



UNIVERSITÀ DEGLI STUDI DI MILANO
DIPARTIMENTO DI INFORMATICA

Towards the Automated Verification of (Ethereum) Smart Contracts

Chiara Braghin, Elvinia Riccobene and Simone Valentini

chiara.braghin@unimi.it

Why smart contract verification?

- Smart contracts can hold *significant financial assets*
- They are *immutable* after deployment
- Source code is *publicly available*
- *Anyone* can submit a transaction to a contract

And...

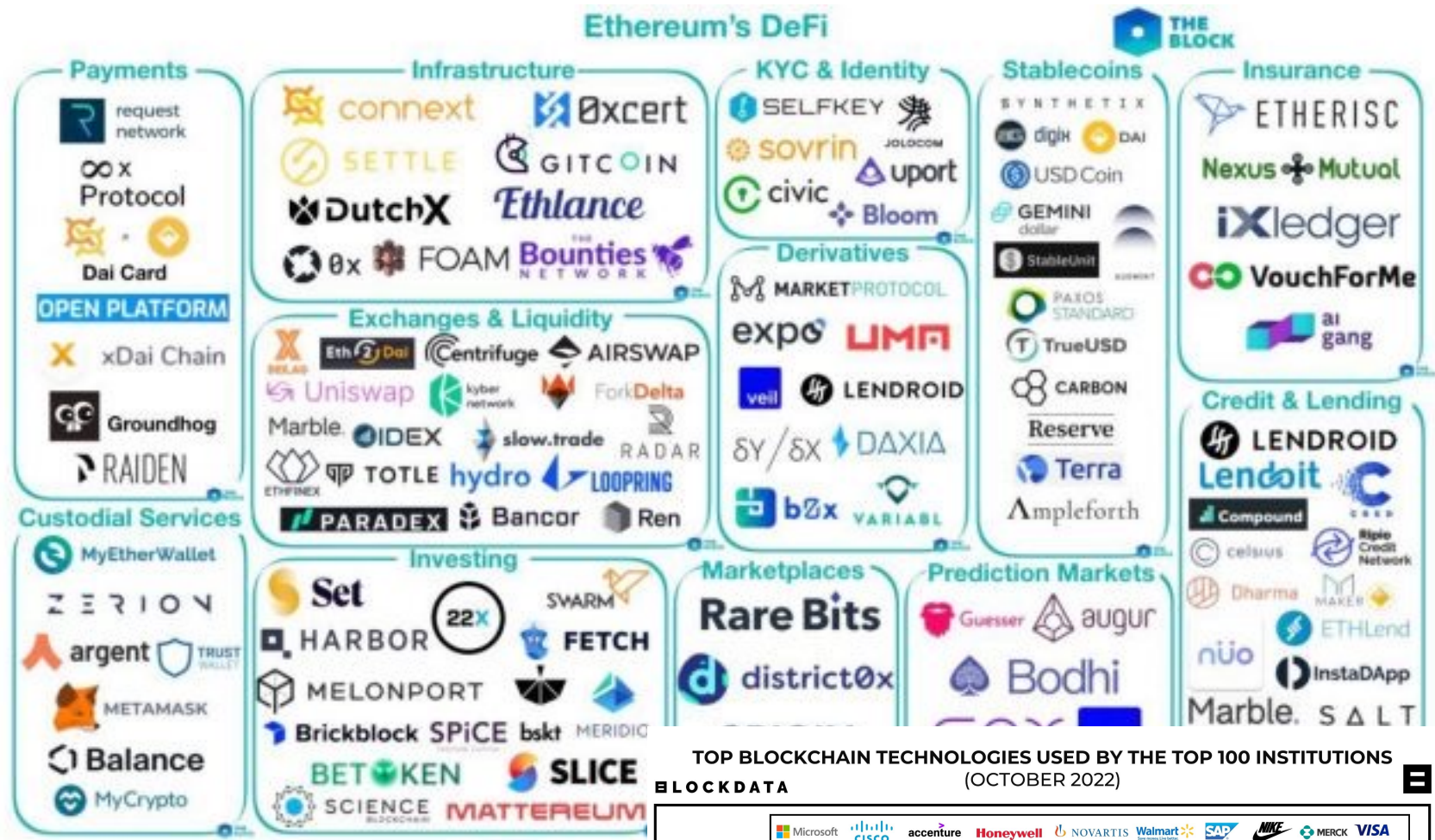
- Writing code correctly is hard
- Often the semantics of a SC language is not fully understood by programmers



Why smart contract verification?

- DAO Hack on June 17, 2016: worth 3.6 million ETH, about \$70 million
- Veritaseum attack on April 2018: worth \$8.4 million
- "Double" bZx DeFi Hack:
 - (1) on February 14, 2020 worth \$6 million
 - (2) on February 18, 2020, additional \$350,000
- Grim Finance on Dec 2021: worth \$30 million
- ...

Why Ethereum?



