Towards parameterized verification of TLA$^+$ specifications

Poonam Kumari
Supervisor: Stephan Merz

July 4, 2017

Array based transition systems

Motivation

- Verify safety properties of system having arbitrary numbers of processes
- No manual intervention
- Convert TLA^+ spec into Cubicle model checker

Array based transition systems

Overview

- Identify a class of TLA + specifications that can be verified using the Cubicle model checker
- Design a translator
- Validate by applying it to suitable case studies

Outline

TLA$^+$

- Specification language for modelling concurrent systems
- Based on set theory, first-order logic and Temporal Logic of Action
- Represent data structures as sets and functions
- 4 specification parts
  - Initial predicate
  - “Next” state relation
  - Safety Properties
  - Liveness Properties
- $Init \wedge \Box[Next]_{vars} \wedge Liveness$

Case study : Two-Phase Commit protocol

Two-Phase Commit

- One process acts as coordinator

Case study : Two-Phase Commit protocol

Two-Phase Commit

- One process acts as coordinator
- The other participating processes are subordinates

Case study : Two-Phase Commit protocol

Two-Phase Commit

- One process acts as coordinator
- The other participating processes are subordinates
- Initially, each participant decides if it wishes to propose commit or propose abort

Case study : Two-Phase Commit protocol

Two-Phase Commit

- One process acts as coordinator
- The other participating processes are subordinates
- Initially, each participant decides if it wishes to propose commit or propose abort
- The transaction is committed by coordinator if all the participants do propose commit. Otherwise, it is aborted

Case study : Two-Phase Commit protocol

Two-Phase Commit

- One process acts as coordinator
- The other participating processes are subordinates
- Initially, each participant decides if it wishes to propose commit or propose abort
- The transaction is committed by coordinator if all the participants do propose commit. Otherwise, it is aborted
- The participants execute the order of the coordinator

TLA + Specification: Data Structure

MODULE *TwoPhaseCommit*

CONSTANT *RM*

VARIABLE *Coordinator*, *rmState*

$Rmmsg \triangleq \{\text{“working”}, \text{“proposeCommit”}, \text{“proposeAbort”}, \text{“commit”}, \text{“abort”}\}$

$Cmsg \triangleq \{\text{“init”}, \text{“ccommit”}, \text{“cabort”}\}$

$TypeOK \triangleq \land\ rmState \in [RM \to Rmmsg]$
$\qquad\qquad \land\ Coordinator \in Cmsg$

- Declaration of parameters
- Definition of operators
- TypeOK defines expected values of variables

TLA + Specification: Behavior

$$Init \triangleq \land rmState = [rm \in RM \mapsto \text{“working”}]$$
$$\land Coordinator = \text{“init”}$$

- Initial condition: defines initial states of system

TLA + Specification: Behavior

$$Init \triangleq \land\ rmState = [rm \in RM \mapsto \text{“working”}]$$
$$\land\ Coordinator = \text{“init”}$$

$$DecisionCommit(rm) \triangleq \land\ rmState[rm] = \text{“working”}$$
$$\land\ rmState' = [rmState \text{ EXCEPT } ![rm] = \text{“proposeCommit”}]$$
$$\land\ Coordinator' = Coordinator$$

$\vdots$

- Initial condition: defines initial states of system
- Action definitions: describe transitions of the algorithm

TLA + Specification: Behavior

$$
\begin{aligned}
&\mathit{Init} \triangleq \land\ \mathit{rmState} = [\mathit{rm} \in \mathit{RM} \mapsto \text{“working”}] \\
&\qquad\qquad \land\ \mathit{Coordinator} = \text{“init”} \\
&\mathit{DecisionCommit}(\mathit{rm}) \triangleq \land\ \mathit{rmState}[\mathit{rm}] = \text{“working”} \\
&\qquad\qquad\qquad\qquad\qquad \land\ \mathit{rmState}' = [\mathit{rmState}\ \text{EXCEPT}\ ![\mathit{rm}] = \text{“proposeCommit”}] \\
&\qquad\qquad\qquad\qquad\qquad \land\ \mathit{Coordinator}' = \mathit{Coordinator} \\
&\vdots \\
&\mathit{Next} \triangleq \lor\ \mathit{CCommit} \lor \mathit{CAbort} \\
&\qquad\qquad \lor\ \exists \mathit{rm} \in \mathit{RM} : \lor\ \mathit{DecisionCommit}(\mathit{rm}) \lor \mathit{DecisionAbort}(\mathit{rm}) \\
&\qquad\qquad\qquad\qquad\qquad \lor\ \mathit{Commit}(\mathit{rm}) \lor \mathit{Abort}(\mathit{rm})
\end{aligned}
$$

