IF: a validation environment for asynchronous real-time systems

L. MOUNIER

Distributed System Group

Grenoble
The Context

Application area:
Telecommunication and distributed systems

Main characteristics:
- asynchronous communications
- "real-time" features
- critical systems

⇒ introduction of validation techniques into the development cycle.
Formal validation: the current situation

Specification formalisms

- use of international standards: Estelle, Lotos, SDL, UML, ...
- a difficult trade-off between:
  • programming facilities (e.g., high-level primitives)
  • validation facilities (e.g., real-time semantics)

Validation tools

- commercial environments:
  • edition, simulation, code generation, test case generation
  • support the existing standards
- academic tools:
  • efficient verification techniques
  • restricted input language
Motivations

**IF**: an intermediate representation for timed asynchronous systems

- a *connection* between commercial and academic tools
  ⇒ bridge the gap between standards and low-level formalisms

- a “*source level*” intermediate representation
  ⇒ allows efficient optimisation and verification techniques

- relies on a powerful and flexible *time model*
  ⇒ a laboratory to study the real-time semantics of high-level formalisms
Outline

• Motivations

• IF: the language

• The IF validation environment

• Some case studies

• Conclusion and perspectives
The IF intermediate representation

Communicating extended timed automata (with urgencies)

Communication
- asynchronous message buffers (reliable/lossy/bounded)
- synchronous rendez-vous
- shared variables

Time model
Timed Automata with urgency attributes on transitions
Timed automata with urgency [BonomiSifakis96]

Time progress depends on urgency of enabled transitions:

- **eager** transitions are urgent as soon as they are enabled and block time progress
- **lazy** transitions never block time progress
- **delayable** transitions allow time progress unless time progress disables it

⇒ allows to express a large spectrum of real-time paradigms.
**Transition Urgency**

- **x:** clock
- **x := 2**
- **1 < x < 3**
- **urgency**

- **Eager**
  - Transition: \(q_0 \rightarrow q_1\)
  - Condition: \(x > 2\)

- **Lazy**
  - Transition: \(q_0 \rightarrow q_1\)
  - Condition: \(1 < x < 2\)

- **Delayable**
  - Transition: \(q_0 \rightarrow q_1\)
  - Condition: \(x < 1\) or \(x > 3\)
Outline

- Context and Motivations
- IF: the language
- The IF validation environment
- Some case studies
- Conclusions and perspectives
Architecture of the toolbox

ObjectGEODE
specification design

SDL

SDL2IF

IF

SLICE
LIVE
static analysis

IF2C

KRONOS

C ADP
model-checking

LTS

state explosion

IF2PML

Promela
Spin

TG V
test generation

specification design

state
explosion
The frontend component: sdl2if

Translation from SDL to IF:

- based on an ObjectGeode API
  ⇒ we follow standard evolution of SDL

- supports a static subset of SDL:
  - limited dynamic process creation/destruction
  - procedures are inlined (no recursion)
  - only static data types are fully translated
The IF level components

- Translation from IF to other tools and formalisms:
  - Promela-Spin (University of Eindhoven)
  - Lash (University of Liege)
  - Agatha (CEA-Leti)
  - ...

- Static analysis and abstractions:
  - live variable computation
  - slicing
Static analysis (1): live variables computation

- strongly preserve the initial behaviour
- drastically reduce the size of the model (more than 2 orders of magnitude)
- easy to compute...

q?m(x, y)
reset(y)

y := z+2
reset(y)

/* y not used here … */

y := 3*x
reset(y)

y := z+2
reset(y)
Static Analysis (2): slicing

- Extract the relevant part of a specification with respect to a slicing criterion:
  - Property under check (test purpose, observer, scenario, ...)
  - Observable events, variables, ...
  - Slicing criterion

- Validation can be performed on the simplified specification
Slicing (example)

**Slicing criteria:**
- **observable events:** in2, out3
  - **var:** u, x, w, y, z
- **environment:** in2, in3, in4
  - **var:** u, x, y, z
  - **var:** x, z
Slicing (example)

Slicing criteria:

- Observables events: in2, out3
  - var: u, x, z

- Environment: in2, in3, in4
  - var: x, z

- Weak bisimulation reduction

var: x, z
The LTS level components

- **Simulator construction:**
  - implements discrete/dense time
  - supports on-the-fly and partial order reductions techniques

- **Model-checking:**
  - temporal-logic properties (Evaluator, Kronos)
  - behavioural specifications (Aldébaran)
  - both including diagnostic capabilities