TOP BLOCKCHAIN TECHNOLOGIES USED BY THE TOP 100 INSTITUTIONS (OCTOBER 2022)



State of Art (1) - Common Vulnerabilities

- Integer Overflow and Underflow
- Default Visibilities
- Race Conditions (Reentrancy, Cross-function race conditions)
- Timestamp Dependence
- DoS with Block Gas Limit
- Forcibly Sending Ether to a Contract



State of Art (2) - Which verification technique?

- Formal verification (of bytecode or Solidity code)
 - Oyente (CCS 2016) – symbolic execution
 - VeriSol (Microsoft 2019) – Boogie intermediate language
 - Solc-Verify (VSSTTE 2019, ESOP 2020)
 - Securify 2.0 (2020) - context-sensitive static analysis in Datalog
 -
- Common features of existing formal techniques:
 - Not fully automated
 - Difficult formal languages, translation task might be error prone
 - Not user friendly
 - Not maintained, code not available
 - Focused only on some types of errors/attacks

Our approach: using ASM and ASMETA

- Abstract State Machine (ASM) [1,2]
 - an extension of Finite State Machines, replacing unstructured FSM control states with *algebraic structures*
 - state transitions are performed by firing *transition rules*
 - different *computational paradigms*: single agent and multi-agent
 - ASM model predefined structure: a *signature* with declarations of domains and functions; a block of *definitions* of static domains and functions, transition rules, state invariants and properties to verify; a *main rule*; a set of *initial states*, one of which is elected as default.

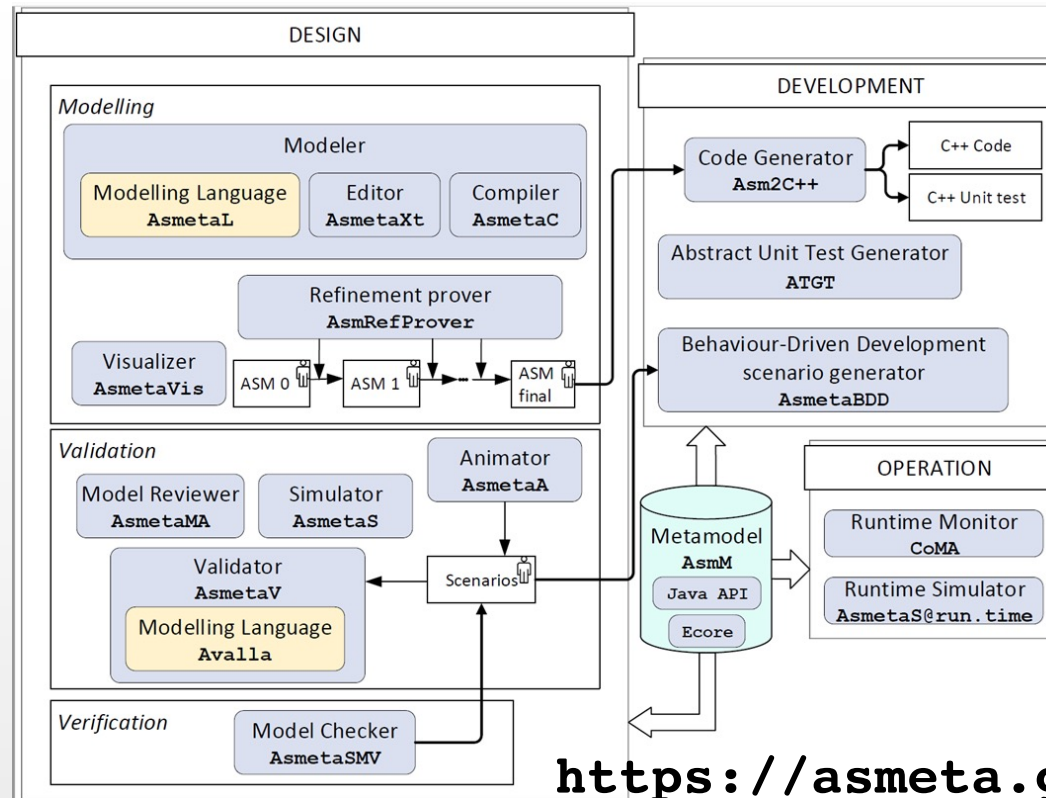
[1] Börger, E. and Raschke, A. (2018). Modeling Companion for Software Practitioners. Springer Verlag.

[2] Börger, E. and Stärk, R. (2003). Abstract State Machines: A Method for High-Level System Design and Analysis. Springer Verlag.



