Tutorial on Apalache

[apalache-mc.org]

igor@konnov.phd

TLA$^+$ Community Meeting 2024

Milan, Sep 10, 2024

2016 2017 2018 2019 2020 2021 2022 2023 2024

TU WIEN

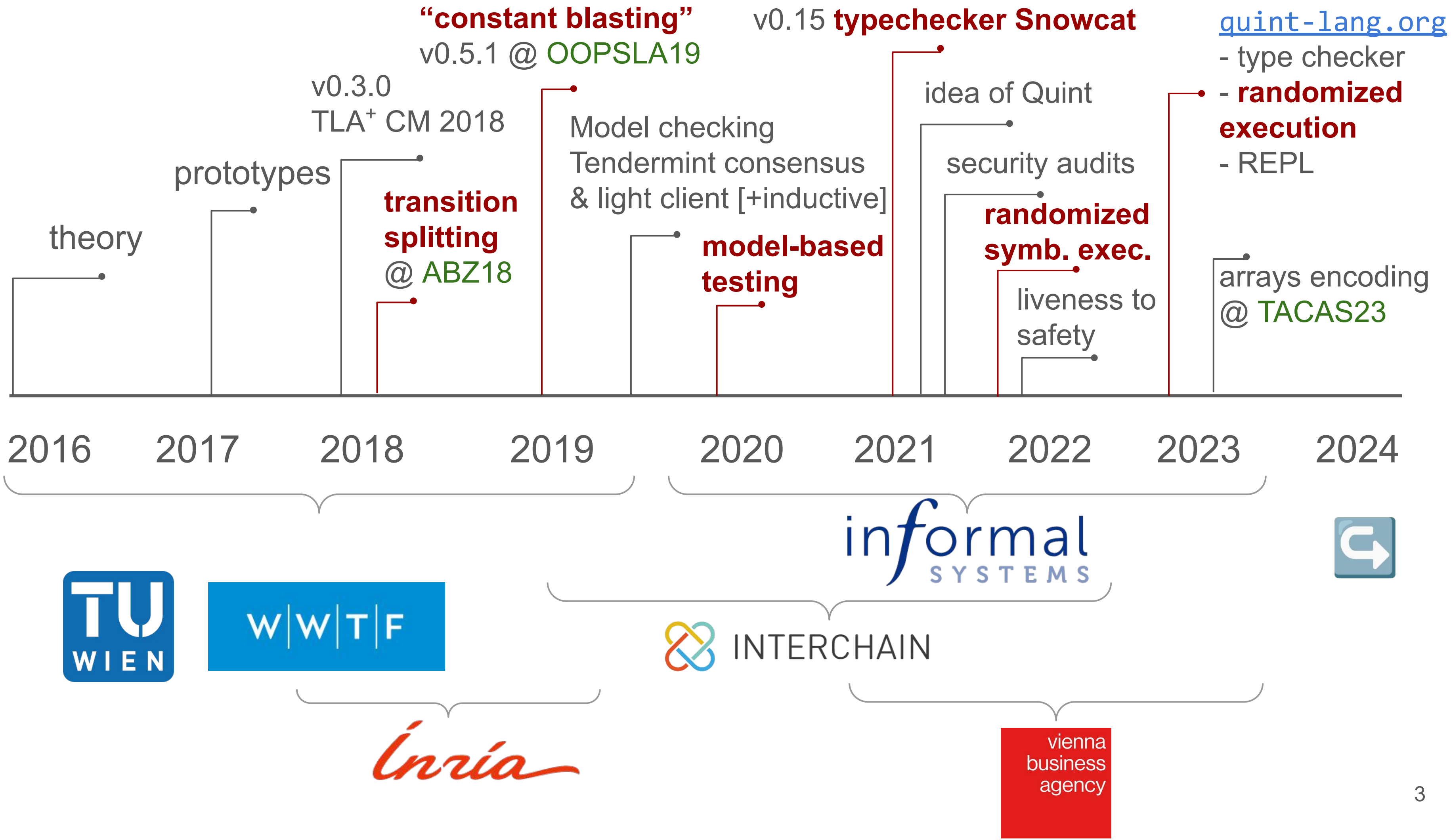
WWTF

Inria

informal SYSTEMS

INTERCHAIN

vienna business agency


theory
prototypes
v0.3.0
TLA+ CM 2018
transition splitting
@ ABZ18
"constant blasting"
v0.5.1 @ OOPSLA19
Model checking
Tendermint consensus
& light client [+inductive]
model-based testing
v0.15 typechecker Snowcat
idea of Quint
security audits
randomized symb. exec.
liveness to safety
quint-lang.org
- type checker
- randomized execution
- REPL
arrays encoding
@ TACAS23
2016
2017
2018
2019
2020
2021
2022
2023
2024
informal SYSTEMS
TU WIEN
W|W|T|F
INTERCHAIN
Inria
vienna business agency

2024: independent researchers


Jure Kukovec


Igor Konnov
@konnov


Thomas Pani


Andrey Kuprianov

[protocols-made-fun.com]

code4rena

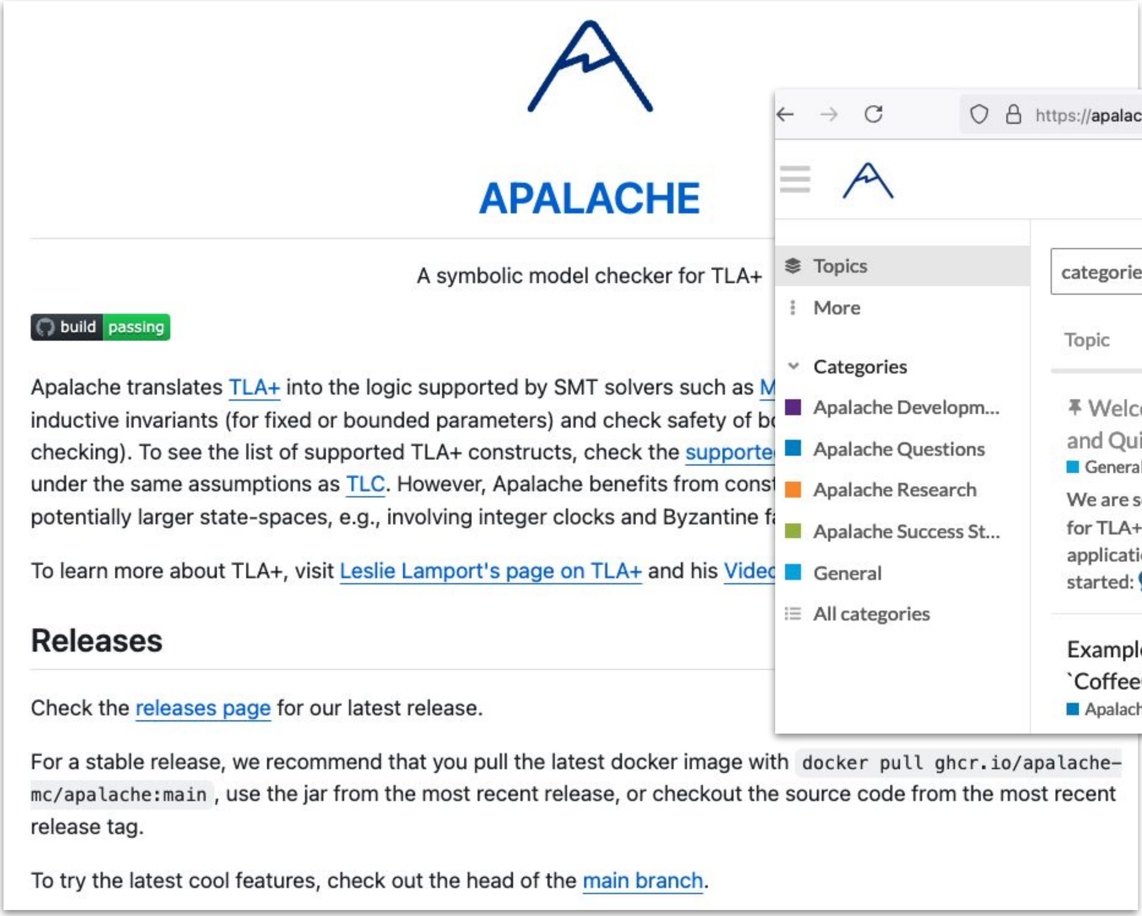
Stellar Community Fund

ethereum foundation

Matter Labs

SHERLOCK

[github.com/apalache-mc/apalache]


