Information Societies Technology (IST) Programme

Future and Emerging Technologies - Open Domain

Project acronym: Advance

Project full title:

Advanced Validation Techniques
for Telecommunication Protocols

Contract no: IST-1999-29082

Deliverable 4

Integration of Abstraction and Symbolic Analysis Tools for Extended Automata

Year 1 : October 2000 - September 2001

Date of preparation: October 2001

Contributors: Paris, Liège, Edinburgh, Grenoble

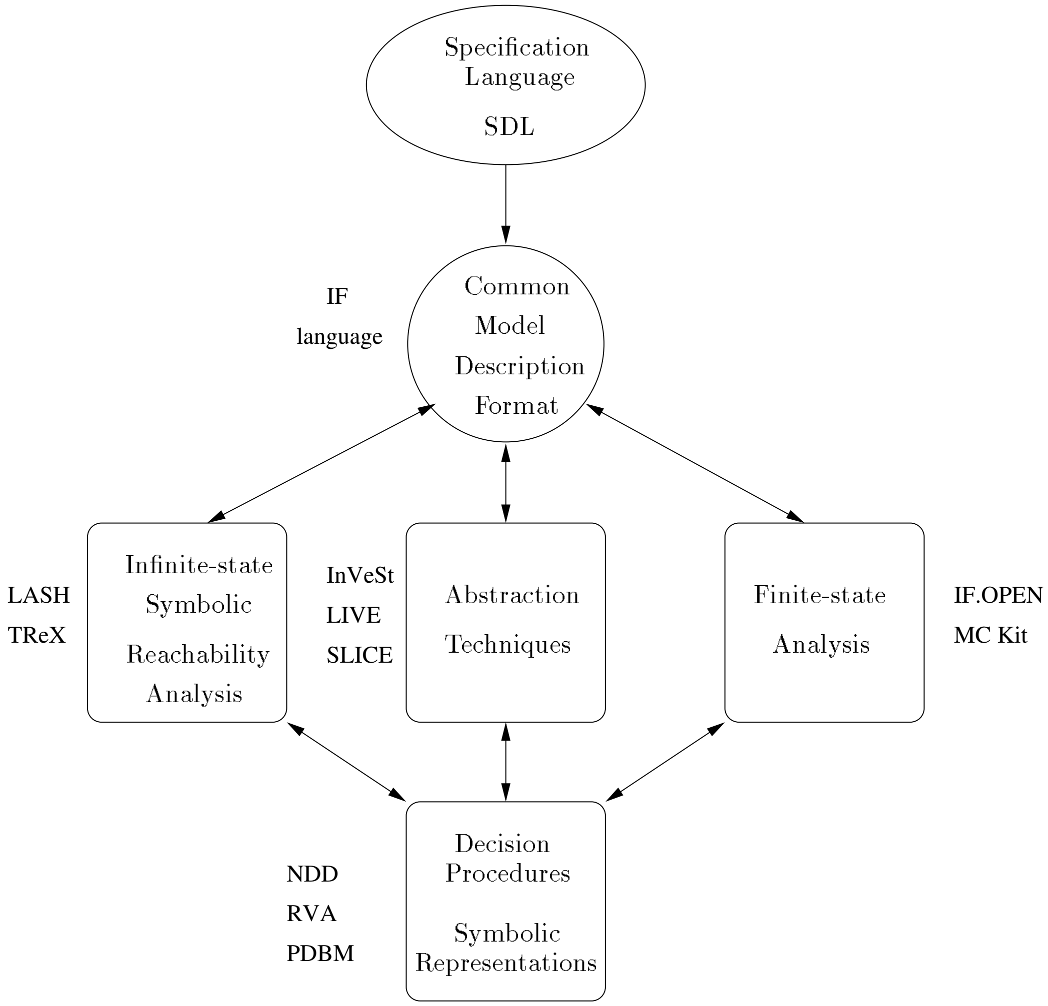

Figure 1: A specification and validation environment

1 Introduction

The final goal of the project is to develop new analysis tools and to integrate them into a common specification and validation environment.

The work carried out this year constitutes an important step toward this goal: (1) Many new techniques and verification algorithms have been implemented, creating original tools, or extending significantly the scope of existing ones. (2) Tools for finite-state and extended automata have been interfaced with the common description language IF. This allows the extension of the IF environment where SDL can be used as a specification language through a connection between IF and the specification tool Object-Geode of Telelogic.

2 Specification and validation environment

The current picture of the specification and validation environment under construction is given in Figure 1. In this environment, several analysis and model-checking tools are interfaced with the common model description language. These tools use various decision procedures and symbolic representation packages. The names of the packages and tools we have developed which are elements of this environment are shown on the picture. We give hereafter a description of each of these components.

2.1 Decision procedures and Symbolic representations

An important effort has been devoted this year to the design of new data structures for symbolic representation of infinite state spaces. These efforts lead to the implementation of several packages allowing the representation and manipulation of sets of integer and real vectors.

The NDD and RVA packages have been developed by the group of Liège in the tool LASH. These data structures are based on finite-state automata and allow to express and manipulate efficiently linear arithmetical constraints on integers and reals.

The PDBM package has been developed by the groups of Paris (**UP7**) and Grenoble (**UJF**). PDBMs are parametric extensions of the Difference Bound Matrices used for the representation of a class of polyhedra called *zones* in the timed automata literature (corresponding to boolean combinations of interval constraints on differences between variables). In PDBMs, bounds of intervals can be parametric (arithmetical expressions including parameters), and parameters can be constrained using arithmetical formulas. A package allowing the manipulation of these data structures has been implemented in the tool TReX. This package uses internal decision procedures (e.g., a local implementation of the Fourier-Motzkin quantifier elimination procedure) as well as external decision procedures for arithmetics (Omega for Presburger arithmetics, and Reduce/Redlog for the arithmetics of reals).

