tinyman 2 structural alloy specification

Intro

This is a specification of tinyman2 structural properties in alloy formalism. Alloy analyzer is a model-finder tool. It can enumerate models, that satisfy specified properties, to search for some examples or counter-examples for a set of assertions.

This specification is aimed at modeling and checking all possible transaction group configuraitons up to ceratin bounds. m At the moment only qualitative properties are modelled (sets and relations under constraints). Quantitative properties (amounts of reserves, values of fees, etc) are not modelled yet. To some extent the modelling of quantitative properties can be added, but Alloy formalism does not support well this type of properties, so it is of lower priority for now.

There are two main reasons to use model-finder:

1. Via enumerating configurations manually review some corner-cases, strange and wrong transaction groups, etc. This allows us to add missing checks, to see possible dangerous/malicios states, etc.
2. Using counter-example search we can prove or refute some our statements (expectations) about state-space of the protocol.

Results

- Several strange configurations were identified, like that `fee_manager`, `fee_setter` and `fee_collector` may be set to the application account or pool account. It is not clear what consequencies are in this case, so the necessity of the corresponding checks is questionable.
- Several assertions are proved to be redundant. For instance assertions for pool address and sender address may be safely removed from `verify_flash_loan` application call. It can help to save some bytecodes and reduce costs of the application deployment.
- Issue with a transaction index in pairs of flash_op/verify_flash_op was identified: from the tealish source code it is unclear whether index can be signed negative integer. Taking into consideration, that AVM using only unsigned 64 bits, it seems that only unsigned non-negative indices are possible, but in the source code reserved word `int` is used, which is rather confusing, because many developers with programming background expects that `int` is a type of signed integers. So, it is recommended to alter the tealish transpiler frontend (`int` -> `uint`), or to put some comments about signedness around corresponding lines of the source code. Moreover, additionally constrain may be added to restrict range of index value: `index >= 1`. This issue initially (until spec was additionally constrained) showed some configurations where flash_loan/verify_flash_loan can be swapped, leading to strange transaction groups state-space with potential of presence of malicious configurations.

Specification

Main ideas

Specification uses next main signatures:

- `Transaction`. It is used for identifying transactions in transaction group. It is ordered, the order is total. So each Transaction atom (atom is an item from `Transaction` set) is in 1-to-1 relation with transaction slot in transaction group.
- `Asset`. Totally ordered signature for assets. There is a distinguished sub-signature `Algo` that always points to first atom in the `Asset` set. So, like in the AVM, Asset0 is reserved

for Algo.

- `Account`. Is is a basic set of accounts in our specification. There are sub-signatures: `Application` and `Pool`. `Application` is one atom set from `Account`, that corresponds to tinyman2 application, `Pool` is a set of several acoounts from `Account` set, that are treated as pools.

Also there are some additional helper signatures and enumerations:

- Enum `LockState` (`Locked`, `Unlocked`) is used for a pool lock variable.
- Enum `TransactionType` (`Transfer`, `AppCall`) is self-describing.
- Signature `Operations` is a 1-to-1 correspondence to application call entries.
- Signature `Checks` is a helper signature that holds information about current assertions in a particular transaction and related statements in the source code.
- `OpCheck` is a macro for relation between `Operation` and `Assert`. So each operation (application call) has corresponding set of asserions.
- Signatures `MainParams`, `AmmParams`, `SwapParams`, etc - are helper signatures that correspond to blocks in the `amm_approval.tl` and contain some variables (important for the specification) declared in these blocks as relations.

In-depth description

Module declaration

Here we open standard alloy module `ordering` and introduce total order over `Transaction`, `Asset`, and `Checks` sets.

Also a helper module `asserts` (see `asserts.md` for more details) is opened.

```
module tinyman
open util/ordering[Transaction] as ot
open util/ordering[Asset] as oa
open util/ordering[Checks] as oc
open asserts
```

Assets

Assets are modelled via ordered set `Asset` with one selected element `Algo`. Algo is always constrained to be the first element.

```
sig Asset{}
one sig Algo in Asset {} {Algo = oa/first}
```

Accounts

Base signature for accounts:

```
sig Account{}
```

`Application` is some selected atom in `Account`:

```
one sig Application in Account {
```

with next relations (fields):

- `fee_setter` - points to one account that corresponds to a fee setter account:

```
  fee_setter: one Account,
```

- `fee_collector` - self-describing:

```
  fee_collector: one Account,
```

- `fee-manager` - self-describing:

```
  fee_manager: one Account
}
```

All the above are additionally constrained (the `amm_approval.tl` has no such checks):

```
fact {
  -- Pool account cannot be set as fee_setter, fee_coolector and
fee_manager
  no Application.(fee_setter + fee_collector + fee_manager) & Pool

  -- Application account cannot be used too
  Application not in Application.(fee_setter + fee_collector +
fee_manager)
}
```

Pools

Locks:

```
enum LockState {Locked, Unlocked}
```

Tokens state:

```
enum TokensIssuedState {Issued, NotIssued}
```

Pools are selected set from `Account` set:

```
sig Pool in Account {
```

Assets of a pool:

```
  asset1 : one Asset,
  asset2 : one Asset,
  pool_token: one Asset,
```

Pool state regarding issued tokens:

```
  tokens_issued: Transaction -> one TokensIssuedState,
```

Lock state for each transaction:

```
  lock: Transaction -> one LockState
} {
```

Constraints imposed on assets:

from `amm_approval.tl:63`:

```
  asset1 not in Algo
```

from `amm_approval.tl:63`, `amm_approval.tl:87` and `amm_approval.tl:108`:

```
  asset1 not in asset2
  pool_token not in asset1 + asset2 + Algo
}
```

Application and pool are different accounts, from `amm_approval.tl:93`:

```
fact {
  no Application & Pool
}
```

Transactions

There are only two types:

```
enum TransactionType {
  Transfer,
  AppCall
}
```

```
sig Transaction {
```

Main fields of a `Transaction`:

```
  type: one TransactionType,
  sender: one Account,
  receiver: lone Account,
  asset: lone Asset,
  op: lone Operation,
```

Names are self-describing, only one remark - `lone` means none or exactly one item.

Next fields are for better vizualization in model vizualizer, their values are taken from corresponding parameters:

```
  -- main
  main_user_address: set Account,

  -- amm
  amm_pool_address: set Account,
  amm_asset1: set Asset,
  amm_asset2: set Asset,
  amm_pool_token: set Asset,
  amm_pool_locked: set LockState,
  amm_tokens_issued: set TokensIssuedState,

  -- swap
  swap_input_txn_index: set Transaction,
  swap_mode: set SwapMode,

  -- flash loan
  fl_verify_txn_index: set Transaction,
  fl_borrowed_assets: set Asset,

  -- verify_flash_loan
  vfl_flash_loan_txn_index: set Transaction,
  vfl_asset1_txn_index: set Transaction,
  vfl_asset2_txn_index: set Transaction,

  -- flash_swap
  fs_verify_txn_index: set Transaction,
  fs_swapped_assets: set Asset,

  -- verify_flash_swap
  vfs_flash_swap_txn_index: set Transaction,

  -- add_liquidity
  al_asset1_txn_index: set Transaction,
  al_asset2_txn_index: set Transaction,
  al_mode: set AddLiquidityMode,

  -- add_initial_liquidity
  ail_asset1_txn_index: set Transaction,
  ail_asset2_txn_index: set Transaction,

  -- checks
  checks: one Checks
} {
```

These constraints are from `AVM` definition:

- Application calls contain no receiver, no assets, and only one operation:

```
  type in AppCall implies no receiver and one op and no asset
```

- Transfers should contain receiver, one asset, and no operation:

```
  type in Transfer implies no op and one receiver and one asset
```

Next constraints are just for taking corresponding values from parameters:

```
  -- main
  main_user_address = MainParams.user_address[this]

  -- amm
  amm_pool_address = AmmParams.pool_address[this]
  amm_asset1 = AmmParams.asset1[this]
  amm_asset2 = AmmParams.asset2[this]
  amm_pool_token = AmmParams.pool_token[this]
  amm_pool_locked = amm_pool_address.lock[this]
  amm_tokens_issued = AmmParams.tokens_issued[this]

  -- swap
  swap_input_txn_index = SwapParams.input_txn_index[this]
  swap_mode = SwapParams.mode[this]

  -- flash loan
  fl_verify_txn_index = FlashLoanParams.verify_txn_index[this]
  fl_borrowed_assets = FlashLoanParams.borrowed_assets[this]

  --verify flash loan
  vfl_flash_loan_txn_index =
VerifyFlashLoanParams.flash_loan_txn_index[this]
  vfl_asset1_txn_index = VerifyFlashLoanParams.asset1_txn_index[this]
  vfl_asset2_txn_index = VerifyFlashLoanParams.asset2_txn_index[this]

  -- flash_swap
  fs_verify_txn_index = FlashSwapParams.verify_txn_index[this]
  fs_swapped_assets = FlashSwapParams.swapped_assets[this]

  -- verify_flash_swap
  vfs_flash_swap_txn_index =
VerifyFlashSwapParams.flash_swap_txn_index[this]

  -- add_liquidity
  al_asset1_txn_index = AddLiquidityParams.asset1_txn_index[this]
  al_asset2_txn_index = AddLiquidityParams.asset2_txn_index[this]
  al_mode = AddLiquidityParams.mode[this]

  -- add_initial_liquidity
  ail_asset1_txn_index = AddInitialLiquidityParams.asset1_txn_index[this]
  ail_asset2_txn_index = AddInitialLiquidityParams.asset2_txn_index[this]

}
```

