NAME

TSIF − *T*ransition *S*ystem *I*nterchange *F*ormat

A symbolic formalism to model hierarchical and coordinated timed Transition Systems.

DESCRIPTION

TSIF is a model to symbolically specify Transition Systems. The model is designed to support hierarchy and process coordination by means of synchronized labels and shared variables. **TSIF** is understood to be a baseline specification formalism to be used as a common ground to map other formalisms on it.

BASIC TRANSITION SYSTEMS

In this initial section we describe the basic features of the **TSIF** format to model Transition Systems that do not require hierarchy nor process synchronization.

TS MODEL DECLARATION

The description of any Transition System must be encapsulated between the *TS* and *END* keywords.

> *TS* <transition system name> {<transition system type>}
>
> {Here we include the declaration of the whole TS}
>
> *END*

The Transition System must have assigned a name <*transition system name*> and a optional type <*transition system type*> that can be either *INTERLEAVED* or *NON-INTERLEAVED* for asynchronous processes, or *SYNCHRONOUS* for synchronous processes. By default, Transition Systems are assumed to be *INTERLEAVED*.

Note that, current, only *INTERLEAVED* Transition Systems are supported.

This encapsulation of each Transition System is necessary because multiple **TSIF** models can be specified in a single file.

VARIABLE DECLARATION

The state of a Transition System is described by a number of variables in a Boolean algebra. Each minterm in the algebra maps onto a state, while characteristic functions describe sets of states.

TSIF requires to have an additional set of variables in order to specify the next-state evolution of the system. Each current-state variable must have an associated next-state variable. Next-state variables can be either specified in the **TSIF** model or internally created.

The next-state variable associated to a current-state variable *var* can be always accessed by using the *NS(var)* operator.

The variable declaration is as follows:

> {*INPUT* | *OUTPUT* | *INTERNAL*} *VARS* <variable declaration list>;

In general, *INTERNAL* and *OUTPUT* variables are used to describe the internal as well as the visible behavior of the system. *INPUT* variables describe the state of the environment. The <*variable declaration list*> specifies a list of declared variables.

LABEL DECLARATION

The dynamic behavior of a Transition System is specified by means of *labels*. Each label represents an abstract operation that can be executed by the system.

The label declaration is as follows:

{*INPUT* | *OUTPUT* | *INTERNAL* | *DUMMY*} *LABELS* <label declaration list>;

In general, *INPUT* and *OUTPUT* labels are used to describe the visible behavior of the system, either expect at the inputs or generated at the outputs. *INTERNAL* and *DUMMY* labels describe the behavior that is hidden inside the system.

EVENT DECLARATION

Since the same operation (*label*) may be executed in different contexts, a second identification level is offered by means of *events*. A number of events could be associated to each label.

The event declaration is as follows:

EVENT [<event name>] <label name>

{Here we include the declaration of the event}

END

The name of the event is just informative. An event name cannot be duplicated for the same label, but there is no restriction between different labels.

If no <event name> is specified, the event to be created will share the name of the label <label name>. This is specially useful when only one events is specified per label.

Each event should have an associated Transition Relation to describe how the state of the Transition System changes (i.e. the variable change) when the event is executed.

Depending on the application, other additional functions could be associated to an event. The list of functions should be encapsulated between the *EVENT* and the *END* keywords.

FUNCTION SPECIFICATION

The **TSIF** format describes Transition Systems by means of Boolean functions. These functions can describe a large variety of elements, e.g. sets of states, transition relations among states, failure conditions, etc.

Boolean functions are internally manipulated by means of Binary Decision Diagrams (BDDs), but should be specified in the *EQN* format in the **TSIF** description.

An *EQN* function declaration is as follows:

EQN <function name> [<function type>]

expression1; expression2; ...

The particular semantic associated to a function is defined by the <function name> parameter. Each function is exclusively associated to a package inside **TSIF**. The most common package is the reachability package (**RGA**). RGA provides support to manipulate Transition Relations (TR), Enabling Functions (EF), Firing functions (FF), the initial state of the system (ISTATE), etc. A complete list of functions and packages can be found at section TS PACKAGES.