Our approach: using ASM and ASMETA

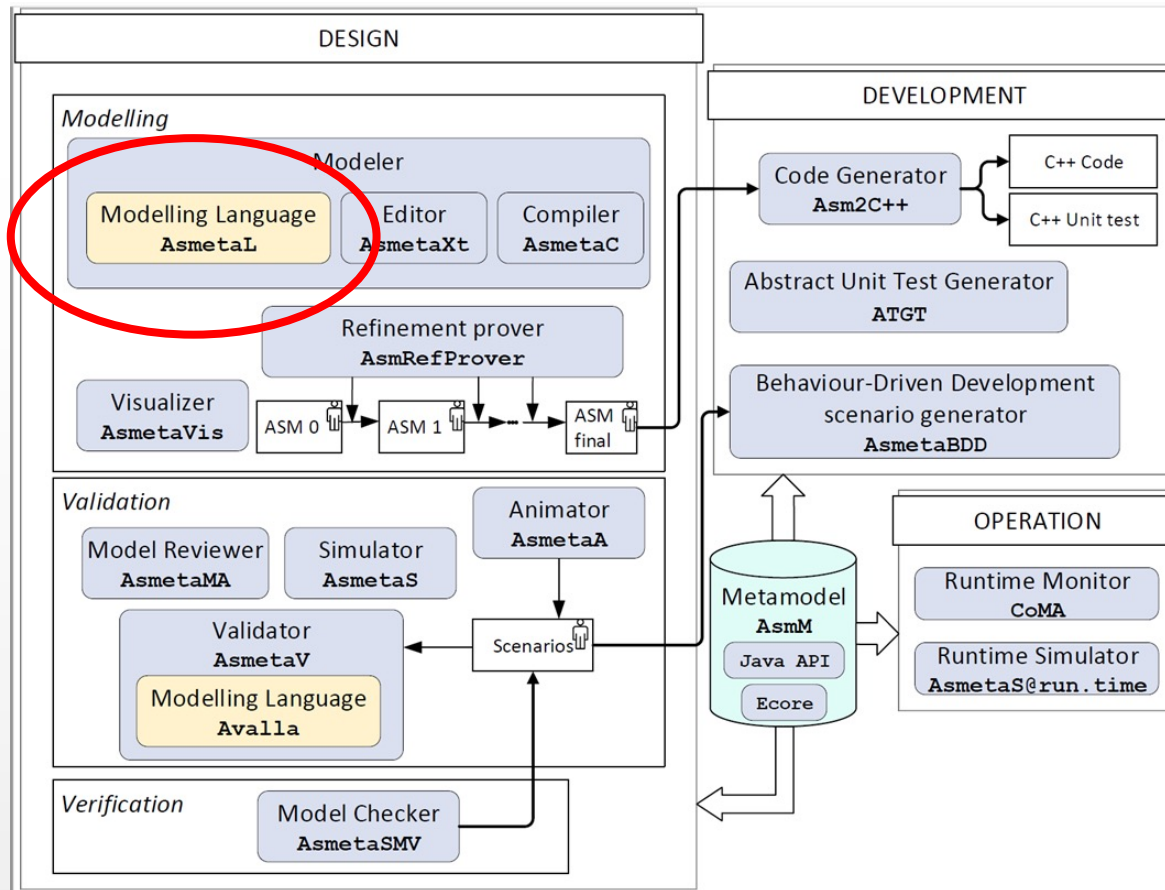
- ASM mETAmodeling: a toolset supporting ASM formal method for model editing, validation and verification [3]



