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 9

Common Interface for a General Validation Environment

Year 2: October 2001 – September 2002

Date of preparation: October 2002

Contributors: Grenoble, Paris, Liège, Uppsala, Edinburgh

Introduction

The common interface of the Advance environment have been built upon the common description language IF-2.0. The interface consists of several Application Programming Interfaces (APIs) allowing to develop or to connect existing validation tools to the common language. The first application, an integrated validation environment (IVE) connecting the main tools of the Advance project (LASH, TReX, IF-2.0 toolbox) have been developed.

The common language API

The common language IF-2.0 comes equipped with a parser allowing to construct the abstract syntax tree from textual specifications. The tree is constructed as a collection of C++ objects and could be accessed, modified or deleted by some user application.

For example, you find below a small C++ program which reads an IF-2 description, build the abstract syntax tree, type check it, then writes out the names of the processes. It could be extended easily to perform more complex analysis on the syntax tree e.g, translate it to some other format, filtering out some constructs, etc.

```
#include <model.h>

void  main(int argc, char *argv[]) {
  IfObject::Initialize();

  IfSystemEntity *entity = Load(stdin);
  if (! entity->Compile())
    return 0;

  for(int i=0;i<entity->GetProcesses()->GetCount();i++) {
    IfProcessEntity* process=entity->GetProcesses()->GetAt(i);
    printf("%s\n", process->GetName());
  }
}
```

The tree interface is included in the standard IF-2.0 distribution (see http://www-verimag.imag.fr/~async/IF). Using this interface, the following tools have been connected to the common language:

- **TReX – connection done by Paris**

TREX [3] (*Tool for Reachability Analysis of Complex Systems*) is a tool for reachability analysis of extended automata. More precisely, the model supported is a (finite) set of (finite control) automata with parameters (i.e. uninstantiated constants), clocks, counters, and lossy fifo queues. The release 1.3 of TREX includes an IF-2.0 language compiler.

The compiler acts in two phases: filtering and translation. The filtering phase consists in checking that the IF-2.0 description can be handled by TREX. For instance, complex data types (i.e., records, arrays, trees), procedures, and communication buffers other than lossy fifo queues are not supported. If the check fails, it outputs a report indicating which features are not supported and the variables that should be sliced in order to obtain an abstract description which can be handled by TREX. If the check is successful, the IF-2.0 description is translated

into the input language of TREX. During the translation, a semantic analysis detects automatically the parameters of the model (i.e., variables which are not assigned).

- **LASH - connection done by Liege**

The Liege Automata-based Symbolic Handler (LASH) [4] is a tool for building and manipulating finite-state representations sets, and exploring the state space of infinite-state systems. The release 0.9 of LASH include an IF-2.0 language compiler.

The IF-2.0 language elements associated to features that are not handled by LASH are ignored, wherever possible. For instance, because the current representation system used by the state-space exploration module of LASH operates on integer vectors, declarations of non-integer variables and operations involving these variables --- although syntactically accepted --- are abstracted out of the compiled model.

The input language is extended with features that are essential to LASH, but that cannot be expressed in IF 2.0. The main extension concerns the definition of *meta-transitions* which are necessary to exploring infinite-state spaces with a finite amount of resources.

- **McKIT - connection done by Edinburgh**

The Model-Checking Kit [1] allows to model finite-state systems using a variety of input languages, translate them for use with a variety of model-checkers (including PEP, SPIN, PROD, SMV), and translate the result back into the terms of the original model. Previously, Edinburgh had added support for IF as an input language (see deliverable D4).

Since most model-checkers featured in the Kit work on finite-state systems, the IF interface of the Kit only supports a finite-state fragment of the language, excluding clocks and other infinite data types. Therefore, the interface has now been extended by a filter. When an IF model uses data types not supported by the Kit (such as a clocks), the Kit can use the Slice tool (developed by Grenoble) to obtain a projection which does not use variables of these types. In such a case, the Kit informs the user about the necessary changes and checks the abstracted model instead.

- **RMC (Regular Model Checking) connection done by Uppsala**

The Uppsala team has implemented part of the IF language with the parameterized extensions as a translator to their own language for reasoning about regular relations. The subset is large enough to cover many classes of parameterised systems, such as mutual exclusion algorithms e.g. Peterson algorithm.