Operations

Enum `Operations` is in 1-to-1 relation with application calls.

```
enum Operation {
-- main
  OpSetFeeCollector,
  OpSetFeeSetter,
  OpSetFeeManager,
  OpClaimFees,
  OpClaimExtra,
  OpSetFee,
-- amm
  OpAddInitialLiquidity,
  OpAddLiquidity,
  OpRemoveLiquidity,
  OpSwap,
  OpFlashLoan,
  OpVerifyFlashLoan,
  OpFlashSwap,
  OpVerifyFlashSwap
}
```

Helper macro for getting all transactions for AMM operations:

```
fun all_transactions_for[o: set Operation] : set Transaction {
  {t: Transaction | t.type = AppCall and t.op in o}
}

let amm_transactions = all_transactions_for[
    OpAddInitialLiquidity
  + OpAddLiquidity
  + OpRemoveLiquidity
  + OpSwap
  + OpFlashLoan
  + OpVerifyFlashLoan
  + OpFlashSwap
  + OpVerifyFlashSwap
]
```

Helper signature Checks

Items from `Checks` set correspond 1-to-1 to items from `Transaction` set.

Signature `Checks` is used as helper for visualizing assertions (and code lines) for a particular operation in a transaction.

For details see `asserts.md`.

```
sig Checks {
  asserts: set Assert,
  assert_txt: set String
} {
  let t = { t: Transaction | t.checks = this } {
    asserts = OpCheck[t.op]
  }

  assert_txt = asserts.assert_line
}
```

This `Checks` signature has a nice view in `Table` mode in visualizer.

Here are some constraints to make `Checks` and `Transaction` items coincide.

```
fact {
  #Transaction = #Checks
  all disj t1, t2: Transaction | t1 in t2.prevs implies t1.checks in
t2.checks.prevs
}
```

Here is a macro with relation beetween operations and corresponding assertions:

```
let OpCheck = {
  -- main
    OpSetFeeCollector -> assert203

  + OpSetFeeSetter -> assert214

  + OpSetFeeManager -> assert225

  + OpClaimFees -> assert244

  + OpClaimExtra -> assert288

  + OpSetFee -> assert296
  + OpSetFee -> assert301
  + OpSetFee -> assert302
  + OpSetFee -> assert303
  + OpSetFee -> assert304

  -- amm
  + OpAddInitialLiquidity -> assert909
  + OpAddInitialLiquidity -> assert914
  + OpAddInitialLiquidity -> assert915
  + OpAddInitialLiquidity -> assert916
  + OpAddInitialLiquidity -> assert917
  + OpAddInitialLiquidity -> assert919
  + OpAddInitialLiquidity -> assert922
  + OpAddInitialLiquidity -> assert923
  + OpAddInitialLiquidity -> assert926
  + OpAddInitialLiquidity -> assert927
  + OpAddInitialLiquidity -> assert928
  + OpAddInitialLiquidity -> assert931
  + OpAddInitialLiquidity -> assert932
  + OpAddInitialLiquidity -> assert936

  + OpAddLiquidity -> assert727
  + OpAddLiquidity -> assert764
  + OpAddLiquidity -> assert765
  + OpAddLiquidity -> assert766
  + OpAddLiquidity -> assert767
  + OpAddLiquidity -> assert773
  + OpAddLiquidity -> assert774
  + OpAddLiquidity -> assert777
  + OpAddLiquidity -> assert778
  + OpAddLiquidity -> assert779
  + OpAddLiquidity -> assert782
  + OpAddLiquidity -> assert871
  + OpAddLiquidity -> assert874

  + OpRemoveLiquidity -> assert965
  + OpRemoveLiquidity -> assert966
  + OpRemoveLiquidity -> assert967
  + OpRemoveLiquidity -> assert968
  + OpRemoveLiquidity -> assert970
  + OpRemoveLiquidity -> assert984
  + OpRemoveLiquidity -> assert996
  + OpRemoveLiquidity -> assert997
  + OpRemoveLiquidity -> assert998
  + OpRemoveLiquidity -> assert999
  + OpRemoveLiquidity -> assert1005
  + OpRemoveLiquidity -> assert1016
  + OpRemoveLiquidity -> assert1034
  + OpRemoveLiquidity -> assert1075
  + OpRemoveLiquidity -> assert1084

  + OpSwap -> assert323
  + OpSwap -> assert351
  + OpSwap -> assert355
  + OpSwap -> assert361
  + OpSwap -> assert362
  + OpSwap -> assert389
  + OpSwap -> assert390
  + OpSwap -> assert391
  + OpSwap -> assert398
  + OpSwap -> assert399
  + OpSwap -> assert400

  + OpFlashLoan -> assert323
  + OpFlashLoan -> assert460
  + OpFlashLoan -> assert462
  + OpFlashLoan -> assert463
  + OpFlashLoan -> assert466
  + OpFlashLoan -> assert467
  + OpFlashLoan -> assert468
  + OpFlashLoan -> assert469
  + OpFlashLoan -> assert471
  + OpFlashLoan -> assert473
  + OpFlashLoan -> assert474
  + OpFlashLoan -> assert477
  + OpFlashLoan -> assert481

  + OpVerifyFlashLoan -> assert323
  + OpVerifyFlashLoan -> assert499
  + OpVerifyFlashLoan -> assert500
  + OpVerifyFlashLoan -> assert501
  + OpVerifyFlashLoan -> assert502
  + OpVerifyFlashLoan -> assert504
  + OpVerifyFlashLoan -> assert506
  + OpVerifyFlashLoan -> assert507
  + OpVerifyFlashLoan -> assert518
  + OpVerifyFlashLoan -> assert528
  + OpVerifyFlashLoan -> assert529
  + OpVerifyFlashLoan -> assert530
  + OpVerifyFlashLoan -> assert531
  + OpVerifyFlashLoan -> assert532
  + OpVerifyFlashLoan -> assert555
  + OpVerifyFlashLoan -> assert562
  + OpVerifyFlashLoan -> assert563
  + OpVerifyFlashLoan -> assert564
  + OpVerifyFlashLoan -> assert565
  + OpVerifyFlashLoan -> assert568
  + OpVerifyFlashLoan -> assert569
  + OpVerifyFlashLoan -> assert570
  + OpVerifyFlashLoan -> assert571
  + OpVerifyFlashLoan -> assert572

  + OpFlashSwap -> assert323
  + OpFlashSwap -> assert606
  + OpFlashSwap -> assert608
  + OpFlashSwap -> assert609
  + OpFlashSwap -> assert610
  + OpFlashSwap -> assert611
  + OpFlashSwap -> assert613
  + OpFlashSwap -> assert615
  + OpFlashSwap -> assert616
  + OpFlashSwap -> assert619
  + OpFlashSwap -> assert622
  + OpFlashSwap -> assert626

  + OpVerifyFlashSwap -> assert646
  + OpVerifyFlashSwap -> assert647
  + OpVerifyFlashSwap -> assert648
  + OpVerifyFlashSwap -> assert649
  + OpVerifyFlashSwap -> assert651
  + OpVerifyFlashSwap -> assert653
  + OpVerifyFlashSwap -> assert687
}
```

Issued pool tokens

Main idea is that state of pool tokens `Issued` / `NotIssued` can be changed only on `AddInitialLiquidity`, `RemoveLiquidity` operations.

```
fact {
  all pool: Pool
  | all t: Transaction - first
  | let ti_prev = pool.tokens_issued[t.prev],
        ti = pool.tokens_issued[t]
    {
      no ti_prev & ti implies {
        t.prev.type = AppCall
        t.prev.op in {OpAddInitialLiquidity + OpRemoveLiquidity}
      }
      t.prev.type = AppCall and t.prev.op = OpAddInitialLiquidity implies
      no ti_prev & ti
    }
}
```

Locks

Locks are treated in a specific way, unlike they appear in code.

As we have static transactions configuration, we cannot reason about locks in dynamic way, so we have to define locks at once for the whole transaction group.

The idea is simple: we take a pairs of corresponding `flash_swap/verify_flash_swap` operations for the same pool, and treat pool state for all transactions in range `(flash_swap, verify_flash_swap]` as locked.

