric clauses or declarations $P1(A), P2(A), \dots Pn(A)$ within a loop, eventually while substituting them with clauses or declarations $Q1(K), Q2(K), \dots Qn(K)$ in the other iterations
<code>when(K in A : B) P(K)</code> <code>when(not K in A : B) P(K)</code> <code>when(K in A : B) P(K) else Q(K)</code> <code>when(not K in A : B) P(K) else Q(K)</code>	Selects or excludes a subset from $A$ to $B$ of parametric clauses or declarations $P(K)$ within a loop, eventually while substituting them with clauses or declarations $Q(K)$ in the other iterations, an underscore “_” can be used to replace $A$ or $B$ with the extremes of the loop
<code>when(K in A : B) { P1(K); P2(K); ... Pn(K); }</code> <code>when(not K in A : B) { P1(K); P2(K); ... Pn(K); }</code> <code>when(K in A : B) { P1(K); P2(K); ... Pn(K); } else</code> <code>    { Q1(K); Q2(K); ... Qn(K); }</code> <code>when(not K in A : B) { P1(K); P2(K); ... Pn(K); } else</code> <code>    { Q1(K); Q2(K); ... Qn(K); }</code>	Selects or excludes a subset from $A$ to $B$ of groups of parametric clauses or declarations $P1(K), P2(K), \dots Pn(K)$ within a loop, eventually while substituting them with clauses or declarations $Q1(K), Q2(K), \dots Qn(K)$ in the other iterations, an underscore “_” can be used to replace $A$ or $B$ with the extremes of the loop

## Declarations

Syntax	Description
input $L1, L2, \dots L_n$	Declares a set of input signals $L1, L2, \dots L_n$ corresponding to each ground clause or function
input [ $O1, O2, O3, O4$ ] $L1, L2, \dots L_n$	Declares a set of input signals $L1, L2, \dots L_n$ corresponding to each ground clause or function with specific options $O1, O2, O3, O4$ (see “Input and output options”)
output $L1, L2, \dots L_n$	Declares a set of output signals $L1, L2, \dots L_n$ corresponding to each ground clause or function
output [ $O1, O2, O3, O4$ ] $L1, L2, \dots L_n$	Declares a set of output signals $L1, L2, \dots L_n$ corresponding to each ground clause or function with specific options $O1, O2, O3, O4$ (see “Input and output options”)
aux $L1, L2, \dots L_n$	Declares a set of auxiliary signals $L1, L2, \dots L_n$ corresponding to each ground clause or function
init $L1 @ T1, L2 @ T2, \dots L_n @ T_n$	Declares as true the ground clauses $L1, L2, \dots L_n$ respectively at absolute times $T1, T2, \dots T_n$
init $\sim L1 @ T1, \sim L2 @ T2, \dots \sim L_n @ T_n$	Declares as false the ground clauses $L1, L2, \dots L_n$ respectively at absolute times $T1, T2, \dots T_n$
init $L1 @ T1 = A1, L2 @ T2 = A2, \dots L_n @ T_n = A_n$	Declares the value $A$ of the real functions $L1, L2, \dots L_n$ respectively at absolute times $T1, T2, \dots T_n$
init $L1 @ R1, L2 @ R2, \dots L_n @ R_n$	Declares as true the ground clauses $L1, L2, \dots L_n$ respectively in absolute intervals $R1, R2, \dots R_n$
init $\sim L1 @ R1, \sim L2 @ R2, \dots \sim L_n @ R_n$	Declares as false the ground clauses $L1, L2, \dots L_n$ respectively in absolute intervals $R1, R2, \dots R_n$
init $L1 @ R1 = A1, L2 @ R2 = A2, \dots L_n @ R_n = A_n$	Declares the value $A$ of the real functions $L1, L2, \dots L_n$ respectively in absolute intervals $R1, R2, \dots R_n$
define $K1 = E1, K2 = E2, \dots K_n = E_n$	Computes the integer or real expressions $E1, E2, \dots E_n$ and assigns their results respectively to the parameters $K1, K2, \dots K_n$



Syntax	Description
<pre> define K( ) = [ E1, E2, ... En ] define K( , ) = [ [ E11, E12, ... E1n ],                   [ E21, E22, ... E2n ] ],                   ...                   [ Em1, Em2, ... Emn ] ] define K(I, ) = [ E1, E2, ... En ] define K( , J) = [ E1, E2, ... Em ] ... </pre>	<p>Computes the integer or real expressions <math>E1</math>, <math>E2</math>, ... <math>En</math> or <math>E11</math>, <math>E12</math>, ... <math>Emn</math> and assigns their results to the elements of the vector <math>K(I)</math>, matrix <math>K(I, J)</math> or other multidimensional array. Partial assignments of a single row or column are allowed</p>

### Other constructs

Syntax	Description
include "F1", "F2", ... "Fn"	Includes in the current program the BTL source files named $F1$ , $F2$ , ... $F_n$
<pre> code(L, "S") code(L(#), "S") code(L(##, #), "S") </pre>	Asserts and negates properly the parametric ground clause either $L(K)$ or $L(H, K)$ so as to represent the string $S$ , with the bit number in $K$ or the byte number in $H$ and its bit number in $K$ respectively, the latter as default
<pre> code(L, N) code(L(#), N) code(L(##, #), N) </pre>	Asserts and negates properly the parametric ground clause either $L(K)$ or $L(H, K)$ so as to represent the unsigned integer $N$ , with the bit number in $K$ or the byte number in $H$ and its bit number in $K$ respectively, the latter as default
<pre> +\$(L, N) +\$(L(#), N) +\$(L(##, #), N) </pre>	Evaluates as a positive time the unsigned integer not greater than $N$ and represented by the parametric ground clause either $L(K)$ or $L(H, K)$ , with the bit number in $K$ or the byte number in $H$ and its bit number in $K$ respectively, the latter as default; <i>no arithmetic is possible on this result</i>
<pre> -\$(L, N) -\$(L(#), N) -\$(L(##, #), N) </pre>	Evaluates as a negative time the unsigned integer not greater than $N$ and represented by the parametric ground clause either $L(K)$ or $L(H, K)$ , with the bit number in $K$ or the byte number in $H$ and its bit number in $K$ respectively, the latter as default; <i>no arithmetic is possible on this result</i>
#	Represents the iterator of the current loop, valid also if no variable is specified. Constructs which operate on arrays present internal iterations and this fact has to be considered when determining the nesting of the loops. Some construct may present more than one level of nesting.
## ### ...	Represent the iterators of the outer loops, valid also if no variable is specified

