Property-agnostic base case extension for scalable verification of distributed systems

Kyle Storey and Eric Mercer[0000−0002−2264−2958]

Brigham Young University, Provo UT 84602, USA
`kyle.r.storey@gmail.com,egm@cs.byu.edu`

Abstract. Many distributed systems require temporal properties to hold for correctness. Model checking can verify these properties on a small system but it doesn't scale for arbitrarily large systems. This work presents a new method for proving that temporal properties verified on a small system extend to an arbitrarily large system when that system has a ring topology. It uses a model checker to prove temporal properties and properties of a partial order of events in the system. It then admits the partial order properties as axioms in a theorem prover and proves a conformance relation between an arbitrary-sized ring of nodes and the model-checked base case. The conformance relation is used to prove that adding a new node to the ring does not affect the possible states of the existing nodes in the system and therefore any properties proven in the small system continue to hold in an arbitrarily large system.
We demonstrate the approach in a case study of a nontrivial distributed protocol that is used by the MyCHIP's digital currency to clear credit.

Keywords: Model checking, interactive theorem provers, formal verification, distributed systems, digital currency

1 Introduction

We present an approach to verification that combines model checking with an interactive theorem prover to verify temporal properties in an arbitrarily large distributed system used to clear credit in a digital currency called MyCHIPs [4].

We model the system with a small fixed number of nodes and verify that the required temporal properties hold. We define a partial order of events with added constraints on which events must be present, parameterized by the size of the system. (Collectively referred to as the *partial order*.) We verify that the *partial order* is respected in the fixed-size system.

In the interactive theorem prover, we use the *partial order* to define the set of possible messages in the system. We construct an inductive proof that the set of messages observed by each node does not change with the number of participating nodes.

We use *conformance* as described by Dill [9] to show that if the set of observed messages for each node is unchanged then the set of internal states of each node is unchanged, which implies that the states considered by the model checker is the complete set of states for nodes in a system of any size and therefore the properties proved in the model checker generalize to the larger system.

MyCHIPs is a novel digital currency that facilitates the trade of goods and services between trusting partners. MyCHIPs has some similarities to time banking systems [22] [18] but introduces a novel debt-clearing protocol, called a *credit lift*, to completely decentralize trade. We verify in this work that the distributed protocol to implement the credit lift preserves the security and functionality of the MyCHIPs digital currency.

A credit lift (*lift* for brevity) in the MyCHIPs protocol operates on a group of nodes arranged in a cycle such that each node is in debt to its successor in the cycle. The protocol arranges for each to forgive their predecessor's debt, and in exchange, their own debt is forgiven by their successor. Each node's debts are recorded in a *tally*, which is a record of transactions that is kept in consensus between its trading partners. The *tally* uses *CHIPs* as a unit of measure for the debt with each *CHIP* being defined to be equal to the value of one hour of an unskilled laborer's time.

The lift operates over two stages once a mutually beneficial cycle of trading nodes is discovered. First, each node *promises* to send CHIPs to their predecessor. And then second, after all have promised, each node *commits* by sending the requisite CHIPs. At the end of the lift each node's *balance*—the difference between the number of sent CHIPs and the number of received CHIPs—remains the same, but all the nodes have reduced their debt liability.

What makes the credit lift complex is that arbitrary nodes may lose connectivity or otherwise become *inactive* at any point in the protocol thus leaving lifts unfinished indefinitely. As such, each lift is given a time limit, represented by a timestamp at which time the lift can no longer be committed. Because we don't expect nodes on the network to have synchronized clocks, a Referee is appointed whose clock is considered authoritative and acts as a consensus object. When a node requests to commit before the timeout, the Referee provides a digital signature that is proof that the lift has been committed. If the lift is not committed before the timeout, the lift becomes *nullified* and the Referee digitally signs a statement to that effect.

1.1 Verified properties and assumptions

The key properties that underpin the security and functionality of the lift protocol, which we prove, are the following:

1. Lifts always eventually are committed or nullified for every active node.
2. At the final state of the lift, every active node agrees that the lift was committed or every active node agrees the lift was nullified.
3. The balance of every active node on the final state of the lift is equal to or greater than its initial balance.
4. Every active pair of nodes on the final state of the lift agree on their shared tally.