All ranges in the form `(verify_flash_swap, flash_swap]`, `[first transaction (assuming it is not verify_flash_swap), flash_swap]` and `(verify_flash_swap, last transaction]` (assuming that there may be several flash swaps for the same pool in a transaction group) are treated as unlocked ones.

```
-- returns a set of transaction from closed interval [t1, t2]
fun tr_range[t1: Transaction, t2: Transaction]: set Transaction {
  t1 in t2.prevs implies {t:Transaction | t in (t1 + t2 + (t1.nexts &
t2.prevs))}
  else none
}

-- simple predicates for readability
pred should_lock_pool[t: Transaction, p: Pool] {
  t.op = OpFlashSwap and AmmParams.pool_address[t] = p
}

pred should_unlock_pool[t: Transaction, p: Pool] {
  t.op = OpVerifyFlashSwap and AmmParams.pool_address[t] = p
}
```

Constraints for locks are in the next fact:

```
fact {
  all pool: Pool
  | all disj t1, t2:Transaction
    | let range = tr_range[t1,t2] {
      {
        some range
        t1.should_lock_pool[pool]
        t2.should_unlock_pool[pool]
        all t: range - t2 | not t.should_unlock_pool[pool]
      } implies {
        all t: range - t1 | pool.lock[t] = Locked
      }
      {
        some range
        t1.should_unlock_pool[pool] or t1 = ot/first
        t2.should_lock_pool[pool] or t2 = ot/last
        all t: range - t2 | not t.should_lock_pool[pool]
      } implies {
        all t: range - t1 | pool.lock[t] = Unlocked
        t1 = ot/first implies pool.lock[t1] = Unlocked
      }
    }
}
```

We assume that there is no possibility to make a `flash_swap` after a `flash_swap`, because of assertion at `amm_approval.tl:323`.

block *main* at amm_apporval.tl:187

For each operation block in `amm_approval.tl` there is a related signature, that have fields for particular parameters in the block.

Parameters are encoded in the form of relation between a transation and parameter values.

```
-- corresponds to block main at amm_approval.tl:187
one sig MainParams {
  user_address : Transaction -> lone Account
} {
  all t: Transaction {
    t.type = AppCall implies {
      -- user_address is defined for all application calls
      t.user_address = t.sender
    }
    t.type = Transfer implies {
      -- if a transaction is not an application call then user_addres is
not set
      no t.user_address
    }
  }
}
```

block *amm* at amm_approval.tl:312

```
-- corresponds to block amm at amm_approval.tl:312
one sig AmmParams {
  pool_address: Transaction -> lone Account,
  asset1: Transaction -> lone Asset,
  asset2: Transaction -> lone Asset,
  pool_token: Transaction -> lone Asset,
  tokens_issued: Transaction -> lone TokensIssuedState
} {
```

A note about assets: in this specification there is no distinction between algos and other assets, because, in general, these details are irrelevant for the current level of structural abstraction.

```
  all t: amm_transactions {
    let pool = t.pool_address {
      one pool
      pool in Pool
      t.asset1 = pool.asset1
      t.asset2 = pool.asset2
      t.pool_token = pool.pool_token
      t.tokens_issued = pool.tokens_issued[t]
    }
  }

  all t: Transaction - amm_transactions {
    no t.pool_address
    no t.asset1
    no t.asset2
    no t.pool_token
    no t.tokens_issued
  }

  -- constraints derived directly from code

  -- assert at amm_approval.tl:323
  all t: amm_transactions
  | t.op = OpVerifyFlashSwap or t.pool_address.lock[t] = Unlocked
}
```

Proof that there are no VerifyFlashSwap operations over unlocked pool.

```
verify_flash_swap_always_under_lock:
check {
  all vfs: all_transactions_for[OpVerifyFlashSwap]
  | AmmParams.pool_address[vfs].lock[vfs] = Locked
} for 16 but 6 int

--   No counterexample found. Assertion may be valid. 481047ms.
```

Proof that there are no operations for the locked pool, except `VerifyFlashSwap`.

```
only_verify_flash_swap_is_possible_for_locked_pool:
check {
  all t: amm_transactions
  | AmmParams.pool_address[t].lock[t] = Locked implies t.op =
OpVerifyFlashSwap
} for 16 but 6 int

-- No counterexample found. Assertion may be valid. 177ms.
```

block *swap* at amm_approval.tl:336

```
enum SwapMode {fixedInput, fixedOutput}

-- corresponds to block swap at amm_approval.tl:336
one sig SwapParams {
  input_txn_index: Transaction -> lone Transaction,
  mode: Transaction -> lone SwapMode
} {
  all t: all_transactions_for[OpSwap] {
    t.input_txn_index = t.prev
    one t.mode
    t.mode in fixedInput + fixedOutput
  }
  all t: Transaction - all_transactions_for[OpSwap] {
    no t.input_txn_index
    no t.mode
  }

  -- constraints derived directly from code
  all t: all_transactions_for[OpSwap]
  | let transfer = t.input_txn_index {
      transfer.type = Transfer
      -- assert at amm_approval.tl:351/355
      transfer.receiver = AmmParams.pool_address[t]
      -- assert at amm_approval.tl:361
      transfer.sender = MainParams.user_address[t]
      -- checks at amm_approval.tl: 366/373
      transfer.asset in AmmParams.(asset1 + asset2)[t]
    }
}
```

block *flash_loan* at amm_approval.tl:447

```
-- corresponds to block flash_loan at amm_approval.tl:447
one sig FlashLoanParams {
  verify_txn_index: Transaction -> lone Transaction,
  borrowed_assets: Transaction -> set Asset
} {
  all t: all_transactions_for[OpFlashLoan] {
    some t.borrowed_assets
    t.borrowed_assets in AmmParams.(asset1 + asset2)[t]
    one t.verify_txn_index
    t.verify_txn_index in t.nexts
  }
  all t: Transaction - all_transactions_for[OpFlashLoan] {
    no t.verify_txn_index
    no t.borrowed_assets
  }

  -- constraints derived directly from code

  all t: all_transactions_for[OpFlashLoan]
  | let verif_tr = t.verify_txn_index {

      -- assertion at amm_approval.tl:466-469
      verif_tr.type = AppCall
      verif_tr.op = OpVerifyFlashLoan

      -- assertion at amm_approval.tl:471
      VerifyFlashLoanParams.flash_loan_txn_index[verif_tr] = t

      -- assertion at amm_approval.tl:473-474
      AmmParams.pool_address[verif_tr] = AmmParams.pool_address[t]
      verif_tr.sender = MainParams.user_address[t]
    }
}
```

block *verify_flash_loan* at amm_approval.tl:487

According to the modeling, some checks are redundant, this fact is noted in comments.

```
-- corresponds to block verify_flash_loan at amm_approval.tl:487
one sig VerifyFlashLoanParams {
  flash_loan_txn_index: Transaction -> lone Transaction,
  asset1_txn_index: Transaction -> lone Transaction,
  asset2_txn_index: Transaction -> lone Transaction
} {
  all t: all_transactions_for[OpVerifyFlashLoan] {
    some t.(asset1_txn_index + asset2_txn_index)
    some t.asset2_txn_index implies {
      t.asset1_txn_index = t.asset2_txn_index.prev
      t.asset2_txn_index = t.prev
    } else {
      t.asset1_txn_index = t.prev
    }
    t.(asset1_txn_index + asset2_txn_index).asset =
FlashLoanParams.borrowed_assets[t.flash_loan_txn_index]
    one t.flash_loan_txn_index
  }
  all t: Transaction - all_transactions_for[OpVerifyFlashLoan] {
    no t.flash_loan_txn_index
    no t.asset1_txn_index
    no t.asset2_txn_index
  }

  -- constraints derived directly from code

  all t: all_transactions_for[OpVerifyFlashLoan]
  | let flash_loan = t.flash_loan_txn_index,
        pool = AmmParams.pool_address[t],
        user_address = MainParams.user_address[t],
        asset1 = AmmParams.asset1[t],
        asset2 = AmmParams.asset2[t]
    {

      -- assert at amm_approval.tl:499-502
      flash_loan.type = AppCall
      flash_loan.op = OpFlashLoan

      -- assert at amm_approval.tl:504
      FlashLoanParams.verify_txn_index[flash_loan] = t

      -- assert at amm_approval.tl:506-507 - redundant
      AmmParams.pool_address[flash_loan] = pool
      flash_loan.sender = user_address

      -- check assets

      let t1 = t.asset1_txn_index,
          t2 = t.asset2_txn_index
      {
        -- amm_approval.tl:528-532
        some t1 implies {
          t1.type = Transfer
          t1.asset = asset1
          t1.receiver = pool
          t1.sender = user_address
        }

        -- amm_approval.tl:562-572
        some t2 implies {
          t2.type = Transfer
          t2.asset = asset2
          t2.receiver = pool
          t2.sender = user_address
        }
      }
    }
}
```