## Input and output options

Option	Category	Description
any	<i>Channel (O1)</i>	Signal input and output channels are determined by external options (default)
ipc	<i>Channel (O1)</i>	Signal input and output occur always by inter process communication
file	<i>Channel (O1)</i>	Signal input and output occur always by shared file
remote	<i>Channel (O1)</i>	Signal input and output occur always by socket
binary	<i>Data (O2)</i>	Input and output signals contain binary symbols (default)
packed	<i>Data (O2)</i>	Input and output signals contain strings of 8-bit characters
false	<i>Default (O3)</i>	The binary signal or packed signal bit is false as default
true	<i>Default (O3)</i>	The binary signal or packed signal bit is true as default
unknown	<i>Default (O3)</i>	The binary signal is unknown as default (default)
default = A	<i>Default (O3)</i>	The real signal has a default equal to number A
raw	<i>Bandwidth (O4)</i>	The binary signal or packed signal bit is always either waited for acquisition for inputs or produced for outputs (default)
filter	<i>Bandwidth (O4)</i>	The binary signal or packed signal bit is either acquired within a time unit or given a default value for inputs and it is produced only if determined of a value different from the default for outputs
omit	<i>Bandwidth (O4)</i>	The binary signal or packed signal bit is either acquired within a time unit or given a default value for inputs and it is produced only if determined of a value different from both unknown and the default for outputs

## String control sequences

Sequence	Description
\\	Backslash
\”	Double quote
\a	Alert
\b	Backspace
\f	Form feed
\n	New line
\r	Carriage return
\t	Horizontal tab
\v	Vertical tab
\x <i>N</i>	Character with hexadecimal code <i>N</i>

## Examples of BTL source file

```
/*
    Benchmark by Andrea Giotti, 2017-2022
    Execute as: tinx -iq -z 10000000 -r 8 minmark
*/

define n = 16;

iter(k on n)
{
    output out(k);
    aux up(k);

    when(k in 1 : _)
        init ~ out(k) @ 0;
}

init up(0) @ 0;

iter(k on n)
{
    up(k) == up((k + 1) % n) @ k + 1;

    up(k) | (out(k) @ -1 & ~ up((k + 1) % n)) --> out(k);
    up((k + 1) % n) | (~ out(k) @ -1 & ~ up(k)) --> ~ out(k);
}
```

```
/*
    Variable window filter of maximum width equal to 2 * maxsize + 1

    The true values of binary input signal "origin" are cut by windows which are centered around
    the true values of a periodic signal and the size of which is dynamically defined by the lower
    bits of the encoded input signal "window" so as to produce the binary output signal "filtered"
*/

define tau = 63, bits = 4, maxsize = 2 ^ bits - 1;

input origin;
output periodic, filtered;

/* Periodic output signal "periodic" of period tau */
init periodic @ 0;

periodic --> ~ periodic @ (0, tau) & periodic @ tau;

/* Encoded window size is sampled at the beginning of each period and hold */
iter(k on bits)
{
    input [packed] window(k);
    aux size(k / 8, k % 8);

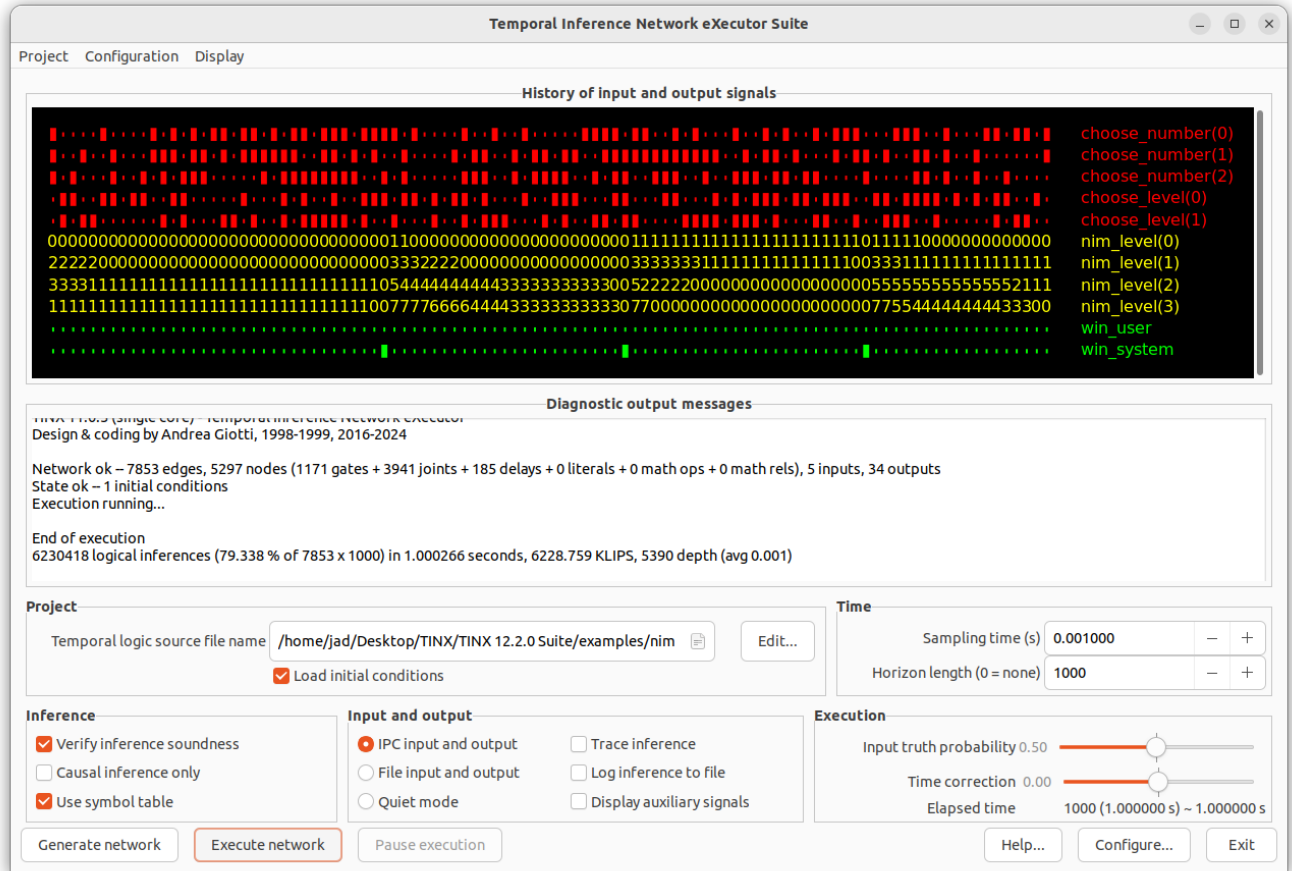
    size(k / 8, k % 8) == since(periodic & window(k), ~ periodic);
}

/* The true values of the periodic signal are the centers of the windows */
filtered == origin & periodic ? [-$(size, maxsize), +$(size, maxsize)];
```