After the function declaration it must follow the function itself. The optional <function type> parameter indicates if the function is partitioned into a conjunction (*CONJUNCTIVE*) or disjunction (*DISJUNCTIVE*) of smaller functions (by default partitioned functions are assumed conjunctive).

In the *EQN* format an *expression* is a logic function with the classical Boolean operators:

expression :: variable
| NS(variable)
| (expression)
| expression'
| expression + expression
| expression * expression
| expression expression
| expression = expression
| expression <> expression
| expression -> expression

PACKAGES:

A set of different functions can be specified in a Transition System. Each function is associated to a particular package in which will offer a certain functionality. By default the **TSIF** format includes the **RGA** (Reachability Graph Analysis) and **DELAY** packages. **RGA** is designed to compute and provide basic symbolic analysis on the state space of the Transition System. **DELAY** provides basic support to specify min/max delays into the events.

Possible functions that can be associated to a Transition System are:

ISTATE Defines the set of initial states of the TS by using its characteristic function. In general more than one initial state is possible.

PARTIAL
Defines a subset of reachable states of the TS (it must be a connected subset including the initial state).

FAIL Defines a failure function of the TS by using the characteristic function of a set of conditions. Failure states can be identified as any situation or condition in which the system behaves incorrectly.

RSTATE
Defines the set of reachable states of the TS by using its characteristic function.

UBOUND
Defines an upper bound for the set of reachable states of the TS by using its characteristic function. In general, upper bounds should improve the symbolic analysis by increasing the set of binary codes that can be considered as don't care during the analysis.

At least one function must be defined associated to the initial state of the TS (ISTATE).

Possible functions that can be associated to an event are:

TR Defines the Transition Relation of some object inside the **TSIF** format. Generally it will be associated to events, but labels, variables, and Transition Systems themselves may have one associated.

EF Defines the Enabling Function of some object inside the **TSIF** format. The Enabling Function specifies the set of states in which the object is defined to be enabled to execute. However, it will no be really executed until the Firing Function (see below) is reached. Generally it will be associated to events, but labels, variables, and Transition Systems themselves may

have one associated.

FF Defines the Firing Function of some object inside the **TSIF** format. The Firing Function specifies the set of states in which the Transition Relation of the object can be effectively applied; that is, it will generate some additional state. The difference between being Enabled or being Firable defines the Lazy Delay Model. For untimed systems the Enabling and Firing functions must coincide. Generally it will be associated to events, but labels, variables, and Transition Systems themselves may have one associated.

FAIL Defines a failure function of the event by using the characteristic function of a set of conditions. Failure states can be identified as any situation or condition in which enabling or executing the event can be considered an error.

At least one function must be defined associated to the transition relation of the event (TR). If no EF/FF function is specified, they are assumed to be equal and will be automatically extracted from the TR.

COORDINATED TRANSITION SYSTEMS

In this section we extend the basic features of the **TSIF** format to cover Transition Systems that exhibit hierarchy and process coordination. The goal of the model is to allow a top-down modeling flow.

TRANSITION SYSTEM INSTANTIATION

A complex Transition System does not need to be specified as a flat collection of events. Following a top-down design philosophy a Transition System may contain a number of events and additional instances of other Transition Systems. In that way we can encapsulate the behavior of other processes in a design.

The process declaration is as follows:

PROCESS <formal-ts-name> : <actual-instance-list>;

The <formal-ts-name> refers to the name of a Transition System that must be already present in **TSIF**. This is mandatory in order to guarantee a complete semantic check when a system is read.

The instantiation process will recursively create as many copies of the formal TS as indicated by the list of actual instances. Note that in the scope of a given TS each process instance must have a unique name.

Given an instance with name <*ts-name*> and instance <*instance-name*>, the internal instance can be access by using both names sequenced by a dot (<*ts-name*>.<*instance-name*>. This method can be used to access any instance in the hierarchy tree, i.e.

<*instance-name*>.<*instance-name*>...

