BMCMT – Bounded Model Checking of TLA$^+$ Specifications with SMT

Jure Kukovec **Igor Konnov** Thanh Hai Tran

work in progress

TLA$^+$ Community Event Oxford, UK, July 2018

Laboratoire lorrain de recherche en informatique et ses applications

informatics mathematics

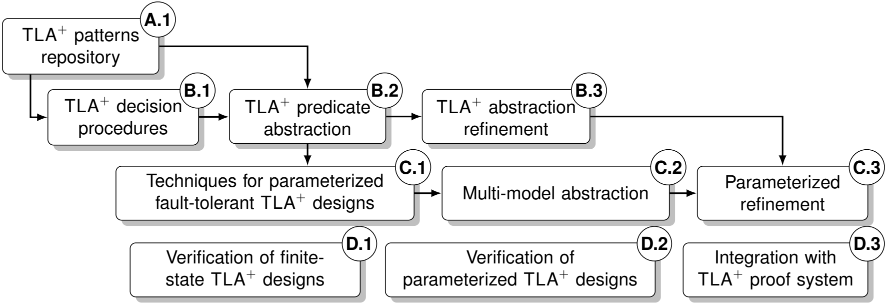
Rigorous Systems Engineering

VIENNA SCIENCE AND TECHNOLOGY FUND

APALACHE

Abstraction-based Parameterized TLA$^+$ Checker

VIENNA SCIENCE
AND TECHNOLOGY FUND



Almost automated verification: using the user input in a sound way

TLA$^+$

First-order logic with sets (ZFC)

Temporal operators:

$\Box$ (always), $\Diamond$ (eventually), $\leadsto$ (leads-to), no *Nexttime*

Syntax for operations on sets, functions, tuples, records

TLA Proof System: TLAPS

Explicit-state model checker: TLC

What is hard about TLA$^+$?

Rich data

sets of sets, functions, records, tuples, sequences

No types

TLA$^+$ is not a programming language

No imperative statements like assignments

TLA$^+$ is not a programming language

No standard control flow

TLA$^+$ is not a programming language

In this talk:

- a model checker like TLC but symbolic
- no abstractions
- nothing parameterized

Our short-term goal

Symbolic model checker that works under the assumptions of TLC:

Fixed and finite constants (parameters)

Finite sets, function domains and **co-domains**

TLC restrictions on formula structure

As few language restrictions as possible

Technically,

Quantifier-free formulas in SMT

Unfolding quantified expressions, e.g., $\forall x \in S : P$ as $\bigwedge_{c \in S} P[c/x]$

Our short-term goal

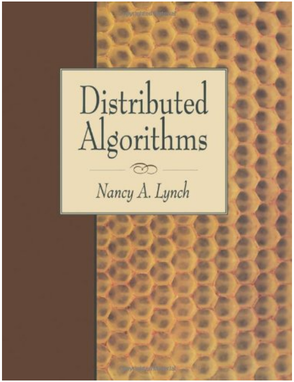
Symbolic model checker that works under the assumptions of TLC:

- **Fixed and finite constants** (parameters)
- **Finite sets, function domains** and **co-domains**
- TLC restrictions on formula structure
- **As few language restrictions as possible**

Technically,

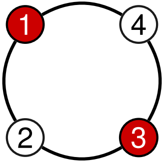
- Quantifier-free formulas in SMT
- Unfolding quantified expressions, e.g., $\forall x \in S : P$ as $\bigwedge_{c \in S} P[c/x]$

an example



Maximal Independent Set

Classical distributed problem **[Lynch, Ch 4]**



N processes placed in the nodes of an undirected graph (V, E)

Processes exchange messages in synchronous rounds

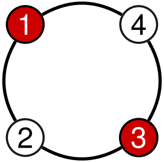
Goal: Find a maximal independent set $I \subseteq V$:

$$(u, v) \in E \rightarrow u \notin I \vee v \notin I \qquad \text{for } u, v \in V \tag{1}$$

$$\text{every larger set } I' \supset I \text{ violates Equation (1)} \tag{2}$$

Example: $I = \{\mathbf{1}, \mathbf{3}\}$

Maximal Independent Set



Classical distributed problem **[Lynch, Ch 4]**

N processes placed in the nodes of an undirected graph (V, E)

Processes exchange messages in synchronous rounds

Goal: Find a maximal independent set $I \subseteq V$:

$$(u, v) \in E \to u \notin I \lor v \notin I \qquad \text{for } u, v \in V \tag{1}$$

$$\text{every larger set } I' \supset I \text{ violates Equation (1)} \tag{2}$$

Example: $I = \{\mathbf{1}, \mathbf{3}\}$

LubyMIS

randomized distributed algorithm **[Lynch, p. 73]**

every process cyclically executes three rounds: 1, 2, 3, 1, 2, 3, ...