The considered fragment of IF supports arrays of finite types, with assignment to individual elements as well as comparison between elements. It also supports quantification, either existential or universal. It is also possible to compare two indices for equality.

The translation tool can be found as a part of the regular model checking toolset, and is available from the web site http://www.regularmodelchecking.com/.

Finally, we mention here the experiment done by Yves-Marie Quemener at France Telecom Lannion, concerning the generation of discriminating test sequences. At Verimag, using this interface we have upgraded the static analysis tools (live variables analysis, dead-code elimination, forward variable elimination).

The exploration API

The exploration interface gives access to the semantic model of IF-2.0 specifications. Semantic models are labelled transition systems. Basically, the interface provides functions to compute 1) the initial state and 2) the set of successors for some given state. Using these functions, it is possible to construct the models *on-the-fly* in an explicit manner.

Below is a small program computing the initial state of the specification as well as the list of transitions outgoing from it. You can easily extend it to perform a complete traversal of the whole model.

```
#include "simulator.h"

class MyTool : public IfDriver {

 public:
  MyTool(const IfEngine* engine)
  : IfDriver( engine ) { }

  virtual void explore(IfConfig* source, IfLabel* label,
                       IfConfig* target)
    { label->print(stdout); printf("\n"); }
};

void main(int argc, char* argv[]) {
  IfIterator iterator;
  MyTool tool( &iterator );

  IfConfig* start = iterator->start();
  start->print(stdout); printf("\n\n");

  iterator->run(start);
}
```

The simulation interface is also included in the standard IF-2.0 distribution (see http://www-verimag.imag.fr/~async/IF).

Using the exploration interface we have implemented several tools allowing mainly to debug IF-2.0 specifications and to construct completely their models, when possible. Debugging can be done either by random or interactive simulation of the specification. Explicit model construction can also be controlled by several parameters such as the traversal mode (depth-first or breadth-first), partial order reduction, output format, etc. Models are generated in Aldebaran format (.aut) thus they are compatible with most of the (finite-state) verification tools included in the CADP toolbox.

