A TLA+ validation of the Chord protocol

Jean-Paul Bodeveix[1], Julien Brunel[2], David Chemouil[2], and Mamoun Filali[1]

[1] IRIT CNRS UPS, Université de Toulouse, France, first.last@irit.fr
[2] ONERA DTIS, Université de Toulouse, France, first.last@onera.fr

In this talk, we report on the mechanization of a substantial case study: the Chord protocol [Zav17, BCT18], through the shallow embedding of TLA+ [Lam02] within the Isabelle-HOL logic [Mer99]. This framework was used, vs TLAPS [CDL+12], because in our work we are mainly interested by liveness properties. To the best of our knowledge, these properties are today better supported by the definitions and the proof rules available in [GM11]. Although, we have already considered the validation of the maintenance operations of the Chord protocol [BBCF19], this work goes further with respect to the proof within the TLA+ logical framework of the liveness properties relying on explicit fairness assumptions. Moreover, this proof relies on new stability steps. Our talk will be structured along the following lines:

1 Introduction

The Chord protocol. Chord is a popular protocol but also, since the seminal work of Zave [Zav], a popular test-bed for formal verification. However, most work has focused on proving safety, sometimes with manual proofs only, while the correctness property for Chord maintenance, addressed here, is a liveness property. Zave [Zav17] carried out a manual proof of liveness and discovered the fundamental notion of *principal*. Informally, a principal is a node that is not skipped by any pair of linked nodes. Then, the correctness of Chord is stated as follows:

- safety: the successor nodes of a node is not empty and the set of principals is not empty.
- stability. When no more joins of fails occur, all the live nodes : members, are eventually linked through a unique ring. Each node successor list is correct with respect to the member nodes.

2 Mechanization engineering

The TLA+ logic. In TLA+ [Lam94], a formula is valid if it is true over all behaviours. A set of proof rules, named, STL1, ..., STL6, TLA1, TLA1, Lattice are proposed for the verification of TLA+ models.

TLA+ mechanization. Within Isabelle-HOL, TLA+ is defined over infinite sequence of states represented as the type $\mathbb{N} \to 'st$ where 'st is the type of states. For proof automation, in order to eliminate this higher order type, we have found convenient to make explicit the state predicate or the action predicate from which a behaviour is generated.

A kernel for TLA+. Again for automation purposes, we have found convenient to express TLA+ actions as guarded commands and elaborate some rules to reason about them [Hes13]. Moreover, for expressing stabilization, Unity [CM89] style predicates, e.g., `stable`, have been encoded and used in proof rules.

Liveness rules. For proving liveness properties usually expressed as a $P \leadsto Q$ property, TLA+ proposes the two rules: WF1 (resp. SF1) establishes such a property under weak (resp. strong fairness) hypotheses. In this study, we will consider the WF1 rule expressed as follows[3]:

$$\frac{\begin{array}{l} P \wedge [\mathcal{N}]_f \Rightarrow (P' \vee Q') \\ P \wedge \langle \mathcal{N} \wedge \mathcal{A} \rangle_f \Rightarrow Q' \\ P \Rightarrow Enabled\langle \mathcal{A} \rangle_f \end{array}}{\Box[\mathcal{N}]_f \wedge \mathrm{WF}_f(\mathcal{A}) \Rightarrow (P \leadsto Q)}$$

Considering that the action $\mathcal{A}$ is a guarded command, the expression $Enabled\langle \mathcal{A} \rangle_f$ can be transformed to $guard_{\mathcal{A}} \wedge f' \neq f$. Thanks to our mechanization, it has been possible to proof meta theorems to facilitate temporal reasoning.

[3] Its mechanization is given in appendix A.3. A primed variable, e.g., P', denotes the value of the variable in the next state.

3 TLA+ validation of Chord

3.1 The TLA+ Chord model

Chord data structure. It consists in an array of node states. Each node state defines : a list of successor nodes, a predecessor node, an inbox for the set of delivered messages, a program counter and for modeling purposes: the node and the network status.

Chord transitions. Following the model of Pamela Save [Zav17], we define the transitions `stabilize`, `from_successor`, `from_predecessor`, `rectify`, `fail`, and `join` as TLA+ actions. Each transition is decomposed as a guarded command. Moreover, in order to model that the system has reached a (virtual) stability point, we introduce the action `no_more_join_or_fail`.

Chord liveness. We express here that each of the transitions `stabilize`, `from_successor`, `from_predecessor`, `rectify` that is eventually always enabled is eventually fired as weak fairness properties. Moreover, we state that each remaining delivered message is eventually received.