- Initial condition: defines initial states of system
- Action definitions: describe transitions of the algorithm
- Next-state relation: contains all allowed transitions

TLA + Specification: Behavior

$$
\begin{aligned}
&\mathit{Init} \triangleq \land\ \mathit{rmState} = [\mathit{rm} \in \mathit{RM} \mapsto \text{“working”}] \\
&\qquad\qquad \land\ \mathit{Coordinator} = \text{“init”} \\
&\mathit{DecisionCommit}(\mathit{rm}) \triangleq \land\ \mathit{rmState}[\mathit{rm}] = \text{“working”} \\
&\qquad\qquad\qquad\qquad\qquad \land\ \mathit{rmState}' = [\mathit{rmState}\ \text{EXCEPT}\ ![\mathit{rm}] = \text{“proposeCommit”}] \\
&\qquad\qquad\qquad\qquad\qquad \land\ \mathit{Coordinator}' = \mathit{Coordinator} \\
&\vdots \\
&\mathit{Next} \triangleq \lor\ \mathit{CCommit} \lor \mathit{CAbort} \\
&\qquad\qquad \lor\ \exists \mathit{rm} \in \mathit{RM} : \lor\ \mathit{DecisionCommit}(\mathit{rm}) \lor \mathit{DecisionAbort}(\mathit{rm}) \\
&\qquad\qquad\qquad\qquad\qquad \lor\ \mathit{Commit}(\mathit{rm}) \lor \mathit{Abort}(\mathit{rm}) \\
&\mathit{Safety} \triangleq \forall \mathit{rm1}, \mathit{rm2} \in \mathit{RM} : \neg\,(\mathit{rmState}[\mathit{rm1}] = \text{“abort”} \land \mathit{rmState}[\mathit{rm2}] = \text{“commit”})
\end{aligned}
$$

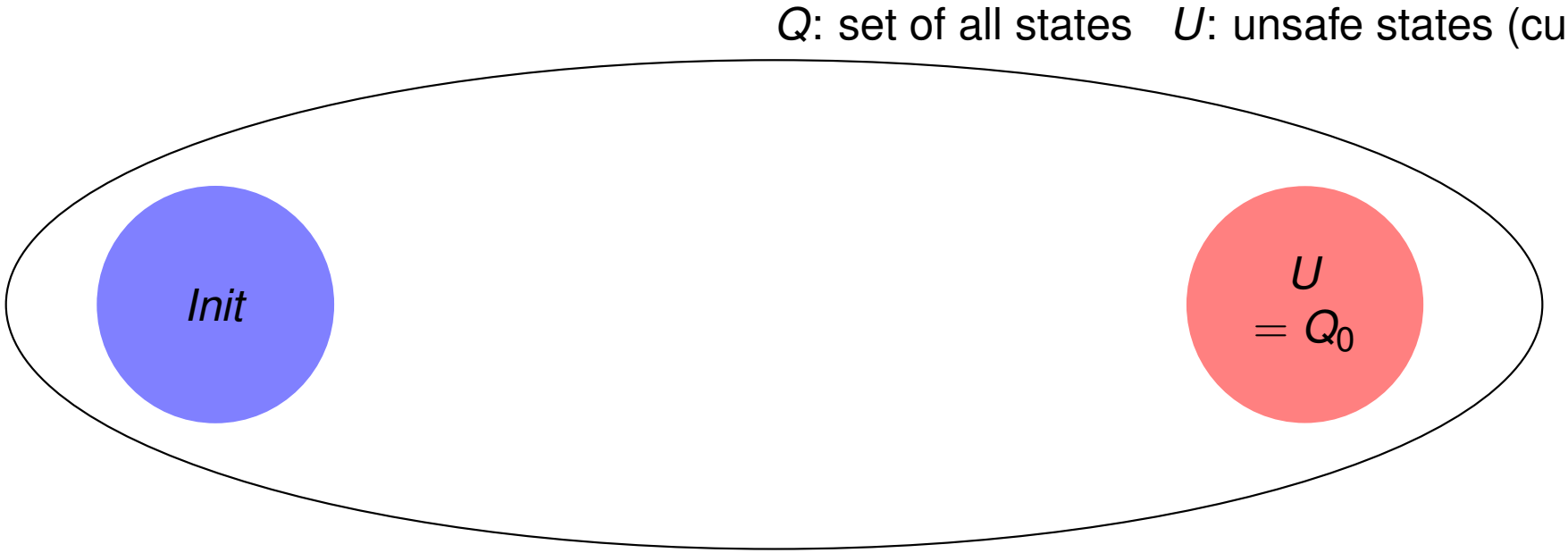
- Initial condition: defines initial states of system
- Action definitions: describe transitions of the algorithm
- Next-state relation: contains all allowed transitions
- Safety Invariant: no two RMs have arrived at conflicting decisions