at every round 1, a process i randomly picks a value $val[i] \in 1..N^4$

round 1: if $val[i] > val[k]$ for every neighbor k of i, i sends "winner" to the neighbors of i

round 2: if a process i receives "winner", it becomes a "loser" and sends "loser" to the neighbors

round 3: a process i removes the losers from its neighbors if i is a winner or a loser, it falls asleep

LubyMIS

randomized distributed algorithm **[Lynch, p. 73]**

every process cyclically executes three rounds: 1, 2, 3, 1, 2, 3, ...

at every round 1, a process *i* randomly picks a value $val[i] \in 1..N^4$

round 1: if $val[i] > val[k]$ for every neighbor *k* of *i*,
i sends “winner” to the neighbors of *i*

round 2: if a process *i* receives “winner”,
it becomes a “loser” and sends “loser” to the neighbors

round 3: a process *i* removes the losers from its neighbors
if *i* is a winner or a loser, it falls asleep

LubyMIS

randomized distributed algorithm **[Lynch, p. 73]**

every process cyclically executes three rounds: 1, 2, 3, 1, 2, 3, . . .

at every round 1, a process i randomly picks a value $val[i] \in 1..N^4$

round 1: if $val[i] > val[k]$ for every neighbor k of i, i sends "winner" to the neighbors of i

round 2: if a process i receives "winner", it becomes a "loser" and sends "loser" to the neighbors

round 3: a process i removes the losers from its neighbors if i is a winner or a loser, it falls asleep

LubyMIS

randomized distributed algorithm **[Lynch, p. 73]**

every process cyclically executes three rounds: 1, 2, 3, 1, 2, 3, ...

at every round 1, a process i randomly picks a value $val[i] \in 1..N^4$

round 1: if $val[i] > val[k]$ for every neighbor k of i,
i sends “winner” to the neighbors of i

round 2: if a process i receives “winner”,
it becomes a “loser” and sends “loser” to the neighbors

round 3: a process i removes the losers from its neighbors
if i is a winner or a loser, it falls asleep

```
---------------------------- MODULE mis ----------------------------
EXTENDS Integers, TLC

N ≜ 3
N4 ≜ 81
Nodes ≜ 1 .. N

VARIABLES Nb, round, val, awake, rem_nbrs, status, msgs

Pred(n) ≜ IF n > 1 THEN n − 1 ELSE N
Succ(n) ≜ IF n < N THEN n + 1 ELSE 1

Init ≜ ∧ Nb = [n ∈ Nodes ↦ {Pred(n), Succ(n)}]
       ∧ round = 1
       ∧ val ∈ [Nodes → 1 .. N4]
       ∧ awake = [n ∈ Nodes ↦ TRUE]
       ∧ rem_nbrs = Nb
       ∧ status = [n ∈ Nodes ↦ "unknown"]
       ∧ msgs = {}

Senders(u) ≜ {v ∈ Nodes : u ∈ rem_nbrs[v] ∧ awake[v]}

SentValues(u) ≜ {val′[w] : w ∈ Senders(u)}

IsWinner(u) ≜ ∀ v ∈ msgs′[u] : val′[u] > v

Round1 ≜
    ∧ round = 1
    ∧ val′ ∈ [Nodes → 1 .. N4]  non-determinism, no randomness
    ∧ msgs′ = [u ∈ Nodes ↦ SentValues(u)]
    ∧ status′ = [n ∈ Nodes ↦
        IF awake[n] ∧ IsWinner(n) THEN "winner" ELSE status[n]]
    ∧ UNCHANGED ⟨rem_nbrs, awake⟩

SentWinners(u) ≜
    IF ∃ w ∈ Senders(u) : awake[w] ∧ status[w] = "winner"
     THEN {"winner"}
     ELSE {}

IsLoser(u) ≜ "winner" ∈ msgs′[u]

Round2 ≜
    ∧ round = 2
    ∧ msgs′ = [u ∈ Nodes ↦ SentWinners(u)]
    ∧ status′ = [n ∈ Nodes ↦
        IF awake[n] ∧ IsLoser(n) THEN "loser" ELSE status[n]]
    ∧ UNCHANGED ⟨rem_nbrs, awake, val⟩

SentLosers(u) ≜
    {w ∈ Senders(u) : awake[w] ∧ status[w] = "loser"}

Round3 ≜
    ∧ round = 3
    ∧ msgs′ = [u ∈ Nodes ↦ SentLosers(u)]
    ∧ awake′ = [n ∈ Nodes ↦
        IF status[n] ∉ {"winner", "loser"} THEN TRUE ELSE FALSE]
    ∧ rem_nbrs′ = [u ∈ Nodes ↦ rem_nbrs[u] \ msgs′[u]]
    ∧ UNCHANGED ⟨status, val⟩

Next ≜
    round′ = 1 + (round%3) ∧ (Round1 ∨ Round2 ∨ Round3) ∧ UNCHANGED ⟨Nb⟩

IsIndependent ≜
    ∀ u ∈ Nodes : ∀ v ∈ Nb[u] :
      (status[u]    ≠ "winner" ∨ status[v] ≠ "winner")

Terminated ≜ ∀ n ∈ Nodes : awake[n] = FALSE

=============================================================================
\* Modification History
\* Last modified Mon Jul 16 19:35:37 CEST 2018 by igor
\* Created Sun Jul 15 17:03:47 CEST 2018 by igor
```