3.2 Chord validation

Safety validation. This is a usual proof showing that each transition[4] preserves the invariant.

Stability validation. The stability proof relies mainly on principals propagation once the stabilization has been reached. Actually, we show that, at stabilization, once a node is principal, its predecessor eventually becomes principal. This proof establishes also that eventually the principal has a correct predecessor. With respect to a principal p, after stabilization, the back propagation of the principal status is decomposed into the following steps: the first element of each successor list becomes a member, the identity of the previous node of p is delivered p, the previous node of p becomes its predecessor, the previous node of p becomes principal.

Thanks to the invariant that the set of principals is not empty, it follows by induction over the finite physical ring that each node eventually becomes principal and each node has a correct predecessor. Then, we establish the correctness of the remaining elements of each successor list.

References

BBCF19. Jean-Paul Bodeveix, Julien Brunel, David Chemouil, and Mamoun Filali. Mechanically verifying the fundamental liveness property of the Chord protocol. In Maurice H. ter Beek, Annabelle McIver, and José N. Oliveira, editors, *Formal Methods - The Next 30 Years - Third World Congress, FM 2019, Proceedings*, volume 11800 of *Lecture Notes in Computer Science*, pages 45–63. Springer, 2019.

BCT18. Julien Brunel, David Chemouil, and Jeanne Tawa. Analyzing the Fundamental Liveness Property of the Chord Protocol. In *Formal Methods in Computer-Aided Design*, Austin (USA), October 2018.

CDL+12. Denis Cousineau, Damien Doligez, Leslie Lamport, Stephan Merz, Daniel Ricketts, and Hernán Vanzetto. TLA+ proofs. In *FM 2012: Formal Methods - 18th International Symposium, Paris, France, August 27-31, 2012. Proceedings*, pages 147–154, 2012.

CM89. K. Mani Chandy and Jayadev Misra. *Parallel program design - a foundation.* Addison-Wesley, 1989.

GM11. Gudmund Grov and Stephan Merz. A definitional encoding of TLA* in Isabelle/HOL. *Arch. Formal Proofs*, 2011, 2011.

Hes13. Wim H. Hesselink. Complete assertional proof rules for progress under weak and strong fairness. *Sci. Comput. Program.*, 78(9):1521–1537, 2013.

Lam94. Leslie Lamport. The temporal logic of actions. *ACM Trans. Program. Lang. Syst.*, 16(3):872–923, May 1994.

Lam02. Leslie Lamport. *Specifying Systems, The TLA+ Language and Tools for Hardware and Software Engineers.* Addison-Wesley, 2002.

Mer99. Stephan Merz. An encoding of TLA in Isabelle. 1999.

Zav. Pamela Zave. Web site Pamela Zave How to Make Chord Correct. http://www.research.att.com/people/Zave_Pamela. Accessed: 2016-09-29.

Zav17. Pamela Zave. Reasoning about identifier spaces: How to make Chord correct. *IEEE Transactions on Software Engineering*, 43(12):1144–1156, Dec 2017.

[4] including `join` and `fail`

The aim of the appendix is to give some elements of our ongoing mechanization on top of the TLA+ framework [GM11].

A Mechanization

A.1 State and action predicate

The following higher order definitions : p_T(resp. r_T) transforms a predicate (of type $'st \Rightarrow bool$), resp. a relation (of type $'st \Rightarrow' st \Rightarrow bool$) into a temporal predicate.

```
definition p_T :: "('st ⇒ bool) ⇒ 'st statefun ⇒ temporal"
  where "p_T p v = TEMP p <$v>"  — p is a predicate on the current state $v

definition r_T  :: "('st ⇒ 'st ⇒ bool) ⇒ 'st statefun ⇒ temporal"
  where "r_T r v = TEMP r <$v,v$>"  — r is a relation over the current state $v and the next state v$
```

A.2 Enabledness

```
theorem  Enabled_act:
  assumes v: "basevars v"
  shows "⊢ Enabled ⟨g<$v> ∧ v$ = c<$v>⟩_v  = (g <$v> ∧ c<$v> ≠ $v)"
```

A.3 Weak Fairness

The WF1 rule is expressed [GM11] as follows:

```
theorem WF1:
  assumes h1: "|~ P ∧ [N]_v ⟶ ⊙P ∨ ⊙Q"
      and h2: "|~ P ∧ ⟨N ∧ A⟩_v ⟶ ⊙Q"
      and h3: "⊢ P ⟶ Enabled ⟨A⟩_v"
      and h4: "|~ P ∧ Unchanged v ⟶ ⊙P"
    shows "⊢ □[N]_v ∧ WF(A)_v ⟶ (P ⇝ Q)"
```