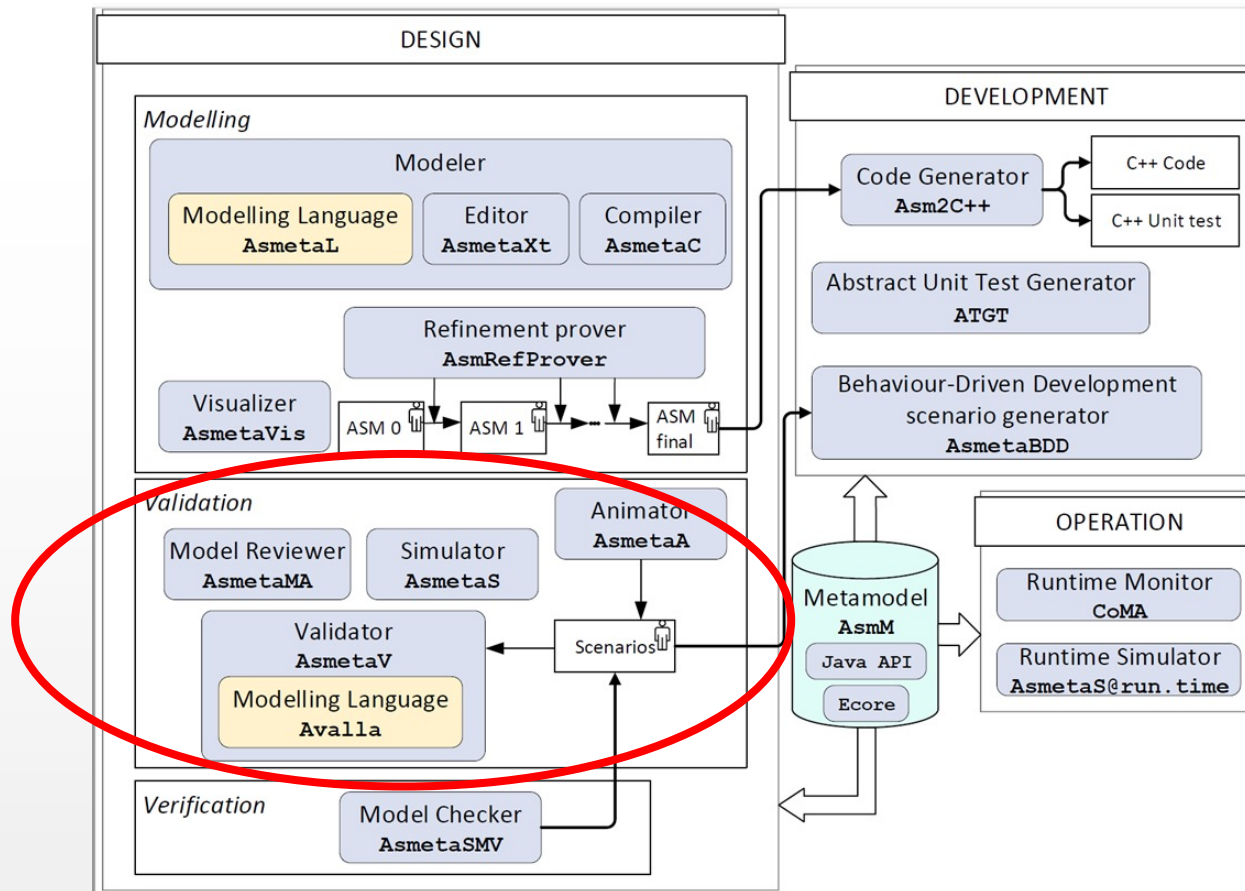
<https://asmeta.github.io/>

- [3] Arcaini, P., Gargantini, A., Riccobene, E., and Scandurra, P. (2011). A model-driven process for engineering a toolset for a formal method. *Software: Practice and Experience*, 41(2):155–166.

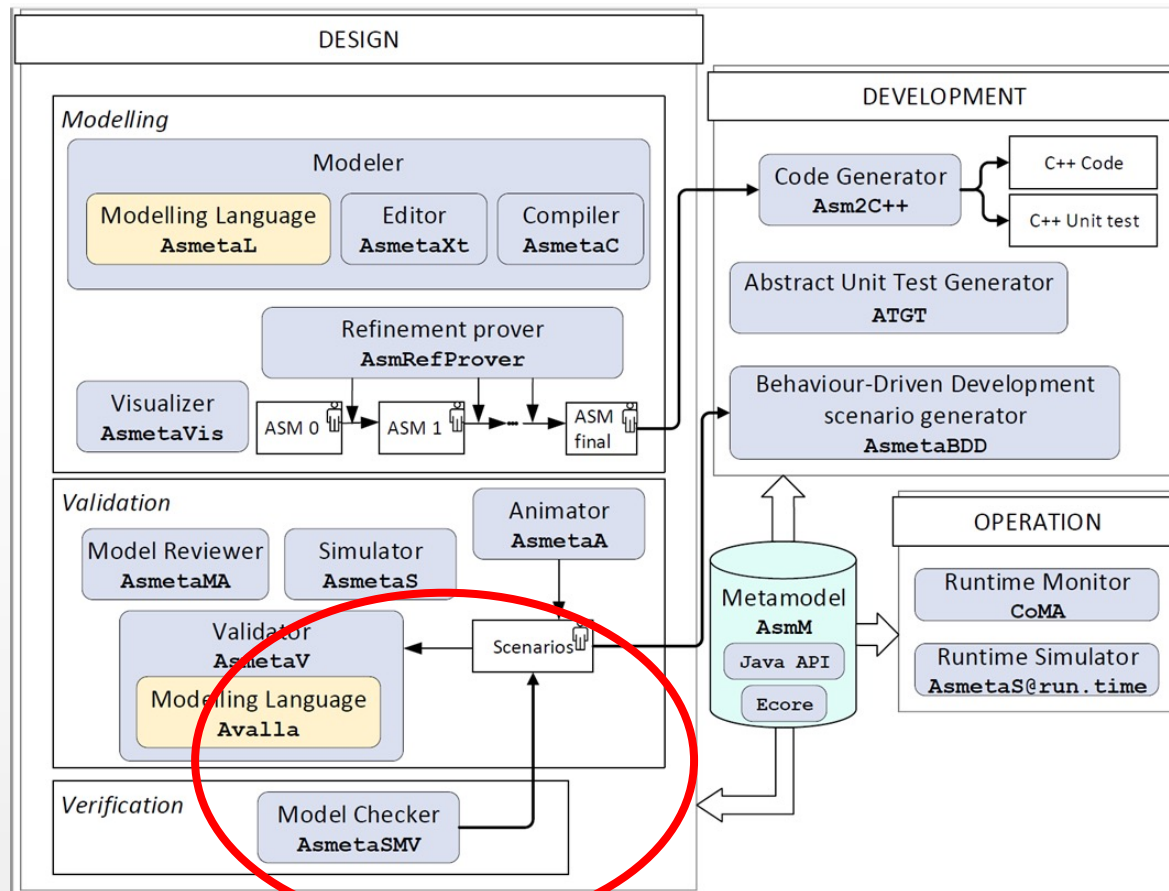
Our approach: using ASM and ASMETA



Our approach: using ASM and ASMETA



Our approach: using ASM and ASMETA



Advantages

- Easy pseudo-code format
- Executable models for different light forms of analysis
- Maintained

