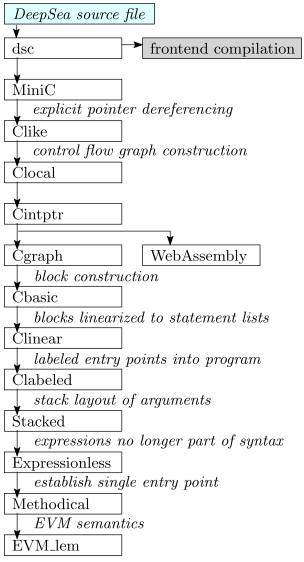
Modelling a Blockchain for Smart Contract Verification using DeepSEA

THE UNIVERSITY OF
WAIKATO
Te Whare Wānanga o Waikato

Daniel Britten (University of Waikato, New Zealand)
Steve Reeves (University of Waikato, New Zealand)

DeepSEA

DeepSEA


DeepSea source file
dsc
frontend compilation
MiniC
explicit pointer dereferencing
Clike
control flow graph construction
Clocal
Cintptr
Cgraph
WebAssembly
block construction
Cbasic
blocks linearized to statement lists
Clinear
labeled entry points into program
Clabeled
stack layout of arguments
Stacked
expressions no longer part of syntax
Expressionless
establish single entry point
Methodical
EVM semantics
EVM_lem

DeepSEA

DeepSEA

Example Context - Crowdfunding [4]:

DeepSEA

Example Context - Crowdfunding [4]:

1. “Safety property: donations record backed by ether”

DeepSEA

Example Context - Crowdfunding [4]:

1. “Safety property: donations record backed by ether”
2. “Donation record preserved over time”

DeepSEA

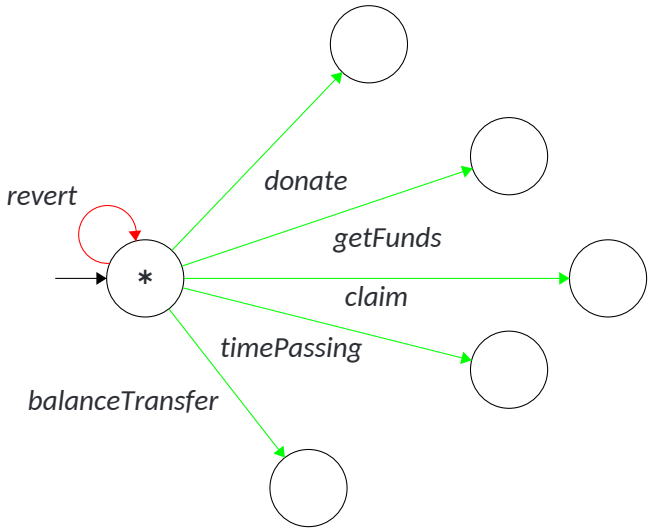
Example Context - Crowdfunding [4]:

1. “Safety property: donations record backed by ether”
2. “Donation record preserved over time”
3. “Backers can claim back donations after campaign fails”

Providing a blockchain model for DeepSEA

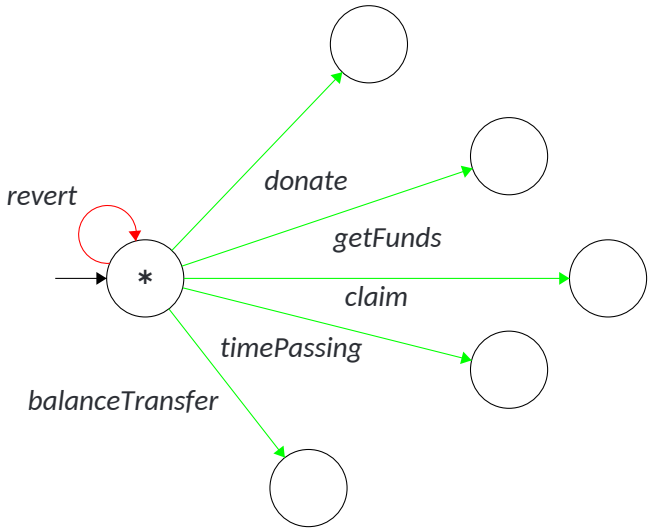
Successful-Calls Approach

Successful-Calls Approach



```
Inductive Action (before : BlockchainState) :=
| call_Crowdfunding_donate (context : CallContext)
  (callvalue_prf : noOverflowOrUnderflowInTransfer ...)
  r
  contract_state_after
  (case_donate_prf : runStateT (Crowdfunding_donate_opt
    (make_machine_env contract_address before context ...))
      (contract_state before)
        = Some (r, contract_state_after))
| call_Crowdfunding_getFunds ...
| call_Crowdfunding_claim ...
| externalBalanceTransfer ...
| timePassing ...
| revert.
```