A.4 Liveness rule

An instance of liveness rules is the following validated theorem:

```
theorem safety_stable_leadsto'':
  assumes N: "⊢ S ⟶ □[r_T N v]_v"
  assumes st: "stable N P"
  assumes lt: "⊢ S ⟶ p_T P v ∧ A ⇝ B"
  shows "⊢ S ⟶ p_T P v ∧ A ⇝ p_T P v ∧ B"
```

B Chord model

B.1 Chord data structure

```
datatype pc_chord = Idle  | FromSuccessor | FromPredecessor
record state =                          — state of a node
  member :: "bool"                      — is the node alive ?
  sl :: "nat list"                      — successor list
  prdc :: "nat"                         — predecessor
  inbox :: "nat set"                    — box of delivered messages
  pc :: pc_chord                        — program counter
  no_more_join_or_fail :: "bool"        — network status
```

B.2 Chord transitions

B.2.1 The `stabilize` transition

```
definition (in Chord) "stabilize self (St::State) (St'::State) =
  (  pc (St self) = Idle
   ∧ self ∈ members St
   ∧ sl (St self) ≠ []
   ∧ (  Query_prdc_sl self (First St self) FromSuccessor St St'
      ∨ (First St self ∉ members St
        ∧  St' = St(self := (St(self))(|
                     pc := Idle,
                     sl := (tl (sl (St self)))@[suc N (List.last (sl (St self)))]  |)))))"

definition (in Chord) "stabilize_guard self St =
                         (pc (St self) = Idle ∧ self ∈ members St ∧ sl (St self) ≠ [])"

definition (in Chord) "stabilize_command self St =
       (if First St self ∈ members St then
           St(self := (St(self))(|
                     pc := FromSuccessor |))
        else St(self := (St(self))(|
                     pc := Idle,
                     sl := (tl (sl (St self)))@[suc N (List.last (sl (St self)))]  |)))"

theorem (in Chord) stabilize_gc:
  shows "stabilize self = gc (stabilize_guard self) (stabilize_command self)"
```

B.3 The next relation

```
definition (in Chord) "Chord_next self =
  (tr_Dl [λ St St'. stabilize self St St',
          λ St St'. from_successor self St St',
          λ St St'. from_predecessor self St St',
          λ St St'. ∃ n_m ∈ Nodes N. rectify self n_m St St',
          λ St St'. fail self St St',
          λ St St'. ∃ newPrdc ∈ Nodes N. join self newPrdc St St',
          λ St St'. no_more_join_or_fail_act St St'])"
```

B.4 Liveness

```
definition (in Chord) Liveness_T  :: "State statefun ⇒ temporal"
  where " Liveness_T v = TEMP (∀ self. #self ∈ Nodes<#N> ⟶WF(r_T (stabilize self) v)_v)
                       ∧ (∀ self. #self ∈ Nodes<#N> ⟶WF(r_T (from_predecessor self) v)_v)
                       ∧ (∀ self. #self ∈ Nodes<#N> ⟶ WF(r_T (from_successor self) v)_v)
                       ∧ (∀ self n_m. #self ∈ Nodes<#N> ⟶ WF(r_T (rectify self n_m) v)_v)"
```

B.5 The Chord model specification

```
definition (in Chord) "Spec_T v =  TEMP (Init_T v ∧ □[r_T Chord_Next v]_(v) ∧ Liveness_T v)"
```

C Chord correctness

C.1 Chord safety

```
definition "Invariant (St::State) =
    (TypeInvariant St ∧ AtLeastOneLiveSuccessor St ∧ PrincipalsNotEmpty St)"
```

C.2 Chord stability

definition *(in Chord) "correctness St =*
(Invariant St
∧ No_more_join_or_fail St
∧ (∀ p ∈ members St. First St p = sucNode (members St) p)
∧ (∀ p ∈ members St. prdc(St p) = prevNode (members St) p)
∧ (∀ p ∈ members St. ∀ j. 0 < j ∧ j < L ⟶
Nth St p j = ((sucNode(members St))^^(j+1))(p))
)"

D Chord validation

D.1 Safety

D.1.1 Assertionnal version

theorem *(in Chord) Invariant_THM:*
shows *"Invariant St ∧ Chord_Next St St' ⟶ Invariant St'"*

D.1.2 Temporal version

theorem *(in Chord) Spec_Inv:*
shows *"⊢ Spec_T v ⟶ □(p_T Invariant v)"*