APALACHE

A symbolic model checker for TLA+

build passing

Apalache translates TLA+ into the logic supported by SMT solvers such as M
inductive invariants (for fixed or bounded parameters) and check safety of bo
checking). To see the list of supported TLA+ constructs, check the supporte
under the same assumptions as TLC. However, Apalache benefits from const
potentially larger state-spaces, e.g., involving integer clocks and Byzantine fa

To learn more about TLA+, visit Leslie Lamport's page on TLA+ and his Video

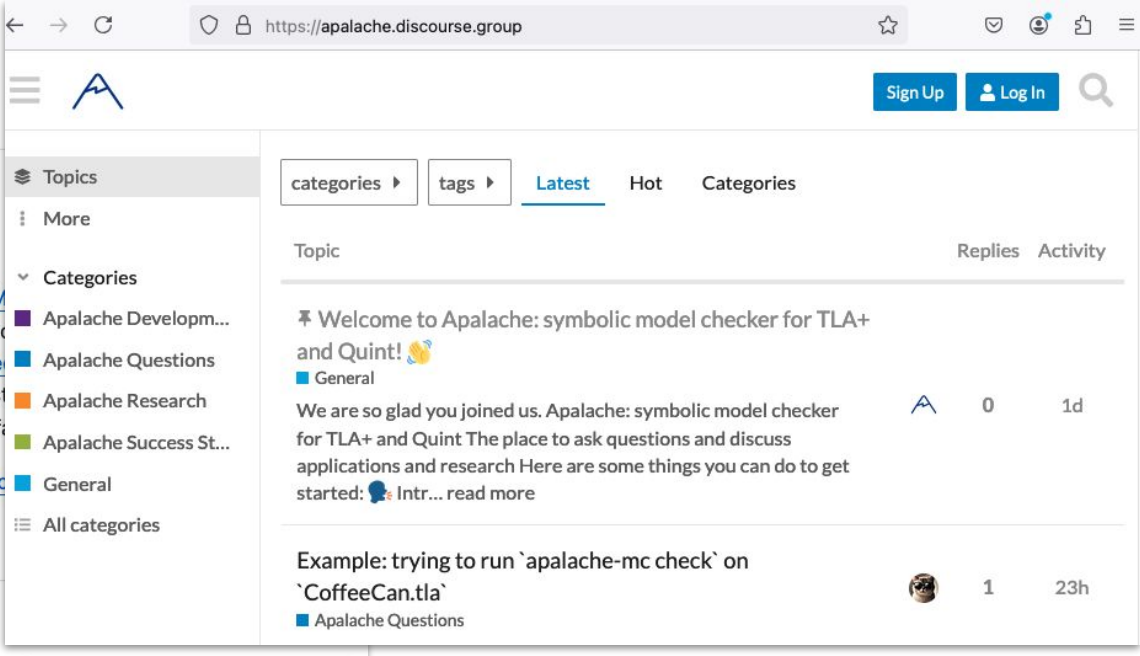
Releases

Check the releases page for our latest release.

For a stable release, we recommend that you pull the latest docker image with `docker pull ghcr.io/apalache-mc/apalache:main`, use the jar from the most recent release, or checkout the source code from the most recent release tag.

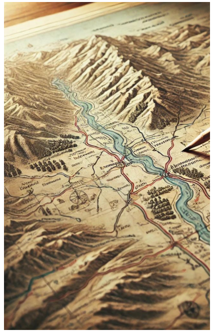
To try the latest cool features, check out the head of the main branch.

[apalache.discourse.group]



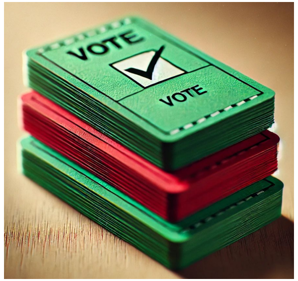
1. explain our “constant blasting”

2. highlight the differences between:



- bounded model checking in Apalache
- randomized symbolic execution in Apalache
- inductive checking in Apalache

3. …using a realistic example closer to the end



Constant blasting

Translation to SMT

APALACHE: model checker for TLA^+

- a symbolic model checker
- parameterized verification

Jure Kukovec Thanh-Hai Tran Marijana Lazić Josef Widder

TLA^+ Model Checking Made Symbolic [OOPSLA'19]

Mimic the semantics implemented by TLC – explicit model checker

Compute **layout of data structures**, constrain **contents with SMT**

Define operational semantics via reduction rules – for bounded data structures

Trade efficiency for expressivity

Trivial example

```
----------------- MODULE t ---------------------
EXTENDS Integers
VARIABLE
  \* @type: Int;
  x
Init ≜ x ∈ 0..2
Next ≜ x' = x - 1
Inv ≜ x ≥ 0
================================================
```


➜ apalache-examples git:(main) ✗

```
$ apalache-mc check --write-intermediate=true --debug --inv=Inv t.tla
...
PASS #13: BoundedChecker
State 0: Checking 1 state invariants
State 0: state invariant 0 holds.
Step 0: picking a transition out of 1 transition(s)
State 1: Checking 1 state invariants
State 1: state invariant 0 violated.
```

```
------------------ MODULE t -----------------------
EXTENDS Integers
VARIABLE
  \* @type: Int;
  x
Init ≜ x ∈ 0..2
Next ≜ x' = x - 1
Inv ≜ x ≥ 0
===================================================
```

```
;; declare cell($C$0): CellTFrom(Bool)
(declare-const $C$0 Bool)
(assert (= $C$0 false))
;; declare cell($C$1): CellTFrom(Bool)
(declare-const $C$1 Bool)
(assert (= $C$1 true))
;; declare cell($C$2): CellTFrom(Set(Bool))
(declare-sort Cell_Sb 0)
(declare-const $C$2 Cell_Sb)
;; declare cell($C$3): InfSet[CellTFrom(Int)]
(declare-sort Cell_Zi 0)
(declare-const $C$3 Cell_Zi)
;; declare cell($C$4): InfSet[CellTFrom(Int)]
(declare-const $C$4 Cell_Zi)
;; assert ¬$C$0
(assert (not false))
;; assert $C$1
(assert true)
;; declare edge predicate in_b0_Sb2: Bool
(declare-const in_b0_Sb2 Bool)
;; declare edge predicate in_b1_Sb2: Bool
(declare-const in_b1_Sb2 Bool)
;; assert Apalache!StoreInSet($C$0, $C$2)
(assert in_b0_Sb2)
;; assert Apalache!StoreInSet($C$1, $C$2)
(assert in_b1_Sb2)
```

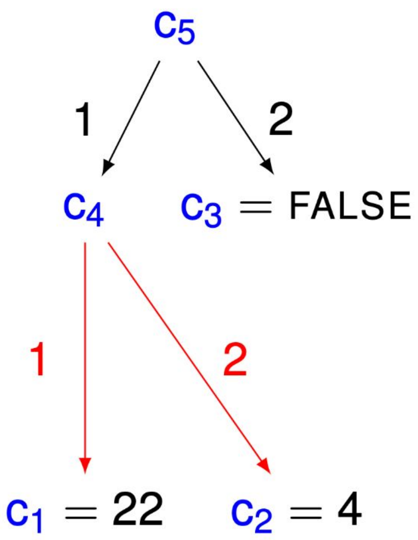
```
; ------- STEP: 0, SMT LEVEL: 2 TRANSITION: 0 {
; QuantRule(Apalache!Skolem(∃t_1$1 ∈ (0 .. 2): (Apalache!:=(x
; skolemizable existential t_1$1 over 0..2
;; declare cell($C$5): CellTFrom(Int)
(declare-const $C$5 Int)
; IntCmpRule($C$5 ≥ 0) {
;; declare cell($C$6): CellTFrom(Bool)
(declare-const $C$6 Bool)
;; assert $C$6 = ($C$5 ≥ 0)
(assert (= $C$6 (>= $C$5 0)))
; } IntCmpRule returns $C$6 [7 arena cells])
;; assert $C$6
(assert $C$6)
; IntCmpRule($C$5 ≤ 2) {
;; declare cell($C$7): CellTFrom(Bool)
(declare-const $C$7 Bool)
;; assert $C$7 = ($C$5 ≤ 2)
(assert (= $C$7 (<= $C$5 2)))
; } IntCmpRule returns $C$7 [8 arena cells])
;; assert $C$7
(assert $C$7)
; AssignmentRule(Apalache!:=(x', t_1$1)) {
; SubstRule(t_1$1) {
; } SubstRule returns $C$5 [8 arena cells])
; } AssignmentRule returns $C$1 [8 arena cells])
; } QuantRule returns $C$1 [8 arena cells])
(push) ;; becomes 3
;; assert $C$1
(assert true)
(check-sat)
;; sat = SATISFIABLE
```

```
; IntCmpRule(x < 0) {
;; declare cell($C$8): CellTFrom(Bool)
(declare-const $C$8 Bool)
;; assert $C$8 = ($C$5 < 0)
(assert (= $C$8 (< $C$5 0)))
; } IntCmpRule returns $C$8 [9 arena cells])
;; assert $C$8
(assert $C$8)
(check-sat)
;; sat = UNSATISFIABLE
```

```
; ------- STEP: 1, SMT LEVEL: 3 TRANSITION: 0 {
; AssignmentRule(Apalache!:=(x', x - 1)) {
; IntArithRule(x - 1) {
;; declare cell($C$10): CellTFrom(Int)
(declare-const $C$10 Int)
;; assert $C$10 = ($C$5 - 1)
(assert (= $C$10 (- $C$5 1)))
; } IntArithRule returns $C$10 [11 arena cells])
; } AssignmentRule returns $C$1 [11 arena cells])
(push) ;; becomes 4
;; assert $C$1
(assert true)
(check-sat)
;; sat = SATISFIABLE
(push) ;; becomes 5
; IntCmpRule(x < 0) {
;; declare cell($C$11): CellTFrom(Bool)
(declare-const $C$11 Bool)
;; assert $C$11 = ($C$10 < 0)
(assert (= $C$11 (< $C$10 0)))
; } IntCmpRule returns $C$11 [12 arena cells])
;; assert $C$11
(assert $C$11)
(check-sat)
;; sat = SATISFIABLE
```