Our verification results are subject to the following assumptions that are explicit in the proofs:

– Only a single lift is in flight with every node in the cycle being distinct.
– Messages may experience arbitrarily long delays except that messages to and from the Referee are always eventually received.
– All nodes follow the protocol though they may do so with arbitrarily long delays between actions again with the exception that the Referee always eventually takes the next action

1.2 Tools and approach

We use the Spin Model Checker to prove these properties hold in a system with one Originator—the node that initiates the lift algorithm—one Referee, and one Relay node—a node that participates in the lift that is not the Originator or the Referee. We then use Coq for our interactive theorem prover and produce an inductive proof that the temporal properties verified in Spin extend to an arbitrarily large system.

In Dill's work, he proves that if one subsystem conforms to another, then it can be safely substituted for the other. By safe substitution, we mean that any behavior of the new subsystem that could cause a failure is also a behavior of the original subsystem. Using the model checking results as a base case, we prove that the set of traces produced by a chain of $n + 1$ Relay nodes conforms to the set of traces produced by n Relay nodes and inductively argue that the properties verified by the model checker hold for any number of nodes.

We admit that using model checkers to verify properties of systems with arbitrarily large or even infinite state has been a common topic within the model-checking communities. To our knowledge, the various published methods reason inductively on the calculus used within the model checker either automatically [12][21], or manually [7]. Our proof approach here deviates from these works in two key ways:

1. We use conformance as the theoretical foundation for the inductive proof
2. We use distinct and separate representations of the nodes for the model checking and the theorem proving, and we use a partial conformance equivalence to provide strong assurance that these representations are interchangeable.

Using this new method in our approach gives two key advantages:

1. New properties may be verified in the system through model checking without needing to change the inductive proof.
2. Using a different representation for the inductive proof allows for more flexibility in proof tactics and structure.

For this method to be sound and complete the representation used in the model checker and the representation used in the inductive proof must be interchangeable. Proving conformance equivalence between these two representations is sufficient to prove interchangeability. However, this work only succeeds in formally proving a partial conformance equivalence. We do however, provide evidence that gives strong assurance of conformance equivalence sufficient for our needs. We leave a method to formally prove conformance equivalence between these representations to future work.

A repository with our Spin models and Coq proof script along with instructions to run the analysis is can be found at https://github.com/kylona/VMCAI-2025

1.3 Organization

The rest of this paper is organized as follows: Section 2 defines the lift protocol. Section 3 is the inductive proof that generalizes the properties to any number of nodes. Section 4 summarizes the Spin verification of the base system. Section 5 describes the Coq proof for the induction. Section 6 gives strong assurance of conformance equivalence between the two representations used in Section 4 and Section 5. Section 7 discusses related work. And finally, Section 8 is the conclusion.