ASM by example (1)

```
contract DAO {
  mapping (address => uint) balances;
  function Deposit() {
    balances[msg.sender] += msg.value;
  }
  function Withdraw(uint amount) {
    if (balances[msg.sender] >= amount) {
      msg.sender.call.value(amount);
      balances[msg.sender] -= amount;
    }
  }
}
```

balances is updated
only after ether
transfer

ASM by example (1)

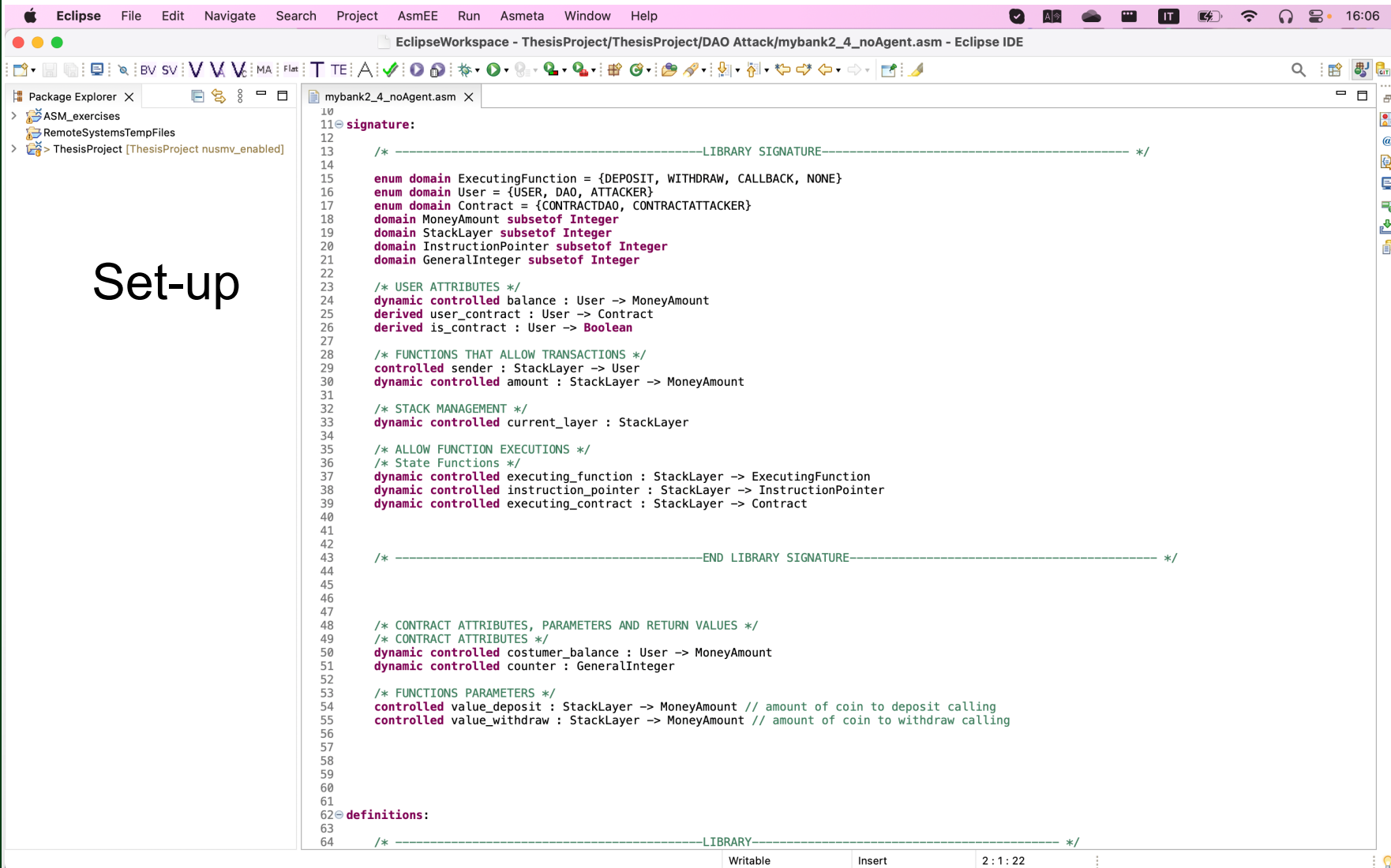
```
contract DAO {
  mapping (address => uint) balances;
  function Deposit() {
    balances[msg.sender] += msg.value;
  }
  function Withdraw(uint amount) {
    if (balances[msg.sender] >= amount) {
      msg.sender.call.value(amount);
      balances[msg.sender] -= amount;
    }
  }
}
```

balances is updated
only after ether
transfer

```
contract Attacker {
  ...
  function moveBalance() {
    dao.Withdraw();
  }
  function () payable {
    dao.Withdraw();
  }
}
```

ASM by example (2)