Declaration and initialization

EXTENDS *Integers*

$N \triangleq 3$

$N4 \triangleq 81$

$Nodes \triangleq 1 \, .. \, N$

VARIABLES Nb, $round$, val, $awake$, rem_nbrs, $status$, $msgs$

$Pred(n) \triangleq$ IF $n > 1$ THEN $n - 1$ ELSE N

$Succ(n) \triangleq$ IF $n < N$ THEN $n + 1$ ELSE 1

$$
\begin{aligned}
Init \triangleq\ & \land Nb = [n \in Nodes \mapsto \{Pred(n), Succ(n)\}] \quad \text{(* a ring of size } N \text{ *)} \\
& \land round = 1 \\
& \land val \in [Nodes \to 1..N4] \\
& \land awake = [n \in Nodes \mapsto TRUE] \\
& \land rem\_nbrs = Nb \\
& \land status = [n \in Nodes \mapsto \text{“}unknown\text{''}] \\
& \land msgs = \{\}
\end{aligned}
$$

Round 1

$$Senders(u) \triangleq \{v \in Nodes : u \in rem\_nbrs[v] \land awake[v]\}$$

$$SentValues(u) \triangleq \{val'[w] : w \in Senders(u)\}$$

$$IsWinner(u) \triangleq \forall v \in msgs'[u] : val'[u] > v$$

$$
\begin{array}{l}
Round1 \triangleq \\
\quad \land\ round = 1 \\
\quad \land\ val' \in [Nodes \rightarrow 1..N4] \quad \text{(* non-determinism instead of randomness *)} \\
\quad \land\ msgs' = [u \in Nodes \mapsto SentValues(u)] \\
\quad \land\ status' = [n \in Nodes \mapsto \\
\qquad\qquad \textsf{IF}\ awake[n] \land IsWinner(n)\ \textsf{THEN}\ \text{"}winner\text{"}\ \textsf{ELSE}\ status[n]] \\
\quad \land\ \textsf{UNCHANGED}\ \langle\langle rem\_nbrs, awake \rangle\rangle
\end{array}
$$

Round 2

$$
\begin{array}{l}
\mathit{SentWinners}(u) \triangleq \\
\quad \text{IF } \exists w \in \mathit{Senders}(u) : \mathit{awake}[w] \land \mathit{status}[w] = \text{“}\mathit{winner}\text{''} \\
\quad \text{THEN } \{\text{“}\mathit{winner}\text{''}\} \\
\quad \text{ELSE } \{\}
\end{array}
$$

$$
\mathit{IsLoser}(u) \triangleq \text{“}\mathit{winner}\text{''} \in \mathit{msgs}'[u]
$$

$$
\begin{array}{l}
\mathit{Round2} \triangleq \\
\quad \land\ \mathit{round} = 2 \\
\quad \land\ \mathit{msgs}' = [u \in \mathit{Nodes} \mapsto \mathit{SentWinners}(u)] \\
\quad \land\ \mathit{status}' = [n \in \mathit{Nodes} \mapsto \\
\qquad\qquad \text{IF } \mathit{awake}[n] \land \mathit{IsLoser}(n) \\
\qquad\qquad \text{THEN } \text{“}\mathit{loser}\text{''} \\
\qquad\qquad \text{ELSE } \mathit{status}[n]] \\
\quad \land\ \text{UNCHANGED } \langle\langle \mathit{rem\_nbrs}, \mathit{awake}, \mathit{val} \rangle\rangle
\end{array}
$$

Round 3

$$SentLosers(u) \triangleq \{w \in Senders(u) : awake[w] \land status[w] = \text{"}loser\text{"}\}$$

$$
\begin{aligned}
Round3 \triangleq\ & \land\ round = 3 \\
& \land\ msgs' = [u \in Nodes \mapsto SentLosers(u)] \\
& \land\ awake' = [n \in Nodes \mapsto \\
& \qquad \text{IF } status[n] \in \{\text{"}winner\text{"}, \text{"}loser\text{"}\} \text{ THEN FALSE ELSE TRUE}] \\
& \land\ rem\_nbrs' = [u \in Nodes \mapsto rem\_nbrs[u] \setminus msgs'[u]] \\
& \land\ \text{UNCHANGED } \langle\langle status, val \rangle\rangle
\end{aligned}
$$