Variables and labels in the interface of an instance can also be acceded by using the same mechanism:

<*instance-name*>.<*var-name*>

<*instance-name*>.<*label-name*>

OVERVIEW ON TS COORDINATION

Transition Systems can be mutually coordinated if they are at the same hierarchy level. **TSIF** offers two coordination mechanisms: label synchronization and variable sharing.

Synchronization refers to the simultaneous execution of events in different instances. This type of coordination is specified by indicating the set of labels that must be synchronized.

Variable sharing refers to the fact that one instance can partially or totally observe the state of another instance. The first instance can take decisions upon the actual state of the second instance. Since the states in **TSIF** are encoded onto Boolean variables, sharing the state between instances implies sharing the

variables.

Both coordination mechanisms are totally independent. Querying the state of an instance does not imply any synchronization; while synchronization is provided as a rendez vous mechanism without any involved data exchange.

Finally, note that coordination can only occur between labels and variables that have been declared to be in the interface of the instances; that is, declared to be either *INPUT* or *OUTPUT*.

LABEL SYNCHRONIZATION

The simultaneous execution of events from different instances is specified by synchronizing among them the corresponding labels. One of the processes is responsible for starting the synchronization (active process), while the rest of processes are stopped until the synchronization is executed (passive processes).

The label synchronization declaration is as follows:

SYNCH <label-name> : <list-of-labels>;

A synchronization statement specifies that a set of labels inside the instances of the TS are to be executed simultaneously. The synchronized labels are specified by a pair <*instance-name*>.<*label-name*>, indicating the actual label inside each instance to be synchronized.

Note that all labels must be inside the scope of the TS; that is, only labels from the immediately preceding hierarchical level can be synchronized in that way.

All synchronized labels are associated to a label <*label-name*> in the current hierarchical level. While all synchronized labels inside instances must be declared either input or output, <*label-name*> could be declared internal if desired, or made visible to upper levels in the hierarchy by declaring <*label-name*> either input or output.

When a set of labels is synchronize, note that only one of them can be an output, while the rest should be inputs.

VARIABLE SHARING

The total or partial observation of the state of an instance from a second instance is specified by sharing variables among them. One of the processes is responsible for defining the value of the variables at each state (writing process), while the rest of processes can only observe its value (listening processes).

The variable sharing declaration is as follows:

SHARED <var-name> : <list-of-variables>;

A shared variable statement specifies that a set of variables inside the instances of the TS are to be considered to be the same variable. The shared variables are specified by a pair <*instance-name*>.<*var-name*>, indicating the actual variable inside each instance to be shared.

Note that all variables must be inside the scope of the TS; that is, only variables in the immediately preceding hierarchical level can be shared in that way.

All shared variables are associated to a variable <*var-name*> in the current hierarchical level. While all shared variables inside instances must be declared either input or output, <*var-name*> could be declared internal if desired, or made visible to upper levels in the hierarchy by declaring <*var-name*> either input or output.

PARAMETERIZED PROCESS INSTANTIATION

Transition Systems may instantiate multiple times a given process. However, it is common to find that this very same process requires to be initialized in different conditions.

To avoid having to duplicate a Transition System for each possible initial condition, the **TSIF** format allows to parameterize the instantiation of a process. Basically, some specific functions can be attached to each instance, in order to refine its initial state.

The syntax of the process instantiation can be extended in the following way:

PROCESS <process-name> : <instance-name> { <list-of-parameters> } ... ;

Each <*instance-name*> can be attached an optional <*list-of-parameters*> that must be encapsulated between brackets. Two different functions (in EQN format) can be attached to an instance:

ISTATE Declares the subset of original initial states of the instantiated TS that must be considered as part of the initial state of the coordinated Transition System.

FORCE Declares a new set of initial states of the instantiated TS. This set is completely unrelated with the original set of initial states.

INITIAL STATE

Coordinating multiple Transition Systems by using variable sharing requires a carefully specified initial state. The initial values specified for a shared variable at each instance must be *consistent*.