Set-up



```
10
11 signature:
12
13 /* ----- LIBRARY SIGNATURE ----- */
14
15 enum domain ExecutingFunction = {DEPOSIT, WITHDRAW, CALLBACK, NONE}
16 enum domain User = {USER, DAO, ATTACKER}
17 enum domain Contract = {CONTRACTDAO, CONTRACTATTACKER}
18 domain MoneyAmount subsetof Integer
19 domain StackLayer subsetof Integer
20 domain InstructionPointer subsetof Integer
21 domain GeneralInteger subsetof Integer
22
23 /* USER ATTRIBUTES */
24 dynamic controlled balance : User -> MoneyAmount
25 derived user_contract : User -> Contract
26 derived is_contract : User -> Boolean
27
28 /* FUNCTIONS THAT ALLOW TRANSACTIONS */
29 controlled sender : StackLayer -> User
30 dynamic controlled amount : StackLayer -> MoneyAmount
31
32 /* STACK MANAGEMENT */
33 dynamic controlled current_layer : StackLayer
34
35 /* ALLOW FUNCTION EXECUTIONS */
36 /* State Functions */
37 dynamic controlled executing_function : StackLayer -> ExecutingFunction
38 dynamic controlled instruction_pointer : StackLayer -> InstructionPointer
39 dynamic controlled executing_contract : StackLayer -> Contract
40
41
42 /* ----- END LIBRARY SIGNATURE ----- */
43
44
45
46
47
48 /* CONTRACT ATTRIBUTES, PARAMETERS AND RETURN VALUES */
49 /* CONTRACT ATTRIBUTES */
50 dynamic controlled costumer_balance : User -> MoneyAmount
51 dynamic controlled counter : GeneralInteger
52
53 /* FUNCTIONS PARAMETERS */
54 controlled value_deposit : StackLayer -> MoneyAmount // amount of coin to deposit calling
55 controlled value_withdraw : StackLayer -> MoneyAmount // amount of coin to withdraw calling
56
57
58
59
60
61
62 definitions:
63
64 /* ----- LIBRARY ----- */
```



ASM by example (2)

```
91
92
93⊖ /*
94  * TRANSACTION RULE
95  */
96⊖ rule r_Transaction($s in User, $r in User, $n in MoneyAmount, $f in ExecutingFunction) =
97⊖   if balance($s) >= $n and $n >= 0 then
98⊖     let ($ucr = user_contract($r), $cl = current_layer) in
99⊖       par
100        balance($s) := balance($s) - $n // subtracts the amount from the sender user balance
101        balance($r) := balance($r) + $n // adds the amount to the dest user balance
102⊖       if is_contract($r) then
103⊖         par
104            sender($cl + 1) := $s // set the transition attribute to the sender user
105            amount($cl + 1) := $n // set the transaction attribute to the amount of coin to transfer
106            current_layer := $cl + 1
107            executing_contract($cl + 1) := $ucr
108            executing_function($cl + 1) := $f
109            instruction_pointer($cl + 1) := 0
110            instruction_pointer($cl) := instruction_pointer($cl) + 1
111        endpar
112       endif
113     endpar
114   endlet
115 endif
116
117
118⊖ /*
119  * RETURN RULE
120  */
121⊖ rule r_Ret =
122   current_layer := current_layer - 1
123
```

Set-up

ASM by example (2)

```
133⓪ /*
134   * DEPOSIT FUNCTION RULE
135   */
136
137⓪ rule r_Deposit_bank1 =
138⓪   let ($cl = current_layer) in
139⓪   let ($scl = sender($cl)) in
140⓪     if executing_function($cl) = DEPOSIT then
141⓪       switch instruction_pointer($cl)
142⓪         case 0 :
143⓪           par
144⓪             value_deposit($cl) := amount($cl)
145⓪             instruction_pointer($cl) := instruction_pointer($cl) + 1
146⓪           endpar
147⓪         case 1 :
148⓪           par
149⓪             costumer_balance($scl) := costumer_balance($scl) + value_deposit($cl)
150⓪             r_Ret[]
151⓪           endpar
152⓪       endswitch
153⓪     endif
154⓪   endlet
155⓪ endlet
156
157
158⓪ /*
159   * WITHDRAW FUNCTION RULE
160   */
161
162⓪ rule r_Withdraw_bank1 =
163⓪   let ($cl = current_layer) in
164⓪   let ($scl = sender($cl)) in
165⓪     if executing_function($cl) = WITHDRAW then
166⓪       switch instruction_pointer($cl)
167⓪         case 0 :
168⓪           par
169⓪             value_withdraw($cl) := 1
170⓪             instruction_pointer($cl) := instruction_pointer($cl) + 1
171⓪           endpar
172⓪         case 1 :
173⓪           if costumer_balance($scl) >= value_withdraw($cl) then
174⓪             r_Transaction[DAO, $scl, value_withdraw($cl), CALLBACK]
175⓪           else
176⓪             r_Ret[]
177⓪           endif
178⓪         case 2 :
179⓪           par
180⓪             costumer_balance($scl) := costumer_balance($scl) - value_withdraw($cl)
181⓪             r_Ret[]
182⓪           endpar
183⓪       endswitch
184⓪     endif
185⓪   endlet
186⓪ endlet
```

```
function Deposit() {
    balances[msg.sender] += msg.value;
}
```

```
function Withdraw(uint amount) {
    if (balances[msg.sender] >= amount) {
        msg.sender.call.value(amount);
        balances[msg.sender] -= amount;
    }
}
```