Cubicle

Cubicle: an open source model checker

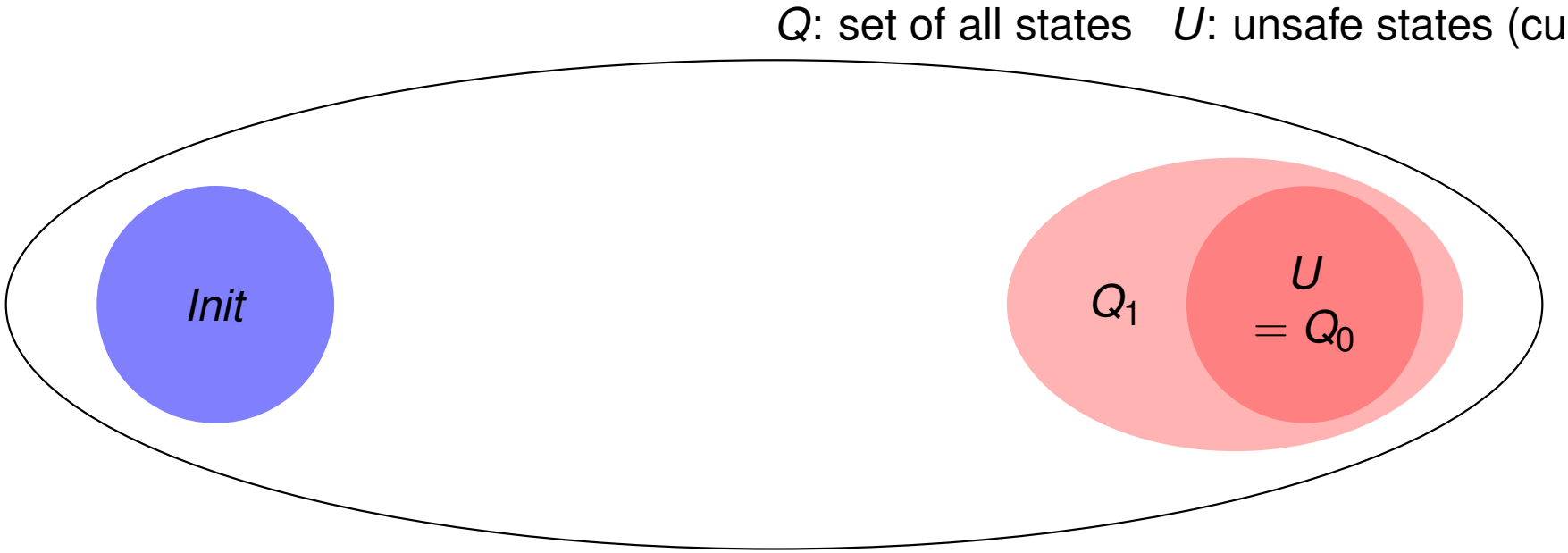
- Verify safety properties of system involving any number of processes
- Its implementation relies on an SMT solver
- Cubicle model-checks by backward reachability analysis on infinite sets of states represented by cubes