# User interface

## Main controls

Module GTINXSH provides a graphical user interface split in three frames. In the top frame signal histories are displayed dynamically during execution, in the middle frame textual output of modules TING and TINX is shown, in the bottom frame controls described in present section are grouped. All controls remain interactive when TINX runs, but only controls appearing in the “Execution” frame and in the “Probabilities” window affects current execution, the others will affect next one. Most systems require the option “Load initial conditions” to be selected for running.



Name	Command	Description
Temporal logic source file name (text field) or Open... (menu item)	ting	Name without extension of the BTL source file, the side icon opens the file chooser dialog
Load initial conditions (check box)	tinx -i	The initial conditions of execution are evaluated, <i>this is necessary for all systems without inputs or with unreachable state</i> (see “init” construct)

Name	Command	Description
Edit... (button)		Opens the text editor on the current BTL source file (see “Text editor” field)
Sampling time (numeric field)	tinx -t	Sampling time in seconds and fractions or zero for fast execution mode
Horizon length (numeric field)	tinx -z	Overall duration of execution expressed in number of samples to process or zero for an unlimited temporal horizon (default option)
Verify inference soundness (check box)	tinx -v	The production of a contradiction at run time interrupts the execution, instead of proceeding anyway
Causal inference only (check box)	tinx -c	The inference proceeds from past or present to present or future only, <i>some system will not work with this performance option enabled</i>
Use symbol table (check box)	ting -x tinx -x	The inference process trace or log includes also BTL clauses
IPC input and output (radio button)	(default)	Input and output of signals unconstrained to a specific method are performed by inter process communication (see “input” and “output” constructs and their options)
File input and output (radio button)	tinx -f	Input and output of signals unconstrained to a specific method are performed by shared files (see “input” and “output” constructs and their options)
	tinx -F	Input and output of signals unconstrained to a specific method are performed by sockets (see “input” and “output” constructs and their options)
Quiet mode (radio button)	tinx -q	No input or output are performed, <i>all systems with inputs will not work with this performance option enabled</i>
Trace inference (check box)	tinx -d	A trace at run time of the inference process is displayed in the message frame, <i>this may break the real time</i> (see “Use symbol table” option)
Log inference to file (check box)	tinx -l	A log at run time of the inference process is produced as a text file (see “Use symbol table” option)
Display auxiliary signals (check box)	ting -b	Also the signals declared as auxiliary are displayed in the history frame as outputs with default options (see “aux” construct)
Input truth probability (slider)		Truth probability of binary symbols in all randomly generated input signals, <i>to be set to one if different values are chosen for each signal</i> (see “Probabilities...” window)
Time correction (slider)		Difference between clocks of the graphical shell and the inference engine, <i>experimental</i>

Name	Command	Description
Generate network (button or menu item)		Compiles the BTL source file to generate a temporal inference network and its execution initial conditions with current options
Execute network (button or menu item)		Activates the inference engine and executes the temporal inference network with current options
Stop execution (button or menu item)		Interrupts the execution of the temporal inference network
Pause execution (button or menu item)		Pauses the execution of the temporal inference network
Resume execution (button or menu item)		Resumes the execution of the temporal inference network
Help... (button or menu item)		Opens the help window (see “Help viewer” field)
About (menu item)		Displays the application description and author
Exit (button or menu item)		Exits from the graphical shell and halts the inference engine if running
Configure... (button or menu item)		Opens the configuration window
Probabilities... (menu item)		<i>See corresponding button in configuration controls</i>
External signals... (menu item)		<i>See corresponding button in configuration controls</i>
Phase plan setup... (menu item)		Opens a window containing the list of the coupled output signals which have to be displayed on the phase plan
Open configuration... (menu item)		Loads the graphical shell configuration from any file
Save configuration (menu item)		<i>See corresponding button in configuration controls</i>
Save configuration as... (menu item)		Saves the graphical shell configuration to any file
Clear configuration (menu item)		<i>See corresponding button in configuration controls</i>
Spectrum analyzer... (menu item)		Opens the spectrum analyzer display window
Phase plan... (menu item)		Opens the phase plan display window, signals are coupled following the setup (see “Phase plan setup...” window)
	ting -h tinx -h	Displays a short synopsis of the command

## Configuration controls

All the remaining options of module GTINXSH can be configured by the following window and the configuration can be saved. Any other option of modules TING and TINX can be accessed by command line only. Some systems require a proper entry in the field “Memory size logarithm” and the options “Generate constant signals”, “Generate suggested defaults” or “Generate user defaults” to be selected for running.