The initial states of a coordinated Transition System is defined as the intersection of the initial states defined for each process. Therefore, the initial states at each process are defined to be consistent if the initial state of the coordinated Transition System is not empty.

FAILURE CONDITIONS

In this section we will describe the mechanisms offered by the **TSIF** format to describe failure conditions.

TSIF supports two built-in methods for the analysis of properties in a Transition System: safety properties and temporal automaton.

Safety properties define either conditions on sets of states or on the potential firing of the events in the system that must be satisfied.

Temporal automaton should be used when more complex properties have to be verified. However, the overhead introduced by this mechanism, prevents its application when the number of properties to verify at the same time is large.

SAFETY PROPERTIES

TSIF supports the automatic verification of safety properties defined as failure conditions. Each condition is specified as a logic proposition that can pose restrictions on the value of the current state variables as well as next state variables. Safety properties can be verified by specifying the characteristic function of the failure condition.

Failure conditions that only depend on current state variables are called STATE conditions because define

conditions on the reachable states of the system.

Failure conditions that depend on a mixture of current and next-state variables are called TRANSITION conditions, because define conditions on possible transitions from reachable states of the system.

ASSOCIATING FAILURES TO A SYSTEM

A failure condition can be associated to different objects in a Transition System. Its semantics will depend on the type of the object. The valid objects to associate a failure are:

A Transition System or any of its instances.

Labels in the transition system.

Events in the transition system.

State conditions associated to a Transition System or to an instance may fail at any reachable state. Transition conditions may fail at any enabled transition from a reachable state.

Failure conditions for transition systems can be specified at any point between the *TS* and the *END* keywords:

TS <transition system name> {<transition system type>}

<list_of_failure_conditions>

END

State conditions associated to labels or events may fail at any reachable state in which the label (or event) is enabled. Transition conditions may fail at any firable transition of the label (or event) from a reachable state.

Failure conditions for events can be specified at any point between the *EVENT* and the *END* keywords:

EVENT [<event name>] <label name>

<list_of_failure_conditions>

END

Failure conditions for labels can only be specified just after the label declaration:

<label type> *LABELS* <label1> {<list_of_failure_conditions_1>} <label2> ... ;

USER SPECIFIED VS. BUILT-IN CONDITIONS

Failure conditions can be specified with an equation in the **TSIF** model or directly provided from the verification environment.

When specified from the **TSIF** model we will use the usual EQN format with the dedicated FAIL keyword.

The *EQN* format also provides support to manipulate the most common objects required in the symbolic analysis of Transition Systems:

expression :: TR(label)
| EF(label)
| FF(label)
| TR(event,label)
| EF(event,label)
| FF(event,label)

However, note that these functions can never be used to specify the Transition Relation or other objects inside the events. They can only be used to specify failure conditions or to analyze the state space after it has been generated.

TSIF also supports several built-in failure conditions commonly used in the analysis of concurrent systems. The available conditions are:

NON PERSISTECY: (keyword CHECK_PER)
Checks that once a label/event is enabled it cannot be disabled without firing. This condition is only applicable on labels or events.

INDUCED NON PERSISTECY: (keyword CHECK_IPER)
Checks that the firing of a label/event does not disable other labels/events. This condition is only applicable on labels or events.

CONFORMANCE: (keyword CHECK_CONF)
Check that whenever the output label in a synchronization is firable, all the input labels in the same synchronization are enabled to accept it. This condition is only applicable on labels. Only applies to label synchronizations.

DEADLOCK (keyword CHECK_DLCK)
Check if there is any state from which no label/event can be fired. This condition is only applicable on transition systems and currently refers to a global deadlock in the global system.

Built-in failure conditions should be specified in the same place than EQN failure equations. The conditions must be in the following format:

{RGA: "failure_condition_keyword"}

FILES

TS/report
TS/examples
TS/benchmarks

SEE ALSO

blif(1), astg(1), transyt(1).

AUTHOR

Enric Pastor
Department of Computer Architecture
Universitat Politecnica de Catalunya
Barcelona, Spain
enric@ac.upc.es