Static picture of TLA$^+$ values and relations between them

Arena:



SMT:

integer `sort Int`

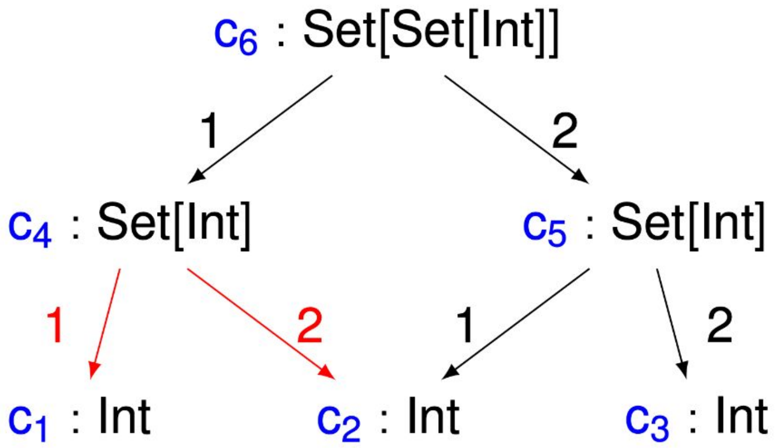
Boolean `sort Bool`

name, e.g., `"abc"`, uninterpreted sort

finite set:

- a constant c of uninterpreted sort set_τ
- propositional constants for members $in_{\langle c_1, c \rangle}, \ldots, in_{\langle c_n, c \rangle}$

Arenas for sets: { { 1, 2 }, { 2, 3} }



SMT defines the contents, e.g., to get $\{\{1\}, \{2\}\}$:

$$in_{\langle c_1, c_4 \rangle} \land \neg in_{\langle c_2, c_4 \rangle} \land in_{\langle c_2, c_5 \rangle} \land \neg in_{\langle c_3, c_5 \rangle}$$

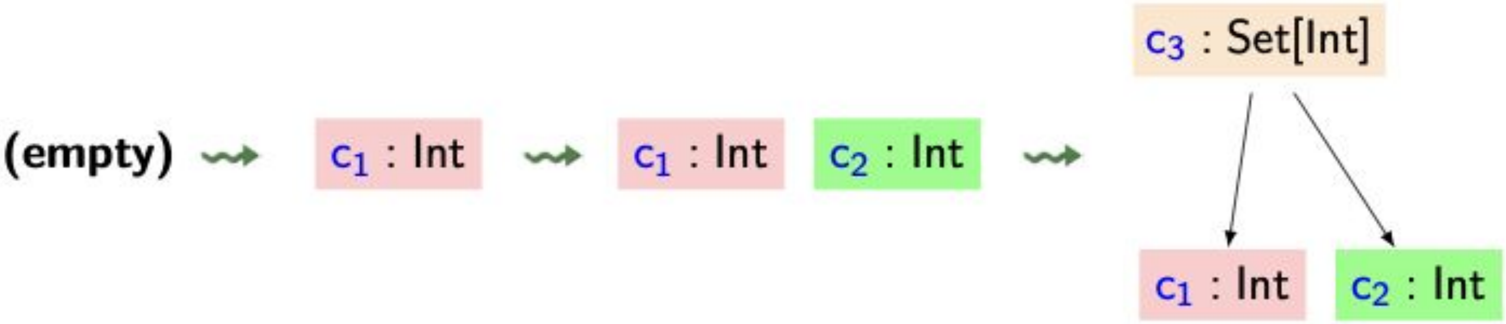
Arenas

- Directed acyclic graphs
- Nodes represent symbolic values of TLA^+ expressions
- Edges represent **potential membership**

Rewriting the set construction

$$\{1, 2\} \leadsto \{c_1, 2\} \leadsto \{c_1, c_2\} \leadsto c_3$$

Corresponding arena

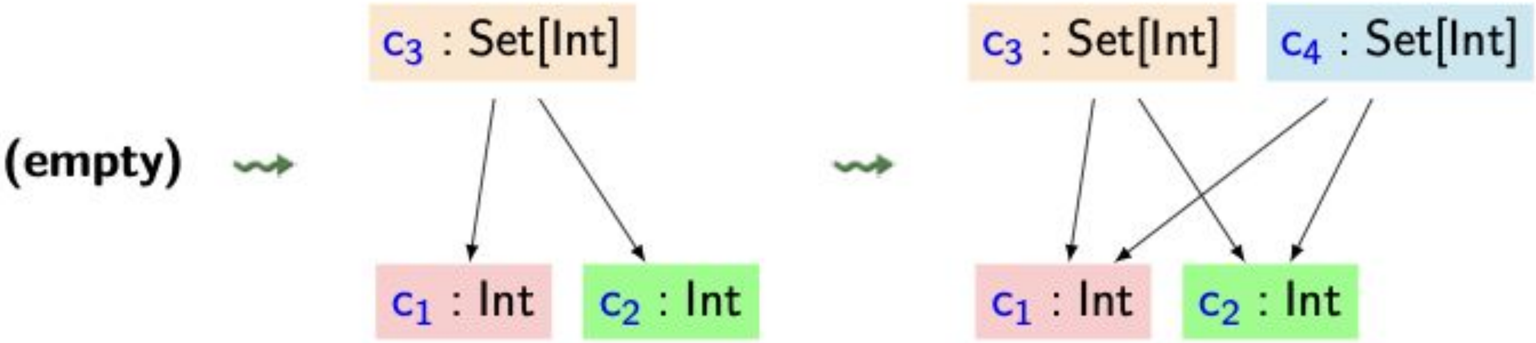


Rewriting and arenas

Rewriting the set filtering

$$\{x \in \{\boxed{1}, \boxed{2}\} : p(x)\} \rightsquigarrow \{x \in \boxed{c_3} : p(x)\} \rightsquigarrow \boxed{c_4}$$

Corresponding arena



SMT constraints

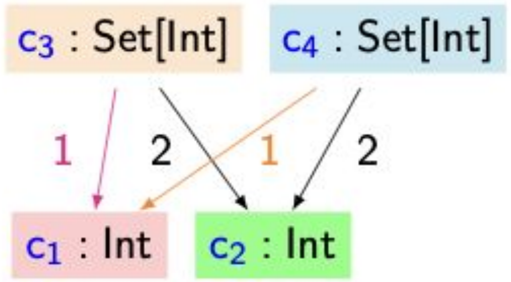
Generated constraints

$$en\langle c_4, 1, c_1\rangle \Leftrightarrow en\langle c_3, 1, c_1\rangle \wedge c_5 = \text{true}$$

$$en\langle c_4, 2, c_2\rangle \Leftrightarrow en\langle c_3, 2, c_2\rangle \wedge c_6 = \text{true}$$