Cubicle: Backward Reachability



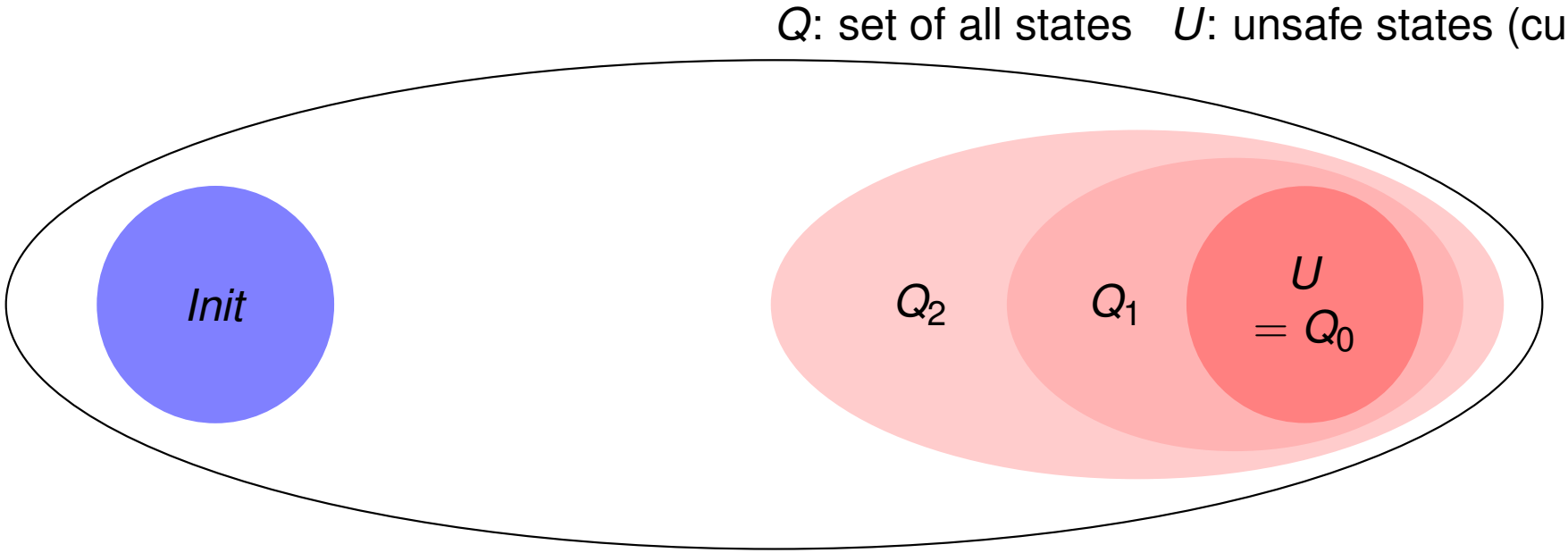
- Q_0: *Unsafe* state

Cubicle: Backward Reachability



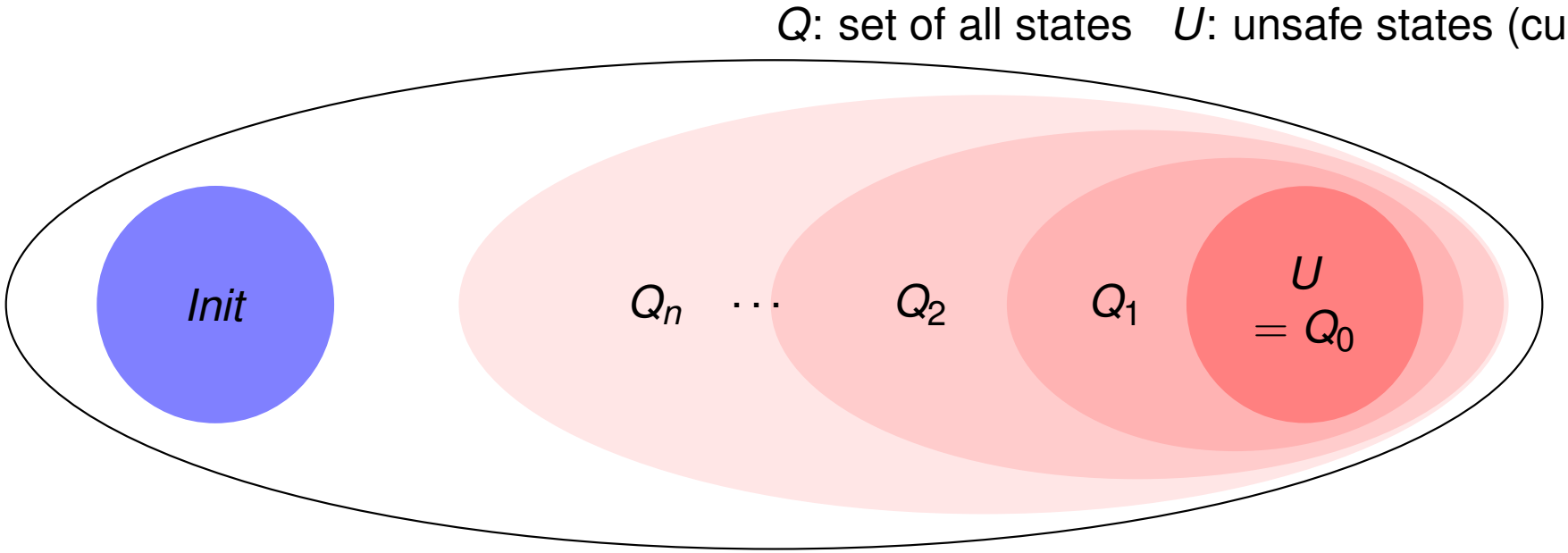
- Q_0: *Unsafe* state
- $Q_{(k+1)}$: union of Q_k and of predecessors of states in Q_k

Cubicle: Backward Reachability



- Q_0: *Unsafe* state
- $Q_{(k+1)}$: union of Q_k and of predecessors of states in Q_k

Cubicle: Backward Reachability



- Q_0: *Unsafe* state
- $Q_{(k+1)}$: union of Q_k and of predecessors of states in Q_k
- some Q_n intersects *Init*: Unsafe
- $Q_{(n+1)} = Q_n$: found all states from which *Unsafe* state can be reached

Cubicle

Cubicle's language description

Cubicle describes a system by:

- type declarations
- global variables and arrays indexed by process identifiers
- a formula for the initial states
- formulas describe the unsafe states
- guard/action transitions parameterized by process identifiers

Cubicle's language description (1)

- Four built-in data types: int, real, bool and proc
- Enumerated data types. e.g

type rmmessage = *Working* | *ProposeCommit* | *ProposeAbort* | *Commit* | *Abort*
type cmessage = *Init* | *Ccommit* | *Cabort*
array RmState[proc]: *rmmessage*