ASM by example (2)

```
arch Project AsmEE Run Asmeta Window Help
EclipseWorkspace - ThesisProject/ThesisProject/DAO Attack/mybank2_4_noAgent.asm - Eclipse IDE

mybank2_4_noAgent.asm X
187
188
189
190  /*
191  * CALLBACK FUNCTION RULE
192  */
193
194  rule r_Callback_bank1 =
195  let ($cl = current_layer) in
196  if executing_function($cl) != DEPOSIT and executing_function($cl) != WITHDRAW then
197  switch instruction_pointer($cl)
198  case 0 :
199      r_Ret[]
200  endswitch
201  endif
202  endlet
203
204
205
206
207
208
209
210
211  /* -----ATTACKER CONTRACT IMPLEMENTATION----- */
212
213  /*
214  * CALLBACK FUNCTION RULE
215  */
216  rule r_Callback_bank2 =
217  let ($cl = current_layer) in
218  switch instruction_pointer($cl)
219  case 0 :
220  if counter < 2 then
221  par
222      r_Transaction[ATTACKER, DAO, 1, WITHDRAW]
223      counter := counter + 1
224  endpar
225  else
226      r_Transaction[ATTACKER, DAO, 1, CALLBACK]
227  endif
228  case 1 :
229      r_Ret[]
230  endswitch
231  endlet
232
233
234
235
236
237
238  /* -----MAINS AND INVARIANTS----- */
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
280
281
282
283
284
285
286
287
288
289
290
291
292
293
294
295
296
297
298
299
300
301
302
303
304
305
306
307
308
309
310
311
312
313
314
315
316
317
318
319
320
321
322
323
324
325
326
327
328
329
330
331
332
333
334
335
336
337
338
339
340
341
342
343
344
345
346
347
348
349
350
351
352
353
354
355
356
357
358
359
360
361
362
363
364
365
366
367
368
369
370
371
372
373
374
375
376
377
378
379
380
381
382
383
384
385
386
387
388
389
390
391
392
393
394
395
396
397
398
399
400
401
402
403
404
405
406
407
408
409
410
411
412
413
414
415
416
417
418
419
420
421
422
423
424
425
426
427
428
429
430
431
432
433
434
435
436
437
438
439
440
441
442
443
444
445
446
447
448
449
450
451
452
453
454
455
456
457
458
459
460
461
462
463
464
465
466
467
468
469
470
471
472
473
474
475
476
477
478
479
480
481
482
483
484
485
486
487
488
489
490
491
492
493
494
495
496
497
498
499
500
501
502
503
504
505
506
507
508
509
510
511
512
513
514
515
516
517
518
519
520
521
522
523
524
525
526
527
528
529
530
531
532
533
534
535
536
537
538
539
540
541
542
543
544
545
546
547
548
549
550
551
552
553
554
555
556
557
558
559
560
561
562
563
564
565
566
567
568
569
570
571
572
573
574
575
576
577
578
579
580
581
582
583
584
585
586
587
588
589
590
591
592
593
594
595
596
597
598
599
600
601
602
603
604
605
606
607
608
609
610
611
612
613
614
615
616
617
618
619
620
621
622
623
624
625
626
627
628
629
630
631
632
633
634
635
636
637
638
639
640
641
642
643
644
645
646
647
648
649
650
651
652
653
654
655
656
657
658
659
660
661
662
663
664
665
666
667
668
669
670
671
672
673
674
675
676
677
678
679
680
681
682
683
684
685
686
687
688
689
690
691
692
693
694
695
696
697
698
699
700
701
702
703
704
705
706
707
708
709
710
711
712
713
714
715
716
717
718
719
720
721
722
723
724
725
726
727
728
729
730
731
732
733
734
735
736
737
738
739
740
741
742
743
744
745
746
747
748
749
750
751
752
753
754
755
756
757
758
759
760
761
762
763
764
765
766
767
768
769
770
771
772
773
774
775
776
777
778
779
780
781
782
783
784
785
786
787
788
789
790
791
792
793
794
795
796
797
798
799
800
801
802
803
804
805
806
807
808
809
810
811
812
813
814
815
816
817
818
819
820
821
822
823
824
825
826
827
828
829
830
831
832
833
834
835
836
837
838
839
840
841
842
843
844
845
846
847
848
849
850
851
852
853
854
855
856
857
858
859
860
861
862
863
864
865
866
867
868
869
870
871
872
873
874
875
876
877
878
879
880
881
882
883
884
885
886
887
888
889
890
891
892
893
894
895
896
897
898
899
900
901
902
903
904
905
906
907
908
909
910
911
912
913
914
915
916
917
918
919
920
921
922
923
924
925
926
927
928
929
930
931
932
933
934
935
936
937
938
939
940
941
942
943
944
945
946
947
948
949
950
951
952
953
954
955
956
957
958
959
960
961
962
963
964
965
966
967
968
969
970
971
972
973
974
975
976
977
978
979
980
981
982
983
984
985
986
987
988
989
990
991
992
993
994
995
996
997
998
999
1000
```