Set filtering

$\{x \in \{\boxed{1}, \boxed{2}\} : p(x)\}$



$p(1) \rightsquigarrow c_5 : \text{Bool}$

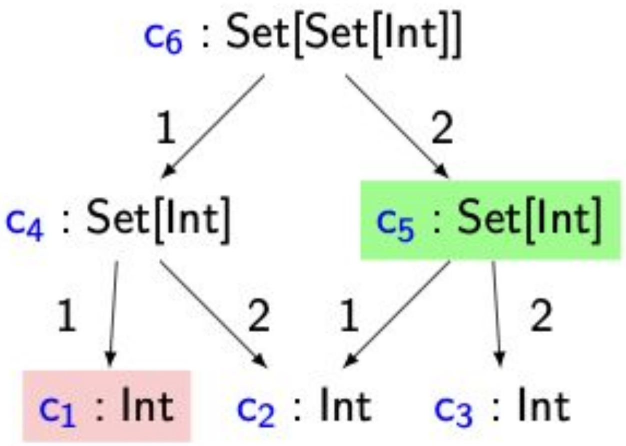
$p(2) \rightsquigarrow c_6 : \text{Bool}$

Arena	SMT encoding
Node	Uninterpreted constant
Edge	Unique Boolean constant

Nested sets

Nested sets

$\{\ \{\boxed{1}, 2\}, \boxed{\{2, 3\}}\ \}$



SMT constraints

$$
\begin{aligned}
& \mathit{en}\langle c_6, 1, c_4\rangle \wedge \mathit{en}\langle c_6, 1, c_5\rangle \\
\wedge\ & \mathit{en}\langle c_5, 1, c_2\rangle \wedge \mathit{en}\langle c_5, 2, c_3\rangle \\
\wedge\ & \mathit{en}\langle c_4, 1, c_1\rangle \wedge \mathit{en}\langle c_4, 1, c_2\rangle \\
\wedge\ & c_1 = 1 \wedge c_2 = 2 \wedge c_3 = 3
\end{aligned}
$$

[8009 arena cells] vs. [6009 arena cells]

```
-------- MODULE t2 --------
EXTENDS Integers
VARIABLE
  \* @type: Set(Int);
  S

Init ≜
  S = {}

Next ≜
  ∃ i ∈ ℤ:
    ∧ 0 ≤ i ∧ i ≤ 5
    ∧ S' = S ∪ { i }

Inv ≜
  2000 ∉ S
============================
```

```
$ hyperfine "apalache-mc check
--length=1000 --inv=Inv t2.tla"
...
```

Time (**mean** ± σ): **120.660 s** ± 0.690 s

```
$ hyperfine "apalache-mc check
--length=1000 --inv=Inv t3.tla"
...
```

Time (**mean** ± σ): **129.939 s** ± 4.982 s

```
-------- MODULE t3 --------
EXTENDS Integers
VARIABLE
  \* @type: Set(Int);
  S

Init ≜
  S = {}

Next ≜
  ∃ i ∈ 0..5:
    S' = S ∪ { i }

Inv ≜
  2000 ∉ S
============================
```

Encoding with SMT arrays

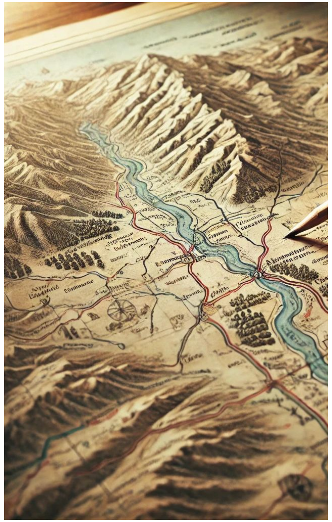
- Using SMT arrays for TLA$^+$ sets and functions: QF_AUFNIA

- Rodrigo Otoni, IK, J. Kukovec, P. Eugster, N. Sharygina

- Working faster on classical fault-tolerant algorithms

Symbolic Model Checking for TLA+ Made Faster [TACAS'23]

Exploration techniques



Bounded model checking

Input: Init, Next and Inv

0. $\text{Init} \wedge \neg\text{Inv}$
1. $(\text{Init} \cdot \text{Next}) \wedge \neg\text{Inv}'$
2. $(\text{Init} \cdot \text{Next} \cdot \text{Next}) \wedge \neg\text{Inv}'$

…

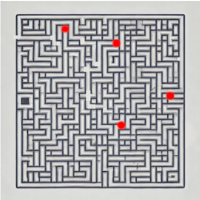
k. $(\text{Init} \cdot \text{Next} \cdot \ldots \cdot \text{Next}) \wedge \neg\text{Inv}'$

k is the bound

Backend:

SMT
(Microsoft z3)

Example: 2D labyrinth



```
CONSTANT
    \* The maximal x-coordinate.
    \* @type: Int;
    MAX_X,
    \* The maximal y-coordinate.
    \* @type: Int;
    MAX_Y,
    \* The set of walls.
    \* @type: Set(<<Int, Int>>);
    WALLS,
    \* The goal coordinates.
    \* @type: <<Int, Int>>;
    GOAL

VARIABLES
    \* @type: Int;
    x,
    \* @type: Int;
    y

Init ≜ x = 0 ∧ y = 0
```

```
Left ≜
    ∃ d ∈ 1 .. MAX_X:
        ∧ x − d ≥ 0 ∧ x' = x − d
        ∧ ∀ i ∈ 1..MAX_X:
            (i ≤ d) ⇒ ⟨x − i, y⟩ ∉ WALLS
        ∧ UNCHANGED y
```

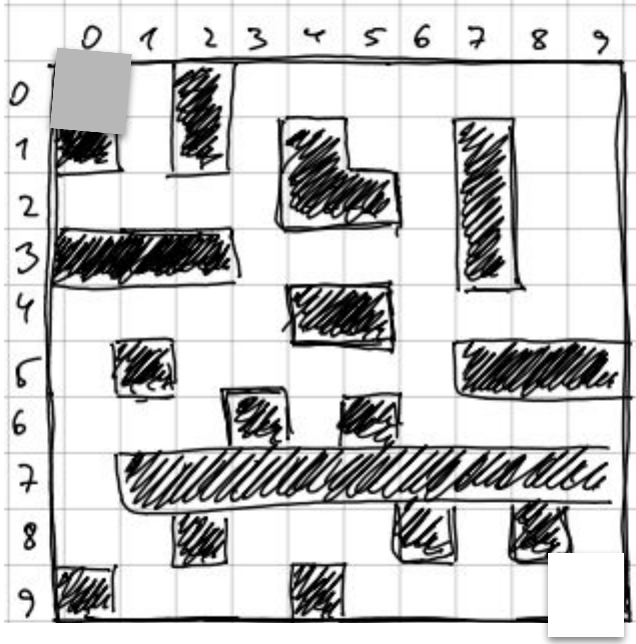
```
Down ≜
    ∃ d ∈ 1 .. MAX_Y:
        ∧ y + d ≤ MAX_Y ∧ y' = y + d
        ∧ ∀ j ∈ 1..MAX_Y:
            (j ≤ d) ⇒ ⟨x, y + j⟩ ∉ WALLS
        ∧ UNCHANGED x

Next ≜ Left ∨ Right ∨ Up ∨ Down
```

```
\* Check this invariant to find a path to the goal:
\* apalache-mc check --length=20 --inv=GoalInv MC_labyrinth_10x10.tla
GoalInv ≜ ⟨x, y⟩ ≠ GOAL

\* Check this invariant to make sure that we do not walk through walls:
\* apalache-mc check --length=20 --inv=NoWallInv MC_labyrinth_10x10.tla
NoWallInv ≜ ⟨x, y⟩ ∉ WALLS
```

Model checking instance

```
------ MODULE MC_labyrinth_10x10 ----
MAX_X ≜ 9
MAX_Y ≜ 9
\* @type: <<Int, Int>>;
GOAL ≜ ⟨9, 9⟩
\* @type: Set(<<Int, Int>>);
WALLS ≜ {
        ⟨0, 1⟩, ⟨2, 0⟩, ⟨2, 1⟩,
        ⟨4, 1⟩, ⟨4, 2⟩, ⟨5, 2⟩,
        ⟨0, 3⟩, ⟨1, 3⟩, ⟨2, 3⟩,
        ⟨4, 4⟩, ⟨5, 4⟩, ⟨1, 5⟩,
        ⟨7, 1⟩, ⟨7, 2⟩, ⟨7, 3⟩,
        ⟨7, 5⟩, ⟨8, 5⟩, ⟨9, 5⟩,
        ⟨3, 6⟩, ⟨5, 6⟩,
        ⟨1, 7⟩, ⟨2, 7⟩, ⟨3, 7⟩,
        ⟨4, 7⟩, ⟨5, 7⟩, ⟨6, 7⟩,
        ⟨7, 7⟩, ⟨8, 7⟩, ⟨9, 7⟩,
        ⟨2, 8⟩, ⟨6, 8⟩, ⟨8, 8⟩,
        ⟨0, 9⟩, ⟨4, 9⟩
}
```

```
VARIABLES
    \* @type: Int;
    x,
    \* @type: Int;
    y

INSTANCE labyrinth
====================================
```