Putting it all together

(The next-state relation *)*

$$
\begin{array}{l}
Next \triangleq \\
\quad \land\ round' = 1 + (round \,\%\, 3) \\
\quad \land\ (Round1 \lor Round2 \lor Round3) \\
\quad \land\ \text{UNCHANGED}\ \langle\langle Nb \rangle\rangle
\end{array}
$$

(An invariant *)*

$$
\begin{array}{l}
IsIndependent \triangleq \\
\quad \forall u \in Nodes : \forall v \in Nb[u] : \\
\qquad\quad (status[u] \neq \text{"}winner\text{"} \lor status[v] \neq \text{"}winner\text{"})
\end{array}
$$

Let's run TLC for $N = 3$...

One day later... still running

Why? Crunching states produced by $\{1, \ldots, N^4\}$, that is, 3^4 integers

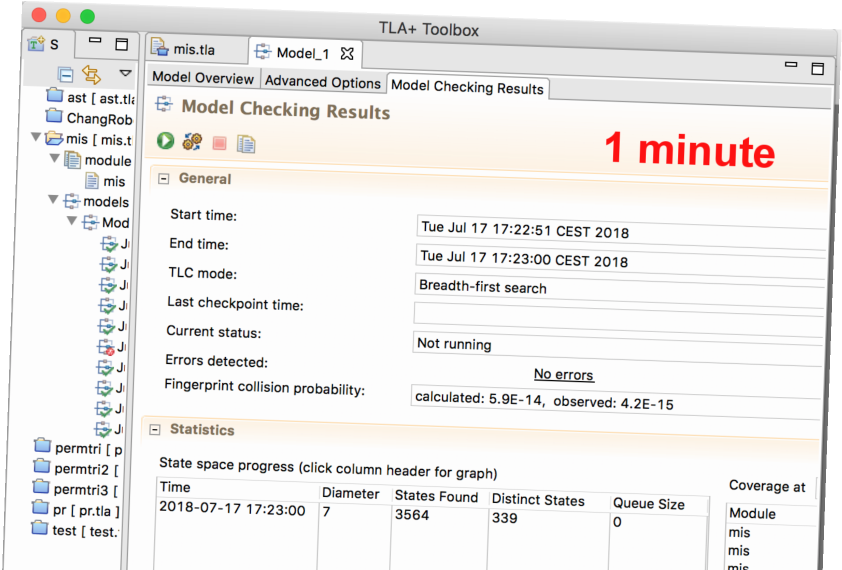
Let's run TLC for $N = 3$...

One day later... still running

Why? Crunching states produced by $\{1, \ldots, N^4\}$, that is, 3^4 integers

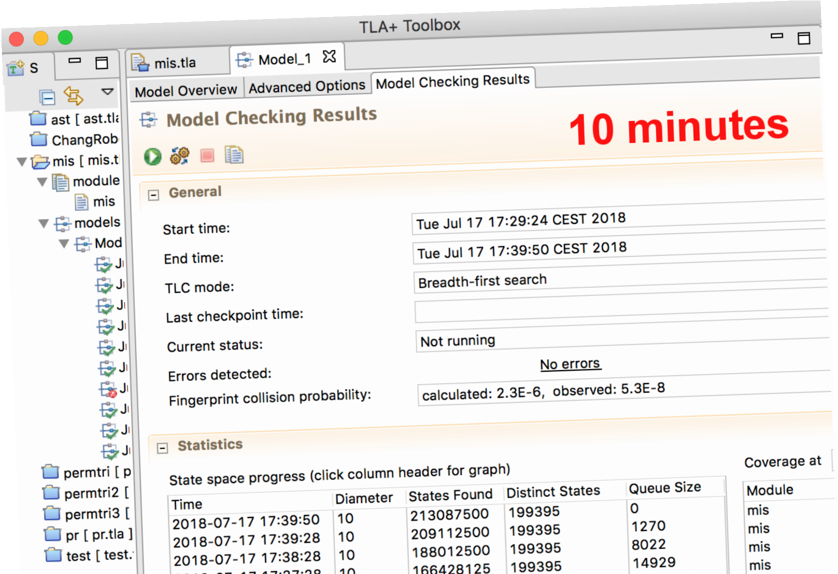
Let’s be more fair to TLC

Let’s set *N*4 to *N* (the smaller values kill progress)



How about larger graphs?