checks to prove redundancy of assertions in *verify_flash_loan*

```
verify_flash_loan_redundant_assertions:
check {
  -- 1-to-1 correspondence
  #all_transactions_for[OpFlashLoan] =
#all_transactions_for[OpVerifyFlashLoan]

  all fl: all_transactions_for[OpFlashLoan]
  | one FlashLoanParams.verify_txn_index[fl]

  all vfl: all_transactions_for[OpVerifyFlashLoan]
  | one VerifyFlashLoanParams.flash_loan_txn_index[vfl]

  all fl: all_transactions_for[OpFlashLoan]
  | let vfl = FlashLoanParams.verify_txn_index[fl]
  | VerifyFlashLoanParams.flash_loan_txn_index[vfl] = fl

  -- flash_loan/verify_flash_loan refer to the same pool
  all fl: all_transactions_for[OpFlashLoan]
  | let vfl = FlashLoanParams.verify_txn_index[fl]
  | AmmParams.pool_address[fl] = AmmParams.pool_address[vfl]

  -- flash_loan/verify_flash_loan have the same sender
  all fl: all_transactions_for[OpFlashLoan]
  | let vfl = FlashLoanParams.verify_txn_index[fl]
  | MainParams.user_address[fl] = MainParams.user_address[vfl]
} for 8 but 6 int
```

block *flash_swap* at amm_approval.tl:597

```
-- corresponds to block flash_swap at amm_approval.tl:597
one sig FlashSwapParams {
  verify_txn_index: Transaction -> set Transaction,
  swapped_assets: Transaction -> set Asset
} {
  all t: all_transactions_for[OpFlashSwap] {
    some t.verify_txn_index
    some t.swapped_assets
    t.swapped_assets in AmmParams.(asset1 + asset2)[t]
  }
  all t: Transaction - all_transactions_for[OpFlashSwap] {
    no t.verify_txn_index
    no t.swapped_assets
  }

  -- constraints derived directly from code

  all t: all_transactions_for[OpFlashSwap]
  | let verify_tr = t.verify_txn_index {

      -- assertions at amm_approval.tl:608-611
      verify_tr.type = AppCall
      verify_tr.op = OpVerifyFlashSwap

      -- assertion at amm_approval.tl:613
      VerifyFlashSwapParams.flash_swap_txn_index[verify_tr] = t

      -- assertion at amm_approval.tl:615-616
      AmmParams.pool_address[verify_tr] = AmmParams.pool_address[t]
      verify_tr.sender = MainParams.user_address[t]
    }
}
```

block *verify_flash_loan* at amm_approval.tl:640

```
-- corresponds to block verify_flash_swap at amm_approval.tl:640
one sig VerifyFlashSwapParams {
  flash_swap_txn_index: Transaction -> set Transaction
} {
  all t: all_transactions_for[OpVerifyFlashSwap] {
    some t.flash_swap_txn_index
    t.flash_swap_txn_index in t.prevs
  }
  all t: Transaction - all_transactions_for[OpVerifyFlashSwap] {
    no t.flash_swap_txn_index
  }

  -- constraints derived directly from code

  all t: all_transactions_for[OpVerifyFlashSwap]
  | let flash_swap_tr = t.flash_swap_txn_index,
        pool = AmmParams.pool_address[t]
    {

      -- assertions at amm_approval.tl:646-649
      flash_swap_tr.type = AppCall
      flash_swap_tr.op = OpFlashSwap

      -- assert at amm_approve.tl:651
      FlashSwapParams.verify_txn_index[flash_swap_tr] = t
      -- is not redundant, removing this assertion breaks 1-to-1
      -- correspondence property

      -- assert at amm_approve.tl:653 -- redundant
      -- to prove redundancy, just comment next line of code
      -- and execute verify_flash_swap_redundant_assertions checks
      AmmParams.pool_address[flash_swap_tr] = pool
    }
}
```

checks that show redundancy of some assertions in *verify_flash_swap*

```
verify_flash_swap_redundant_assertions:
check {
  -- 1-to-1 correspondence
  #all_transactions_for[OpFlashSwap] =
#all_transactions_for[OpVerifyFlashSwap]

  all fs: all_transactions_for[OpFlashSwap]
  | one FlashSwapParams.verify_txn_index[fs]

  all vfs: all_transactions_for[OpVerifyFlashSwap]
  | one VerifyFlashSwapParams.flash_swap_txn_index[vfs]

  all fs: all_transactions_for[OpFlashSwap]
  | let vfs = FlashSwapParams.verify_txn_index[fs]
  | VerifyFlashSwapParams.flash_swap_txn_index[vfs] = fs

  -- flash_swap/verify_flash_swap refer to the same pool
  all fs: all_transactions_for[OpFlashSwap]
  | let vfs = FlashSwapParams.verify_txn_index[fs]
  | AmmParams.pool_address[fs] = AmmParams.pool_address[vfs]

  -- flash_swap/verify_flash_swap have the same sender
  all fs: all_transactions_for[OpFlashSwap]
  | let vfs = FlashSwapParams.verify_txn_index[fs]
  | MainParams.user_address[fs] = MainParams.user_address[vfs]
} for 8 but 6 int
```

block *add_liquidity* at amm_approval.tl:712

```
-- corresponds to block add_liquidity at amm_approval.tl:712
enum AddLiquidityMode {Flexible, Single}

one sig AddLiquidityParams {
  asset1_txn_index: Transaction -> lone Transaction,
  asset2_txn_index: Transaction -> lone Transaction,
  mode: Transaction -> lone AddLiquidityMode
} {
  all t: all_transactions_for[OpAddLiquidity] {
    some t.(asset1_txn_index + asset2_txn_index)
    no t.asset1_txn_index & t.asset2_txn_index
    one t.mode
  }
  all t: Transaction - all_transactions_for[OpAddLiquidity] {
    no t.asset1_txn_index
    no t.asset2_txn_index
    no t.mode
  }

  -- constraints derived directly from code

  all t: all_transactions_for[OpAddLiquidity]
  | let asset1_txn = t.asset1_txn_index,
        asset2_txn = t.asset2_txn_index,
        pool = AmmParams.pool_address[t],
        asset1 = AmmParams.asset1[t],
        asset2 = AmmParams.asset2[t],
        tokens_issued = AmmParams.tokens_issued[t],
        mod = t.mode,
        user_address = MainParams.user_address[t]
    {
      -- from amm_approval.tl:727
      tokens_issued = Issued

      -- from amm_approval.tl:744-747
      mod = Flexible implies {
        one asset1_txn
        one asset2_txn
        asset1_txn = asset2_txn.prev
        asset2_txn.next = t
      }
      -- from amm_approval.tl:749-758
      mod = Single implies {
        one (asset1_txn + asset2_txn)
        (asset1_txn + asset2_txn).next = t
      }

      -- from amm_approval.tl:764-767
      some asset1_txn implies {
        asset1_txn.type = Transfer
        asset1_txn.asset = asset1
        asset1_txn.receiver = pool
        asset1_txn.sender = user_address
      }

      -- from amm_approval.tl:773-782
      some asset2_txn implies {
        asset2_txn.type = Transfer
        asset2_txn.asset = asset2
        asset2_txn.receiver = pool
        asset2_txn.sender = user_address
      }

    }
}
```

block *add_initial_liquidity* at amm_approval.tl:896

```
-- corresponds to block add_initial_liquidity at amm_approval.tl:896

one sig AddInitialLiquidityParams {
  asset1_txn_index: Transaction -> lone Transaction,
  asset2_txn_index: Transaction -> lone Transaction,
} {
  all t: all_transactions_for[OpAddInitialLiquidity] {
    one t.asset1_txn_index
    one t.asset2_txn_index
    t.asset1_txn_index = t.asset2_txn_index.prev
    t.asset2_txn_index.next = t
    no t.asset1_txn_index & t.asset2_txn_index
  }
  all t: Transaction - all_transactions_for[OpAddInitialLiquidity] {
    no t.asset1_txn_index
    no t.asset2_txn_index
  }
  all t: all_transactions_for[OpAddInitialLiquidity]
  | let asset1_txn = t.asset1_txn_index,
        asset2_txn = t.asset2_txn_index,
        pool = AmmParams.pool_address[t],
        asset1 = AmmParams.asset1[t],
        asset2 = AmmParams.asset2[t],
        toks_issued = AmmParams.tokens_issued[t],
        user_address = MainParams.user_address[t]
    {
      -- from amm_approval.tl:909
      toks_issued = NotIssued

      -- from amm_approval.tl:914-917
      asset1_txn.type = Transfer
      asset1_txn.asset = asset1
      asset1_txn.receiver = pool
      asset1_txn.sender = user_address

      -- from amm_approval.tl:922-928
      asset2_txn.type = Transfer
      asset2_txn.asset = asset2
      asset2_txn.receiver = pool
      asset2_txn.sender = user_address
    }
}
```

block *remove_liquidity* at amm_approval.tl:949

Since inner transactions modelling is not implemented yet, we won’t differentiate between requests with one and two assets.

```
-- corresponds to block remove_liquidity at amm_approval.tl:949
one sig RemoveLiquidityParams {
  token_txn_index: Transaction -> lone Transaction
} {
  all t: all_transactions_for[OpRemoveLiquidity] | some t.token_txn_index
  all t: Transaction - all_transactions_for[OpRemoveLiquidity] | no
t.token_txn_index
  all t: all_transactions_for[OpRemoveLiquidity]
  | let token_txn = t.token_txn_index,
        pool = AmmParams.pool_address[t],
        pool_token = pool.pool_token,
        user_address = MainParams.user_address[t]
    {
      -- from amm_approval.tl:965-968
      token_txn.type = Transfer
      token_txn.asset = pool_token
      token_txn.receiver = pool
      token_txn.sender = user_address

      -- from amm_approval.tl:977
      -- issued tokens may be zero in next transaction
      t not in last implies {
        -- strictly speaking this statement is not necessary to
        -- put explicitly, but may be helpful in future refactoring
        AmmParams.tokens_issued[t] in Issued + NotIssued
      }
    }
}
```