ASM by example (3)

The screenshot shows the Eclipse IDE with the following components:

- Package Explorer:** Shows the project structure with folders for 'ASM_exercises', 'RemoteSystemsTempFiles', and 'ThesisProject'.
- Editor:** Displays the file 'mybank2_4_noAgent.asm' with the following code:

```
188
189
190
191  /* CALLBACK FUNCTION RULE
192  */
193
194  rule r_Callback_bank1 =
195  let ($cl = current_layer) in
196  if executing_function($cl) != DEPOSIT and executing
197  switch instruction_pointer($cl)
198  case 0 :
199      r_Ret[]
200  endswitch
201  endif
202  endlet
203
204
205
206
207
208
209
210
211
212
213  /* -----ATTACKER CONTI
214  */
215  /* CALLBACK FUNCTION RULE
216  */
217  rule r_Callback_bank2 =
218  let ($cl = current_layer) in
219  switch instruction_pointer($cl)
220  case 0 :
221      if counter < 2 then
222          par
223              r_Transaction[ATTACKER, DAO, 1, WITHDRAW]
224              counter := counter + 1
225          endpar
226      else
227          r_Transaction[ATTACKER, DAO, 1, CALLBACK]
228      endif
229  case 1 :
230      r_Ret[]
231  endswitch
232  endlet
233
234
235
236
237
238  /* -----MAINS AND INVARIANTS----- */
239
240
241
242
```
- Console:** Shows the Asmeta console output:

```
Asmeta console
-----
INVARIANT violations
FINAL STATE: amount(1)=1
amount(2)=1
amount(3)=1
amount(4)=1
amount(5)=1
amount(6)=1
balance(ATTACKER)=2
balance(DAO)=3
balance(USER)=1
costumer_balance(ATTACKER)=-1
counter=2
current_layer=1
executing_contract(1)=CONTRACTATTACKER
executing_contract(2)=CONTRACTDAO
executing_contract(3)=CONTRACTATTACKER
executing_contract(4)=CONTRACTDAO
executing_contract(5)=CONTRACTATTACKER
executing_contract(6)=CONTRACTDAO
executing_function(1)=CALLBACK
executing_function(2)=WITHDRAW
executing_function(3)=CALLBACK
executing_function(4)=WITHDRAW
```

VALIDATION

```
/*
 * INVARIANT
 */

invariant over costumer_balance : costumer_balance(ATTACKER) >= 0

/*
 * CTLSPEC
 */
CTLSPEC ef(costumer_balance(ATTACKER) < 0)
```



ASM by example (3)

```
Hyper Shell Edit View Tools Window Help
EclipseWorkspace - ThesisProject/ThesisProject/DAO Attack/mybank2_4_noAgent.asm - Eclipse IDE
~/DJUM/N/bin
$ cd Documents/UNI/ModellazioneSistemi/NuSMV-2.6.0-Darwin/bin/
$ ./NuSMV -dynamic ../../ModellazioneSistemi/mybank2_4_noAgent.smv
*** This is NuSMV 2.6.0 (compiled on Wed Oct 14 15:32:58 2015)
*** Enabled addons are: compass
*** For more information on NuSMV see <http://nusmv.fbk.eu>
*** or email to <nusmv-users@list.fbk.eu>.
*** Please report bugs to <Please report bugs to <nusmv-users@fbk.eu>>

*** Copyright (c) 2010-2014, Fondazione Bruno Kessler

*** This version of NuSMV is linked to the CUDD library version 2.4.1
*** Copyright (c) 1995-2004, Regents of the University of Colorado

*** This version of NuSMV is linked to the MiniSat SAT solver.
*** See http://minisat.se/MiniSat.html
*** Copyright (c) 2003-2006, Niklas Een, Niklas Sorensson
*** Copyright (c) 2007-2010, Niklas Sorensson

-- specification EF costumer_balance_ATTACKER < 0 is true
-- specification AG costumer_balance_ATTACKER >= 0 is false
-- as demonstrated by the following execution sequence
Trace Description: CTL Counterexample
Trace Type: Counterexample
-> State: 1.1 <-
  amount_0 = -2147483647
  amount_1 = -2147483647
  amount_2 = -2147483647
  amount_3 = -2147483647
  amount_4 = -2147483647
  amount_5 = -2147483647
  amount_6 = -2147483647
  amount_7 = -2147483647
  amount_8 = -2147483647
```

VERIFICATION

```
/*
 * INVARIANT
 */
invariant over costumer_balance : costumer_balance(ATTACKER) >= 0

/*
 * CTLSPEC
 */
CTLSPEC ef(costumer_balance(ATTACKER) < 0)
```



Future Works

- Generalize the approach
 - Model the full semantics of SC
 - Identify common patterns in the structure of the contract to build an ASM library
 - Build a catalog of common vulnerabilities and express it in terms of properties to check
 - A GUI for specification of verified smart contracts?
 - What about other blockchains?

