

Mechanized Formalization of a Transformation from an Extensible Spi Calculus to Java

Master Thesis Defense Talk

Alex Busenius

Supervisor: Prof. Dr. Michael Backes

Advisor: Cătălin Hrițcu, M.Sc.

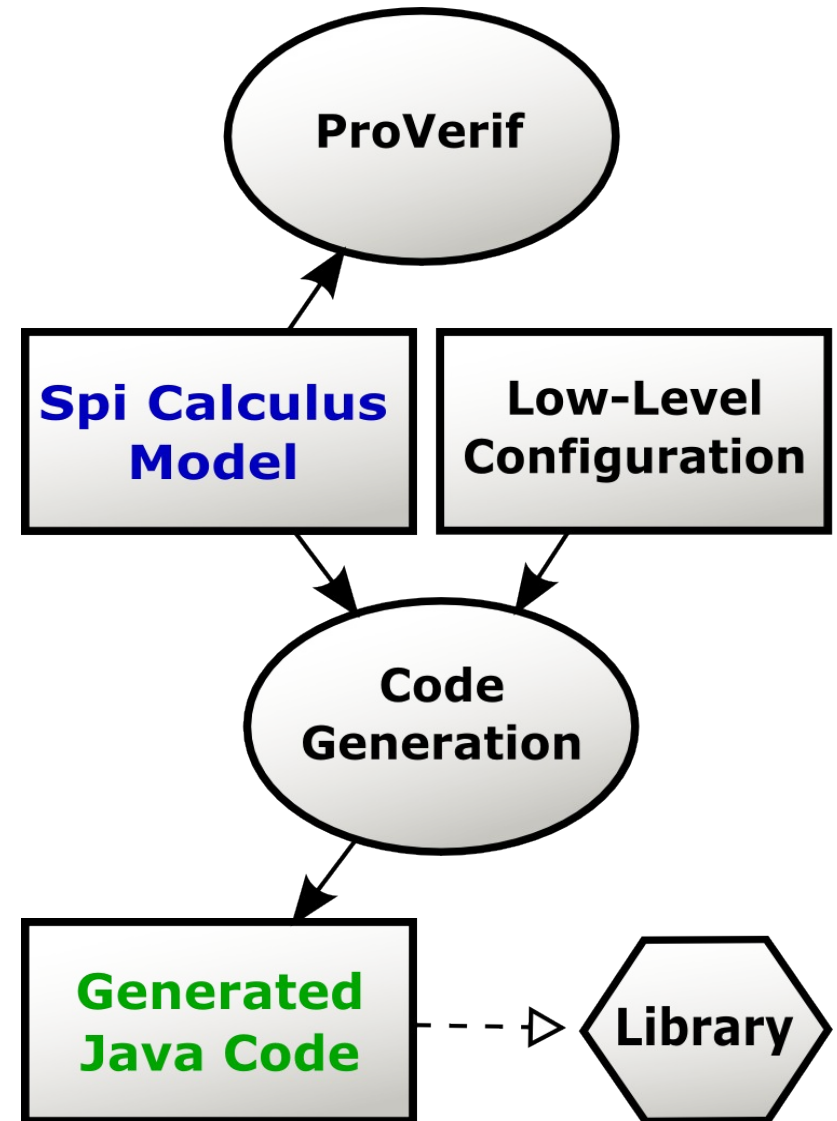
2nd Reviewer: Matteo Maffei, Ph.D

Outline

- Motivation
 - ▶ Expi2Java
- Formalization
 - ▶ Expi calculus
 - ▶ Variant Parametric Jinja
 - ▶ Transformation
- Future Work
- Summary

Expi2Java

- A code generator for security protocols
- Highly customizable and extensible
 - ▶ User-defined types, cryptographic primitives and low-level configuration
 - ▶ Code generation using code templates
 - ▶ User-provided implementation classes



Expi2Java

- Input language: A variant of Spi calculus (Expi)
- Simple type system (with type inference)
 - ▶ Prevents incorrect use of cryptographic primitives and configurations
- Produces interoperable Java code
 - ▶ Common cryptographic primitives and data types supported out of the box
 - ▶ Full control over the low-level data format
- Mature project
 - ▶ 6 major releases since December 2008