**Configuration**

**Files**

Network object file name:

Initial conditions file name:

Log file name:

Symbol table file name:

Module file path:

Text editor:

Help viewer:

Logic compiler:

Monoprocess executor:

High throughput monoprocess executor:

Multiprocess executor:

High throughput multiprocess executor:

External startup program:

External shutdown program:

**Input and output parameters**

IPC prefix:  ☐ External inputs ☐ External outputs ☒ Display unknowns as dots

I/O file path:  ☐ System V IPC ☐ Ignore IPC errors

False char:  True char:  Unknown char:  End char:  ☐ Display internal signals ☐ Display full names

**Resources**

Number of processes	<input type="text" value="1"/>	<input type="text" value="-"/>	<input type="text" value="+"/>
Memory size logarithm	<input type="text" value="8"/>	<input type="text" value="-"/>	<input type="text" value="+"/>
History window columns	<input type="text" value="100"/>	<input type="text" value="-"/>	<input type="text" value="+"/>
History window max rows	<input type="text" value="15"/>	<input type="text" value="-"/>	<input type="text" value="+"/>

**Optimizations**

☒ Optimize joints in intervals ☒ Optimize joints in recursions

☒ Optimize delays ☒ Post optimization

☒ Generate constant signals ☒ Generate suggested defaults

☒ Generate user defaults

**Miscellaneous**

☐ High throughput

☐ Wait running

☐ Hard real time (root access only)

Save configuration Probabilities... External signals... Clear configuration

Name	Command	Description
Network object file name (text field)	ting -o tinx	Name without extension of the temporal inference network object file, the side icon opens the file chooser dialog
Initial conditions file name (text field)	ting -I tinx -I	Name without extension of the execution initial conditions file, the side icon opens the file chooser dialog (see “Load initial conditions” option)
Log file name (text field)	tinx -L	Name without extension of the inference process log file, the side icon opens the file chooser dialog (see “Log inference to file” option)



Name	Command	Description
Symbol table file name (text field)	ting -X tinx -X	Name without extension of the symbol table file, the side icon opens the file chooser dialog (see “Use symbol table” option)
Module file path (text field)	ting -P	Name of the module file path, the side icon opens the file chooser dialog (see “include” construct)
Text editor (text field)		Name of the executable file of the text editor for BTL source files (see “Edit...” command)
Help viewer (text field)		Name of the executable file of the PDF viewer for help files (see “Help...” command)
Logic compiler (text field)		Name of the executable file of the logic compiler (see “Generate network” command)
Monoprocess executor (text field)		Name of the executable file of the monoprocess executor (see “Execute network” command)
High throughput monoprocess executor (text field)		Name of the executable file of the high throughput monoprocess executor (see “Execute network” command)
Multiprocess executor (text field)		Name of the executable file of the multiprocess executor (see “Execute network” command and “Number of processes” field)
High throughput multiprocess executor (text field)		Name of the executable file of the high throughput multiprocess executor (see “Execute network” command and “Number of processes” field)
External startup program (text field)		Name of the executable file of the optional startup sequence, launched before the run of the executor (see “Execute network” command)
External shutdown program (text field)		Name of the executable file of the optional shutdown sequence, launched after the run of the executor (see “Execute network” command)
IPC prefix (text field)	tinx -e	Prefix for inter process communication port names, it has to start with a “/” (see “IPC input and output” option and “input” and “output” constructs)
I/O file path (text field)	tinx -p	Path for shared file names used for input and output signals (see “File input and output” option and “input” and “output” constructs)
	tinx -j	Network prefix for socket names used for input and output signals (see “File input and output” option and “input” and “output” constructs)
False char (text field)	tinx -a	Symbol used to represent the false in binary mode, it has to be the first character of the command line argument

Name	Command	Description
True char (text field)	tinx -a	Symbol used to represent the true in binary mode, it has to be the second character of the command line argument if present
Unknown char (text field)	tinx -a	Symbol used to represent the unknown in binary mode, it has to be the third character of the command line argument if present
End char (text field)	tinx -a	Symbol used to represent the end of input or output signals in binary mode, it has to be the fourth character of the command line argument if present
External inputs (check box)		The inference engine gathers all its input signals from external programs or files and no random input signal is generated or displayed by the graphical shell, <i>disabling this option may overwrite predetermined external files</i>
External outputs (check box)		The inference engine provides all its output signals to external programs or files and no output signal is displayed by the graphical shell
System V IPC (check box)	tinx -V	Inter process communication follows the System V protocol instead of the POSIX protocol, <i>the uniqueness of the port identifiers is no more granted</i>
Ignore IPC errors (check box)	tinx -y	Inter process communication errors are ignored instead of causing the end of input or output signals
Display internal signals (check box)	ting -B	Also the internal signals used in the decomposition of the temporal operators are displayed in the history frame as outputs with default options
Display full signal names (check box)		Signal names are displayed with port identifiers or path and extension in the history frame
Display unknowns as dots (check box)		Unknown logical values are displayed as dots in the history frame
Number of processes (numeric field)	tinx_mt -n tinx_zt -n	If equal to one the single or dual core version of the inference engine are used, if more than one such a number of cores are allocated to the inference process by the multiple core version <i>plus one core for input and output</i>
Memory size logarithm (numeric field)	tinx -r	Two raised to this number is proportional to the memory allocated for the inference process, <i>this result should be more than four times the maximum time present in the arguments of temporal interval operators (see “@” and “?” constructs)</i>
History window columns (numeric field)		Number of samples displayed in the history frame
History window max rows (numeric field)		Maximum number of signals displayed in the history frame