2 MyCHIPs Protocol

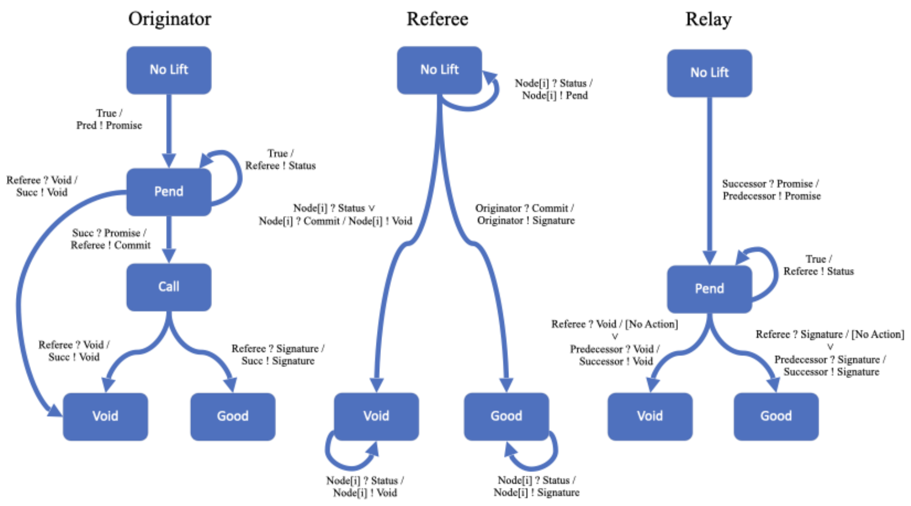


Fig. 1: Lift States Diagram

Figure 1 shows the Mealy machines that define the behavior for each type of node in the credit lift protocol: the Originator, the Referee, and any number of Relays [5]. Additional background for MyCHIPs can be found at the *gotchoices* website [4] or in Appendix B. Each box in Figure 1 represents the state of the lift from the perspective of the node type, and each arrow represents a change in state and an associated action that is taken during the transition from one state to the next. A '/' character delineates the condition that must hold for the transition to be taken from the action that is taken during the transition.

The state of each node type includes the network ID of its predecessor (Pred), its successor (Succ), and the Referee. A transition may be conditional on receiving a particular message from a particular network ID. This case is written as *sender* ? *message_ type*. Additionally, actions may transmit a particular message to a particular network ID. This case is written as *recipient* ! *message_ type*.

Each node in the lift begins in the *No Lift* state. The originator has an edge with *True* as its condition so at any time the originator can start the lift by sending a *Promise* to its predecessor in the cycle and transition to the *Pend* state. Each relay node when it receives this *Promise* message forwards the message to its predecessor and transitions to the *Pend* state.

The *Pend* state indicates that the node has not yet determined whether to commit or roll back the lift. At any time when a node is in the *Pend* state it can send a message to the Referee to ask if the lift has been completed. The referee will respond with one of three messages: *Void*, *Signature*, or *Pend*. The *Pend* message means no decision has been made, so this induces no state transition and no action. If the message is *Void* or *Signature*, this triggers the node to mark the lift as *Void* or *Good* respectively. This path through the states allows for lifts to be completed even when messages to and from peers in the cycle are unreliable. Note that transitions in this case have no action. So a node only propagates a *Void* or *Signature* message to its successor when it has received it from a peer, not from the referee.

If the messages to peers are successful, once the *Promise* message propagates around the cycle to the originator, this satisfies the transition condition so the originator can transition to the *Call* state. When it does so it sends a *Commit* message to the referee to request its signature. The referee can then return either a *Void* or *Signature* message. If *Void*, the originator transitions to the *Void* state. If *Signature*, the originator transitions to the *Good* state. In either case, it propagates the message to its successor. When these messages reach the relay nodes they similarly transition to *Void* or *Good* and forward the message to their successor. Eventually reaching the originator which ignores this message and the lift is complete.

3 Conformance Equivalence for Representation Switching

We would like to verify the stated properties of the lift protocol for a cycle of any number of nodes. To do this, we need to construct an inductive proof. The base case for the proof is the smallest complete system for which a lift can be performed: an Originator, a Referee, and a Relay. We generalize this step of the proof to an arbitrary set of properties and not just those stated in the introduction. For this generalization, we define a predicate $Prop(Ref, Orig, R_1, \cdots)$ that should only be true if the properties checked by the predicate hold in the given lift under all possible circumstances allowed by our assumptions. The lift itself is defined by the arguments to the predicate that name the Referee, Originator, and some sequence of one or more Relay nodes. The Originator is the head of the cycle for the lift and is followed by the sequence of relays.

Verifying properties in the base system is a natural fit for model checking. Spin is able to readily verify each of the properties stated at the outset of this paper by describing the Mealy machines directly in its input language Promela and expressing the desired properties in linear temporal logic. We state the result in the following lemma.

Lemma 1 (Base Case). *The base system consisting of one Referee, one Originator, and one Relay is correct.*

$$\mathit{Prop}(\mathrm{Ref}, \mathrm{Orig}, R_1) \text{ holds}$$

Proof. See Section 4

The question now is how to prove out the inductive step that generalizes the model checking results to arbitrary lifts? For that, we turn to theorem proving. We tried defining the state machines as Automaton and composing them using Athalye's framework[2], but it didn't scale.