- **Test case generation (TGV)**
A validation "methodology"

- Specification
  - live analysis, dead code elimination...
- Environment
  - guided simulation, deadlock detection...
- Requirements
  - advanced static analysis
    - slicing, abstraction...
      - partial order, model-checking, test generation...
    - model generation + validation
  - model exploration
    - basic static analysis
      - static analysis
      - slicing,
        abstraction...
Outline

- Motivations
- IF: the language
- The IF validation environment
- Some case studies
  - a distributed leader election algorithm
  - the SSCOP protocol
  - the Ariane-5 flight controller [slides not available here]
- Conclusions and perspectives
A distributed leader election algorithm

mutual exclusion access to a shared resource on an unreliable circular network:

- approx. 200 lines of SDL
- 4 processes:
  - 3 states
  - 3 variables, 1 timer
Leader election algorithm: modelling problems

1. modelling unreliable channels: an IF buffer attribute

2. modelling time progress:

```
T := 10 sec

input token

[T := 0] output claim ...
```

(timer T)

(lazy) ... (eager)
## Results for live analysis

<table>
<thead>
<tr>
<th></th>
<th>Object GEODE</th>
<th>IFF</th>
<th>IFF + live analysis</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Reliable channels</strong></td>
<td>1731 st.</td>
<td>618 st.</td>
<td>292 st.</td>
</tr>
<tr>
<td><strong>Maximal urgency</strong></td>
<td>3822 tr.</td>
<td>1256 tr.</td>
<td>756 tr.</td>
</tr>
<tr>
<td><strong>1 sec.</strong></td>
<td></td>
<td>0.4 sec.</td>
<td>0.2 sec.</td>
</tr>
<tr>
<td><strong>Lossy channels</strong></td>
<td>3018 145 st.</td>
<td>537 891 st.</td>
<td>4943 st.</td>
</tr>
<tr>
<td><strong>Maximal urgency</strong></td>
<td>7119 043 tr.</td>
<td>2298 348 tr.</td>
<td>19664 tr.</td>
</tr>
<tr>
<td><strong>18 mn 7 sec.</strong></td>
<td></td>
<td>9 mn 7 sec.</td>
<td>4.8 sec.</td>
</tr>
<tr>
<td><strong>Lossy channels</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Weak urgency</strong></td>
<td>not available</td>
<td>too large!</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>54591 st.</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>250 016 tr.</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>54.9 sec.</td>
<td></td>
</tr>
</tbody>
</table>
Leader election algorithm: verification

Property:

Accesses to the resource are performed in mutual exclusion, i.e., there is always a close action between two open actions.

Graph obtained by weak bisimulation minimisation.
Outline

• Motivations
• IF: the language
• The IF validation environment
• Some case studies
  - a distributed leader election algorithm
  - the SSCOP protocol
  - the Ariane-5 flight controller [slides not available here]
• Conclusions and perspectives
SSCOP Protocol

- case-study provided by France Telecom R&D within the FORMA Research Action

- part of ATM Adaptation Layer (AAL), normalised by ITU Q2110

- aims were both formal verification and test generation
SSCOP Protocol: overview

- several services have to be provided
  - connection control (establishment, flow-control, maintenance)
  - data transfer
  - error detection and recovery

- described as a single SDL process
  - 10 states, 134 variables, 4 timers
  - 2000 lines of code
SSCOP Protocol: Verification Steps

- **direct generation using ObjectGEODE fails**
  - 2kB / state vector: only 50,000 states could be generated

- **static analysis simplifications**
  - “aggressive” abstraction by variable elimination
  - variable resetting using live information
  - slicing wrt specific properties

- **model generation using ObjectGEODE**
  - 0.2 kB / state vector: 1,000,000 could be generated
  - several functioning phases were completely verified
Example of property: connection establishment

Each connection request input to a SSCOP entity is followed by a connection response output by the same entity.

Verification: about 15,000 states generated (2 mn)
Conclusion

IF Validation environment

• open validation environment, connecting design and verification tools (ObjectGeode, Spin, CADP, TGV, ...)

• provides automatic program level optimisations: static analysis, slicing

• able to deal with realistic size case studies ...
Perspectives

• more static analysis (invariant generation, ...)
• general abstractions (InVeSt)
• connection with other tools

• definition of dynamicIF, for the description of dynamic and parameterised systems and for connection with UML and Java
The END

http://www-verimag.imag.fr/DISTSYS