Name	Command	Description
Optimize joints in intervals (check box)	ting -w	The number of joints of the temporal inference network produced by the interval operators is optimized (see “@” and “?” constructs)
Optimize joints in recursions (check box)	ting -W	The number of joints of the temporal inference network produced by the recursive operators is optimized, <i>this may cause problems with some initial conditions</i> (see “since” and “until” constructs)
Optimize delays (check box)	ting -u	The number of delays of the temporal inference network is optimized, <i>this may cause problems with the selected memory size</i>
Post optimization (check box)	ting -O	The global post optimizer is run after the other optimizations
Generate constant signals (check box)	ting -k	Constant output signals are generated anyway as such with deduced defaults and not removed by the optimizer
Generate suggested defaults (check box)	ting -K	Other either constant or unknown output signals are generated anyway as constants with deduced defaults and not removed by the optimizer
Generate user defaults (check box)	ting -m	Remaining either constant or unknown output signals are generated anyway as constants with user suggested defaults and not removed by the optimizer
Hard real time (check box)	tinx -s	A fair scheduling is requested to the operating system, <i>this option is available at root access only</i>
Wait running (check box)	tinx -S	The inference engine busy waits, this will increase the system overhead but decrease the inference process latency
High throughput (check box)	tinx_dt tinx_zt	The optimized dual core version or the multiple core version of the inference engine for high throughput applications are used, in the first case one core is allocated to the inference process and one core for input and output, for the second case see the parameter “Number of processes” description
Save configuration (button)		Saves the graphical shell configuration to an hidden file of current directory
Probabilities... (button)		Opens a window containing the truth probabilities of binary symbols in each single randomly generated input signal, so different values can be chosen (see “Input truth probability” slider, <i>which has to be set to one for this option</i> )
External signals... (button)		Opens a window containing the list of input and output signals which have to be managed by external programs and not by the graphical shell (see “External inputs” and “External outputs” options)

<b>Name</b>	<b>Command</b>	<b>Description</b>
Clear configuration (button)		Deletes the graphical shell configuration from current directory
Clear (button)		Resets to one the truth probabilities of binary symbols in all randomly generated inputs
Clear (button)		Resets to internals all the inputs and the outputs of the system
Clear (button)		Resets to empty the set of couples used to display the phase plan
	tinx_mt -D	Displays debug information of the multiple core version only
	tinx -g	Sets the origin of the temporal reference system, equal to current time if omitted
	ting -M1 ting -M2	The logic compiler produces an output enriched of markup sequences for a light theme (-M1) or a dark one (-M2), instead of a pure textual one

## Display format

Drawing	Description
Red sample sequence	Binary or real input signal history
Green sample sequence	Binary or real output signal history
Orange sample sequence	8-bit character input signal history
Yellow sample sequence	8-bit character output signal history
Turquoise sample sequence	Auxiliary or internal signal history (with “Display auxiliary signals” or “Display internal signals” options enabled)
Small rectangle	False bit
Big rectangle	True bit
Dot	Unknown bit (with “Display unknowns as dots” option enabled)
Yellow circle	Loss of phase in output signal and real time broken
Yellow ring	Binary output signal of unknown value in phase
Elapsed time	Partial duration of execution expressed in number of samples processed and number of corresponding seconds, compared to measured time expressed in seconds
Red exclamation point	Warning symbol asserting it is necessary to regenerate the temporal inference network so that the requested options can take place

## Examples of sample display

Example	Drawings
Binary input false and true	
Binary output false, true, unknown and alerts of loss of phase and unknown output	
Binary auxiliary false, true, unknown and alerts of loss of phase and unknown output	
8-bit character input of “A” and “B”	
8-bit character output of “A” and “B” and alert of loss of phase	

## File formats

### Inference engine trace and log file format

Record	Description
$(V, W) \# K @ I$ $(V, W) \# K @ I: P \rightarrow Q$ $(V, W) \# K @ I = A$ $(V, W) \# K @ I = A: F \rightarrow G$	In the log file, the edge from vertex $V$ to vertex $W$ with index $K$ representing clause $Q$ or function $G$ at absolute time $I$ , clause or function which has been deduced from clause $P$ or function $F$ at some time, is selected and processed for further inference. Function $G$ is evaluated as a real number $A$
$> T: (V, W) \# K @ I$ $> T: (V, W) \# K @ I: P \rightarrow Q$ $> T: (V, W) \# K @ I = A$ $> T: (V, W) \# K @ I = A: F \rightarrow G$ $\#N > T: (V, W) \# K @ I$ $\#N > T: (V, W) \# K @ I: P \rightarrow Q$ $\#N > T: (V, W) \# K @ I = A$ $\#N > T: (V, W) \# K @ I = A: F \rightarrow G$	In the execution trace, described processing is performed at external absolute time $T$ by the core number $N$ for the multiple core version
$*$ $\#N *$	In the execution trace, no further inference is possible until new inputs and this is noted by the core number $N$ for the multiple core version
$ $ $\#N  $	In the execution trace, no further inference will be done until external time advances and this is noted by the core number $N$ for the multiple core version

## Temporal inference network file format

Record	Description
V0: G ; V1 # K1, V2 # K2, V3 # K3 V0: J ; V1 # K1, V2 # K2, V3 # K3 V0: D T ; V1 # K1, V2 # K2 V0: E T ; V1 # K1, V2 # K2 V0: U ; V1 # K1, V2 # K2 V0: V ; V1 # K1, V2 # K2 V0: T ; V1 # K1, V2 # K2 V0: A ; V1 # K1, V2 # K2, V3 # K3 V0: B ; V1 # K1, V2 # K2, V3 # K3 V0: C ; V1 # K1, V2 # K2, V3 # K3 V0: Z ; V1 # K1, V2 # K2 V0: X ; V1 # K1, V2 # K2 V0: M ; V1 # K1, V2 # K2 V0: W ; V1 # K1, V2 # K2 V0: L A ; V1 # K1 V0: L F ; V1 # K1	Vertex V0 of class gate (“G”), joint (“J”), delay (“D”) or mathematical delay (“E”) of degree <i>T</i> , unary minus (“U”), inverse (“V”), trigonometric sine (“T”), addition (“A”), multiplication (“B”), exponentiation (“C”), equality with zero (“Z”), inequality with zero (“X”), equality or relation of greater than zero (“M”), relation of less than zero (“W”) or literal (“L”) of constant value <i>A</i> or name <i>F</i> is connected to parent vertex <i>V1</i> and possibly to sons vertexes <i>V2</i> and <i>V3</i> by edges with indexes <i>K1</i> and possibly <i>K2</i> and <i>K3</i>
! P (V, W) # K / O1, O2', O2", O3, O4 ? P (V, W) # K / O1, O2', O2", O3, O4 . P (V, W) # K / O1, O2', O2", O3, O4 !^ F (V, W) # K / O1, O2', O2", O3, O4 ?^ F (V, W) # K / O1, O2', O2", O3, O4 .^ F (V, W) # K / O1, O2', O2", O3, O4 _^ F (V, W) # K / O1, O2', O2", O3, O4 ? P (*, *) # 0 / O1, O2', O2", O3, O4	Input signal (“!”), output signal (“?”) , auxiliary signal (“.”) or hidden signal (“_”) corresponding to clause <i>P</i> or function <i>F</i> is represented by the edge from vertex <i>V</i> to vertex <i>W</i> with index <i>K</i> and options <i>O1</i> , <i>O2</i> , <i>O3</i> and <i>O4</i> ( <i>O2'</i> is the packed signal number if greater than zero and <i>O2''</i> the packed signal bit for that number). If <i>O2'</i> is equal to zero the signal is binary, if less than zero the signal is real. Real signals in feedback are marked by a caret (“^”) and vertexes are replaced by a pair of asterisks (“*”) for constant outputs