Bounded model checking

$ apalache-mc **check --length=30** --inv=**GoalInv** MC_labyrinth_10x10.tla

```
PASS #13: BoundedChecker
State 0: Checking 1 state invariants
State 0: state invariant 0 holds.
Step 0: picking a transition out of 1 transition(s)
Step 1: Transition #0 is disabled
State 1: Checking 1 state invariants
State 1: state invariant 0 holds.
Step 1: Transition #2 is disabled
Step 1: Transition #3 is disabled
Step 1: picking a transition out of 1 transition(s)
State 2: Checking 1 state invariants
State 2: state invariant 0 holds.
Step 2: Transition #1 is disabled
Step 2: Transition #2 is disabled
State 2: Checking 1 state invariants
State 2: state invariant 0 holds.
Step 2: picking a transition out of 2 transition(s)
State 3: Checking 1 state invariants
State 3: state invaria
State 3: Checking 1 st
State 3: state invaria
State 3: Checking 1 st
State 3: state invariant 0 holds.
State 3: Checking 1 state invariants
State 3: state invariant 0 holds.
Step 3: picking a transition out of 4 transition(s)
```

```
Step 12: picking a transition out of 4 transition(s)
State 13: Checking 1 state invariants
State 13: state invariant 0 holds.
State 13: Checking 1 state invariants
Check the trace in: /Users/igor/devl/apalache-examples/
06-25_1791134631953794594l/violation1.tla, /Users/igor/
0x10.tla/2024-09-06T09-06-25_1791134631953794594l/MCvio
alache-out/MC_labyrinth_10x10.tla/2024-09-06T09-06-25_17
e-examples/labyrinth/_apalache-out/MC_labyrinth_10x10.t
on I@09:07:05.152
State 13: state invariant 0 violated.
Found 1 error(s)
The outcome is: Error
Checker has found an error
It took me 0 days  0 hours  0 min 39 sec
Total time: 39.830 sec
```

```
Benchmark 1: ~/devl/apalache/bin/apalache-mc check --length=30 --inv=GoalInv MC_labyrinth_10x10.tla
  Time (mean ± σ):     30.723 s ±  1.948 s    [User: 34.087 s, System: 0.418 s]
  Range (min … max):   26.730 s … 33.271 s    10 runs
```

Why slow down? ⏱️

Init

R

L ∨ **R** ∨ **U** ∨ **D**

L ∨ **R** ∨ **U** ∨ **D**

L ∨ **R** ∨ **U** ∨ **D**

L ∨ **R** ∨ **U** ∨ **D**

L ∨ **R** ∨ **U** ∨ **D**

Randomized symbolic execution

Init

R

L ∨ R ∨ U ∨ D

L ∨ R ∨ U ∨ D

L ∨ R ∨ U ∨ D

L ∨ R ∨ U ∨ D

L ∨ R ∨ U ∨ D

¬Invariant

[bounded model checking]

```
$ hyperfine -i -r 100 "apalache-mc check --length=30 --inv=GoalInv MC_labyrinth_10x10.tla"
```

```
Time (mean ± σ):     35.575 s ±  5.655 s    [User: 38.293 s, System: 0.623 s]
Range (min … max):   25.423 s … 52.707 s    100 runs
```

[randomized symbolic execution]

```
$ hyperfine -i -r 100 "apalache-mc simulate --length=50 --inv=GoalInv MC_labyrinth_10x10.tla"
```

```
Time (mean ± σ):     134.850 s ± 112.288 s    [User: 134.858 s, System: 3.582 s]
Range (min … max):    5.845 s … 406.693 s    100 runs
```

Randomized symbolic execution in parallel

```
$ seq 0 30 | parallel --delay 1 --halt now,fail=1
  apalache-mc simulate --out-dir=s/{} \
  --length=50 --inv=GoalInv MC_labyrinth_10x10.tla

----------------------------
Symbolic runs left: 99
State 38: state invariant 0 violated. ❌
Total time: 17.457 sec

$ hyperfine -i "[...above...]"
```

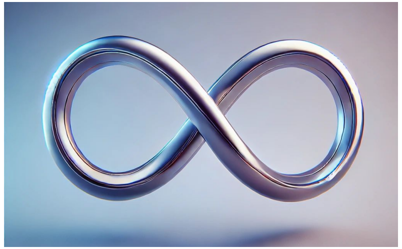
```
Time (mean ± σ):     40.474 s ± 21.095 s    [User: 31.413 s, System: 1.407 s]
Range (min … max):    9.508 s … 79.283 s    10 runs
```

32 cores, 128G RAM

Recall the figures for "check":

```
Time (mean ± σ):     35.575 s ±  5.655 s    [User: 38.293 s, System: 0.623 s]
Range (min … max):   25.423 s … 52.707 s    100 runs
```

Unbounded executions?



Inductive invariants

```
\* Check this invariant to find a path to the goal:
\* apalache-mc check --length=20 --inv=GoalInv MC_labyrinth_10x10.tla
GoalInv ≜ ⟨x, y⟩ ≠ GOAL

\* Check this invariant to make sure that we do not walk through walls:
\* apalache-mc check --length=20 --inv=NoWallInv MC_labyrinth_10x10.tla
NoWallInv ≜ ⟨x, y⟩ ∉ WALLS

\* A few definitions to check NoWallInv for arbitrary long executions.

TypeOK ≜
    ∧ x ∈ 0..MAX_X
    ∧ y ∈ 0..MAX_Y

\* 1. apalache-mc check --length=0 --init=IndInv --inv=NoWallInv MC_labyrinth_10x10.tla
\* 2. apalache-mc check --length=0 --init=Init --inv=IndInv MC_labyrinth_10x10.tla
\* 3. apalache-mc check --length=1 --init=IndInv --inv=IndInv MC_labyrinth_10x10.tla
IndInv ≜ TypeOK ∧ NoWallInv
```

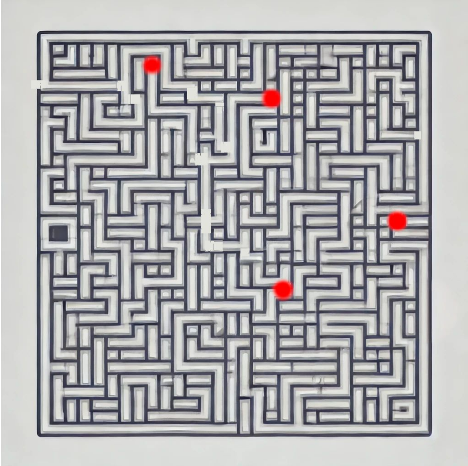
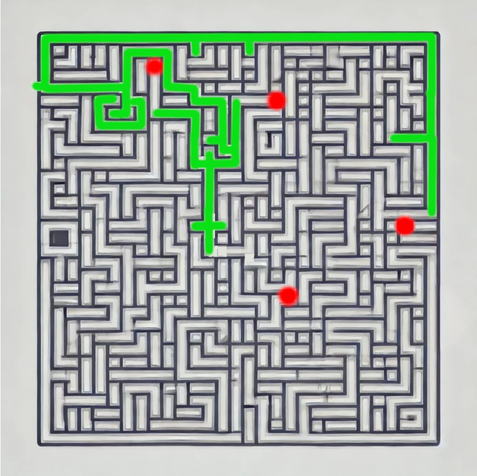
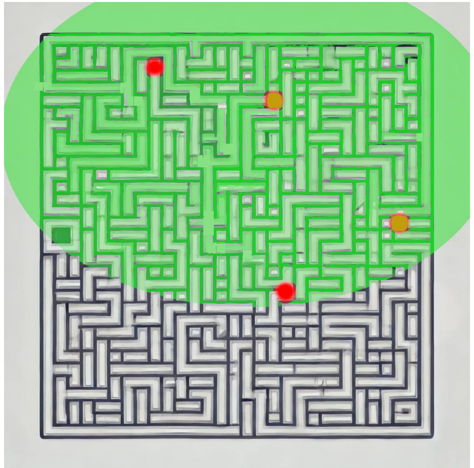
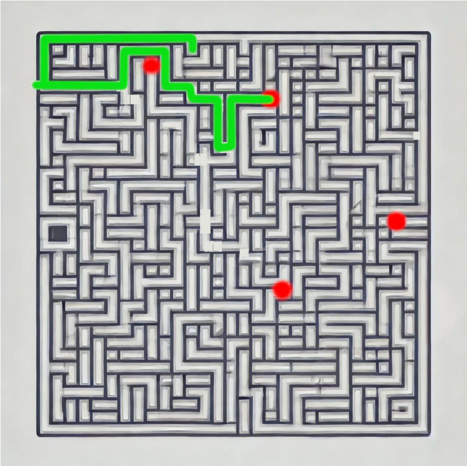
Gotchas with inductive invariants