Case Study: TLS Protocol

- Fully-functional TLS v1.0 client and server
 - ▶ Started in Bachelor's thesis (client only)
 - ▶ With dynamic encryption scheme selection
- Verified the TLS handshake with ProVerif
 - ▶ Secrecy of nonces and the private key

```
(** Protocol start: Receive ClientHello **)
let server =
  new s_time      : $Timestamp;
  new session_id : $TLSSession;
  new server_rnd  : $TLSSRandom;
  let server_nonce = (s_time, server_rnd) in
    (* read certificate and secret key *)
    in(chFileSKey, server_rsa_key);
    in(chFileCert, certificate);
    event BeginServer();
  ...
```

```
config TLS_AES extends AES(
  keylength   = "256",
  provider    = "SunJCE",
  type SymEnc(
    class     = "TLSSymEnc",
    mode      = "CBC",
    padding   = "TLSSv1Padding"),
  type SymKey(class = "SymKey"),
  type Int(
    class    = "BigNonce",
    size     = "16" (* IV *))).
```

Formalization

- Source language
 - ▶ Expi Calculus
- Target language
 - ▶ Variant Parametric Jinja (a subset of Java)
- Transformation
 - ▶ In 2 steps
- Mechanized in Coq proof assistant
 - ▶ Formalization: \approx 10700 lines of Coq code
 - ▶ Proofs: \approx 6500 lines of Coq code

Expi Calculus

- Locally nameless representation of binders
 - ▶ Using Ott and LNgen
- Type system with parametric polymorphism
- Customizability

```
proc, P, Q, R :: 'proc_' ::=
| out( y , t ); P      ::      :: out
| in( y , x ); P      ::      :: in          (+ bind x in P +)
| !in( y , x ); P     ::      :: bangin      (+ bind x in P +)
| let x = g in P else Q ::      :: let          (+ bind x in P +)
| new a : T ; P       ::      :: new          (+ bind a in P +)
| P '|' Q            ::      :: fork
| 0                  ::      :: null
| ( P )              :: S     :: parenproc  {{ coq ([[P]]) }}
| P [ x := t ]       :: M     :: substproc  {{ coq (subst_t [[t]] [[x]] [[P]]) }}
```

Expi Calculus

- Customizability is an important feature
 - ▶ User-defined types, constructors and destructors
 - ▶ Low-level configuration
- Formalized using a “configuration”
 - ▶ Sets of type, constructor and destructor identifiers
 - ▶ Functions defining the behavior of types and terms
 - ▶ Destructor reduction relation
 - ▶ Consistency assumptions
- Proved consistency of a “default” configuration
 - ▶ Type preservation

Variant Parametric Jinja

- Full-fledged Java is too complex
 - ▶ Simplified the transformation
- Still need a very expressive subset
 - ▶ User-defined classes
 - ▶ Field update
 - ▶ Concurrency
 - Shared memory
 - Synchronization
 - ▶ Parametric polymorphism and subtyping
 - ▶ Exception handling

Variant Parametric Jinja

- No existing subset supported all needed features
- Jinja with Threads [Lochbihler, 2008]
 - ▶ Based on Jinja [Nipkow et al, 2006]
 - ▶ In Isabelle/HOL
- Variant Parametric FJ [Igarashi et al, 2006]
 - ▶ Based on Featherweight Java [Igarashi et al, 2001]
 - ▶ The type system is closely related to Java wildcards
 - ▶ Not mechanized
- Combined and extended both formalizations
 - ▶ Variant Parametric Jinja

Variant Parametric Jinja

- Translated the formalization of Jinja to Coq
 - ▶ Used functional maps instead of lists
 - ▶ Tried to avoid using classical lemmas
- Added new type (String)
 - ▶ For configuration names
- Removed support for arrays
- Introduced variant parametric types

VPJ: Example

```
(* public void send(T msg) *)  
Definition esend : expr :=  
  (* synchronized (this) ... *)  
  jsync (jthis) (  
    (* this.msg = msg; *)  
    FAss jthis "msg" "AbstractChannel" (* := *) (Var "msg") ;  
    (* semaRecv.release(); *)  
    Call (FAcc jthis "semaRecv" "AbstractChannel")  
      nil "release" nil ;  
    (* semaReady.acquire(); *)  
    Call (FAcc jthis "semaReady" "AbstractChannel")  
      nil "acquire" nil ;  
    (* this.msg = null; *)  
    FAss jthis "msg" "AbstractChannel" (* := *) jnull  
  ).
```

```
Definition msend :=  
  ("send",  
    (nil, ("msg", TVar "T") :: nil, Void, esend)).
```