Let's set *N* and *N4* to 5

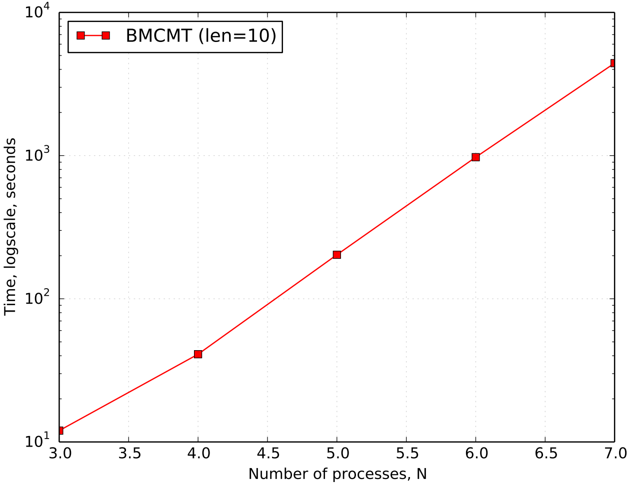


Let's run BMCMT

for $N = 5$ and $N4 = 5^4$ **3 minutes**

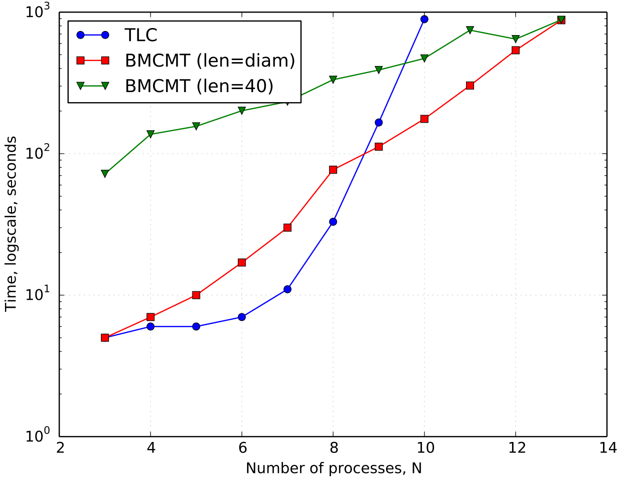
```
./bin/apalache-mc check --inv=IsIndependent mis.tla

PASS #1: AssignmentFinder
Found 1 initializing transitions and 3 next transitions
PASS #2: Grade
PASS #3: SimpleSkolemization
Found 2 free existentials in the transitions
PASS #4: BoundedChecker
The outcome is: NoError
PASS #5: Terminal
Checker reports no error up to computation length 10
```



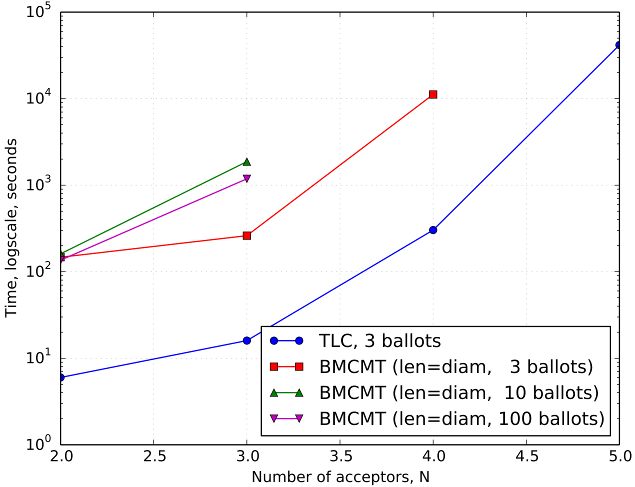
LubyMIS: N processes and the range $1..N^4$

Invariant: independence for executions of length up to 10



EWD840: Dijkstra's termination detection in a ring of N nodes
Invariant: when termination is detected, all nodes are inactive

Diameter is $3N$, as shown by TLC for $N \leq 10$



Simple Paxos of N acceptors, from the TLA$^+$ benchmarks

Just computing reachable states (SAT) for 3, 10, and 100 ballots

Can I run it?

Yes! [forsyte.at/research/apalache]

Beware: it is fresh and crashes more often than it works

how does it work?

