Interfacing with Proof Assistants for Domain Specific Programming Using EventML

Vincent Rahli

PRL team - Cornell University

July 13, 2012
Credits

- Mark Bickford
- Robert Constable
- David Guaspari
- Richard Eaton
- Vincent Rahli
- Robbert Van Renesse
- Nicolas Schiper
- Jason Wu
Problem

Problem: unverified protocols are wrong.

Goal: automatic synthesis of verified diversifiable distributed systems.

Our solution: building tools that cooperate with a Logical Programming Environment (e.g., a constructive theorem prover).
EventML: specification and programming language

- A ML-like functional programming language.
- Features logical constructs (Logic of Events combinators).
- To specify/code distributed protocols.
- EventML translates specifications into event classes.

**Logical aspect**
- EventML synthesizes distributed programs (in the model underlying the Logic of Events) from specifications.

**Computational aspect**
Cooperation with a Logical Programming Environment

EventML

- Library
  - Type checker
  - Typable specification
  - Code synthesizer
  - Logical translator
  - Event Logic predicates
  - Message system
  - Simulator
    - Evaluator 1
    - Evaluator 2
    - Evaluator 3
    - ...

Nuprl

- Library
  - Extraction
  - Extracted code
  - Programmability lemma
  - Event Logic predicates
  - ILF
  - Distributed system properties

Reiners
Accomplishments

We have specified many distributed protocols. We have proved the correctness of the following protocols:

- Leader election in a ring.
- Two-thirds consensus protocol.
- Paxos (in progress).

The methodology works!

Nicolas Schiper (Cornell postdoc) has implemented a replicated database (ShadowDB) on top of our synthesized two-thirds consensus protocol.

It is used!
An example: Maximum using Memory

We have defined state machines in the Logic of Events.
E.g., Memory1.

We have automated some reasoning on state machines.
Maximum

```plaintext
input int : Int

class Maximum =
    Memory1 (\loc.{0})
    (\loc.x.s. imax x s)
    int'd base
;;

Intuition: at any event, computes the maximum of the integers received in the past.
```
class Maximum =
Memory1 (\loc.\{0\})
(\loc.\x.\n.imax x n)
int'base

{0}
{3}
{3}
{5}
Maximum

class Obs1 =
let F loc x n = if n > 3 & n < 20
    then {imax x n}
    else {}
in F o (int'base,Maximum)

\{\} \{\} \{\} \{\}(3,5)\{12\}(5,12)
input start : Unit
internal inc : Unit

class input = start'base
|| inc'base

class IncState =
Memory1 \loc.{0}  
(Voc.\().\n.n+1)  
Input

Class Increment =
let F loc () n =
{ int'send A n
; inc'send loc ()
}  
in F o (Input,IncState)

class Obs1 =
let F loc x n = if n > 3 & n < 20
then {imax x n}
else {}
in F o (int'base,Maximum)

class Obs2 =
let F loc x n = if n >= 3 & x >= 3
then {(n,x)}
else {}
in F o (int'base,Maximum)

{}    {}

{}    {}

{}    {(3,5)}

{}    {(5,12)}
Maximum
One can specify state machine invariants in EventML:

\texttt{invariant pos\_max on n in Maximum}
\hspace{1em}== n >= 0;;

\texttt{progress inc\_max on n1 then n2 in Maximum}
\hspace{1em}with \hspace{1em}n in \hspace{1em}int'base \hspace{1em}and \hspace{1em}s \Rightarrow n > s
\hspace{1em}== n2 > n1;;

\texttt{memory mem\_max on n1 then n2 in Maximum}
\hspace{1em}with \hspace{1em}n in \hspace{1em}int'base
\hspace{1em}== n2 >= n \hspace{1em}\land \hspace{1em}n2 >= n1;;

\textbf{Nuprl automatically proves these invariants.}
What’s next?

- Automation.
- Correct-by-construction optimizations.
- More expressive types: refinement types, dependent types...