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 specification is introduced in [4] along with a TLA+ specification. We converted the specification to Ivy so we can compare *endive* with other invariant inference tools. We targeted Ivy 1.8, the latest version of Ivy to date, during our conversion process.

The conversion from TLA+ to Ivy can be summarized as a mapping from each component of a TLA+ specification to a component in an Ivy specification. We show the mapping at a high level in Figure 1.1, and provide detail in the sections below.

1.1.1 Sorts And Types

TLA+ is an untyped specification language while Ivy is typed, and thus we need to introduce several types. The first is for the sort *Server* that MLDR accepts as a parameter. The sort is declared in TLA+ using the CONSTANT keyword, and maps to the `type` keyword in Ivy. Second, we include a `state_type` type in Ivy to denote primary versus secondary servers. Third, we introduce a `nat` type to represent natural numbers, which we equip with barebones axioms and operators, including `succ`. Finally, we include the types `quorum` and `conf` to represent quorums and configurations. While quorums and configurations are conceptually the same type–the power set of *Server*–we found that separating the two types yields a cleaner design in Ivy.

1.1.2 State Variables

State variables are encoded in TLA+ using the VARIABLE keyword, and map to functions or relations in Ivy. MLDR includes a *TypeOK* property that places constraints on the universe of values for each state variable; we use this property as a direct guide in our 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:
$\land currentTerm \in [Server \to \mathbb{N}]$	function $currentTerm(S : server) : nat$
$\land state \in [Server \to \{Secondary, Primary\}]$	function $state(S : server) : state\_type$
$\land config \in [Server \to \text{SUBSET } Server]$	function $config(S : server) : conf$
$\land configVersion \in [Server \to \mathbb{N}]$	function $config\_version(S : server) : nat$
$\land configTerm \in [Server \to \mathbb{N}]$	function $config\_term(S : server) : nat$

1.1.3 The Initial Constraint And Transition Relation

The initial constraint is encoded in TLA+ as a conjunction of individual constraints on state variables, and maps to a sequence of individual constraints in Ivy, wrapped in an `after init` block. The transition relation is encoded in TLA+ as a disjunction of individual actions, each of which map to an exported Ivy `action` block. Within each action, guards are specified in TLA+ as a constraint on unprimed variables, and map to the `assume` keyword in Ivy. The actions themselves are described symbolically in TLA+ as a relation between unprimed and primed state variables, and map to assignment operators on functions and relations in Ivy (recall that functions and relations represent state variables in Ivy).

1.1.4 Auxiliary Operators

In TLA+ it is standard to wrap the initial constraint, transition relation, and each individual action in an operator. However, operators are also often used as "helpers" as a means to make a specification modular and readable. We will refer to any such "helper" operator as an *auxiliary operator*. Auxiliary operators directly map to functions and relations in Ivy.

1.2 Inductive Invariant Conversion

Endive successfully synthesized an inductive invariant for MLDR which we tried to convert from TLA+ to Ivy. Unfortunately, this inductive invariant introduces a cycle in the quantifier alternation graph of the MLDR Ivy specification, and hence falls outside of Extended EPR [2]. Although techniques exist that may fit a protocol into EPR–such as splitting code with mutually dependent types into modules [1]–these techniques may require significant effort and, in general, are not guaranteed to be complete [2]. Instead of manually using Ivy to find an inductive invariant in Extended EPR, we left this task for the automatic synthesis tools.

1.3 Challenges

During the conversion process we faced three main challenges which we discuss below.

1.3.1 EPR

Designing MLDR to fit into Extended EPR was not clear at first given that we are not experts in this area. One of the key hurdles we faced was modeling quorums and configurations without a power set operator; we ultimately converged on using separate types for quorums and configurations which led to a clean solution that fit into Extended EPR.

1.3.2 Validating The Translation

Once we completed the conversion process, we sought to verify that the Ivy specification was correct. The ideal test would have been to encode the inductive invariant that *endive* synthesized to prove correctness, yet unfortunately, the inductive invariant places the specification outside of Extended EPR. Instead, we leveraged IC3PO for our sanity checks. IC3PO may accept a protocol outside of FAU, and hence we were able to pass the MLDR Ivy specification along with *endive*'s synthesized inductive invariant. We performed sanity checks by running IC3PO for several finite instances and confirming that it terminated successfully without adding additional invariants. This sanity check gave us confidence that our conversion process for MLDR was successful.

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.