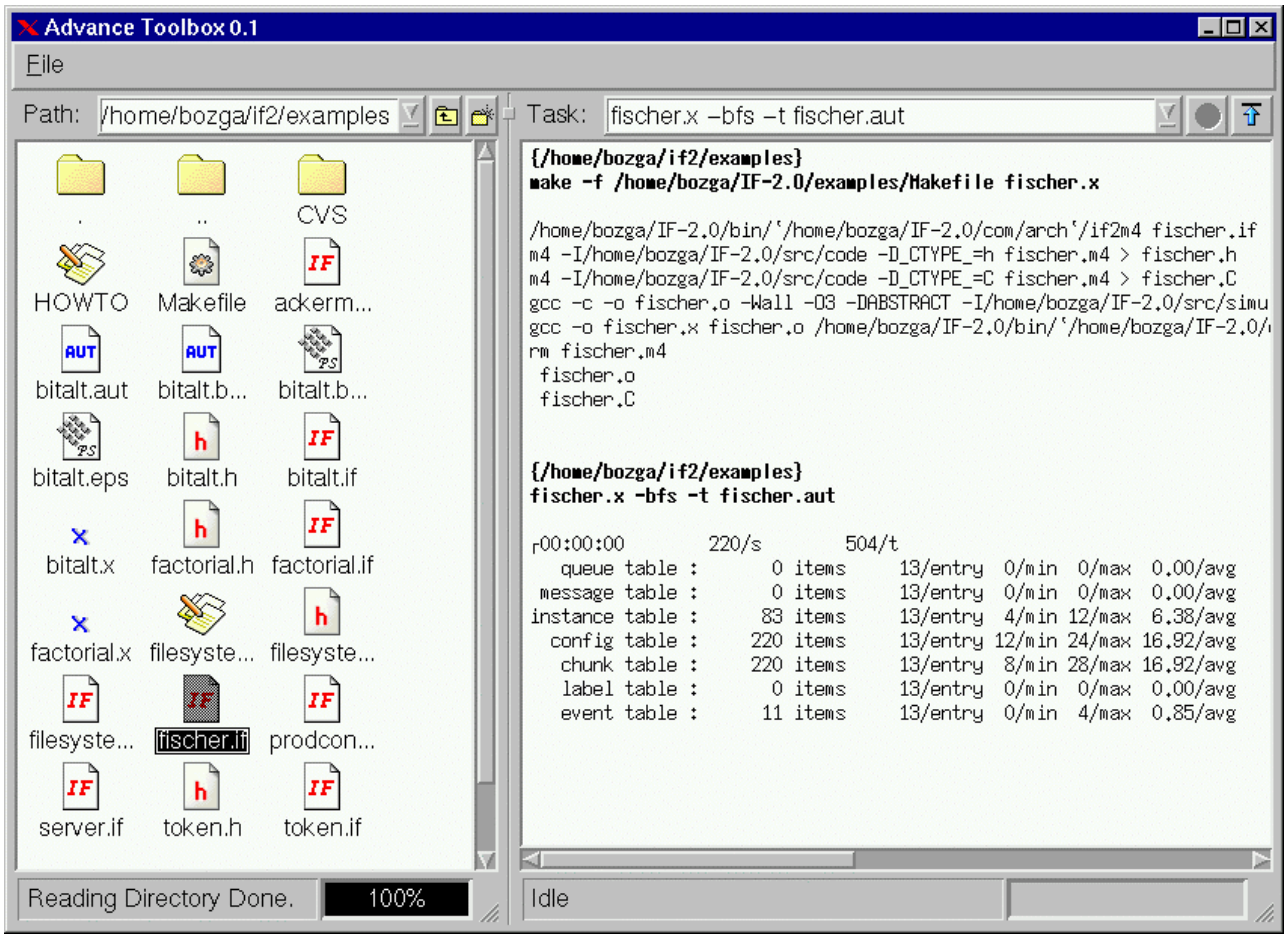
The model-construction tools are fully integrated in the verification environment, see below. Debugging tools are not connected yet.