Instead, we used trace sets as our representation because our prior work has shown theorem provers tend to reason well over sets of traces [17,14]. We manually extract from the Mealy machines a set of constraints on sequential traces of events in the lift. These extracted constraints define what traces are allowed by the lift while the Mealy machines define how such traces are generated.

We compare this relationship between the two representations to black-box and white-box testing. The model checker is white-box meaning that it uses the internal state of each node to reason about all reachable message traces, while the theorem prover is black-box meaning that it uses constraints on, or properties of, traces to reason about all message traces. We differentiate these two models in our notation with R to denote a node as defined in the model checker and B to denote a node as defined in the theorem prover.

To prove that these two representations are equivalent, we utilize Dill's work on trace theory. In Dill's work, he defines a sufficient condition for one subsystem to conform to another. Dill assumes that if a subsystem experiences an internal failure the system as a whole will experience a failure. However, the goal of our analysis is to prove the properties of the system as a whole hold even when individual nodes in the system may fail. All of the behavior of each node, including the behavior of a node that experiences an internal error is considered in the possible behavior and not considered a failure. This with this assumption we can reduce the necessary condition for conformance to:

Definition 1 (Conformance).

$$\mathit{Proj}(P(R), R') \subseteq P(R') \longrightarrow R \preceq R'$$

Where $P(R)$ and $P(R')$ are the sets of possible traces in R and R' respectively, and $\mathit{Proj}(T, R')$ projects *the set of traces T to omit events that are not in the alphabet of R'.*

Dill's work proves that conformance implies safe substitution so we admit the following theorem:

Theorem 1 (Conformance implies safe-substitution). *In the context of any system, if the component R conforms to the component R', then R' can be replaced by R, and all properties of the system are preserved.*

$$R \preceq R' \longrightarrow Prop(\cdots, R', \cdots) \longrightarrow Prop(\cdots, R, \cdots)$$

Conformance is a helpful tool in the inductive step as seen in Section 5, but it also proves useful to bridge the gap between our model checker and theorem prover representations. By showing *conformance equivalence* between each of these representations we can show that a node as defined in the model checker can safely be substituted with a node as defined in the theorem prover. Using this technique we can define Lemma 5 which allows for our inductive proof in Theorem 2 to be proven.

Theorem 2 (The properties hold on an arbitrarily large system). *The desired properties hold for an arbitrarily large number of nodes utilizing the MyCHIPs protocol.*

$$\forall n \in \mathbb{N}, n \geq 1 \longrightarrow Prop(\text{Ref}, \text{Orig}, R_1, \cdots, R_n)$$

Proof. *By induction.*

Base Case: $Prop(\text{Ref}, \text{Orig}, R_1)$

See Section 4

Inductive Step:

$$\forall n \in \mathbb{N}, Prop(\text{Ref}, \text{Orig}, R_1, \cdots, R_n) \longrightarrow Prop(\text{Ref}, \text{Orig}, R_1, \cdots, R_n, R_{n+1})$$

See Lemma 5

The proof for Lemma 5 relies on three Conformance properties defined in Lemma 2, Lemma 3 and Lemma 4.

Lemma 2 (Two nodes conform to one node). *In the context of a MyCHIPs system, a chain of two black-box Relay nodes conform to a single black-box Relay node.*

$$\forall n \in \mathbb{N}, (B_n, B_{n+1}) \preceq B_n$$

Proof. *See Section 5*

Lemma 3 (Inductive node conforms to model checker node). *The representation of a node B as defined in the theorem prover conforms to the representation of a node R as defined in the model checker.*

$$B \preceq R$$

Proof. *See Section 6*

Lemma 4 (Model checker node conforms to inductive node). *The representation of a node R as defined in the model checker conforms to The representation of a node B as defined in the theorem prover.*

$$R \preceq B$$

Unproven. See Section 6

Using Theorem 1, Lemma 2, Lemma 3 and Lemma 4 we can show:

Lemma 5 (Inductive Step).

$$\forall n \in \mathbb{N}, \mathit{Prop}(\text{Ref}, \text{Orig}, R_1, \cdots, R_n) \longrightarrow \mathit{Prop}(\text{Ref}, \text{Orig}, R_1, \cdots, R_n, R_{n+1})$$

