CAV 2022 Protocol Conversion

Ian Dardik

April 12, 2022

1 Converting MLDR From TLA+ To Ivy

Converting the *MongoLoglessDynamicRaft* (MLDR) 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 MLDR Protocol is introduced in [4] accompanied with a TLA+ specification. We converted this TLA+ specification to Ivy by creating a direct mapping of keywords and concepts in TLA+ to the appropriate construct in Ivy. We targeted Ivy 1.8 in our mapping, the most recent version of Ivy to date. The mapping is summarized at a high level in Figure 1.1. We provide a detailed explanation of each individual mapping in the sections below.

1.1.1 Sorts And Types

MLDR is parameterized by a single sort *Server*, denoted in TLA+ using the CONSTANT keyword which maps to the `type` keyword in Ivy. We also include a `state_type` type in Ivy to denote primary and secondary servers, as well as a `nat` type to represent natural numbers with some barebones axioms and operators, including `succ`. We also include a separate `quorum` and `conf` type to represent quorums and configs, even though these two types are conceptually identical. We will discuss the need to separate types for quorums and configs in Section 1.3.1.

1.1.2 State Variable Mapping

State variables are encoded in TLA+ using the VARIABLE keyword, and must be translated to functions or relations in Ivy. Ivy, unlike TLA+, is a typed language so we must assign a type to each function and relation. While TLA+ is not typed, the MLDR protocol includes a standard *TypeOK* property that constrains the universe of values for each state variable. We use this property as a guide for state variable conversion, which we show in Figure 1.1.2.

TLA+		Ivy 1.8
Sort	→	Type
State Variables	→	Functions
Initial Constraint Operator	→	`after init`
Transition Relation Operator	→	Actions
Auxiliary Operators	→	Relations

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

1.1.3 Initial Constraint And Transition Relation Mapping

The initial constraint is mapped from a conjunction of constraints in TLA+ to a sequence of constraints in Ivy, wrapped in a after init block. The transition relation is encoded in TLA+ as a disjunction of individual action, each of which map to an exported Ivy action block. Within each action, guards are described in TLA+ with a constraint on non-primed variables, and map to the assume keyword. The actions themselves are described symbolically in TLA+ as a relation between unprimed and primed state variables, and map to assignment of functions and relations in Ivy.

1.1.4 Auxiliary Operator Mapping

In TLA+ it is standard to wrap the initial constraint, the transition relation, and each individual action in an operator. However, operators are often used quite liberally as a means to make a specification modular and readable. Any operator that does not refer to a standard TLA+ construct–such as the initial constraint, TypeOK, etc.–we will refer to as an *auxiliary operator*. Auxiliary operators directly map to functions and relations in Ivy. The mapping is straightforward, except for the quorum_overlap operator which we describe in more detail in section 1.3.1.

1.2 Inductive Invariant Conversion

The work in [3] includes a TLA+ inductive invariant for the *MongoRaftReconfig* (MRR) protocol. MRR is a large protocol that is composed of two subprotocols, one of which is MLDR. We use the conjuncts of the inductive invariant for MRR that exclusively reference the state variables of MLDR to create an inductive invariant specifically for MLDR. Unfortunately, this inductive invariants includes a cycle in the quantifier alternation graph, and hence falls outside of Extended EPR [2]. Although techniques exist that may fit a protocol into EPR–such as splitting dependent code into modules [1]–these techniques are expensive and, in general, are not necessarily guaranteed to work; this is because modeling a protocol from an aribtrary logic into EPR, and more generally FOL, cannot be guaranteed to be complete [2]. Instead of continuing to manually find an inductive invariant in EPR using Ivy, we left this task for the automatic synthesis tools.

1.3 Challenges

During the conversion process we faced two main challenges: fitting MLDR into EPR and validating the translation. In this section we discuss these two challenges as well as our solutions.

1.3.1 EPR

Fitting MLDR into EPR was not obvious at first given that we are not experts in this area. One of the large issues was correctly modeling quorums and configurations without a powerset operator; once we figured out how to model this cleanly, the protocol fell into EPR.

1.3.2 Validating The Translation

Once we completed the conversion process, it was important to test the Ivy specification to make sure it is correct. The best litmus test for correctness would be to encode the inductive invariant into Ivy and confirm that it holds. Unfortunately, the inductive invariant causes the protocol to fall outside of EPR, so we needed an alternative. Instead, we chose to include two sanity check proofs and leverage IC3PO to speculate whether our inductive invariant is correct based on its finite convergence checks.

1.3.3 Do Ivy Synthesis Tools Have A Fighting Chance?

TLA+ is a rather expressive language where we can explicitly create complicated constructs, including second order formulas. For example, quorums and configurations are key concepts in MLDR, both of whose types are the power set of Server. In TLA+, the power set–and hence quorums and configurations–can be explicitly defined by using the SUBSET keyword. In Ivy, however, there is no support for power sets or second order logic; instead, a protocol designer can use a binary *member* relation on quorums and configurations to define key axioms that Ivy can use for proving theorems.

The Ivy method–in which we provide key axioms instead of explicitly defining key higher order constructs–begs an interesting question: how do we know that we know that the axioms that we have provided will be sufficient for an automatic invariant synthesis tool to succeed? When using Ivy to create an inductive invariant by hand, presumably it will be clear that more axioms are needed during the Ivy workflow. When the task is handed to a synthesis tool, this workflow becomes opaque which implies that insufficient user provided axioms must be added to the long list of possible causes upon failure.

Furthermore, SWISS and DistAI both operate within EPR, and it may be the case that IC3PO and FOL-IC3 operate more efficiently in EPR. However, even in the case that a protocol and its key safey property fall into EPR, it is not clear whether an inductive invariant can be synthesized in EPR. It is possible that *every* possible inductive invariant introduces a quantifier alternation graph cycle, and that the protocol must be split into separate modules as a prerequisite for *any* inductive invariant to possibly be in EPR. This is also an issue that would presumably be ironed out during manual search for an inductive invariant using the Ivy workflow, and now becomes opaque when handed to an auomatic synthesis tool.

References

[1] Kenneth L. McMillan and Oded Padon. Deductive verification in decidable fragments with ivy. In Andreas Podelski, editor, *Static Analysis - 25th International Symposium, SAS 2018, Freiburg, Germany, August 29-31, 2018, Proceedings*, volume 11002 of *Lecture Notes in Computer Science*, pages 43–55. Springer, 2018.

[2] Oded Padon, Giuliano Losa, Mooly Sagiv, and Sharon Shoham. Paxos Made EPR: Decidable Reasoning about Distributed Protocols. *Proc. ACM Program. Lang.*, 1(OOPSLA), oct 2017.

[3] William Schultz, Ian Dardik, and Stavros Tripakis. Formal Verification of a Distributed Dynamic Reconfiguration Protocol. In *Proceedings of the 11th ACM SIGPLAN International Conference on Certified Programs and Proofs*, CPP 2022, page 143–152, New York, NY, USA, 2022. Association for Computing Machinery.

[4] 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.