CAV 2022 Protocol Conversion

Ian Dardik

No Institute Given

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] 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, and provide detail in the sections below.

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

Fig. 1. High Level Conversion Mapping

Sorts And Types TLA+ is untyped while Ivy is typed, and thus we need to introduce several types into the new Ivy specification. The first type 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 `t`ype keyword in Ivy. Second, we include a `s`tate_type type in Ivy to denote primary versus secondary servers. Third, we introduce a `n`at type to represent natural numbers, which we equip with total order axioms and the `s`ucc operator. Finally, we include the types `q`uorum and `c`onf 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.

State Variables State variables are encoded in TLA+ using the VARIABLE keyword, and map to functions and 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 2.

$TypeOK ==$	Ivy translation:
$\land currentTerm \in [Server \to \mathbb{N}]$	function $current\_term(S : server) : nat$
$\land state \in [Server \to \{Secondary, Primary\}]$	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 $config\_term(S : server) : nat$

Fig. 2. State Variable Conversion Mapping

The Initial Constraint And Transition Relation The initial constraint is encoded in TLA+ as a conjunction state variables constraints, and maps to Ivy as a sequence of function constraints wrapped in an `after init` block. The transition relation is encoded in TLA+ as a disjunction of actions, each of which maps to an exported Ivy `action` block. Within each action, guards are specified in TLA+ as a constraint on unprimed state variables, and map to the `assume` keyword in Ivy. The actions 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).

Helper 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. These helper 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 [3]. Although techniques exist that may fit a protocol into EPR–such as splitting code with mutually dependent types into modules [2]–these techniques may require significant effort and, in general,

are not guaranteed to be complete [3]. 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 two main challenges: fitting MLDR into Extended EPR and validating our newly translated Ivy specification. We discuss these two challenges below,and incldue a higher level discussion afterward.

EPR Designing MLDR to fit into Extended EPR was not simple, considering that we are newcomers to Ivy. 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 resulted in the quantifier alternation graph $\mathtt{conf}\rightarrow\mathtt{quorum}\rightarrow\mathtt{server}$. This graph is stratified, and hence fits into Extended EPR.

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 as we noted in section 1.2. Instead, we leveraged IC3PO for our sanity checks. IC3PO may accept a protocol outside of Ivy's decidable fragment (FAU [1]), 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.

Discussion TLA+ is a rather expressive language where we can explicitly create a protocol and model check safety properties to gain confidence. Ivy, on the other hand, is a more restricted language, and forces protocol designers to write complex code within an individual module in a more implicit fashion. This implicit fashion entails defining bear bones constructs and providing key axioms–this workflow is aimed at keeping each construct powerful enough to prove key properties, but light enough to remain in the decidable fragment FAU. For example, consider quorums in the MLDR protocol, whose type is the power set of *Server*. In TLA+, we can explicitly use a power set operator, making quorums simple to define. In Ivy, however, there is no support for power sets; instead, a protocol designer can define a binary $member \in \mathtt{server}\times\mathtt{quorum}$ relation as a means for providing key axioms that will be necessary for any proofs about quorums.

The implicit definition workflow of Ivy begs an important question: how does a protocol designer know when she has provided sufficent axioms on implicitly defined types in order to prove key theorems? Presumably, the designer who

uses Ivy's interactive workflow to prove theorems can discover when the set of axioms are insufficient. Invariant inference tools, however, provide little to no insight into whether a failure is due to insufficient axioms versus other potential reasons.

Furthermore, it is not clear that a protocol in EPR is guaranteed to have an inductive invariant that is also in EPR. For any tool that exclusively operates in EPR–or even *works better* in EPR–the tool may experience issues if it is tough, or even impossible, for an inductive invariant to exist in EPR. It may be the case that the protocol must be split into organized modules in order for *any* inductive invariant to fit into EPR. In this case, a designer might naturally split a protocol into organized modules by following the Ivy workflow, while invariant inference tools offer little to no insight.

We do not know the answer to the questions posed above, however, we consider these aspects to be potential limmitations of any tool that operates within EPR or relies on SMT solvers. Thus we believe that, to the best of our abilities, we provide a fair comparison for the MLDR protocol between *endive* and any tool that accepts the Ivy version as input.

References

1. Yeting Ge and Leonardo Moura. Complete instantiation for quantified formulas in satisfiabiliby modulo theories. In *Proceedings of the 21st International Conference on Computer Aided Verification*, CAV '09, page 306–320, Berlin, Heidelberg, 2009. Springer-Verlag.
2. 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.
3. 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.
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.