Proof.

$$\mathit{Prop}(\text{Ref}, \text{Orig}, R_1, \cdots, R_n) \ \wedge \ B \preceq R \ \longrightarrow \\ \mathit{Prop}(\text{Ref}, \text{Orig}, R_1, \cdots, R_{n-1}, B_n)$$

$$\mathit{Prop}(\text{Ref}, \text{Orig}, R_1, \cdots, R_{n-1}, B_n) \ \wedge \ (B_n, B_{n+1}) \preceq (B_n) \longrightarrow \\ \mathit{Prop}(\text{Ref}, \text{Orig}, R_1, \cdots, R_{n-1}, B_n, B_{n+1})$$

$$\mathit{Prop}(\text{Ref}, \text{Orig}, R_1, \cdots, R_{n-1}, B_n, B_{n+1}) \ \wedge \ R \preceq B \ \longrightarrow \\ \mathit{Prop}(\text{Ref}, \text{Orig}, R_1, \cdots, R_n, R_{n+1})$$

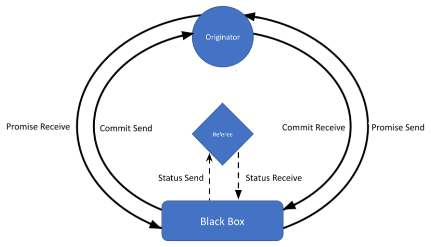
$$\therefore \forall n \in \mathbb{N}, \mathit{Prop}(\text{Ref}, \text{Orig}, R_1, \cdots, R_n) \longrightarrow \\ \mathit{Prop}(\text{Ref}, \text{Orig}, R_1, \cdots, R_n, R_{n+1})$$

Using Figure 2 to help illustrate, the remainder of this section will describe the intuition of Lemma 5.

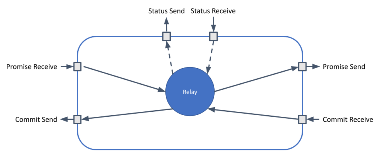
First, conformance requires that the alphabet of events of these subsystems be equivalent. Said differently, we define a standard interface that is sufficient for all the submodules we want to substitute. That interface includes 6 unique types of events: *Promise Send*, *Promise Receive*, *Commit Send*, *Commit Receive*, *Status Send*, and *Status Receive*. Figure 2a illustrates how these events map to messages passed in the system. The events are written from the perspective of the black box at the bottom of the figure. When the black box emits a *Promise Send* event a *Promise* type message is sent to the originator as indicated by the arrow. Similarly when *Commit Receive* event is emitted by the black box this means that the originator has sent a *Commit* type message. Likewise, when a *Status Send* event is emitted a *Status* type message is sent to the referee. This is an optional event so the arrow is drawn with a dotted line.

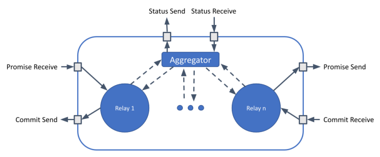
Lemma 2 considers two subsystems where the first has exactly one Relay node and the second has a chain of two (or more) Relay nodes. These take the form of Figure 2b and Figure 2c, respectively. In the second subsystem, we introduce an *Aggregator* which connects requests made from any node inside the module to a single external channel that communicates with the Referee. We hide the details of the internal signals so that the alphabet of events matches the interface required for the black-box model. Lemma 2 states that the chain of 2 Relay nodes conforms to the single Relay node.

Lemma 3 and Lemma 4 consider two subsystems that each have exactly one Relay node. Both subsystems take the form visualized in Figure 2b. The first module contains a Relay R using the model checking representation. The second module contains a Relay B that uses the theorem prover representation. Showing conformance equivalence between these two subsystems verifies that the two representations can be interchanged freely without changing the behavior of the system.