Verification of structural properties

Flash loans

```
-- check flash loan configurations

flash_loan1:
check {
  OpFlashLoan not in Transaction.op
  implies OpVerifyFlashLoan not in Transaction.op
}
      for 16 but 6 int
-- No counterexample found. Assertion may be valid. 686ms.

flash_loan2:
check {
  OpVerifyFlashLoan not in Transaction.op
  implies OpFlashLoan not in Transaction.op
}
      for 16 but 6 int
-- No counterexample found. Assertion may be valid. 605ms.

flash_loan3:
check {
  all fl: all_transactions_for[OpFlashLoan] {
    let vfl = FlashLoanParams.verify_txn_index[fl] {
      MainParams.user_address[fl] = MainParams.user_address[vfl]
    }
  }
} for 16 but 6 int
-- No counterexample found. Assertion may be valid. 2665ms.

flash_loan4:
check {
  all fl: all_transactions_for[OpFlashLoan] {
    let vfl = FlashLoanParams.verify_txn_index[fl] {
      AmmParams.pool_address[fl] = AmmParams.pool_address[vfl]
    }
  }
} for 16 but 6 int
-- No counterexample found. Assertion may be valid. 2232ms.
```

Flash swaps

```
-- check flash loan configurations
flash_swap1:
check {
  all fs: all_transactions_for[OpFlashSwap] {
    let vfs = FlashSwapParams.verify_txn_index[fs] {
      MainParams.user_address[fs] = MainParams.user_address[vfs]
    }
  }
} for 16 but 6 int
-- No counterexample found. Assertion may be valid. 4546ms.

flash_swap2:
check {
  all fs: all_transactions_for[OpFlashSwap] {
    let vfs = FlashSwapParams.verify_txn_index[fs] {
      AmmParams.pool_address[fs] = AmmParams.pool_address[vfs]
    }
  }
} for 16 but 6 int
-- No counterexample found. Assertion may be valid. 3849ms.
```

Enumeration of transaction groups

Only flash loans

```
ex_flash_loan1:
run {
  some Transaction.type & AppCall
  all t: Transaction
  | some t.op implies t.op in
    (
       OpFlashLoan
     + OpVerifyFlashLoan
    )
} for
9 -- maximum size of unconstrained sets (Account, Pool, Transaction ...)
but
5 int -- 5 bit in signed ints, so range is [-16..15]
```

Only flash swaps

```
ex_flash_swap1:
run {
  some Transaction.type & AppCall
  all t: Transaction
  | some t.op implies t.op in
    (
       OpFlashSwap
     + OpVerifyFlashSwap
    )
} for 4 but 5 int
```

Generic

```
ex_all1:
run {
  -- amount of selected operations
  #all_transactions_for[ OpAddInitialLiquidity ] = 1
  #all_transactions_for[ OpAddLiquidity        ] = 1
  #all_transactions_for[ OpRemoveLiquidity     ] = 1
  #all_transactions_for[ OpSwap                ] = 0
  #all_transactions_for[ OpFlashLoan           ] = 0
  #all_transactions_for[ OpVerifyFlashLoan     ] = 0
  #all_transactions_for[ OpFlashSwap           ] = 0
  #all_transactions_for[ OpVerifyFlashSwap     ] = 0
  #all_transactions_for[ OpSetFeeCollector     ] = 0
  #all_transactions_for[ OpSetFeeSetter        ] = 0
  #all_transactions_for[ OpSetFeeManager       ] = 0
  #all_transactions_for[ OpClaimFees           ] = 0
  #all_transactions_for[ OpClaimExtra          ] = 0
  #all_transactions_for[ OpSetFee              ] = 0

  #Pool = 1 -- Pools amount is set as a predicate, because it is a sub-signature

} for 16 -- common setting
but
16 Transaction,
5 Account,
4 Asset,
6 int -- [-16..15]
```

How to use this specification

This sepcification is intendent to be used in Alloy Analyzer.

Installation of Alloy Analizer

Alloy Analyzer can be installed from GitHub

It is just single Java archive file `jar` and Java Runtime Environment (JRE) is needed to execute the program.

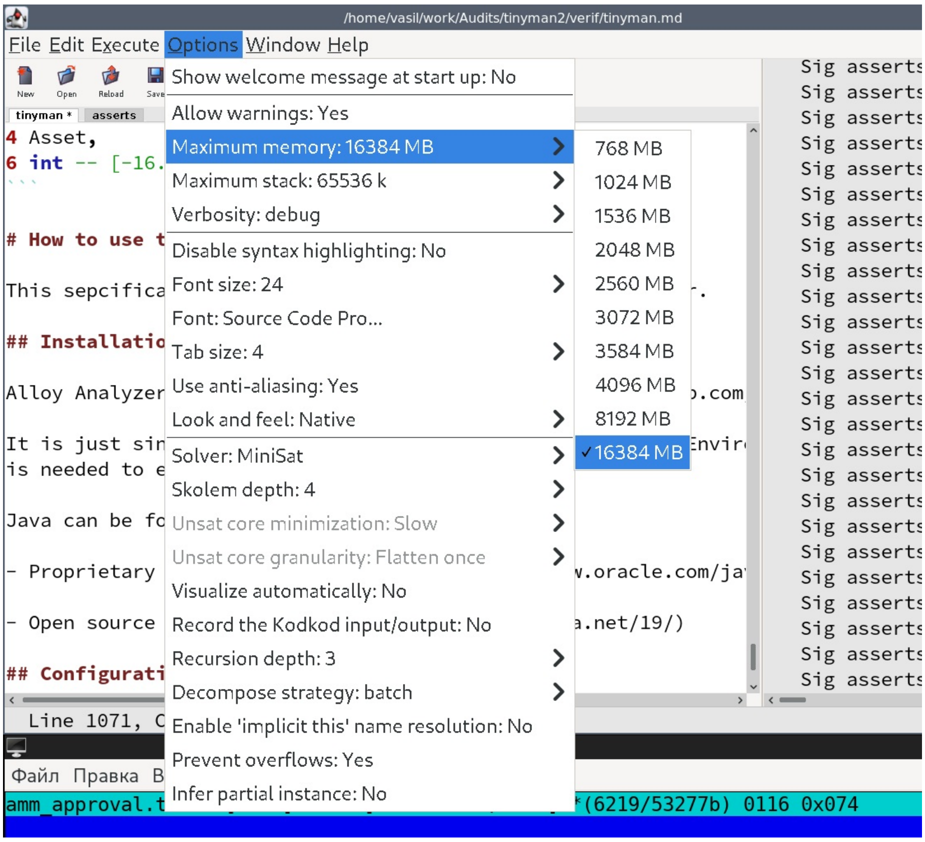
Java can be found in:

- Proprietary version on Oracle site
- Open source one from JDK site

Configuration

Alloy Analyzer should be configured according to the next instructions:

- Maximum memory is recommended to be set to 16Gb

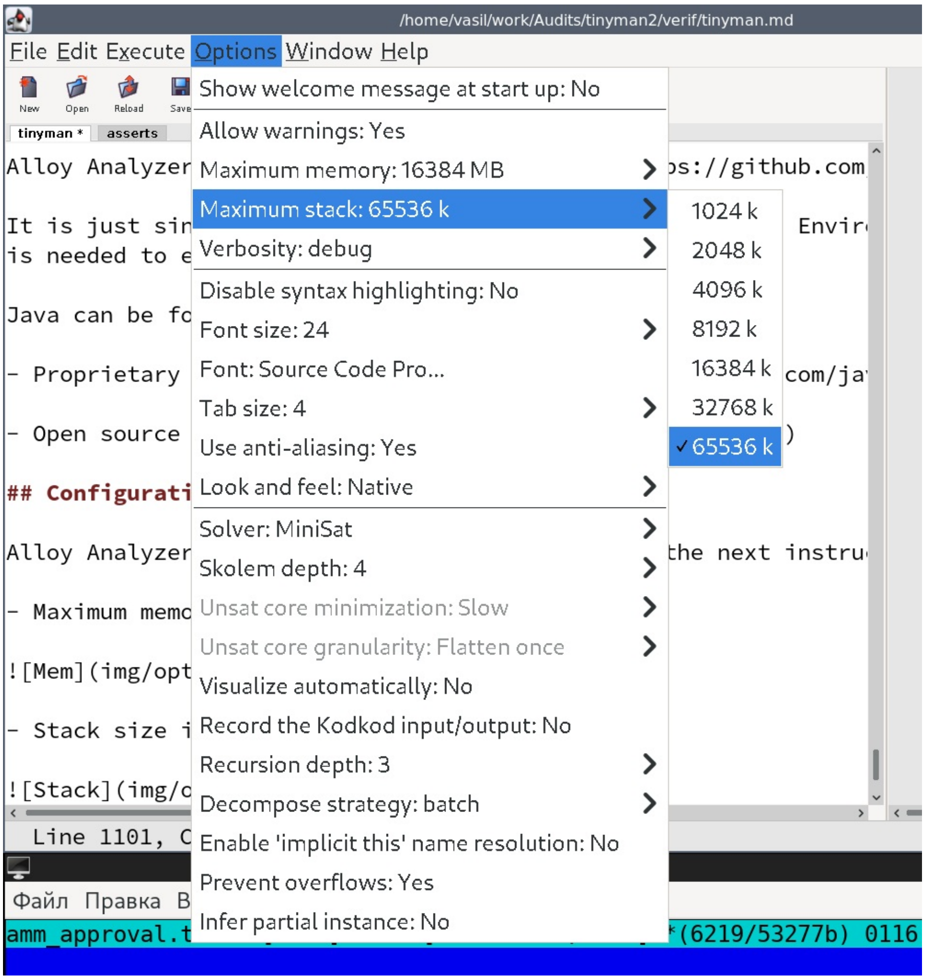

/home/vasil/work/Audits/tinyman2/verif/tinyman.md
File Edit Execute Options Window Help
New
Open
Reload
Save
tinyman *
asserts

Show welcome message at start up: No
Allow warnings: Yes
Maximum memory: 16384 MB
Maximum stack: 65536 k
Verbosity: debug
Disable syntax highlighting: No
Font size: 24
Font: Source Code Pro...
Tab size: 4
Use anti-aliasing: Yes
Look and feel: Native
Solver: MiniSat
Skolem depth: 4
Unsat core minimization: Slow
Unsat core granularity: Flatten once
Visualize automatically: No
Record the Kodkod input/output: No
Recursion depth: 3
Decompose strategy: batch
Enable 'implicit this' name resolution: No
Prevent overflows: Yes
Infer partial instance: No

768 MB
1024 MB
1536 MB
2048 MB
2560 MB
3072 MB
3584 MB
4096 MB
8192 MB
✓16384 MB

Mem

- Stack size is 16Mb



Show welcome message at start up: No

Allow warnings: Yes

Maximum memory: 16384 MB

Maximum stack: 65536 k

Verbosity: debug

Disable syntax highlighting: No

Font size: 24

Font: Source Code Pro...

Tab size: 4

Use anti-aliasing: Yes

Look and feel: Native

Solver: MiniSat

Skolem depth: 4

Unsat core minimization: Slow

Unsat core granularity: Flatten once

Visualize automatically: No

Record the Kodkod input/output: No

Recursion depth: 3

Decompose strategy: batch

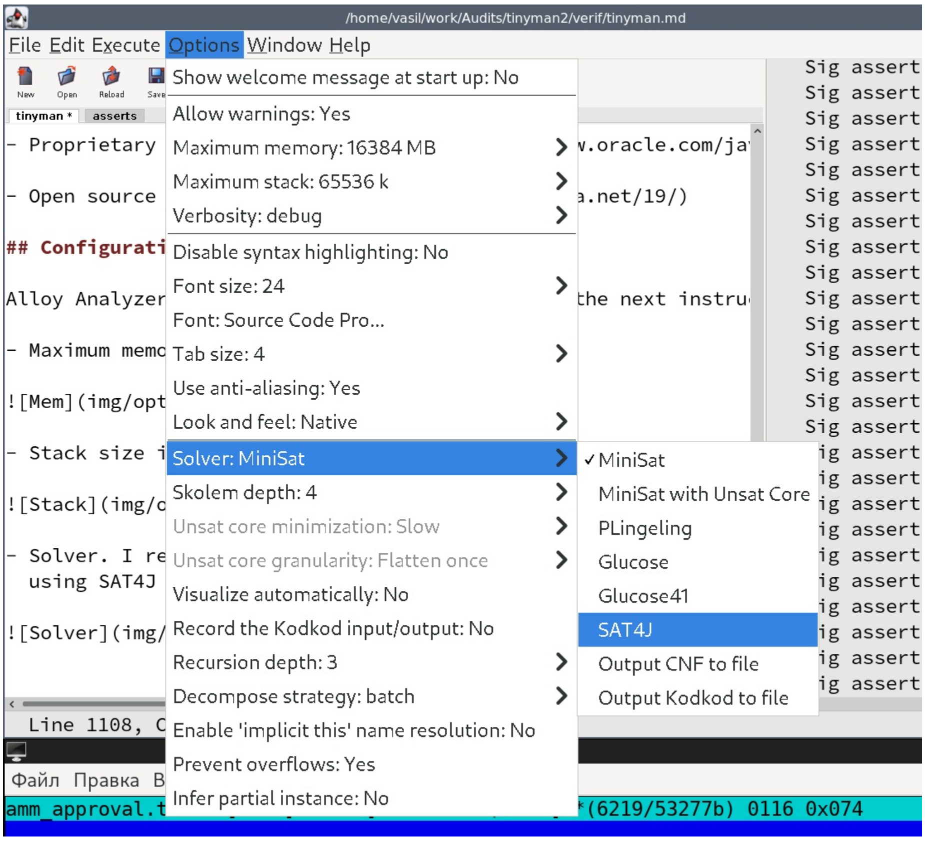
Enable 'implicit this' name resolution: No

Prevent overflows: Yes

Infer partial instance: No

Stack

- Solver. I recommend `SAT4J` as main solver, or `MiniSat` when using SAT4J takes a long time to get the result.



Show welcome message at start up: No

Allow warnings: Yes

Maximum memory: 16384 MB

Maximum stack: 65536 k

Verbosity: debug

Disable syntax highlighting: No

Font size: 24

Font: Source Code Pro...

Tab size: 4

Use anti-aliasing: Yes

Look and feel: Native

Solver: MiniSat

Skolem depth: 4

Unsat core minimization: Slow

Unsat core granularity: Flatten once

Visualize automatically: No

Record the Kodkod input/output: No

Recursion depth: 3

Decompose strategy: batch

Enable 'implicit this' name resolution: No

Prevent overflows: Yes

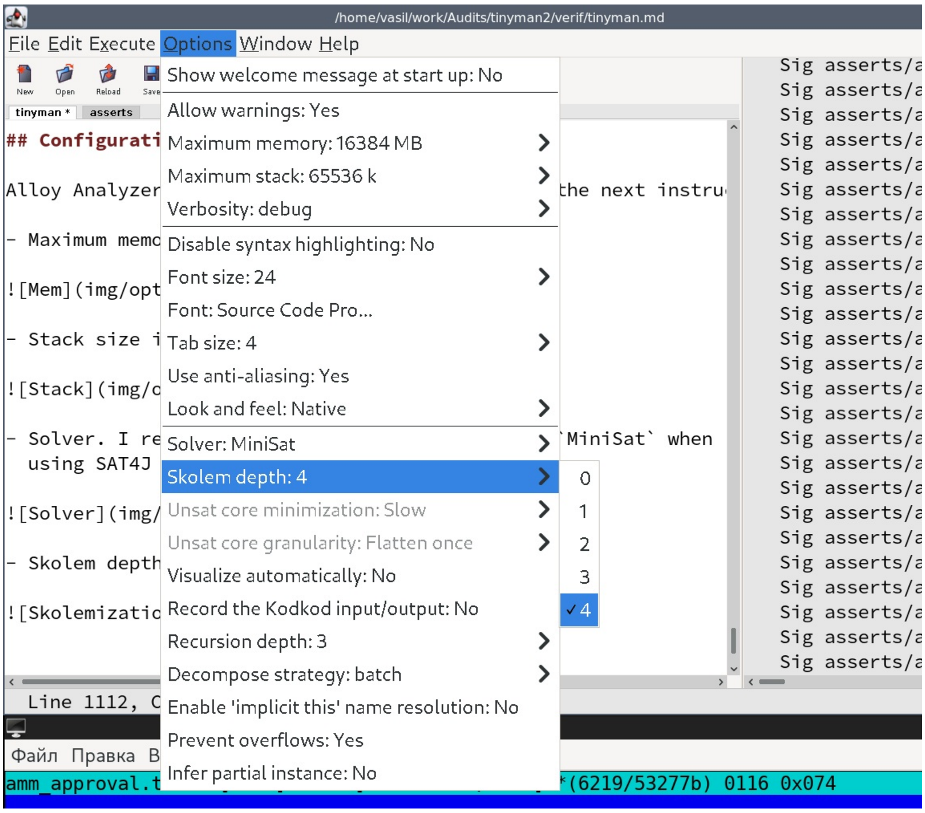
Infer partial instance: No

```
- Proprietary                    w.oracle.com/ja
- Open source                    a.net/19/)
## Configurati
Alloy Analyzer                   the next instru
- Maximum memo
![Mem](img/opt
- Stack size i
![Stack](img/o
- Solver. I re
  using SAT4J
![Solver](img/
```

```
Sig assert
Sig assert
Sig assert
Sig assert
Sig assert
Sig assert
Sig assert
Sig assert
Sig assert
Sig assert
Sig assert
Sig assert
Sig assert
Sig assert
Sig assert
ig assert
ig assert
ig assert
ig assert
ig assert
ig assert
ig assert
ig assert
ig assert
ig assert
```