Essential steps

Extracting assignments and symbolic transitions

somewhat similar to TLC

Simple type inference

as we go, for every step

Bounded model checking

we track potential contents of data structures

assignments & symbolic transitions

Symbolic transitions

$$
\begin{array}{l}
Next \triangleq \\
\quad \land\ round' = 1 + (round \% 3) \\
\quad \land\ (Round1 \lor Round2 \lor Round3) \\
\quad \land\ \text{UNCHANGED } \langle\langle Nb \rangle\rangle
\end{array}
$$

Intuitively, we reason about the three cases:

$$
\begin{array}{l}
\land\ round' = 1 + (round \% 3) \\
\land\ (Round1 \lor \cancel{Round2 \lor Round}3) \\
\land\ \text{UNCHANGED } \langle\langle Nb \rangle\rangle
\end{array}
$$

$$
\begin{array}{l}
\land\ round' = 1 + (round \% 3) \\
\land\ (\cancel{Round1} \lor Round2 \lor \cancel{Round3}) \\
\land\ \text{UNCHANGED } \langle\langle Nb \rangle\rangle
\end{array}
$$

$$
\begin{array}{l}
\land\ round' = 1 + (round \% 3) \\
\land\ (\cancel{Round1 \lor Round2} \lor Round3) \\
\land\ \text{UNCHANGED } \langle\langle Nb \rangle\rangle
\end{array}
$$

Symbolic transitions

$$
\begin{aligned}
&\mathit{Next} \triangleq \\
&\quad \land\ \mathit{round}' = 1 + (\mathit{round} \% 3) \\
&\quad \land\ (\mathit{Round}1 \lor \mathit{Round}2 \lor \mathit{Round}3) \\
&\quad \land\ \text{UNCHANGED } \langle\langle \mathit{Nb} \rangle\rangle
\end{aligned}
$$

Intuitively, we reason about the three cases:

$$
\begin{aligned}
&\land\ \mathit{round}' = 1 + (\mathit{round} \% 3) \\
&\land\ (\mathit{Round}1 \lor \cancel{\mathit{Round}2 \lor \mathit{Round}3}) \\
&\land\ \text{UNCHANGED } \langle\langle \mathit{Nb} \rangle\rangle
\end{aligned}
$$

$$
\begin{aligned}
&\land\ \mathit{round}' = 1 + (\mathit{round} \% 3) \\
&\land\ (\cancel{\mathit{Round}1} \lor \mathit{Round}2 \lor \cancel{\mathit{Round}3}) \\
&\land\ \text{UNCHANGED } \langle\langle \mathit{Nb} \rangle\rangle
\end{aligned}
$$

$$
\begin{aligned}
&\land\ \mathit{round}' = 1 + (\mathit{round} \% 3) \\
&\land\ (\cancel{\mathit{Round}1 \lor \mathit{Round}2} \lor \mathit{Round}3) \\
&\land\ \text{UNCHANGED } \langle\langle \mathit{Nb} \rangle\rangle
\end{aligned}
$$

How does TLC find assignments?

TLC detects assignments as it explores a formula:

- from left to right:

$$x' = 1 \land x' \in \{1, 2, 3\}$$

- treating action-level disjunctions as non-deterministic choice

$$\big(x' = 1 \lor x' = 2\big) \land x' \geq 2$$

- expecting the same kind of assignments on all branches

$$(x' = 1 \land y' = 2) \lor x' = 3$$

Anything similar with SMT?

Looking for assignment strategies that:

- cover every Boolean branch (not easy to define)
- have exactly one assignment per variable per branch
- do not contain cyclic assignments

$$\big((\underline{y' = x'} \land x' \in \{2, 3, y'\}) \lor (x' = 2 \land \underline{y' \in \{x'\}})\big) \land \underline{x' = 3}$$

Sometimes, we do better than TLC (above)

Sometimes, worse, e.g., when $x = 0$:

$$x > 0 \lor (x' = x + 1 \lor y' = x - 1)$$

[Kukovec, K., Tran, ABZ'18]

Anything similar with SMT?

Looking for assignment strategies that:

- cover every Boolean branch (not easy to define)
- have exactly one assignment per variable per branch
- do not contain cyclic assignments

$$\big((\underline{y' = x'} \wedge x' \in \{2, 3, y'\}) \vee (x' = 2 \wedge \underline{y' \in \{x'\}})\big) \wedge \underline{x' = 3}$$

Sometimes, we do better than TLC (above)

Sometimes, worse, e.g., when $x = 0$:

$$x > 0 \vee (x' = x + 1 \vee y' = x - 1)$$

[Kukovec, K., Tran, ABZ'18]

Simple types

Types: scalars and functions

Basic:

constants: *Const* — “a”, “hello”

integers: *Int* — -1, 1024

Booleans: *Bool* — FALSE, TRUE

Functions:

functions: $\tau_{set} \rightarrow \tau_{set}$ — $\mathit{FinSet}(\mathit{Int}) \rightarrow \mathit{FinSet}(\mathit{Bool})$

tuples: $\langle \tau, \dots, \tau \rangle$ — $\langle \mathit{Int}, \mathit{Bool}, \mathit{FinSet}(\mathit{Int}), \mathit{Int} \rightarrow \mathit{FinSet}(\mathit{Int}) \rangle$

records: $[\mathit{Const} \mapsto \tau, \dots, \mathit{Const} \mapsto \tau]$ — $[\text{“a”} \mapsto \mathit{Int}, \text{“b”} \mapsto \mathit{Bool}]$