(a) Black-Box System Model





(b) Single-node Contents

(c) Multi-node Contents

Fig. 2: Utilizing Conformance

To understand the intuition of Lemma 5 first consider the system visualized in Figure 2a with a Relay node R in place for the black box. This is equivalent to the system $(\mathit{Ref}, \mathit{Orig}, R_1)$ of Lemma 1. Using Lemma 4 we substitute in a Relay node B. Using Lemma 2 we substitute a chain of two of these type of relay nodes (B_n, B_{n+1}) in place of a single relay node B. This substitution is proven safe with a machine-checked proof described in Section 5. Finally, we consider each of those relay nodes B_n and B_{n+1} as individual black boxes and using Lemma 4 we switch back to the model checker representation to complete the proof that the properties hold for the system with one more relay node.

4 Model Checking

We would like to prove that the properties hold on a base system consisting of one Originator, one Relay node, and one Referee. $\mathit{Prop}(\mathit{Ref}, \mathit{Orig}, R_1)$. The temporal properties required for the MyCHIPs lift protocol can be proven by an exhaustive search of all possible states in this base system. The state required by the system can be split into two classes: the part of the state that is evaluated when checking the properties, and the state that is used only to determine which state transition to take. We provide a formal definition of the former and a description of the latter. The source code for the complete Spin model can be found in the code repository.

The system state is a combination of the individual states for each node. The state for the Originator and each Relay node can be represented as:

$$\textit{Node state} \equiv \langle S, \Delta^s, \Delta^p \rangle; \quad S \in \{\textit{No Lift}, \textit{Pend}, \textit{Call}, \textit{Void}, \textit{Good}\}; \quad \Delta^s, \Delta^p \in \mathbb{Z}$$

Here S represents the Mealy-machine state and Δ^s and Δ^p represent the change in balance with the node's successor or predecessor respectively. The state of the Referee is similar except that it omits the balances

$$\textit{Referee state} \equiv \langle S \rangle$$

These are then composed together as a system state:

$$\textit{System state} \equiv \langle S_O, S_1, S_{\textit{Ref}}, \Delta^s_O, \Delta^p_O, \Delta^s_1, \Delta^p_1 \rangle$$

With the O subscript representing the *Originator*, 1 representing the *Relay* node, and *Ref* representing the *Referee*.

Each Mealy machine defined in Section 2 was implemented directly. This implementation consisted of three process types: Originator, Relay, and Referee. Each process has a unique set of edges marked as non-deterministic choices that are enabled only when the current state S matches the state of the edge in the diagram, and—if applicable—when the triggering message is present at the front of a message queue.

The message passing between nodes is modeled by creating three message channels for each node. One for messages received from the node's successor in the cycle, one for messages from the predecessor and one for messages from the Referee. The Referee also has a channel where it receives messages.

The details of the message are not important for the model, instead, each message will have a specific type. It is expected that nodes in a real system will verify that messages are valid as part of processing a received message. For the model, we will consider invalid messages as if they were never received. Once a valid message is received, the type of that message will inform the actions a node will take in response to the message.

When sending messages we model the possibility that a message might be lost in transit. One unique case is the possibility that all messages from a given node are dropped. This models the case where a node crashes or otherwise loses connection to the network.

The protocol requires that the Referee is consistently available so messages to and from the Referee are never dropped. Without this concession, the problem is isomorphic with the *two generals problem* [13], and it is impossible to guarantee eventual consensus on the lift status. However, the model will evaluate the possibilities that arise should those messages to and from the Referee take a very long time. Time is not modeled directly; rather, we allow all decisions that would be made by a node based on time to be a non-deterministic choice thus allowing for arbitrary, and even infinite, delays.

The Spin model verifies the following properties:

Definition 2 (Property 1).

$P_1 \equiv$ always eventually,

$$S_O \neq \text{Pend} \wedge\ S_1 \neq \text{Pend} \wedge\ S_{Ref} \neq \text{No Lift}$$

Definition 3 (Property 2).

$P_2 \equiv$ always eventually,

$$(S_O = \text{Good} \leftrightarrow S_1 = \text{Good} \leftrightarrow S_{Ref} = \text{Good})$$

Definition 4 (Property 3).

$P_3 \equiv$ always eventually,

$$(\varDelta^s_O + \varDelta^p_O \geq 0) \wedge (\varDelta^s_1 + \varDelta^p_1 \geq 0)$$

Definition 5 (Property 4).