## Input and output codes

Option	Category	Code
any	<i>Channel (O1)</i>	0
ipc	<i>Channel (O1)</i>	1
file	<i>Channel (O1)</i>	2
remote	<i>Channel (O1)</i>	3
binary	<i>Data (O2')</i>	0
packed	<i>Data (O2')</i>	$\geq 1$ ( <i>signal number + 1</i> )
( <i>real stream</i> )	<i>Data (O2')</i>	-1
	<i>Data (O2'')</i>	$\geq 0$ ( <i>bit number</i> )
false	<i>Default (O3)</i>	0
true	<i>Default (O3)</i>	1
unknown	<i>Default (O3)</i>	2
default = A	<i>Default (O3)</i>	A
raw	<i>Bandwidth (O4)</i>	0
filter	<i>Bandwidth (O4)</i>	1
omit	<i>Bandwidth (O4)</i>	2

## Symbol table file format

Record	Description
V: P V: F	The edge from vertex <i>V</i> to its parent with index 0 represents clause <i>P</i> or function <i>F</i>



## Some suggestion to run

### General use

- If the source file contains “init” constructs you have to select the “Load initial conditions” check box on the main panel to execute the described system, if not some signal will result unknown. However, source files without this construct will not run with this option enabled.
- If some signal declared in the source file results to be constant you have to select the “Generate constant signals” check box on the configuration panel to execute the described system, if not packed I/O may be compromised by the removal of that signal. For example, this often happens if the source file contains “code” constructs to represent I/O symbols.
- If this is not sufficient, you can select the “Generate suggested defaults” check box on the configuration panel to execute the system, but some partially unknown signal will be generated as a constant signal with a logical value compatible with its known logical values.
- If the executor refuses to run a system because the maximum time displacement specified in its source file exceeds the temporal buffer size, you have to increase it by setting the “Memory size logarithm” numeric field in the configuration panel. Some system could require an even larger buffer size for internal computations, necessary to determine all its signals to known logical values. An excessive temporal buffer size has to be anyway avoided, because it would bring to virtualize memory and to break the real time.
- The “External startup program” and “External shutdown program” text fields on the configuration panel are meant to launch and terminate the external I/O clients of the executor, for example by inserting their path and name in the first field or by the use of a specific “pkill” command to be inserted in the second field, respectively.

### Logical coding

- When writing the description of a system, it is suggested to use intermediate logical variables as arguments of the operators and to define these variables in separate clauses instead of using complex expressions for arguments of the same operators, so as to limit the size of the generated network.
- When writing the description of a system, it is suggested to use specific temporal operators instead of common logical operators and simple time displacements, since the former are optimized in space and time.
- When writing the description of a system, it is suggested to limit recursion depth on indexes of clauses, nesting of iterations on them and length of time intervals, since all these parameters affect the network size and subsequently both the memory required for the execution and the minimum sampling time.
- When writing the description of a system, it is necessary to remember that not explicitly declared logical values are assumed unknown, so use double implication ( $P == Q$ ) rather than simple one ( $P \rightarrow Q$ ) if you are not covering separately the other case ( $R \rightarrow \sim Q$ ).
- When writing the description of a system, it is suggested to avoid unknown logical values for output variables since they can increase significantly the phase lag of the corresponding signal.
- When writing the description of a system, it is suggested to use the “filter” or “omit” options

only if it is required to limit the I/O bandwidth, because the synchronization among the executor and its external I/O clients may be broken by these options if the selected sampling time is too short.

- If you find verbose the coding style described in these cards you can shorten it by the use of iterators, as an example the statement “one( $P(K)$ ,  $K$  is  $A$ ,  $K$  on  $N$ )” can be rewritten as “one( $P(\#)$ ,  $A$ ,  $N$ )” and similarly for the other logic constructs.

## Logical debug

- It is suggested to debug your system with the “Verify inference soundness” check box selected on the main panel, if not most logical contradiction will not be detected.
- It is suggested to debug your system with both “Use symbol table” and “Trace inference” check boxes selected on the main panel to help you to trace the bugs, if the chosen sampling time is not too short.
- It is suggested to debug your system with the “Display auxiliary signals” check box selected on the main panel to monitor also the intermediate logical variables of the described system, if they are not too many.
- If the number of I/O signals exceeds the maximum number of POSIX IPC message queues but not the maximum number of open files you can select the “File input and output” check box on the main panel to switch to the latter method.
- Bugs which are hard to diagnose can be found by decomposing the system in separate modules and then by testing the subsystems.