Successful-Calls Approach



Successful-Calls Approach

```
Fixpoint step
  (before : BlockchainState) (action : Action before)
    : BlockchainState :=
match action with
| call_Crowdfunding_donate context
    callvalue_prf r d_after case_donate_prf =>
      next_blockchain_state before d_after
| call_Crowdfunding_claim ...
| call_Crowdfunding_getFunds ...
| timePassing ...
| externalBalanceTransfer ...
| revert => before
end.
```

Usage in a Crowdfunding Proof

```
let donate () =
  ...
  if (blk > _max_block) then
    fail
  else
      if (backers[msg_sender] = 0) then
          backers[msg_sender] := msg_value
      else
        fail
```

Usage in a Crowdfunding Proof

```
Theorem donation_preserved : forall (a : addr) (d : Z),
  (donation_recorded a d) `since` (donation_recorded a d) `as-long-as` (no_claims_from a).
Proof.
unfold since_as_long. intros. induction H; [assumption|]. assert(donation_recorded a d prevSt)
  by (apply IHReachableFromBy; intros; apply H1; apply in_cons; assumption).
clear H0 IHReachableFromBy. unfold donation_recorded in *. Hlinks.
assert (no_claims_from a prev) by (apply H1; destruct HL; subst; right; left; auto).
destruct prev; autounfold in *; simpl in *. clear H1 H HL. unfold no_claims_from in H0.
unfold donation_recorded in *. destruct Step_action0; simpl in *; rewrite <- HS in *; try assumption.
- Transparent Crowdfunding_donate_opt. unfold Crowdfunding_donate_opt in *.
  ds_inv; subst; simpl; inv_FT. destruct (a =? (caller context)) eqn:Case.
  + exfalso. apply Int256eq_true in Case. simpl in *. apply Z.eqb_eq in Heqb0. rewrite <- Case in *.
    rewrite Heqb0 in H2. lia.
  + simpl in *. apply Int256eq_false in Case. rewrite get_default_so; assumption.
- Transparent Crowdfunding_getFunds_opt. unfold Crowdfunding_getFunds_opt in *.
  ds_inv; subst; simpl; lia.
- contradiction.
Qed.
```

Usage in a Crowdfunding Proof

```
Proof.
- (* donate *)
  +
Heqb0
     : get msg_sender backers = 0

  +
Heqb0
     : get msg_sender backers = 0

- (* getFunds *)
- (* claim *)
Qed.
```

Usage in a Crowdfunding Proof

```
let donate () =
  ...
  if (blk > _max_block) then
    fail
  else
      if (backers[msg_sender] = 0) then
          backers[msg_sender] := msg_value
      else
        fail
```

Usage in a Crowdfunding Proof

```
Theorem donation_preserved : forall (a : addr) (d : Z),
  (donation_recorded a d) `since` (donation_recorded a d) `as-long-as` (no_claims_from a).
Proof.
unfold since_as_long. intros. induction H; [assumption|]. assert(donation_recorded a d prevSt)
  by (apply IHReachableFromBy; intros; apply H1; apply in_cons; assumption).
clear H0 IHReachableFromBy. unfold donation_recorded in *. Hlinks.
assert (no_claims_from a prev) by (apply H1; destruct HL; subst; right; left; auto).
destruct prev; autounfold in *; simpl in *. clear H1 H HL. unfold no_claims_from in H0.
unfold donation_recorded in *. destruct Step_action0; simpl in *; rewrite <- HS in *; try assumption.
- Transparent Crowdfunding_donate_opt. unfold Crowdfunding_donate_opt in *.
  ds_inv; subst; simpl; inv_FT. destruct (a =? (caller context)) eqn:Case.
  + exfalso. apply Int256eq_true in Case. simpl in *. apply Z.eqb_eq in Heqb0. rewrite <- Case in *.
    rewrite Heqb0 in H2. lia.
  + simpl in *. apply Int256eq_false in Case. rewrite get_default_so; assumption.
- Transparent Crowdfunding_getFunds_opt. unfold Crowdfunding_getFunds_opt in *.
  ds_inv; subst; simpl; lia.
- contradiction.
Qed.
```