- Apalache still needs bounded data structures in `Init`
- Unbounded integers are OK, but only a bounded number of them
- Disproving `TypeOK` is often too hard

- Apalache-specific hack: `Gen(k)` produces data structures of width ≤ k

TypeOK that is “too much”

```
TypeOK ≜
  ∧ value ∈ [ CORRECT → VALUES ]
  ∧ decision ∈ [ ALL → VALUES ∪ { NO_DECISION } ]
  ∧ round ∈ [ CORRECT → ROUNDS ]
  ∧ step ∈ [ CORRECT → { S1, S2, S3 } ]
  ∧ ∃ A1 ∈ SUBSET [ src: ALL, r: ROUNDS, v: VALUES ]:
        msgs1 = [ r ∈ ROUNDS ↦ { m ∈ A1: m.r = r } ]
  ∧ ∃ A1D ∈ SUBSET [ src: ALL, r: ROUNDS, v: VALUES ],
          A1Q ∈ SUBSET [ src: ALL, r: ROUNDS ]:
        msgs2 = [ r ∈ ROUNDS ↦
            { D2(mm.src, r, mm.v): mm ∈ { m ∈ A1D: m.r = r } }
                ∪ { Q2(mm.src, r): mm ∈ { m ∈ A1Q: m.r = r } }
        ]
```

Realistic example:

Ben-Or's Consensus

[github.com/konnov/apalache-examples]

Fault-tolerant distributed systems

Distributed

logically and geographically

Fault-tolerant

individual machines may crash and even act malicious

Safe and live

e.g., no double spending

every transaction is eventually committed

Properties of Distributed Consensus

A distributed algorithm for N replicas

every replica proposes a value $w \in V$

Termination

every correct replica eventually decides on a value $v \in V$

Agreement

if a replica decides on v, no replica decides on $V \setminus \{v\}$

Validity

if a replica decides on v, the value v was proposed earlier

B — Byzantine Protocol

Process P: Initial value x_P.

step 0: set $r := 1$.

step 1: Send the message $(1, r, x_P)$ to all the processes.

step 2: Wait till messages of type $(1, r, *)$ are received from $N - t$ processes. If more than $(N + t)/2$ messages have the same value v, then send the message $(2, r, v, D)$ to all processes. Else send the message $(2, r, ?)$ to all processes.

step 3: Wait till messages of type $(2, r, *)$ arrive from $N - t$ processes.

(a) If there are at least $t+1$ D-messages $(2, r, v, D)$, then set $x_P := v$.

(b) If there are more than $(N + t)/2$ D-messages then **decide** v.

(c) Else set $x_P = 1$ or 0 each with probability $\frac{1}{2}$.

step 4: Set $r := r + 1$ and go to step 1.

$n > 5 \cdot t \wedge t \geq f$

Another Advantage of Free Choice:
Completely Asynchronous Agreement Protocols

(Extended Abstract)

Michael Ben-Or [†]

Type definitions

```
------------------------------- MODULE typedefs --------------------------------
EXTENDS Variants

(*
 * Type definitions:
 *
 * Type-1 messages.
 * @typeAlias: msgA = { src: REPLICA, r: Int, v: Int };
 *
 * Type-2 messages.
 * @typeAlias: msgB = Q({ src: REPLICA, r: Int }) | D({ src: REPLICA, r: Int, v: Int });
 *)
typedefs_aliases ≜ TRUE

\* predefined constants for the steps
S1 ≜ "S1_OF_STEP"
S2 ≜ "S2_OF_STEP"
S3 ≜ "S3_OF_STEP"
```

Auxiliary type constructors

[typedefs.tla]

```
\* @type: (REPLICA, Int, Int) => $msgA;
M1(src, round, value) ≜ [ src ↦ src, r ↦ round, v ↦ value ]

\* @type: (REPLICA, Int) => $msgB;
Q2(src, round) ≜ Variant("Q", [ src ↦ src, r ↦ round ])

\* @type: $msgB => Bool;
IsQ2(msg) ≜ VariantTag(msg) = "Q"

\* @type: $msgB => { src: REPLICA, r: Int };
AsQ2(msg) ≜ VariantGetUnsafe("Q", msg)

\* @type: (REPLICA, Int, Int) => $msgB;
D2(src, round, value) ≜ Variant("D", [ src ↦ src, r ↦ round, v ↦ value ])

\* @type: $msgB => Bool;
IsD2(msg) ≜ VariantTag(msg) = "D"

\* @type: $msgB => { src: REPLICA, r: Int, v: Int };
AsD2(msg) ≜ VariantGetUnsafe("D", msg)
```

[preambule in Ben_or83.tla]

```
\* The set of values to choose from
VALUES == { 0, 1 }

CONSTANTS
    \* The total number of replicas.
    \* @type: Int;
    N,
    \* An upper bound on the number of faulty replicas.
    \* @type: Int;
    T,
    \* The actual number of faulty replicas (unknown to the replicas).
    \* @type: Int;
    F,
    \* The set of the correct (honest) replicas.
    \* @type: Set(REPLICA);
    CORRECT,
    \* The set of the Byzantine (faulty) replicas.
    \* @type: Set(REPLICA);
    FAULTY,
    \* The set of rounds, which we bound for model checking.
    \* @type: Set(Int);
    ROUNDS

ALL == CORRECT \union FAULTY
NO_DECISION == -1
```

[variables in Ben_or83.tla]

```
VARIABLES
    \* The current value by a replica, called $x_P$ in the paper.
    \* @type: REPLICA -> Int;
    value,
    \* The decision by a replica, where -1 means no decision.
    \* @type: REPLICA -> Int;
    decision,
    \* The round number of a replica, called $r$ in the paper.
    \* @type: REPLICA -> Int;
    round,
    \* The replica step: S1, S2, S3.
    \* @type: REPLICA -> STEP;
    step,
    \* Type-1 messages sent by the correct and faulty replicas, mapped by rounds.
    \* @type: Int -> Set($msgA);
    msgs1,
    \* Type-2 messages sent by the correct and faulty replicas, mapped by rounds.
    \* @type: Int -> Set($msgB);
    msgs2
```