var Coordinator: *cmessage*

Cubicle's language description (1)

- Four built-in data types: int, real, bool and proc
- Enumerated data types. e.g

type rmmessage = *Working* | *ProposeCommit* | *ProposeAbort* | *Commit* | *Abort*
type cmessage = *Init* | *Ccommit* | *Cabort*
array RmState[proc]: *rmmessage*
var Coordinator: *cmessage*

- Initial states described with universally-quantified formula over processes

init (*rm*) { RmState[rm]=Working && Coordinator=Init }

Cubicle's language description (1)

- Four built-in data types: int, real, bool and proc
- Enumerated data types. e.g

type rmmessage = *Working* | *ProposeCommit* | *ProposeAbort* | *Commit* | *Abort*
type cmessage = *Init* | *Ccommit* | *Cabort*
array RmState[proc]: *rmmessage*
var Coordinator: *cmessage*

- Initial states described with universally-quantified formula over processes

init (*rm*) { RmState[rm]=Working && Coordinator=Init }

- Unsafe states described with existentially-quantified formulas over processes

unsafe (*rm*1 *rm*2) {
  RmState[*rm*1] = *Commit* && *RmState*[*rm*2] = *Abort*
}

Cubicle's language description (2)

- Transitions in the form of guard and update action

Cubicle's language description (2)