Solver

- Skolem depth. Set it to 4 (max)


/home/vasil/work/Audits/tinyman2/verif/tinyman.md
File Edit Execute Options Window Help
New
Open
Reload
Save
tinyman *
asserts
Line 1112, C
Файл Правка В

Show welcome message at start up: No
Allow warnings: Yes
Maximum memory: 16384 MB
Maximum stack: 65536 k
Verbosity: debug
Disable syntax highlighting: No
Font size: 24
Font: Source Code Pro...
Tab size: 4
Use anti-aliasing: Yes
Look and feel: Native
Solver: MiniSat
Skolem depth: 4
Unsat core minimization: Slow
Unsat core granularity: Flatten once
Visualize automatically: No
Record the Kodkod input/output: No
Recursion depth: 3
Decompose strategy: batch
Enable 'implicit this' name resolution: No
Prevent overflows: Yes
Infer partial instance: No

0
1
2
3
✓ 4

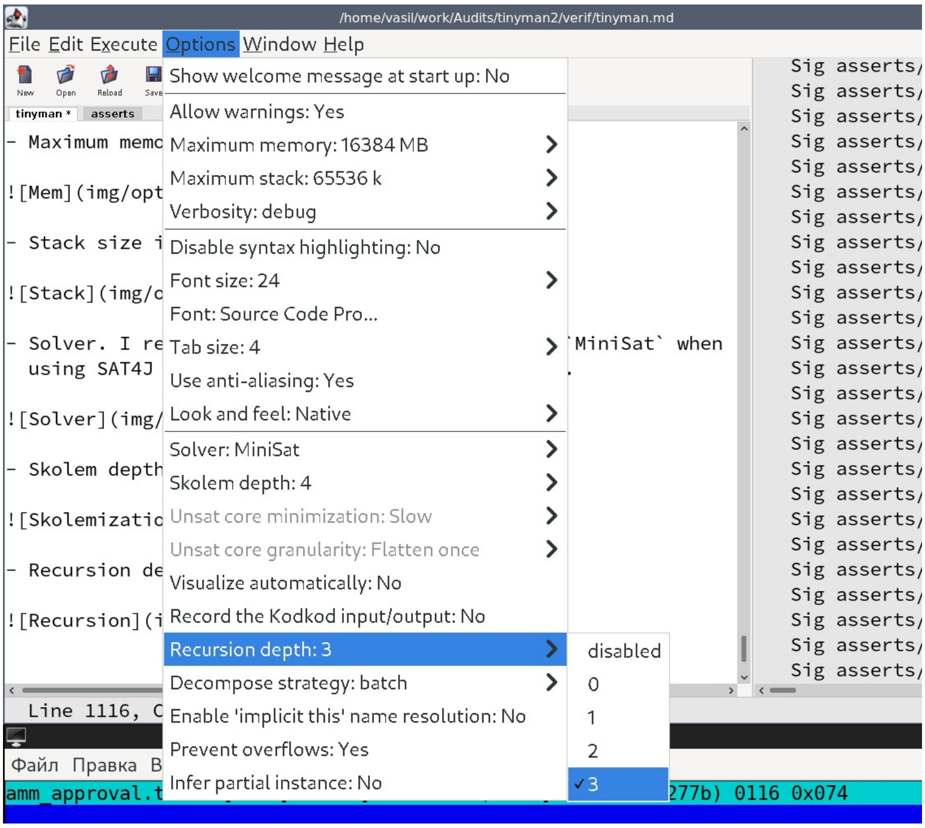
```
## Configurati
Alloy Analyzer                                the next instru
- Maximum memo
![Mem](img/opt
- Stack size i
![Stack](img/c
- Solver. I re                                `MiniSat` when
  using SAT4J
![Solver](img/
- Skolem depth
![Skolemizatic
```

```
Sig asserts/a
```

```
amm_approval.t                               *(6219/53277b) 0116 0x074
```

Skolemization

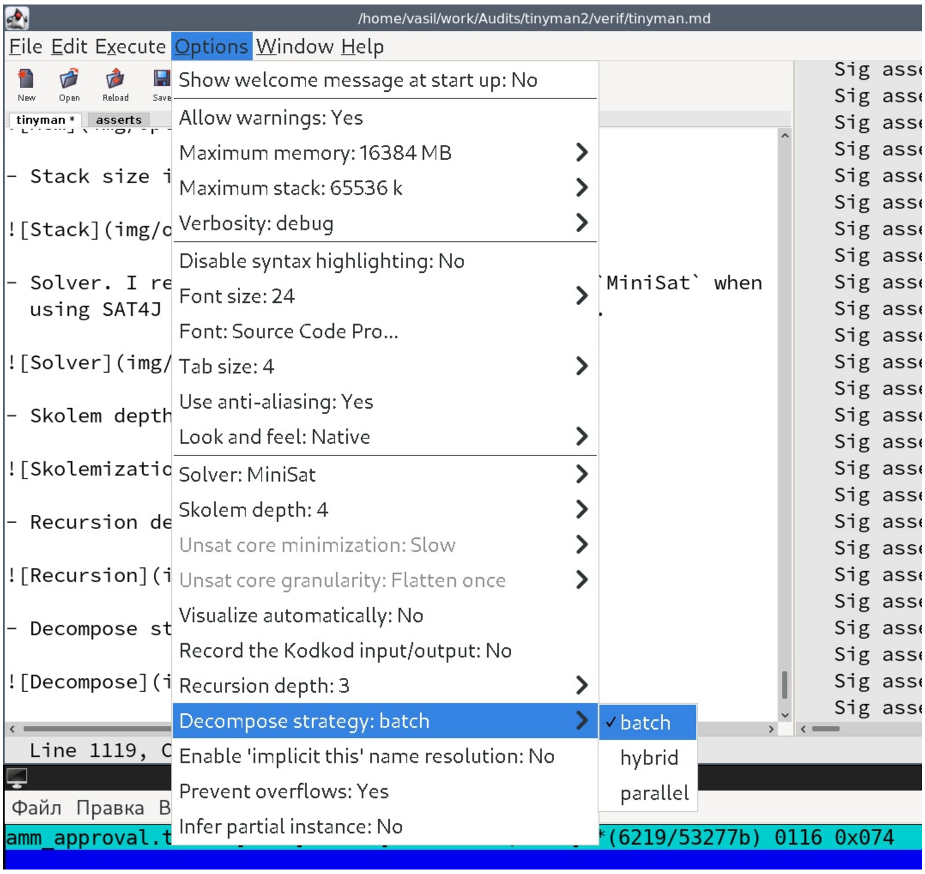
- Recursion depth. Set it to 3 (max)



Show welcome message at start up: No
Allow warnings: Yes
Maximum memory: 16384 MB
Maximum stack: 65536 k
Verbosity: debug
Disable syntax highlighting: No
Font size: 24
Font: Source Code Pro...
Tab size: 4
Use anti-aliasing: Yes
Look and feel: Native
Solver: MiniSat
Skolem depth: 4
Unsat core minimization: Slow
Unsat core granularity: Flatten once
Visualize automatically: No
Record the Kodkod input/output: No
Recursion depth: 3
Decompose strategy: batch
Enable 'implicit this' name resolution: No
Prevent overflows: Yes
Infer partial instance: No

Recursion

- Decompose strategy: `batch`


/home/vasil/work/Audits/tinyman2/verif/tinyman.md
File Edit Execute Options Window Help
New
Open
Reload
Save
tinyman *
asserts
Line 1119, C
Файл Правка В

Show welcome message at start up: No
Allow warnings: Yes
Maximum memory: 16384 MB
Maximum stack: 65536 k
Verbosity: debug
Disable syntax highlighting: No
Font size: 24
Font: Source Code Pro...
Tab size: 4
Use anti-aliasing: Yes
Look and feel: Native
Solver: MiniSat
Skolem depth: 4
Unsat core minimization: Slow
Unsat core granularity: Flatten once
Visualize automatically: No
Record the Kodkod input/output: No
Recursion depth: 3
Decompose strategy: batch
Enable 'implicit this' name resolution: No
Prevent overflows: Yes
Infer partial instance: No