[Ben_or83.tla]

```
Init ≜
  \* non-deterministically choose the initial values
  ∧ value ∈ [ CORRECT → VALUES ]
  ∧ decision = [ r ∈ CORRECT ↦ NO_DECISION ]
  ∧ round = [ r ∈ CORRECT ↦ 1 ]
  ∧ step = [ r ∈ CORRECT ↦ S1 ]
  ∧ msgs1 = [ r ∈ ROUNDS ↦ {}]
  ∧ msgs2 = [ r ∈ ROUNDS ↦ {}]
```

```
\* @type: REPLICA => Bool;
Step1(id) ≜
  LET r ≜ round[id] IN
  ∧ step[id] = S1
  \* "send the message (1, r, x_P) to all the processes"
  ∧ msgs1' = [msgs1 EXCEPT ![r] = @ ∪ { M1(id, r, value[id]) }]
  ∧ step' = [step EXCEPT ![id] = S2]
  ∧ UNCHANGED ⟨ value, decision, round, msgs2 ⟩
```

```
Step2(id) ≜
  LET r ≜ round[id] IN
  ∧ step[id] = S2
  ∧ ∃ received ∈ SUBSET msgs1[r]:
     \* "wait till messages of type (1, r, *) are received from N − T processes"
     ∧ Cardinality(Senders1(received)) ≥ N − T
     ∧ LET Weights ≜ [ v ∈ VALUES ↦
          Cardinality(Senders1({ m ∈ received: m.v = v })) ]
        IN
        ∨ ∃ v ∈ VALUES:
          \* "if more than (N + T)/2 messages have the same value v..."
          ∧ 2 * Weights[v] > N + T
          \* "...then send the message (2, r, v, D) to all processes"
          ∧ msgs2' = [msgs2 EXCEPT ![r] = @ ∪ { D2(id, r, v) }]
        ∨∧ ∀ v ∈ VALUES: 2 * Weights[v] ≤ N
          \* "Else send the message (2, r, ?) to all processes"
         ∧ msgs2' = [msgs2 EXCEPT ![r] = @ ∪ { Q2(id, r) }]
  ∧ step' = [ step EXCEPT ![id] = S3 ]
  ∧ UNCHANGED ⟨ value, decision, round, msgs1 ⟩
```

```
Step3(id) ≜
  LET r ≜ round[id] IN
  ∧ step[id] = S3
  ∧ ∃ received ∈ SUBSET msgs2[r]:
    \* "Wait till messages of type (2, r, *) arrive from N − T processes"
    ∧ Cardinality(Senders2(received)) ≥ N − T
    ∧ LET Weights ≜ [ v ∈ VALUES ↦
              Cardinality(Senders2({ m ∈ received: IsD2(m) ∧ AsD2(m).v = v })) ]
      IN
      ∨ ∃ v ∈ VALUES:
          \* "(a) If there are at least T+1 D-messages (2, r, v, D),
          \* then set x_P to v"
          ∧ Weights[v] ≥ T + 1
          ∧ value' = [value EXCEPT ![id] = v]
          \* "(b) If there are more than (N + T)/2 D-messages..."
          ∧ IF 2 * Weights[v] > N + T
              \* "...then decide v"
              THEN decision' = [decision EXCEPT ![id] = v]
              ELSE decision' = decision
      ∨ ∧ ∀ v ∈ VALUES: Weights[v] < T + 1
        ∧ ∃ next_v ∈ VALUES:
            \* "(c) Else set x_P = 1 or 0 each with probability 1/2."
            \* We replace probabilites with non-determinism.
            ∧ value' = [value EXCEPT ![id] = next_v]
            ∧ decision' = decision
    \* the condition below is to bound the number of rounds for model checking
    ∧ r + 1 ∈ ROUNDS
    \* "Set r := r + 1 and go to step 1"
    ∧ round' = [ round EXCEPT ![id] = r + 1 ]
    ∧ step' = [ step EXCEPT ![id] = S1 ]
    ∧ UNCHANGED ⟨ msgs1, msgs2 ⟩
```

```
FaultyStep ≜
    \* the faulty replicas collectively inject messages for a single round
    ∧ ∃ r ∈ ROUNDS:
        ∧ ∃ F1 ∈ SUBSET [ src: FAULTY, r: { r }, v: VALUES ]:
            msgs1' = [ msgs1 EXCEPT ![r] = @ ∪ F1 ]
        ∧ ∃ F2D ∈ SUBSET { D2(src, r, v): src ∈ FAULTY, v ∈ VALUES }:
            ∃ F2Q ∈ SUBSET { Q2(src, r): src ∈ FAULTY }:
                msgs2' = [ msgs2 EXCEPT ![r] = @ ∪ F2D ∪ F2Q ]
    ∧ UNCHANGED ⟨ value, decision, round, step ⟩

CorrectStep ≜
    ∃ id ∈ CORRECT:
        ∨ Step1(id)
        ∨ Step2(id)
        ∨ Step3(id)

Next ≜
    ∨ CorrectStep
    ∨ FaultyStep
```

```
\* No two correct replicas decide on different values
AgreementInv ≜
    ∀ id1, id2 ∈ CORRECT:
        ∨ decision[id1] = NO_DECISION
        ∨ decision[id2] = NO_DECISION
        ∨ decision[id1] = decision[id2]
```

Model checking instance

```
---------------------------------- MODULE MC_n6t1f1 ----------------------------------
EXTENDS Integers, typedefs

N == 6
T == 1
F == 1
CORRECT == {
    "0_OF_REPLICA", "1_OF_REPLICA", "2_OF_REPLICA", "3_OF_REPLICA", "4_OF_REPLICA"
}
FAULTY == {"5_OF_REPLICA"}
ROUNDS == 1..3
VARIABLES
  \* The current value by a replica, called $x_P$ in the paper.
  \* @type: REPLICA -> Int;
  value,
  \* The decision by a replica, where -1 means no decision.
  \* @type: REPLICA -> Int;
  decision,
  \* The round number of a replica, called $r$ in the paper.
  \* @type: REPLICA -> Int;
  round,
  \* The replica step: S1, S2, S3.
  \* @type: REPLICA -> STEP;
  step,
  \* Type-1 messages sent by the correct and faulty replicas, mapped by rounds.
  \* @type: Int -> Set($msgA);
  msgs1,
  \* Type-2 messages sent by the correct and faulty replicas, mapped by rounds.
  \* @type: Int -> Set($msgB);
  msgs2

INSTANCE Ben_or83
=====================================================================================
```

Checking “falsy” invariants

```
\* An example of a replica that has made a decision
DecisionEx ≜
    ¬(∃ id ∈ CORRECT: decision[id] ≠ NO_DECISION)

\* An example of all correct replicas having made a decision
AllDecisionEx ≜
    ¬(∀ id ∈ CORRECT: decision[id] ≠ NO_DECISION)
```

$ apalache-mc **check** --inv=**DecisionEx** \
--init=**Init** --next=**Next** MC_n6t1f1.tla

26 min 16 sec, 1.2G ❌

apalache-mc **simulate** --length=30 --inv=**AllDecisionEx** \
--init=**Init** --next=**Next** MC_n6t1f1.tla

1 min 9 sec, 1.4G ❌

$ apalache-mc **check** --inv=**DecisionEx** \
--init=**InitWithFaults** --next=**CorrectStep** MC_n6t1f1.tla

39 sec, 1.1G ❌

apalache-mc **simulate** --length=30 --inv=**AllDecisionEx** \
--init=**InitWithFaults** --next=**CorrectStep** MC_n6t1f1.tla

1 min 9 sec, 1.4G ❌

apalache-mc **check** --length=30 --inv=**AllDecisionEx** \
--init=**InitWithFaults** --next=**CorrectStep** MC_n6t1f1.tla

1 hour 55 min, 1.7G ❌

Injecting faults in the initial states

```
InitWithFaults ≜
  \* non-deterministically choose the initial values
  ∧ value ∈ [ CORRECT → VALUES ]
  ∧ decision = [ r ∈ CORRECT ↦ NO_DECISION ]
  ∧ round = [ r ∈ CORRECT ↦ 1 ]
  ∧ step = [ r ∈ CORRECT ↦ S1 ]
  \* non-deterministically initialize the messages with faults
  ∧ ∃ F1 ∈ SUBSET [ src: FAULTY, r: ROUNDS, v: VALUES ]:
        msgs1 = [ r ∈ ROUNDS ↦ { m ∈ F1: m.r = r } ]
  ∧ ∃ F1D ∈ SUBSET [ src: FAULTY, r: ROUNDS, v: VALUES ],
        F1Q ∈ SUBSET [ src: FAULTY, r: ROUNDS ]:
        msgs2 = [ r ∈ ROUNDS ↦
              { D2(mm.src, r, mm.v): mm ∈ { m ∈ F1D: m.r = r } }
                  ∪ { Q2(mm.src, r): mm ∈ { m ∈ F1Q: m.r = r } }
        ]
```

Main litmus test

Check AgreementInv for n = 6, f = 2

It must be ❌

```
$ seq 0 9 | parallel --delay 1 apalache-mc simulate --out-dir=o/{} \
  --length=30 --init=InitWithFaults --next=CorrectStep \
  --inv=AgreementInv MC_n6t1f2.tla
```

```
State 10: state invariant 0 violated.
Total time: 17.859 sec
```
❌

```
$ apalache-mc check --length=30 --init=InitWithFaults \
  --next=CorrectStep --inv=AgreementInv MC_n6t1f2.tla
```

```
State 10: state invariant 0 violated.
It took me 0 days  0 hours 11 min 39 sec
```
❌