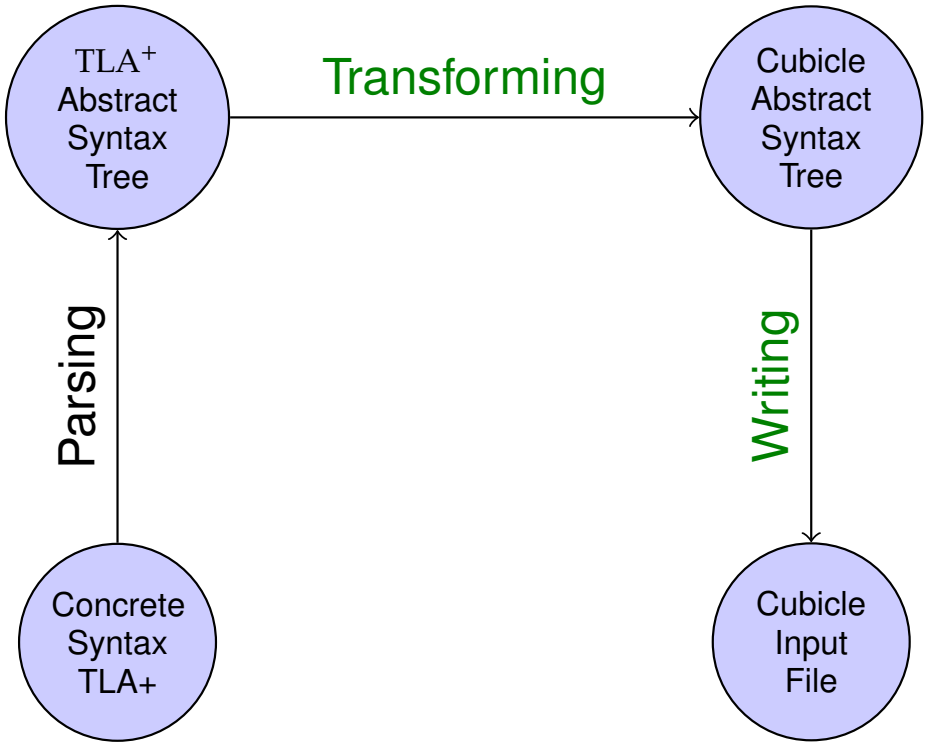
- Transitions in the form of guard and update action
- Guards must be in the form of $F \wedge \forall x.(\Delta \Rightarrow F')$
  where F and F' are conjunctions of literals and Δ means every x-variable is **distinct** from every parameter of the transition

Cubicle's language description (2)

- Transitions in the form of guard and update action
- Guards must be in the form of $F \wedge \forall x.(\Delta \Rightarrow F')$
  where F and F' are conjunctions of literals and Δ means every x-variable is **distinct** from every parameter of the transition
- Update actions are assignments to variables or arrays

```
transition Ccommit ()
requires{forall_other rm. (RmState[rm] = ProposeCommit)
                        && Coordinator = Init }
{
  Coordinator := Ccommit;
}
```

Translation : TLA$^+$ to Cubicle's input language



- TLA$^+$ spec file must be in specific format
- 4 properties
  - TypeOk Invariant
  - Init
  - Next
  - Safety Invariant

Translation to Cubicle: TypeOK

- Declaration of variables and arrays with their types

Translation to Cubicle: TypeOK

- Declaration of variables and arrays with their types
- Enumerated set T is translated into a *type* declaration

Translation to Cubicle: TypeOK

- Declaration of variables and arrays with their types
- Enumerated set T is translated into a *type* declaration
- $x \in [\, S \rightarrow T \,]$ is translated into an array parameterized by proc

