CAV 2022 Protocol Conversion

Ian Dardik

April 8, 2022

1 Converting MLDR From TLA+ To Ivy

Converting the *MongoLoglessDynamicRaft* protocol from TLA+ to Ivy was a nontrivial exercise. In this appendix we describe the conversion process as well as the challenges we faced.

1.1 Protocol Conversion

The *MongoLoglessDynamicRaft* protocol that is introduced in [1] is encoded as a TLA+ specification. The specification encodes the STS using TLA+ constructs; the sort *Server* is encoded as a CONSTANT, each state variable is encoded as a VARIABLE, and the initial constraint and transition relation are each encoded as a TLA+ *operator*. In addition, the TLA+ spec also includes several auxiliary operators that make the spec more readable. We can convert by mapping the sort *Server* to Ivy's *type* construct, the state variables to Ivy *functions*, the initial constraint and transition relation to Ivy *actions*, and the auxiliary operators to relations. We describe each mapping in a separate section below, except for the sort *Server* because it maps directly to an Ivy *type*. We provide a summary of the mapping below:

TLA+		Ivy
Sort	$\rightarrow$	Type
State Variables	$\rightarrow$	Functions
Initial Constraint Operator	$\rightarrow$	"after init" Action
Transition Relation Operator	$\rightarrow$	Actions
Auxiliary Operators	$\rightarrow$	Relations

1.1.1 State Variable Mapping

TLA+ encodes state variables using the VARIABLE keyword. Ivy does not have a designated keyword for declaring state variables; instead we use functions. Ivy is a typed language so we must assign a type to each function. TLA+ is not typed, however we do encode a constraint on the universe of values for each state varaible in the *TypeOK* property. We use this property as guide for implementing each state variable as an Ivy function.

$TypeOK ==$	Ivy translation:
$\wedge\ currentTerm \in [Server \rightarrow \mathbb{N}]$	function $currentTerm(S : server) : nat$
$\wedge\ state \in [Server \rightarrow \{Secondary, Primary\}]$	function $state(S : server) : state\_type$
$\wedge\ config \in [Server \rightarrow \text{SUBSET } Server]$	function $config(S : server) : conf$
$\wedge\ configVersion \in [Server \rightarrow \mathbb{N}]$	function $config\_version(S : server) : nat$
$\wedge\ configTerm \in [Server \rightarrow \mathbb{N}]$	function $config\_term(S : server) : nat$

This mapping implies that we also need three more types: *nat*, *state_type*, and *conf*, which we encode as Ivy types.

1.1.2 Initial Constraint And Transition Relation Mapping

The initial constraint is encoded as a conjunction of constraints on the state variables in TLA+. This remains the same in Ivy; we simply wrap the constraint in a "after init" action. The transition relation for MLDR–as is typical in TLA+–is encoded as a disjunction of several individual transitions. In Ivy, we enumerate each individual transition as an exported action. Transition guards are described in TLA+ with a constraint on non-primed variables, while guards in Ivy are described using the "assume" keyword to describe any constraints/preconditions for actions. Transitions in TLA+ are described symbolically as a relation between unprimed and primed state variables, while transitions in Ivy are described via assignment. The translations are straightforward.

1.1.3 Auxiliary Operator Mapping

Operators in TLA+ can be encoded as relations with definitions. Give some examples. In particular, give quorum_overlap as an example because we will hone in on this in a later section.

1.2 Inductive Invariant Conversion

This was straightforward, especially given the auxiliary operators that were previously translated to Ivy. Unfortunately, our inductive invariant does not fall into Extended EPR. TODO did we try to put it into EPR at all?

1.3 Challenges

In this section we cover the challenges we faced during conversion.

1.3.1 EPR

Fitting MLDR into EPR was especially tough because of the quorum overlap property that must be encoded into MLDR. They key here is to separate the type for configs and quorums, even though their types are both subsets of *Server*.

1.3.2 Testing Our Ivy Spec

Once we completed the conversion process, it was not clear how we might test our spec since our inductive invariant puts our spec outside of EPR. One option would be to develop a new inductive invariant within EPR using Ivy, but this is a costly process and it is not clear whether we are guaranteed to find such an inductive invariant. Instead we gained intuition by proving simple properties, and by confirming our inductive invariant "works" in IC3PO for sufficiently large sort sizes.

References

[1] William Schultz, Siyuan Zhou, Ian Dardik, and Stavros Tripakis. Design and Analysis of a Logless Dynamic Reconfiguration Protocol. In Quentin Bramas, Vincent Gramoli, and Alessia Milani, editors, *25th International Conference on Principles of Distributed Systems (OPODIS 2021)*, volume 217 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 26:1–26:16, Dagstuhl, Germany, 2022. Schloss Dagstuhl – Leibniz-Zentrum für Informatik.