Checking AgreementInv

Check AgreementInv for n = 6, t = 1, f = 1

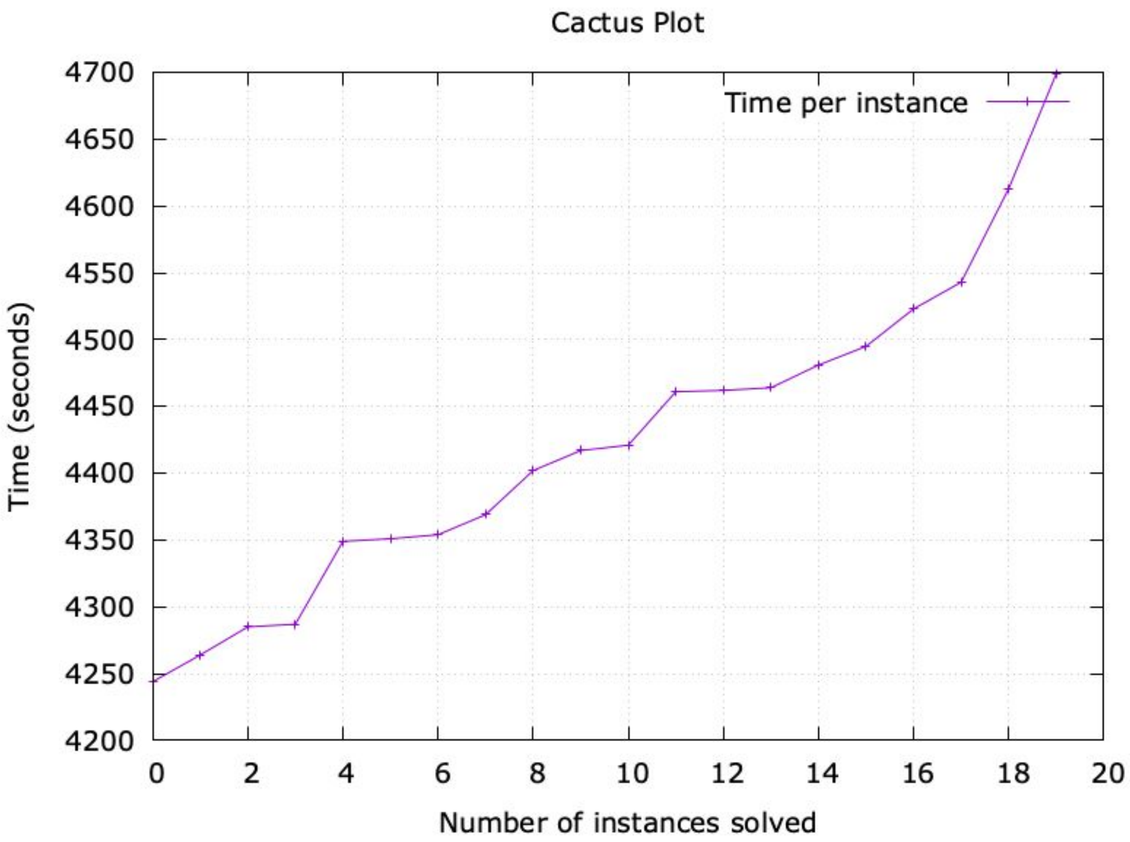
It must be ✅

```
$ apalache-mc check --length=18 --init=InitWithFaults \
  --next=CorrectStep --inv=AgreementInv MC_n6t1f1.tla
```

OK in 18h 53 min ✅

```
$ seq 0 19 | parallel --delay 1 --halt now,fail=1 \
  apalache-mc simulate –out-dir=s/{} --length=25 \
  --init=InitWithFaults --next=CorrectStep --inv=AgreementInv MC_n6t1f1.tla
```

OK in 3h - 3h 20 min ✅


Cactus Plot
Time per instance
Time (seconds)
Number of instances solved
4200
4250
4300
4350
4400
4450
4500
4550
4600
4650
4700
0
2
4
6
8
10
12
14
16
18
20

Inductive invariant?

```
IndInv ≜
  ∧ Lemma2_NoEquivocation1ByCorrect
  ∧ Lemma3_NoEquivocation2ByCorrect
  ∧ Lemma4_MessagesNotFromFuture
  ∧ Lemma5_RoundNeedsSentMessages
  ∧ Lemma6_DecisionDefinesValue
  ∧ Lemma7_D2RequiresQuorum
  ∧ Lemma8_Q2RequiresNoQuorum
  ∧ Lemma9_RoundsConnection
  ∧ Lemma10_M1RequiresQuorum
  ∧ Lemma11_ValueOnQuorum
  ∧ Lemma12_CannotJumpRoundsWithoutQuorum
  ∧ Lemma13_ValueLock
  \* this lemma is rather slow
  ∧ Lemma1_DecisionRequiresLastQuorum
```

```
TypeOK ≜
  ∧ value ∈ [ CORRECT → VALUES ]
  ∧ decision ∈ [ ALL → VALUES ∪ { NO_DECISION } ]
  ∧ round ∈ [ CORRECT → ROUNDS ]
  ∧ step ∈ [ CORRECT → { S1, S2, S3 } ]
  ∧ ∃ A1 ∈ SUBSET [ src: ALL, r: ROUNDS, v: VALUES ]:
        msgs1 = [ r ∈ ROUNDS ↦ { m ∈ A1: m.r = r } ]
  ∧ ∃ A1D ∈ SUBSET [ src: ALL, r: ROUNDS, v: VALUES ],
          A1Q ∈ SUBSET [ src: ALL, r: ROUNDS ]:
        msgs2 = [ r ∈ ROUNDS ↦
            { D2(mm.src, r, mm.v): mm ∈ { m ∈ A1D: m.r = r } }
              ∪ { Q2(mm.src, r): mm ∈ { m ∈ A1Q: m.r = r } }
        ]
```

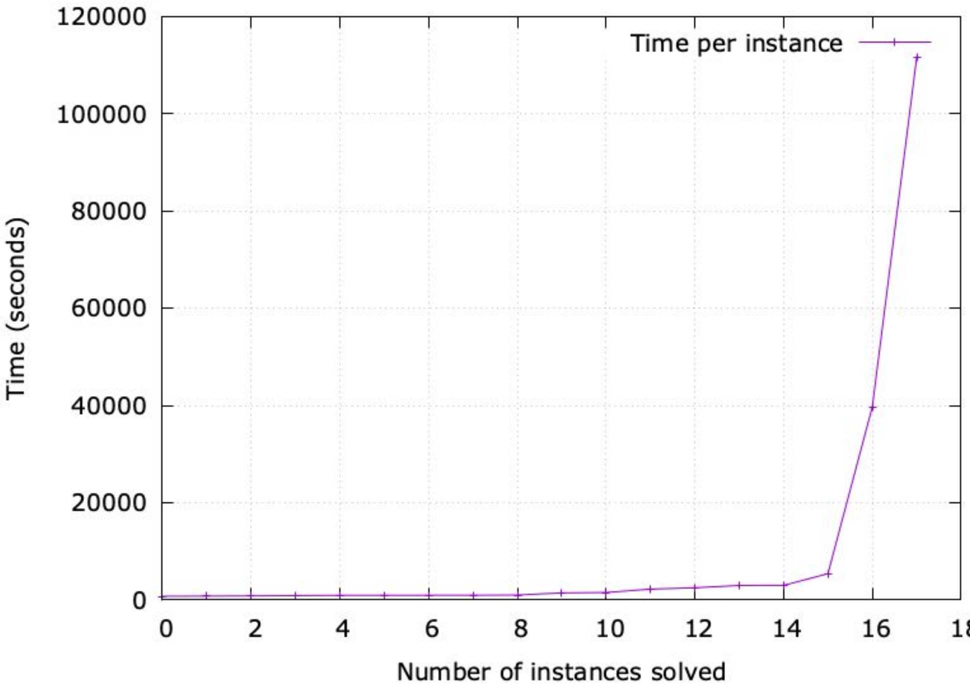
```
IndInit ≜
  ∧ TypeOK
  ∧ IndInv
```

[Ben_or83_inductive.tla]

Performance: n = 6, f = 1

- 13 lemmas, about 13h of my time
- dozens of attempts
- up to 20 model checker runs each
- **Challenge:** fix the lemmas by looking at the counterexamples
- not trying to write a proof on paper

1. $\mathit{Init} \Longrightarrow \mathit{IndInv}$: 13 sec
2. $\mathit{IndInit} \Longrightarrow \mathit{AgreementInv}$: 21 min
3. $\mathit{IndInit} \land \mathit{Next} \Longrightarrow \mathrm{IndInv}$



Fineprint

We have proven *AgreementInv* by induction for:

- 5 correct and 1 faulty replica
- 3 rounds

It’s not a parameterized proof!

Sufficient in practice?

💜⏳ ⏱️💰

Heavy-weight verification

- Complete functional verification
- Hoare-style proofs

Mid-weight verification

- Complete model checking
- Conformance testing

👆

- Model checking
- Stateless exploration
- Bug finding

Lightweight verification

- Static analysis
- Property-based testing

Conclusions

Directions

- Make “simulate” learn about good/bad executions – fuzzing
- Introduce a less explosive encoding, e.g., Alloy’s?
- Better support for parallel runs

- There is no company to dictate a roadmap, it’s truly open source

Jure Kukovec

Thanh-Hai Tran

Marijana Lazić

Josef Widder

TU WIEN

Jure Kukovec

Shon Feder

Gabriela Moreira
@bugarela

Igor Konnov
@konnov

Thomas Pani

Andrey Kupriyanov

Philip Offtermatt

Rodrigo Otoni

Inria

INTERCHAIN

informal SYSTEMS

[github.com/apalache-mc/apalache]

[apalache.discourse.group]