The integrated validation environment

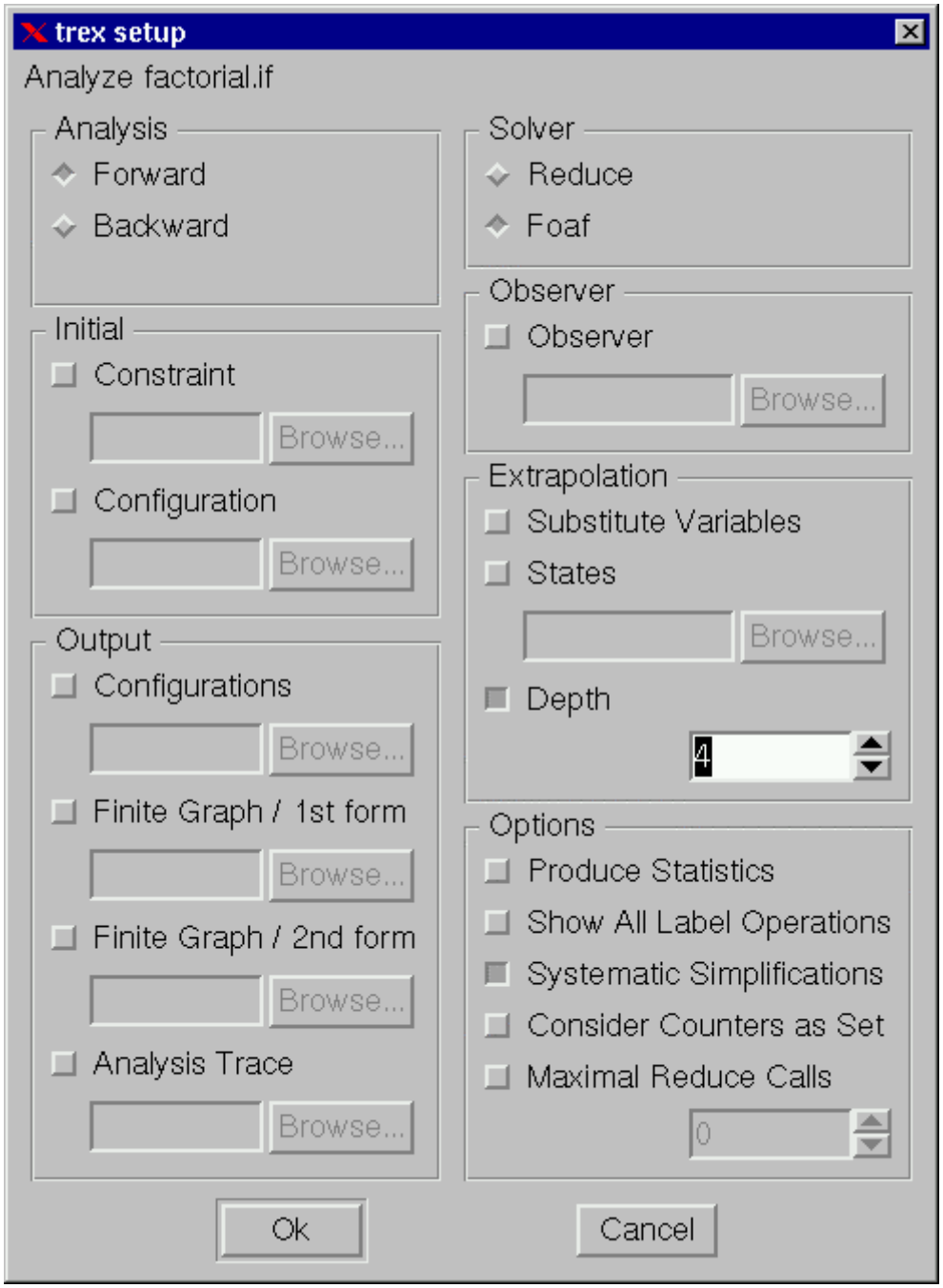
We have implemented a first prototype of a integrated validation environment. It is built upon the common language IF-2.0 and rely on the two application programming interfaces described before.

Currently, the environment provides *general* functionalities such as file browsing, file editing or task management as well as *specific* validation functionalities such as explicit model construction, symbolic analysis (using TReX and LASH), model minimisation (using Aldebaran), model

visualisation (using dot). The figure below gives an overview of the main window. By clicking on IF files, the tool will open a menu from where the user can invoke one of the available tools. For each tool, the environment provide a configuration window where the user may choose the right parameters for his IF file.



The next figure presents the TReX configuration window. It is rather complex but provides a complete control over all the TReX running options (analysis method, input and output files, extrapolation, initial constraints, backend solver, etc). Similar windows exist for all the currently integrated tools.



The environment is completely open, in particular, new functionalities or tools could be added with little effort. We envisage to integrate all the Advance tools.

The environment (tools and interface) could be freely downloaded from the Advance project Web site http://verif.liafa.jussieu.fr/~haberm/ADVANCE/main.html.

References

1. C. Schroter, S. Schwoon and J. Esparza. **The Model Checking Kit**. In *Proceedings of Tools Day Workshop* (CONCUR'02), Masaryk University, Brno, FIMU-RS-2002-05. See also http://wwwbrauer.in.tum.de/gruppen/theorie/KIT

2. M. Bozga, S.Graf, L. Mounier. **IF-2.0: A Validation Environment for Component-Based Real-Time Systems**. In *Proceedings of CAV'02, Copenhagen, Demark.* LNCS 2404. See also http://www-verimag.imag.fr/~async/IF

3. A. Annichini, A. Bouajjani, M.Sighireanu **TreX : A tool for Reachability Analysis of Complex Systems**. In *Proceedings of CAV'01, Paris, France*. LNCS 2102. See also http://www.liafa.jussieu.fr/~sighirea/trex

4. B. Boigelot, L. Latour. **The Liege Automata-based Symbolic Handler (LASH)**. Available at http://www.montefiore.ulg.ac.be/~boigelot/research/lash

5. M. Nilsson, Julien d'Orso. **Regular Model Checking (RMC) tool**. Available at http://www.regularmodelchecking.com/