## Estimate of quantum

The number of edges ( $E$ ) plus the square number of mathematical literals ( $L$ ) of the temporal inference network generated by compiling the source file is roughly proportional to the minimum sampling time ( $T$ ) you can set in the “Sampling time” numeric field on the main panel of the user interface without breaking the real time. The number of binary<sup>2</sup> I/O signals ( $S$ ) and the maximum temporal displacement appearing in the network ( $D$ ) also affects this parameter, following:

$$T = \lambda (E + \gamma L^2) + \mu S$$

$$\lambda = \alpha (1 + \delta D)$$

$$\mu = \beta (1 + \varepsilon S)$$

with  $\alpha$ ,  $\beta$ ,  $\gamma$ ,  $\delta$  and  $\varepsilon$  coefficients to be determined experimentally. Factor  $\lambda$  represents the maximum time required for each single inference step, factor  $\mu$  represents the maximum I/O time per signal sample and depends on the selected I/O method, hence the notation  $\mu_{IPC}$  to estimate  $T_{IPC}$  and  $\mu_{file}$  to estimate  $T_{file}$ .

The maximum number of inference steps necessary to determine all the output signal samples for one single instant of time is the term  $E + \gamma L^2$ , since each clause represented by an edge is evaluated only once for each instant and the square being justified by the fact that each mathematical literal occurrence has to be put in relation with each other one for copying purposes. Relations among logical literals are resolved by the compiler and represented by a subpart of the network structure. In the expression of  $\lambda$ , coefficient  $\alpha$  represents the constant time component of the inference step, coefficient  $\delta$  the weight on the temporal search of maximum depth ( $D$ ) in the variable time component, which is overestimated.

The relation between the number of I/O signals and the time required to perform their acquisition or generation is not linear and has to be corrected because of random retries during input polling or output attempt. This relation is modeled to be quadratic following the hypothesis that each signal becomes available for reading or writing with equal probability  $1 / (1 + \varepsilon S)$  and is so processed in an average number of I/O operations equal to  $1 + \varepsilon S$ , or after  $\varepsilon S$  retries on average. In the expression of  $\mu$ , coefficient  $\beta$  ( $\beta_{IPC}$  or  $\beta_{file}$ )<sup>3</sup> represents the constant time component of the I/O signal sample processing, coefficient  $\varepsilon$  the weight on the number of signals in the variable time component, which is averaged.

A possibly stricter estimate of the inference step time can be obtained by considering average values and not maximum values for temporal displacements. By denoting with  $B$  the average temporal search depth per inference step and with  $\eta$  the relative weight required per single deepening of the search algorithm, it holds  $B \leq D$  and  $\lambda$  is rewritten as:

$$\lambda = \alpha (1 + \eta B)$$

*This factor is equal to the reciprocal of measured KLIPS in quiet mode divided for one thousand. When  $D \ll 1 / \delta$  or  $B \ll 1 / \eta$ , the factor  $\lambda$  can be assumed to be about equal to  $\alpha$  which depends on the*

- 
- 2 The laws hold for any case but the numeric estimates are made for binary case only while  $n$ -ary case involve a system call only once for each eight internal I/O operations or more, thus bringing to possibly different parameters.
  - 3 Both success in input or output than failure in input involve system calls and cost at most  $\beta$ , while failures in output are much cheaper, so output only may admit a better model. Differences in relative costs can be averaged and included in  $\varepsilon$ , so  $\varepsilon S$  no more represents the number of retries but the product  $\beta \varepsilon S$  represents anyway their cost in time.

specific system to be executed, with similar systems presenting similar values. This factor grows with the size  $E$  of the network for the length  $2^R$  of the temporal buffer (see option “-r”) due to hardware cache reasons. Since it holds  $D < 2^R / 4$ , the product  $E \cdot D$  constitutes a lower bound to determine both the memory usage of the inference engine ( $E \cdot 2^R$ ) and the coefficient  $\alpha$ . After an execution, module TINX reports the measured KLIPS and the index  $B$  on exit for further estimates.

To determine  $D$ , it is necessary to examine the temporal operators present in the source file and consider the longest finite relative time  $\Delta$  which appears there. If it refers to a single instant,  $D = |\Delta|$  but if  $\Delta_A$  and  $\Delta_B$  are the extremes of an interval it holds  $D = \max\{\min\{|\Delta_A|, |\Delta_B|\}, 2^N\}$  with  $N$  integer part of  $\log_2(1 + |\Delta_B - \Delta_A|)$ .

For further refinements, the executor produces as a statistic the average ratio  $\rho$  of filling of the network in percentage during the run, *which has to be in interactive mode by IPC or file to provide meaningful results* and if the temporal horizon  $H \gg 2^R$ , this coefficient can be used to weight the number of edges  $E$ . Also the overall number of mathematical literals  $L$  present in the network is reported by the compiler, as a sum on  $k$  of the number of occurrences  $L_k$  of each different literal. These numbers of occurrences can be extracted from the source file and the square sum  $L^2 = (\sum L_k)^2$  in previous formulas can be substituted by the smaller sum of squares  $\sum (L_k^2)$ , following:

$$T = \lambda (\rho E + \gamma \sum L_k^2) + \mu S$$

If a measure of KLIPS during an interactive execution is available and by denoting with  $\tau$  its reciprocal divided for one thousand, factor  $\mu$  can be estimated with better results by the formula:

$$\mu = (\tau - \lambda) (\rho E + \gamma \sum L_k^2) / S$$

About the other versions of the inference engine, the optimized dual core module TINX\_DT exhibits better performance<sup>4</sup> by parallelizing inference and I/O processes, but is characterized by the same experimental coefficients, while the multiple core module TINX\_MT does not perform as well for locking reasons. The basic formula for TINX\_DT becomes:

$$T = \max\{\lambda (E + \gamma L^2 + \zeta S_{in}), \mu S\}$$

where  $S_{in}$  is the number of binary input signals and  $\zeta$  coefficient to be determined experimentally.