Translation to Cubicle: TypeOK

- Declaration of variables and arrays with their types
- Enumerated set T is translated into a *type* declaration
- $x \in [\, S \to T \,]$ is translated into an array parameterized by proc
- $x \in T$ is translated into a variable

$$
\begin{array}{l}
\mathit{Rmmessage} \triangleq \{\text{“working”},\ \text{“proposeCommit”},\ \text{“proposeAbort”},\ \text{“Commit”},\ \text{“Abort”}\} \\
\mathit{Cmessage} \triangleq \{\text{“init”},\ \text{“Ccommit”},\ \text{“Cabort”}\} \\
\mathit{TypeOK} \triangleq \land\ \mathit{rmState} \in [\mathit{RM} \to \mathit{Rmmessage}] \\
\qquad\qquad\ \land\ \mathit{Coordinator} \in \mathit{Cmessage}
\end{array}
$$

$$
\begin{array}{l}
\mathit{type\ Rmmessage} = \mathit{Working} \mid \mathit{ProposeCommit} \mid \mathit{ProposeAbort} \mid \mathit{Commit} \mid \mathit{Abort} \\
\mathit{type\ Cmessage} = \mathit{Init} \mid \mathit{Ccommit} \mid \mathit{Cabort} \\
\mathit{array\ RmState}[\mathit{proc}] : \mathit{RmMessage} \\
\mathit{var\ Coordinator} : \mathit{Cmessage}
\end{array}
$$

Translation to Cubicle: Init

- Describes the initial state of the system
- $x = [\, s \in S \mapsto e \,]$ is translated into an array parameterized by s
- $x = e$ is translated into a variable

$$Init \triangleq \land\ rmState = [rm \in RM \mapsto \text{“working”}] \\ \land\ Coordinator = \text{“init”}$$

```
init (rm) {
  RmState[rm] = Working
  && Coordinator = Init
}
```

Translation to Cubicle: Transition

- ”Next” state relation is translated into transitions
- State predicate transformed into the guard command
- Action predicate transformed into set of assignments that update state of that variable
  - $x' = e$ is translated into a variable update
  - $g' = [f \text{ EXCEPT } ![x] = e]$ is translated into an array update using case construct and default case
  - *Unchanged* clauses are ignored

Translation to Cubicle: Transition(1)

$$
\begin{aligned}
Next \triangleq\ & \lor CCommit \lor CAbort \\
& \lor \exists rm \in RM : \lor DecisionCommit(rm) \lor DecisionAbort(rm) \\
& \qquad \lor Commit(rm) \lor Abort(rm)
\end{aligned}
$$

$$
\begin{aligned}
& CCommit \triangleq \\
& \quad \land \forall\, rm \in RM : rmState[rm] = \text{''}proposeCommit\text{''} \\
& \quad \land Coordinator = \text{“init”} \\
& \quad \land Coordinator' = \text{“Ccommit”} \\
& \quad \land rmState' = rmState
\end{aligned}
$$

$$
\begin{aligned}
& transition\ CCommit\ () \\
& requires\{\ forall\_other\ rm.\ (RmState[rm] = ProposeCommit) \\
& \qquad\qquad \&\&\ Coordinator = Init\ \} \\
& \{ \\
& \quad Coordinator := Ccommit; \\
& \}
\end{aligned}
$$

$$DecisionCommit(rm) \triangleq$$
$$\land\ rmState[rm] = \text{“working”}$$
$$\land\ rmState' = [rmState \text{ EXCEPT } ![rm] = \text{“proposeCommit”}]$$
$$\land\ UNCHANGED\ Coordinator$$

transition DecisionCommit (rm)
requires $\{RmState[rm] = Working\}$
$\{$
$RmState[e] := case$
$|e = rm : ProposeCommit$
$|\_ : RmState[e];$
$\}$

Translation to Cubicle: Safety Invariant

- Users give safety properties of TLA^+ spec to create unsafe state
- The safety properties should be in the form of below example:

$$Safety \triangleq \forall rm1, rm2 \in RM : \neg (rmState[rm1] = \text{“abort”} \wedge rmState[rm2] = \text{“commit”})$$

```
unsafe (rm1 rm2) {
   RmState[rm1] = Abort && RmState[rm2] = Commit
}
```

Summing Up

- Analyzed both languages
- Found correspondence between TLA^+ and Cubicle
- Designed translator from class of TLA^+ specs into Cubicle

Conclusion & Future work

Conclusion

- Translation from TLA^+ specifications into an array based parameterized system
- Successfully verified safety properties of Two-phase commit protocol, Mutual Exclusion and Simple Allocator case studies

Future work

- Consider more complex TLA^+ spec
- Automatic execution of translated cubicle's input file
- Translation of Cubicle counter-examples back into TLA^+ state sequences

Thank You!