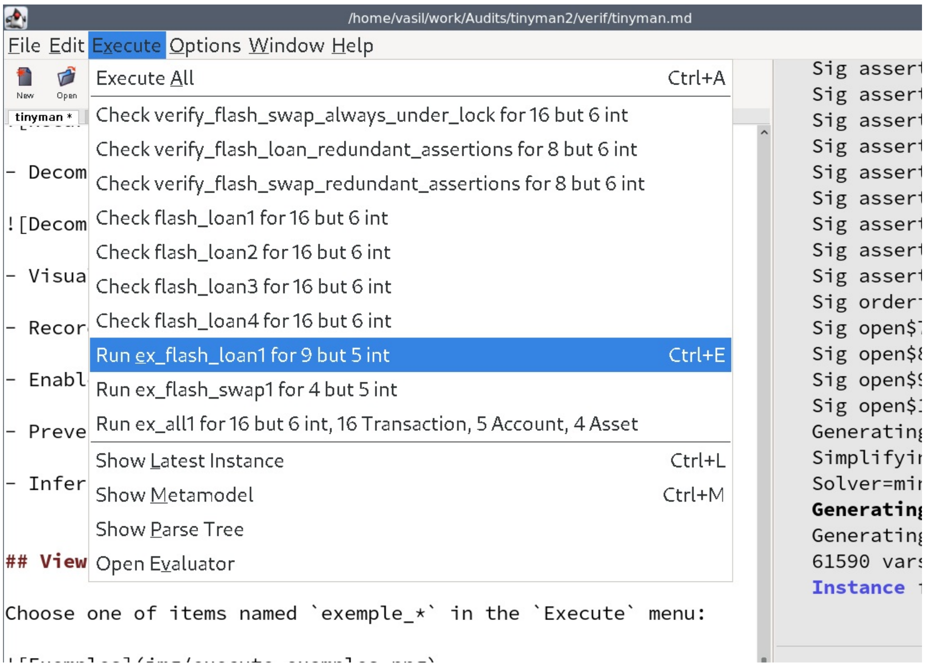
✓ batch
hybrid
parallel

Decompose

- Visualize automatically: `No`
- Record the Kodkod input/output: `No`
- Enable ‘implicit this’ name resolution: `No`
- Prevent overflows: `Yes`
- Infer partial instance: `No`

Viewing of possible transaction groups

Choose one of items named `Run ex_*` in the `Execute` menu:

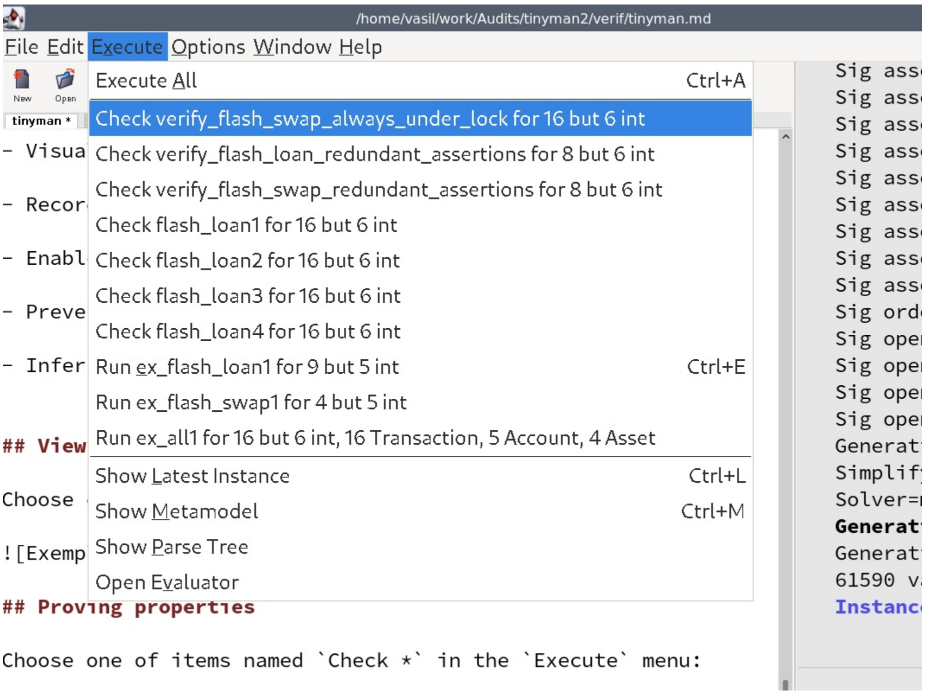

/home/vasil/work/Audits/tinyman2/verif/tinyman.md
File Edit Execute Options Window Help
New Open
tinyman *
Execute All Ctrl+A
Check verify_flash_swap_always_under_lock for 16 but 6 int
Check verify_flash_loan_redundant_assertions for 8 but 6 int
Check verify_flash_swap_redundant_assertions for 8 but 6 int
Check flash_loan1 for 16 but 6 int
Check flash_loan2 for 16 but 6 int
Check flash_loan3 for 16 but 6 int
Check flash_loan4 for 16 but 6 int
Run ex_flash_loan1 for 9 but 5 int Ctrl+E
Run ex_flash_swap1 for 4 but 5 int
Run ex_all1 for 16 but 6 int, 16 Transaction, 5 Account, 4 Asset
Show Latest Instance Ctrl+L
Show Metamodel Ctrl+M
Show Parse Tree
Open Evaluator
- Decom
![Decom
- Visua
- Recor
- Enabl
- Preve
- Infer
## View
Sig assert
Sig assert
Sig assert
Sig assert
Sig assert
Sig assert
Sig assert
Sig assert
Sig assert
Sig order
Sig open$7
Sig open$8
Sig open$9
Sig open$1
Generating
Simplifyin
Solver=min
Generating
Generating
61590 vars
Instance f

```
Choose one of items named `exemple_*` in the `Execute` menu:
```

Exemples

Proving properties

Choose one of items named `Check *` in the `Execute` menu:


/home/vasil/work/Audits/tinyman2/verif/tinyman.md
File Edit Execute Options Window Help
New Open
tinyman *
Execute All Ctrl+A
Check verify_flash_swap_always_under_lock for 16 but 6 int
Check verify_flash_loan_redundant_assertions for 8 but 6 int
Check verify_flash_swap_redundant_assertions for 8 but 6 int
Check flash_loan1 for 16 but 6 int
Check flash_loan2 for 16 but 6 int
Check flash_loan3 for 16 but 6 int
Check flash_loan4 for 16 but 6 int
Run ex_flash_loan1 for 9 but 5 int Ctrl+E
Run ex_flash_swap1 for 4 but 5 int
Run ex_all1 for 16 but 6 int, 16 Transaction, 5 Account, 4 Asset
Show Latest Instance Ctrl+L
Show Metamodel Ctrl+M
Show Parse Tree
Open Evaluator
- Visua
- Recor
- Enabl
- Preve
- Infer
## View
Choose
![Exemp
## Proving properties
Sig ass
Sig ord
Sig ope
Generat
Simplif
Solver=
Generat
Generat
61590 v
Instanc

```
Choose one of items named `Check *` in the `Execute` menu:
```

Checks

Tweaking examples

Go to the one of `run` blocks, that labeled by `ex_*` label, alter some parameters (according to descriptions around them) and rerun corresponding item in `Execute` menu.

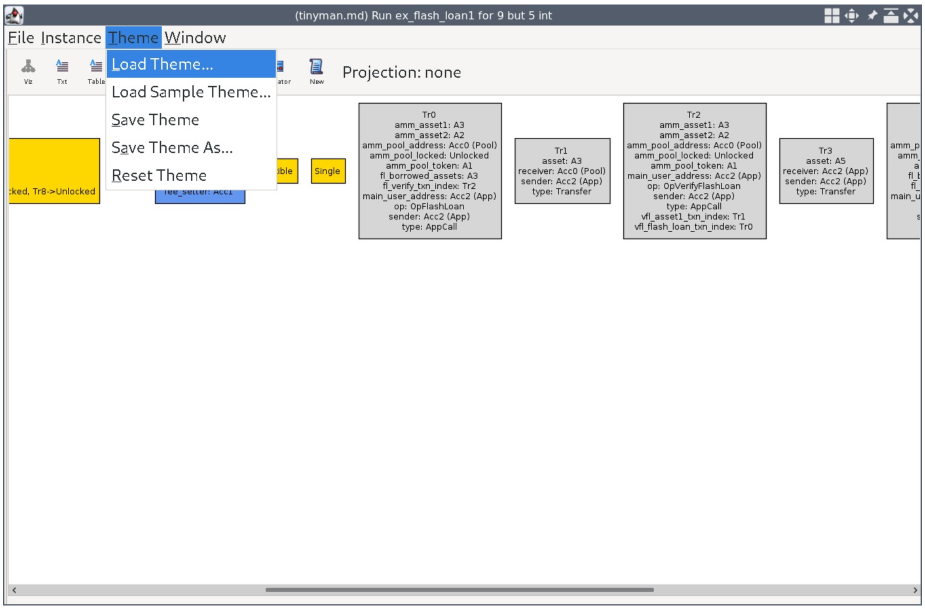
NB: befor rerunning examples, close visualizer window, rerun example and open visualizer again.

Visualizer

Visualizer is invoked by pressing `Ctrl+L` after running the example.

Visualizer allows to use themes for better representation of found models.

Changing theme in visualizer:



Theme

Useful links

- Alloy analyzer

TODO

1. Add modeling of inner transactions
2. Add modeling of some quantitative properties