---

4 This does not happen on every hardware / Linux kernel / compiler version.

Coefficient	Estimate
$\alpha$	Between $\approx 10 \cdot 10^{-9}$ ( $E \cdot D < 10^3$ ) and $\approx 60 \cdot 10^{-9}$ seconds ( $E \cdot D > 10^6$ ) are typical values on a 3 GHz processor
$\beta$	$\beta_{IPC} \approx 200 \cdot 10^{-9}$ seconds for binary IPC output and $\beta_{file} \approx 500 \cdot 10^{-9}$ seconds for binary file output on current Linux kernel and a 3 GHz processor with SSD as mass memory
$\gamma$	$\approx 1^5$
$\delta$	$\approx 10^{-2}$ but systems having different ratios $B / D$ will exhibit different values
$\varepsilon$	$< 5 \cdot 10^{-2}$ for binary output
$\eta$	$< 0.25$
$\rho$	Between $\approx 0.75$ and 1 for most systems
$\zeta$	$\approx 1$

---

5 This notation means the two values have the same order of magnitude.

## Benchmarks

The computer used for benchmarking has 8 cores Intel i5 at 3.1 GHz as processors, 8 GB of central memory and a 512 GB SSD as mass memory. As an example, the minimum sampling time of four binary dynamic systems has been first estimated by previous formula and then measured by command line, since the graphical shell drains part of the computing power of the inference engine and may provide less accurate figures. Even simpler systems can reach a number of KLIPS up to about  $96 \cdot 10^6$  on the adopted hardware, but real ones are bound in the range below.

Parameter	Minmark 16	Parbench 16	Hanoi 4	Hanoi 8
$E$	352	2554	4350	16053
$V$	256	1884	3058	11134
$V_{gate}$	32 (12.5 %)	288 (15.3 %)	505 (16.5 %)	2000 (18.0 %)
$V_{joint}$	160 (62.5 %)	1052 (55.8 %)	2079 (68.0 %)	7838 (70.4 %)
$V_{delay}$	64 (25.0 %)	544 (28.9 %)	474 (15.5 %)	1296 (11.6 %)
$L$	0 (0.0 %)	0 (0.0 %)	0 (0.0 %)	0 (0.0 %)
$D$	16	128	4	64
$S$	16 (1-bit output)	48 (1-bit output)	96 (12 · 8-bit output)	192 (24 · 8-bit output)
$R$	8	10	6	10
KLIPS (measured)	$\approx 64 \cdot 10^6 \text{ s}^{-1}$	$\approx 27 \cdot 10^6 \text{ s}^{-1}$	$\approx 75 \cdot 10^6 \text{ s}^{-1}$	$\approx 17 \cdot 10^6 \text{ s}^{-1}$
$B$ (measured)	$\approx 0.70$	$\approx 2.49$	$\approx 0.03$	$\approx 0.02$
$\rho$ (measured)	$\approx 95\%$	$\approx 97\%$	$\approx 77\%$	$\approx 79\%$
$\alpha$ (estimated)	$\approx 13 \cdot 10^{-9} \text{ s}$	$\approx 24 \cdot 10^{-9} \text{ s}$	$\approx 13 \cdot 10^{-9} \text{ s}$	$\approx 58 \cdot 10^{-9} \text{ s}$
$\lambda$ (estimated)	$\approx 16 \cdot 10^{-9} \text{ s}$	$\approx 37 \cdot 10^{-9} \text{ s}$	$\approx 13 \cdot 10^{-9} \text{ s}$	$\approx 58 \cdot 10^{-9} \text{ s}$
$\mu_{IPC}$ (estimate A)	$\approx 356 \cdot 10^{-9} \text{ s}$	$\approx 671 \cdot 10^{-9} \text{ s}$	$\approx 1143 \cdot 10^{-9} \text{ s}$	$\approx 2087 \cdot 10^{-9} \text{ s}$
$\mu_{IPC}$ (estimate B)	$\approx 401 \cdot 10^{-9} \text{ s}$	$\approx 725 \cdot 10^{-9} \text{ s}$	$\approx 899 \cdot 10^{-9} \text{ s}$	$\approx 4195 \cdot 10^{-9} \text{ s}$
$\lambda \rho E / T_{IPC}$ (est. B)	$\approx 45 \%$	$\approx 73 \%$	$\approx 34 \%$	$\approx 48 \%$
$\mu_{IPC} S / T_{IPC}$ (est. B)	$\approx 55\%$	$\approx 27 \%$	$\approx 66 \%$	$\approx 52 \%$
$T_{IPC}$ (estimate A)	$\approx 10.9 \mu\text{s}$	$\approx 124 \mu\text{s}$	$\approx 155 \mu\text{s}$	$\approx 1135 \mu\text{s}$
$T_{IPC}$ (estimate B)	$\approx 11.6 \mu\text{s}$	$\approx 127 \mu\text{s}$	$\approx 131 \mu\text{s}$	$\approx 1539 \mu\text{s}$
$T_{IPC}$ (measured)	$\approx 12.5 \mu\text{s}$	$\approx 127 \mu\text{s}$	$\approx 132 \mu\text{s}$	$\approx 1540 \mu\text{s}$

Measured KLIPS have been obtained by a run in quiet mode and do not depend from random factors related to I/O. Estimate A is based on formula  $T_{IPC} = \alpha (1 + \eta B) \rho E + \beta_{IPC} (1 + \varepsilon S) S$ , estimate B is more accurate and follows the formula  $T_{IPC} = \tau_{IPC} \rho E$  but requires a set of measures of KLIPS in interactive mode by IPC. Both estimates are computed in a spreadsheet included in the documentation.

## Credits and references

### Credits

TINX suite was designed and coded by Prof. Andrea Giotti, PhD, who wrote the theory behind its algorithms (procedure of progressive closure) and data structures (temporal inference networks) for his degree thesis in 2000. Basic Temporal Logic has been inspired in some of its conventions by TILCO language, previously developed by university of Florence (various contributors) for system specification. TINX project was started in 1998 by the author on an Amiga 4000 with SAS/C compiler and has been renewed from 2016 on a standard hardware with Linux operating system, GCC compiler, Flex + Bison and GTK libraries. If you find useful this development tool and you share the idea of free software, please send me some system specification written by you and it will be included among the examples of this package. Bug reports are welcome too.

### External references

- Tower of Hanoi is modeled as a constrain satisfaction problem and executed by TINX suite as described in the following article: </usr/share/doc/tinx/hanoi.pdf>
- Degree thesis of the author, which discusses the theory behind this software, is available in Italian only: </usr/share/doc/tinx/thesis.pdf>
- You can contact the author at the email address: <mailto:andrea.giotti@tin.it>