$P_4 \equiv$ always eventually,

$$(\varDelta^s_O = -\varDelta^p_1) \wedge (\varDelta^s_1 = -\varDelta^p_O)$$

To verify each of these properties, Spin evaluated 16K unique states with 173K edges between those states. Evaluating each property Spin found that there were no counterexamples. This result from Spin is the proof for Lemma 1 as described in Section 3.

5 Two nodes conform to one node

We would like to prove that a chain of two Relay nodes conforms to a single Relay node in the context of a MyCHIPs system as implemented in the theorem prover. $\forall n \in \mathbb{N}, (B_n, B_{n+1}) \preceq (B_n)$. By Definition 1 this is can be reduced to: $\mathit{Proj}(P(B_n, B_{n+1}), B_n) \subseteq P(B_n) \longrightarrow (B_n, B_{n+1}) \preceq B_n$

The *partial order* defines the set possible traces of the model.

$$\forall T, \mathbb{V}(T, n) \leftrightarrow T \in P(B_n) \tag{1}$$

Where $\mathbb{V}(T, n)$ is predicate that is true when a trace T, satisfies the *partial order* for a system of size n. When the predicate holds for a trace we say that that trace is *valid* for the MyCHIPs protocol.

We use this predicate to state the conformance property necessary to prove safe substitution.

Theorem 3 (Valid projection implies conformance).

$$[\forall T \in P(B_n, B_{n+1}), \mathbb{V}(\mathbb{P}(T, n), n)] \longrightarrow (B_n, B_{n+1}) \preceq B_n$$

Where $\mathbb{P}(T, n)$ *is a projection function that removes events from the trace that are not part of the alphabet of events in a system of size n. This is equivalent to a combination of* hide *and* rename *operations as described by Dill.*

Proof.
Introduce:

$$\forall T \in P(B_n, B_{n+1}), \mathbb{V}(\mathbb{P}(T, n), n) \tag{2}$$

This is equivalent to:

$$\forall T \in Proj(P(B_n, B_{n+1}), B_n), \mathbb{V}(T, n) \tag{3}$$

Apply (1) and (3):

$$\forall T, T \in Proj(P(B_n, B_{n+1}), B_n) \longrightarrow T \in P(B_n) \tag{4}$$

By Definition 1 this is equivalent to:

$$Proj(P(B_n, B_{n+1}), B_n) \subseteq P(B_n) \tag{5}$$

$$\therefore (B_n, B_{n+1}) \preceq B_n \tag{6}$$

□

5.1 Machine-Checked Proof

We use Coq to prove that a chain of $n + 1$ Relay nodes can be projected onto a chain of n nodes.

$$\forall T \in P(B_n, B_{n+1}), \mathbb{V}(T, n + 1) \longrightarrow \mathbb{V}(\mathbb{P}(T, n), n) \tag{7}$$

This allows us to prove that the left-hand side of Theorem 3 holds for all valid systems, which in turn allows us to prove conformance.

The *partial order* and the *projection* are formally defined in Sections 5.2 and 5.3. Section 5.4 describes the proof script briefly. For details of the proofs, we direct the reader to examine the machine-checked proof script which can be found in the code repository.

5.2 Partial Order Description

The Coq proof operates on the set of all traces of all events that might occur in the system. Each trace is represented by an ordered list of events.

The *partial order* is defined in *acts_ valid*. There are eight rules that define the *partial order*:

1. *has_ required_ actions*,

2. *has_ no_ duplicate_ receives*,
3. *all_ receives_ causal*,
4. *all_ sends_ triggered*,
5. *all_ ids_ in_ range*,
6. *promise_ forward_ commit_ backward*,
7. *phase_ sequence_ correct*,
8. *all_ ref_ receives_ causal.*

All must hold for a trace of events to be considered valid. The rules were designed to match the possible traces that could be generated by a system of nodes for the given size. For brevity, only the most interesting rule *all_ receives_ causal* is defined here. For definitions of the other rules we direct the user to the machine-checked proof script in the code repository.

