Verification using the

Lennart Beringer

PRINCETON UNIVERSITY

CPS Workshop @ CMU, May 12, 2017
VST verification flow

Example.c

```c
void foo(int x) {...}
```

CompCert Frontend

Example.v

( Clight AST )

Definition `Foo body := ...`

CompCert Backend

Assembly Code
VST verification flow

Domain-specific definitions

Example.c

\texttt{void foo(int x) \{ \ldots \}}

\texttt{foo\_spec := \{ pre \} \{ post \}}

Example Spec.v

CompCert Backend

\texttt{( Clight AST )}

\texttt{Domain\_defs.v}

\texttt{Domain\_defs.v}

CompCert Frontend

Example.v

\texttt{Definition Foobody := \ldots}

Assembly Code
VST verification flow

Domain-specific definitions

Example.c
void foo(int x) {...}

Imports

Example_spec.v
Definition
foo_spec := "{ pre } { post }"

CompCert
Frontend
( Clight AST )
Definition Foobody := ...

Imports

Assembly Code

Verifiable C program logic

Proof automation “Floyd”

CompCert
Backend

Domain_defs.v

Imports

Imports
VST verification flow

**Domain-specific definitions**

- **Example.c**
  - `void foo(int x) {...}

**CompCert Frontend**

- **Example.v**
  - (Clight AST)
    - Definition `Foobody := ...

**CompCert Backend**

- **Assembly Code**

**Imports**

**Definition**

- `foo_spec := "{ pre } { post }"

**Example_spec.v**

**Imports**

**Example_proof.v**

**Imports**

**Lemma FooCorrect:**

- `semax foobody foo_spec.
- Proof. ..... Qed.

**Imports**

**Proof automation “Floyd”**
Intros.
assert PROP (isPtr k) as isPtrK by entail!.
forward call (buf, k, kl, key, kv, HMACAbs nil nil nil).
  { apply isPtrD in isPtrK. destruct isPtrK as [Ab [kofs HK]]. rewrite HK.
    unfold initPre. entail!.
  }
assert PROP (s256a.len (absctxt (hmacInit key)) = 512).
  { unfold hmacstate. Intros r. apply prop_right. apply H.
    rename H into absH_len512.
  }
forward call (hmacInit key, buf, msg, dl, data, kv).
  { rewrite absH_len512. assumption. }
(* Call to HMAC_Final*)
assert PROP (field compatible CompSpecs (Tstruct _hmac_ctx_st noattr) nil buf)
  { unfold hmacstate. Intros r; entail!.. }
rename H into FC_buf.
forward call (hmacUpdate data (hmacInit key), buf, md, shmd, kv).
  { remember (hmacFinal (hmacUpdate data (hmacInit key))) as RES.
    destruct RES as [h2 dig].
    simpl.
  }
forward call (h2, buf).
freeze [0; 1; 2; 3; 4] FR1.
forward.
(*assert PROP (field compatible (tarray tuchar (sizeof t_struct_hmac_ctx_st) stmt)))
  { unfold data block at 1. unfold ZLength. simpl. apply prop_right. assumption.
    rename H5 into FBUF.*}
specialize (hmac_sound key data). unfold hmac.
  { rewrite <- EqRES. simpl. intros.
    Exists buf dig. thaw FR1. entail!.. }
  { subst.
  split. unfold bitspec. simpl. rewrite Equivalence.
    f_equal. unfold HMAC_msg_absprot.HMAC_Abstract.Message2Blist.
    remember (mkCont data) as dd. destruct dd. destruct a; subst x.
    rewrite ByteBitRelations.bytes_bits_bytes_id.
    rewrite HMAC_equivalence.of_length_proof_irrel.
    rewrite ByteBitRelations.bytes_bits_bytes_id. reflexivity.
}

1 subgoal
Espec : OracleKind
k : val
kl : Z
key : list Z
msg : val
dl : Z
data : list Z
kv : val
shmd : share
md : writable_shmd shmd
KL : has_lengthH KL key
DL : has_lengthD 512 dl data
Delta specs := abbreviate : PTREE.T fumspec
POSTCONDITION := abbreviate : ret_assert
Pmd : isPtr md
isByteZ key : Forall isByteZ key
Pbuf : isPtr buf
isPtrK : isPtr k
Delta := abbreviate : tycontext
MORE COMMANDS := abbreviate : statement
absH_len512 : s256a_len (absctxt (hmacInit key)) = 512

semax Delta
(PROP ()
  LOCAL [var _c t_struct_hmac_ctx_st buf; temp_md md; temp_key k;
    temp_key_len (Vint (Int.repr kl)); temp_d msg;
    temp_n (Vint (Int.repr dl)); gvar sha. K256 kv]
  SEP (hmacstate_ (hmacInit key) buf; initPostKey k key; K_vector kv;
    data_block Tsh data msg; memory_block shmd 32 md))

(Ssequence
  (Scall None
    (Evar _HMAC_Update
      (Tfunction
        (Tcons (tptr (Tstruct _hmac_ctx_st noattr)))
    )
  )
)

Messages Errors Jobs
VST: key properties

• Program logic:
  – expressive concurrent separation logic
  – \{ \text{Prop } P \ \text{Local} \ Q \ \text{Sep} \ R \} \ c \ { \text{Prop } P' \ \text{Local} \ Q' \ \text{Sep} \ R' } \}
  – specifications: extractable / executable programs, or propositional
  – formal soundness proof in Coq w.r.t. Clight op.semantics

• Proof automation “Floyd”:
  – forward symbolic execution
  – domain specific tactics for entailments etc

• Guarantees:
  – safety: no undefined behavior, memory safety (buffer overrun,...), div-by-0,...
  – confinement: no read/writes outside specified memory region

• Major design goals: compositionality, support of “real C”
Compositional verification, end to end

- **horizontally**: compose verified code modules
- **vertically**: separate code verification from model-level reasoning
- Example: implementations of cryptographic primitives
Automation & Performance

- assertions in canonical form: PROP (P) LOCAL (Q) SEP (R)
- SL proof rules for C complex! Many entailments!
- full employment theorem for tactics programmers
- horizontal frame, not vertical: PROP (P) LOCAL (Q) SEP (R) FR (F)

![Graph showing performance metrics](image)

SLOW
- Time (Sec)
- Slope = 1
- Geomean = 0.77

FAST

N=27
VST 4 CPS (1): communication protocols for sensor networks

- CSL rules for lock-acquire / release: local thread gains / relinquishes resources
- also applicable to CAS, **atomic exchange**, ...
VST 4 CPS (2):
Linking code verification & model-level reasoning

Model – level reasoning:
crypto (probability & number theory)
abstract protocols (distr. systems)
physical world (ODE’s, …)
etc.

Code verification: \{ P \} C \{ Q \}
VST 4 CPS (2): Linking code verification & model-level reasoning

Model – level reasoning:
- crypto (probability & number theory)
- abstract protocols (distr. systems)
- physical world (ODE’s, …)
- etc.

Specification for domain-level reasoning

“Obvious” specification

Executable specification

Concrete functional specification for C

Code verification: \( \{ P \} \{ Q \} \)
Vision

- Interactive proof assistant as “integrated development environment” for high-assurance software
- Other components: foundationally verified
  - hypervisors/OS-kernels
  - processor implementations
  - network stacks and other libraries
- Exemplary interface specifications under development in NSF Expedition
(Reader allowed to miss values. Existence of N+2 Buffers ensures non-blocking)
(Reader allowed to miss values. Existence of N+2 Buffers ensures non-blocking)