Transformation

- Need a transformation that is easy to work with
- Known challenges:
 - ▶ Structural equivalence
 - $(P \mid Q) \mid R \equiv P \mid (Q \mid R)$
 - $P \equiv P \mid 0$
 - ▶ Restrictions
 - $\text{new } a; P$
 - ▶ Scope extrusion/intrusion
 - $a \notin \text{fn } Q \Rightarrow (\text{new } a; P) \mid Q \equiv \text{new } a; (P \mid Q)$
- Idea: Use an alternative semantics for Expi

Transformation

- Step 1: Expi \rightarrow Global Expi
 - ▶ Replace restrictions with unique name generation
 - $\text{step}_1(\text{new } a; P) = \text{gen } x \text{ in } \text{step}_1(P\{x/a\})$
 - ▶ This makes the calculus closer to implementation
 - ▶ Simpler semantics
 - No scope extrusion
 - ▶ Fully-abstract translation
 - Old Names for Nu [Lucian Wischik, 2004]
 - $P \approx_{\text{spi}} Q \Leftrightarrow \text{step}_1(P) \approx_{\text{spi}'} \text{step}_1(Q)$

Transformation

- Step 2: Global Expi \rightarrow VPJ
 - ▶ Implements the behavior of Expi terms and processes in VPJ
 - Channels (communication between threads)
 - Pattern matching
 - Structural equality
 - ▶ Symbolic library
 - Cryptographic operations as abstract terms
 - Channel implementation

Transformation

Global Expi	VPJ
Configuration	
Type identifiers	Class declarations
Configuration names	String constants
Constructors	Special methods in the class Fun
Destructors	Special methods in the class Fun
Terms	Expressions (variables, method calls)
Processes	Expressions (variable declarations, control flow)
Parallel composition	Threads
Input / Output	Communication between threads
Free names	Variables in main expression

Transformation: Example

```
(* out( c , t ); G
   Evaluates [[t]] and sends the result over channel [[c]] *)
Definition out_expr (c : vname) (et : expr) (eg : expr) : expr :=
  (* c.send(et) ; *)
  Call (Var c) nil "send" (et :: nil) ;
  (* [[G]] *)
  eg.

Inductive spitoj_gproc : ... :=
...
| spitoj_gproc_out : forall c cns L vns g ch t G cn et Tt Tt' J eg,
  lc_term ch ->
  spitoj_term c g ch = Some (Var cn, TAbstractChannel (JVInv, Tt'))->
  lc_term t ->
  spitoj_term c g t = Some (et, Tt) ->
  spitoj_gproc c cns L vns g G J eg ->
  spitoj_gproc c cns L vns g (out( ch , t ); G) J (out_expr cn et eg)
...
```

Transformation: Proofs

- Transformation preserves well-typedness
 - ▶ Symbolic library is well-typed
 - ▶ Produced VPJ code is well-typed
- Important consistency check
 - ▶ Can be used to prove correctness in the future
- Quite long and tedious proofs (3.7k LOC)
- Not proved in all details, omitted:
 - ▶ Complicated list rewriting, substitution rewriting
 - ▶ Invariants for transformation of terms and types

Future Work

- Transformation preserves trace properties
 - ▶ Proof technique: Weak labeled (bi)simulation
- Transformation preserves security (robust safety)
 - ▶ Or a weaker notion: Robust safety wrt. Expi contexts
- Functional correctness of the transformation
 - ▶ Also using trace properties

Summary

- Formalized
 - ▶ Expi Calculus 1.0k + 2.8k LOC
 - ▶ Variant Parametric Jinja 8.2k LOC
 - ▶ Transformation
 - Expi → Global Expi → VPJ 1.0k LOC
 - Symbolic library 0.5k LOC
- Proved Transformation Well-Typed 3.7k LOC
- Improved Implementation
 - ▶ Type inference, usability improvements
 - ▶ TLS server, dynamic encryption scheme selection

Questions?

- Formalized
 - ▶ Expi Calculus 1.0k + 2.8k LOC
 - ▶ Variant Parametric Jinja 8.2k LOC
 - ▶ Transformation
 - Expi → Global Expi → VPJ 1.0k LOC
 - Symbolic library 0.5k LOC
- Proved Transformation Well-Typed 3.7k LOC
- Improved Implementation
 - ▶ Type inference, usability improvements
 - ▶ TLS server, dynamic encryption scheme selection