Types: sets

finite sets: $\mathit{FinSet}(\tau)$ $\qquad$ $\mathit{FinSet}(\mathit{Int})$

power sets: $\mathit{PowSet}(\tau)$ $\qquad$ $\mathit{PowSet}(\mathit{FinSet}(\mathit{Int}))$

function sets: $[\tau_{set} \rightarrow \tau_{set}]$ $\qquad$ $[\mathit{PowSet}(\mathit{FinSet}(\mathit{Int})) \rightarrow \mathit{FinSet}(\mathit{Int})]$

products: $\tau_{set} \times \cdots \times \tau_{set}$ $\qquad$ $[\mathit{FinSet}(\mathit{Int}) \times \mathit{FinSet}(\mathit{Bool})]$

record sets: $[\mathit{Const} : \tau_{set}, \ldots, \mathit{Const} : \tau_{set}]$

$[\text{“a”} : \mathit{FinSet}(\mathit{Int}), \text{“b”} : \mathit{FinSet}(\mathit{Bool})]$

Simple type inference

Knowing the types at the current state

Compute the types of the expressions and of the primed variables

e.g., if X has type $\mathit{FinSet}(\mathit{Int})$, then

$X' = [X \to X]$ has type $\mathit{Fun}(\mathit{FinSet}(\mathit{Int}), \mathit{FinSet}(\mathit{Int}))$

y in $\{y \in X : y > 0\}$ has type Int

$\{\}$ has type $\mathit{FinSet}(\mathit{Unknown})$

hence, IF P THEN $\{1\}$ ELSE $\{\}$ fails

one can hack it by writing $\{1\} \setminus \{1\}$

Simple type inference

Knowing the types at the current state

Compute the types of the expressions and of the primed variables

e.g., if X has type $\mathit{FinSet}(\mathit{Int})$, then

$X' = [X \to X]$ has type $\mathit{Fun}(\mathit{FinSet}(\mathit{Int}), \mathit{FinSet}(\mathit{Int}))$

y in $\{y \in X : y > 0\}$ has type Int

$\{\}$ has type $\mathit{FinSet}(\mathit{Unknown})$

hence, IF P THEN $\{1\}$ ELSE $\{\}$ fails

one can hack it by writing $\{1\} \setminus \{1\}$

Bounded model checking

Old recipe for bounded symbolic computations

Two symbolic transitions that assign values to x

$$Next \triangleq A \vee B$$

Translate TLA$^+$ expressions to SMT with some $[\![\cdot]\!]$

$[\![Init]\!]$	$x \mapsto i_0$
$[\![A[i_0/x]]\!]$	$x \mapsto a_1$
$[\![B[i_0/x]]\!]$	$x \mapsto b_1$
$[\![x' \in \{a_1, b_1\}]\!]$	$x \mapsto c_1$
$[\![A[c_1/x]]\!]$	$x \mapsto a_2$
$[\![B[c_1/x]]\!]$	$x \mapsto b_2$
$[\![x' \in \{a_2, b_2\}]\!]$	$x \mapsto c_2$
$\ldots$	$\ldots$

What is $\llbracket\cdot\rrbracket$?

Our idea

Let's mimic the explicit model checker TLC

Explicitely compute the memory layout of data structures

Restrict memory contents with SMT

Define operational semantics (for finite models)

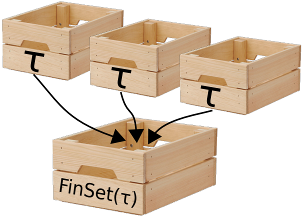
Static picture of TLA$^+$ objects and relations between them

Arena:

SMT:

integer `sort Int`

Boolean `sort Bool`



finite set: ¢ of uninterpreted sort

uninterpreted function

in :  ×  → **Bool**

Arenas: functions

SMT:

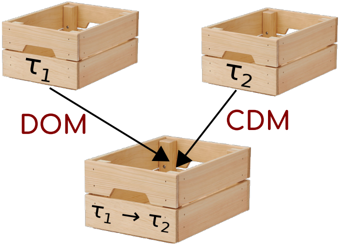
function: ¢ of uninterpreted sort

uninterpreted function

fun$_{¢}$:  $\rightarrow$ 

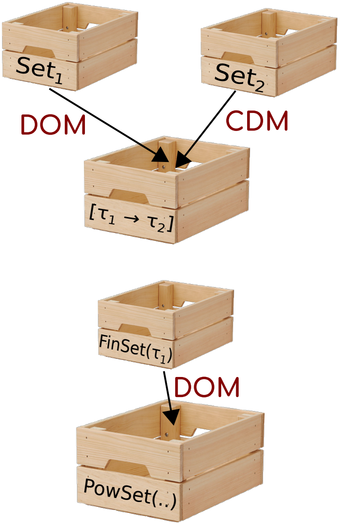
that keeps tracks of values

Arena:



Arenas: set of functions and powerset

Arena:



SMT:

cells of uninterpreted sort

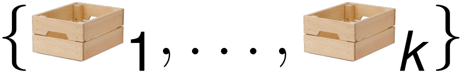
$[\{1,2,3\} \rightarrow \{4,5,6\}]$

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

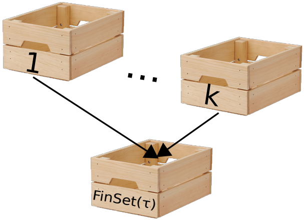
just tracking the structure

some rules for sets

Set constructor

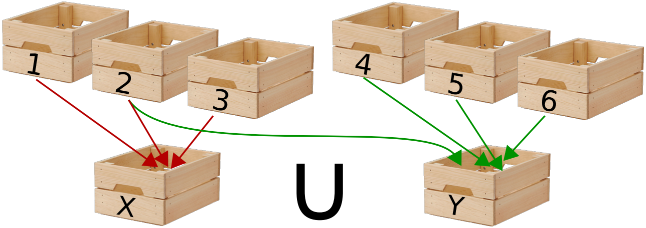


Arena:

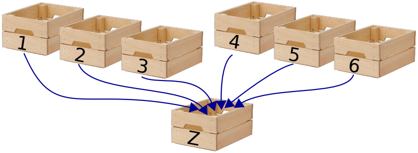


SMT:

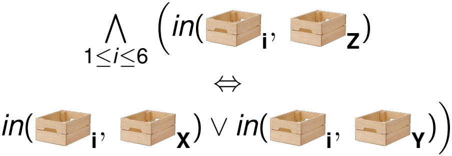
Set union: $Z = X \cup Y$



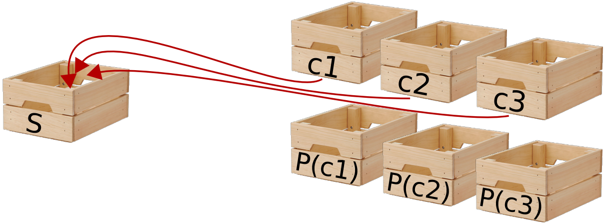
Arena:



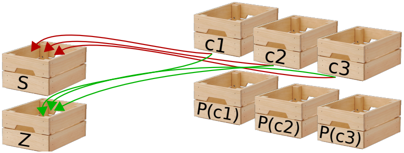
SMT:



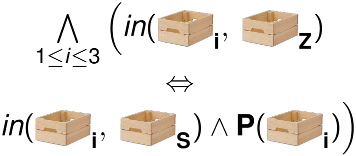
Set filter: $Z = \{x \in S \colon P\}$



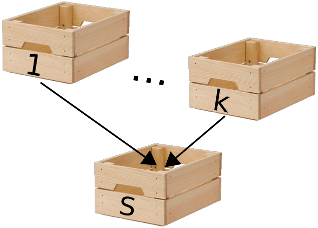
Arena:



SMT:



Set membership: $¢_e \in ¢_S$



( $_e =$  $_i$) $\rightsquigarrow$ eq_i

for $1 \leq i \leq k$

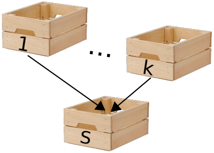
Arena:

 $_?$

SMT:

 $_? \leftrightarrow \bigvee_{1 \leq i \leq k} \Big(eq_i \wedge in($  $_{\mathbf{i}},$  $_{\mathbf{S}}) \Big)$

Set inclusion and equality



$S \subseteq T \quad \leftrightarrow \quad \bigwedge_{1 \le i \le k} \Big(in($  $_i,$  $_S) \to$  $_i \in$  $_T \Big)$

$$S = T \quad \leftrightarrow \quad S \subseteq T \wedge T \subseteq S$$

Equality in general

Integers, Booleans, string constants

- SMT equality (=)

Sets, functions, records, tuples

- **lazy**, define $X = Y$ when needed
- avoid redundant constraints
- exploit locality thanks to arenas
- cache constraints

Implementation

about 100 rewriting rules, to encode semantics

still, some features not covered:

- recursive functions
- sequences
- set cardinalities (any ideas?)
- operations with modules

Conclusions

TLA$^+$ can be checked symbolically

TLC works surprisingly well

Covering all TLA$^+$ features is hard!

We are preparing a technical report and hope to release a stable version soon

We need benchmarks from you!