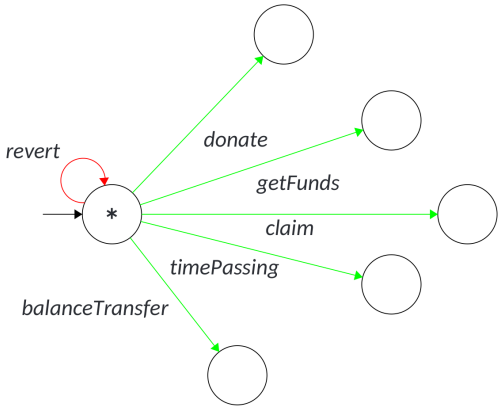
Usage in a Crowdfunding Proof

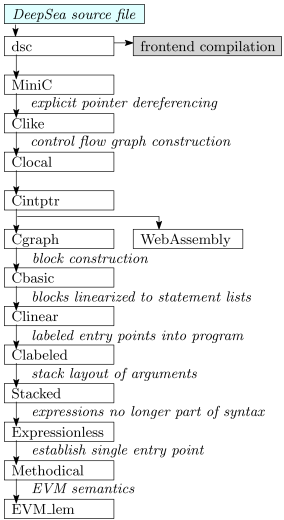
```
Transparent Crowdfunding_donate_opt. unfold Crowdfunding_donate_opt in *.
ds_inv; ...
```

Usage in a Crowdfunding Proof

```
match goal with
  | H : context[runStateT ?X ] |- _
    => destruct (runStateT X) eqn:SSSCase end; [|inversion H0; assumption].
destruct p.
Transparent Crowdfunding_donate_opt.
unfold Crowdfunding_donate_opt in SSSCase.
ds_inv; ...
```

Summary


revert
donate
getFunds
claim
timePassing
balanceTransfer
*


DeepSea source file
dsc
frontend compilation
MiniC
explicit pointer dereferencing
Clike
control flow graph construction
Clocal
Cintptr
Cgraph
WebAssembly
block construction
Cbasic
blocks linearized to statement lists
Clinear
labeled entry points into program
Clabeled
stack layout of arguments
Stacked
expressions no longer part of syntax
Expressionless
establish single entry point
Methodical
EVM semantics
EVM_lem

Thank you

Thank you to Vilhelm Sjöberg and Steve Reeves for their input and also to the University of Auckland and Jing Sun for kindly hosting me during this research.

Selected references:

[1] Daniel Britten.
Crowdfunding Proof GitHub Repository.
https://github.com/Coda-Coda/Crowdfunding/tree/FTSCS-2022. Accessed 1 December 2022.

[2] Shentu Chain, CertiK.
DeepSEA GitHub Repository.
https://github.com/ShentuChain/deepsea. Accessed 1 December 2022.

For a full list of references, see the paper [3].

Additional references I

[3] Daniel Britten, and Steve Reeves.
Modelling a Blockchain for Smart Contract Verification using DeepSEA.
Proceedings of the 8th ACM SIGPLAN International Workshop on Formal Techniques for Safety-Critical Systems (FTSCS '22), December 07, 2022, Auckland. Forthcoming.

[4] Ilya Sergey, Vaivaswatha Nagaraj, Jacob Johannsen, Amrit Kumar, Anton Trunov, and Ken Chan Guan Hao.
Safer smart contract programming with Scilla.
Proceedings of the ACM on Programming Languages, 3(OOPSLA):1–30, 2019.

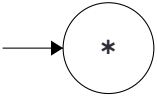
[5] Leroy, Xavier and Blazy, Sandrine and Kästner, Daniel and Schommer, Bernhard and Pister, Markus and Ferdinand, Christian
CompCert – A Formally Verified Optimizing Compiler
ERTS 2016: Embedded Real Time Software and Systems, 8th European Congress

[6] Vít Novotný
Fibeamer for the Faculty of Education at the Masaryk University in Brno.
Latex theme, used under CC BY 4.0.

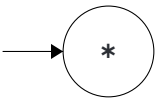
[7] Clément Pit-Claudel
Untangling mechanized proofs (Alectryon)
Proceedings of the 13th ACM SIGPLAN International Conference on Software Language Engineering, 2020.

Snapshot Approach

Snapshot Approach

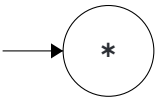


Snapshot Approach



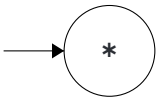

```
Context
  (snapshot_timestamp : int256)
  (snapshot_number : int256)
  (snapshot_blockhash : int256 -> int256)
  (snapshot_balances : addr -> wei).
```

Snapshot Approach




```
Record BlockchainState := mkBlockchainState {
  timestamp : int256;
  block_number : int256;
  balance : addr -> wei;
  blockhash : int256 -> int256;
  contract_state : global_abstract_data_type
}.
```

Snapshot Approach




```
Definition initial_state : BlockchainState :=
  mkBlockchainState
    snapshot_timestamp
    snapshot_number
    snapshot_balances
    snapshot_blockhash
    init_global_abstract_data
.
```

Usage in a Crowdfunding Proof

```
Proof.
```

```
snapshot_timestamp, snapshot_number: int256
snapshot_blockhash: int256 -> int256    snapshot_balances: addr -> wei
```

```
forall (a : addr) (d : Z),
(donation_recorded a d) `since`
donation_recorded a d `as-long-as`
(no_claims_from a)
```