2.2 Infinite-state Symbolic Reachability Analysis

The tool LASH is developed in Liège. It allows analysis of extended automata, such as automata with counters and clocks, using automata-based symbolic representation structures (NDD, RVA). Acceleration techniques are used in order to help termination of the computation of reachability sets. The techniques used in LASH are based on the concept of *meta-transition* corresponding to computing (in one step) the effect of iterating an arbitrary number of times a sequence of actions (a control loop).

The tool TReX is developed in Paris (**UP7**) and in Grenoble (**UJF**). This tool allows the analysis of extended automata manipulating clocks and integer counters with parametric guards, communicating through shared variables and unbounded FIFO channels. TReX allows the construction of the reachability sets and finite abstract models of the analyzed systems. TReX uses PDBMs to represent sets of integer and real vectors, and it allows to manipulate nonlinear arithmetical constraints. TReX uses accurate extrapolation techniques to accelerate fixpoint calculations during the reachability analysis.

Both tools, LASH and TReX are interfaced with the common model description language IF.

2.3 Finite-state Analysis and Model-checking

2.3.1 The Model Checking Kit

The Model Checking Kit (MCK) is a collection of programs which allow to model a finite-state system using a variety of modeling languages, and verify it using a variety of checkers, including deadlock-checkers, reachability-checkers, and model-checkers for the temporal logics CTL and LTL. The most interesting feature of the Kit is that, independently of the description language chosen by the user, (almost) all checkers can be applied to the same model. The checkers that are currently available include PEP, PROD, SMV, SPIN, and others.

We have implemented an interface between the common description language (IF) and the Kit. It includes a a compiler from IF into safe Petri Nets (the basic model of the Kit), a formula converter which translates properties of the original model into Petri net properties, and a trace interpreter which translates error traces of the Petri net back into traces of the original model. The fragment covered by the compiler excludes the real-time features of the language and is limited to finite data types (since these cannot be modeled in the target language), but includes important features like dynamic process creation or different signal delivery disciplines.

More precisely, the list of supported features is as follows:

- Systems consist of multiple processes running in parallel; multiple instances of a process can exist simultaneously, but the maximum number of concurrently active instances has to be fixed a priori.

- Processes can communicate by exchanging signals or through shared variables. Signals (which may take arguments) can pass either through signalroutes or through direct communication between processes. Incoming signals are stored in a message queue. A limitation is that the capacity of the message queue has to be fixed a priori.

- Signalroute can have different delivery disciplines in the common language most of which are supported by the compiler: FIFO and multiset queueing, reliable and lossy transmission, peer-to-peer, unicast, and multicast delivery. Moreover, the common language provides for urgent (immediate) and delayed delivery. In the case of delayed delivery, the language allows to place temporal constraints on the amount of delay and the transmission rate. These temporal restrictions are not supported by the compiler, but distinction is made between immediate and delayed delivery where signals are stored in a buffer (of fixed size).

- The following data types are supported by the compiler:

  - finite integer fragments
  - process/signalroute ids
  - booleans
  - enum and range types

  The following types are not supported

  - floats, clocks and unbounded integer types
  - arrays and records (though these could be added if necessary)
  - strings and abstract types

- Procedures (which are used to import C code into the system) are not supported.

- Of the guards that control entry into a state transition, we support the input and the provided guards. The temporal guards tpc, deadline, or when are not supported. Save sets are not currently supported, but can be considered for implementation if needed.

- Of the actions available inside state transitions, we support manipulation of variables (task), sending of signals (output), and the dynamic creation of processes (complete with the passing of arguments) and signalroutes, as well as process/signalroute destruction (kill). Actions to do with clocks (set/reset) are not supported.

2.3.2 The IF.OPEN tool-set

IF.OPEN is a set of modules for the exhaustive simulation and validation of IF specifications:

- **libifmodel.a** : is a library providing an access to the abstract tree of IF specifications.
- **libifsimulator.a** : is a library of components needed for the simulation of IF specifications. It provides implementation for structures such as strings, events, labels, etc but also for various kind of hash-tables.
- **if2c** : is a program which generates the C primitives needed for the exhaustive simulation of IF specifications. Basically, it generates a C type encoding the state of the system and an iterator function over states. This C code is intended to be compiled and linked together with some specialized application (e.g. generator) to obtain an executable for the IF specification.
- **generator** : is a reachability analysis tool of IF specifications.

2.4 Abstraction and reduction techniques

2.4.1 InVeSt

InVeSt is a tool for automatic generation of abstractions, given a concrete (infinite-state) model and a state abstraction relation. This tool is implemented on top of the theorem prover PVS which is used to build the abstract transition relation. When a property is not satisfied by the abstract model, the tool provides a counter-example which can be analyzed automatically to check if it corresponds to a counter-example in the concrete system, and if not, to refine the considered abstraction relation. Connections between IF and InVeSt are under constructions.

2.4.2 The LIVE and SLICE tools

LIVE and SLICE are static analysis tools for IF specifications developed by Grenoble (**UJF**).

LIVE performs live variables optimization: it transforms an IF specification into an equivalent one by adding systematic resets of non-live variables. This transformation preserves completely the behavior of the original specification while the global state space (and further, the exploration time) are dramatically reduced.

SLICE performs reduction techniques inspired from program slicing. These techniques allow to extract from a model portions which are relevant to the validation of the properties under consideration. This extraction is done by projecting a model on a set of relevant actions ; it can be seen as an action-based abstraction technique. Slicing allows to obtain smaller IF models with manageable state graphs.