For the following definitions we use the notation $\mathbb{A}$ to represent the action type.

$$\mathbb{A} \equiv \{(Send, s, d, m), (Receive, d, m), (SendRef, s)(ReceiveRef, d)\}$$

Where $s \in \mathbb{N}$ is the source identifier $0 \leq s < size$, $d \in \mathbb{Z}$ is the destination identifier $-1 \leq d \leq size$, and m is a message $m \in \{Promise, Commit\}$.

We also use the notation T where $T \equiv (a_1, a_2, ..., a_n)$, $a_i \in \mathbb{A}$, to represent a *trace* of events and denote the size of the system $S \in \mathbb{N}$ which is a count of the number of nodes in the cycle.

Definition 6 (all_receives_causal). *Every* Receive *action is preceded by a corresponding* Send *action. Given* $r = (\text{Receive}, d, m), r \in T \longrightarrow \exists a = (\text{Send}, s, d^*, m), a \in T \wedge d^* = d \mod S \wedge a \prec_T r$*. Where* $\prec_T$ *indicates that* a *occurs before* r *in the trace of events.*

Because we allow the destination of a *Send* to be an integer between -1 and the size *inclusive* this means that a node can send a message with a destination -1 and this always corresponds with the node with the maximum ID in the system. Additionally, a node might send a message with the destination n, but when we project to size n the node with ID n is removed from the system. This send—instead of being sent to a node that doesn't exist—now gets mapped to node 0 because the destination need only be equal modulo the size. This allows for the projection to work without the need to mutate events, which makes proving properties about the projected system much simpler.

5.3 Projection Definition

Projection is defined with the *projected* function. Given an action and the size of the system we would like to project onto the *projected* function determines if the given action is kept or omitted in the projected trace. It uses a special type $option\mathbb{A} \equiv \{None, Some\ a\}$ with $a \in \mathbb{A}$.

projected returns *None* if the given action, a, should be omitted and *Some a* if the action should be kept in the projected trace.

Definition 7 (projected).

$$\begin{array}{l}
a = (\text{Send}, s, d, m) \wedge s \geq S \longrightarrow \text{projected}(a, S) = \text{None} \\
a = (\text{Send}, s, d, m) \wedge d > S \longrightarrow \text{projected}(a, S) = \text{None} \\
a = (\text{Receive}, d, m) \wedge d \geq S \longrightarrow \text{projected}(a, S) = \text{None} \\
a = (\text{SendRef}, s) \wedge s \geq S \longrightarrow \text{projected}(a, S) = \text{None} \\
a = (\text{ReceiveRef}\, d) \wedge d \geq S \longrightarrow \text{projected}(a, S) = \text{None} \\
\text{Otherwise} \longrightarrow \text{projected}(a, S) = \text{Some } a
\end{array}$$

Definition 8 (project_to_size). *Given a size and a list of events,* project_to_size*, denoted* $\mathbb{P}$*, returns a new list of events that omits all events for which* projected *returns None.*

$$a \in T \wedge projected(a, S) = Some\ a \longrightarrow a \in \mathbb{P}(T, s)$$

$$projected(a, S) = None \longrightarrow a \notin \mathbb{P}(T, s)$$

5.4 Proofs

The machine-checked proof consists of a set of lemmas that progressively build from basic principles to Theorem 4. We support this claim with Lemmas 6–16.

Theorem 4. *larger_ conforms_ to_ smaller*

$$\forall T \in [\mathbb{A}]\ \mathbb{V}(T, n+1) \wedge n > 1 \longrightarrow \mathbb{V}(\mathbb{P}(T, n), n).$$

Lemmas 6 through 12 all have the form

Lemma 6 (through 12). *property_ independent_ of_ proj*

$$\forall T \in [\mathbb{A}]\ n \in \mathbb{N}, n > 1 \wedge property(T, n+1) \longrightarrow property(\mathbb{P}(T, n), n)$$

For each of the properties: has_ required_ actions, has_ no_ duplicate_ receives, all_ ids_ in_ range, promise_forward_ commit_ backward, phase_ sequence_ correct, all_ ref_ receives_ causal and all_ sends_ triggered.

Lemma 13. *valid_ implies_ all_ receives_ causal_ in_ proj*

$$\forall T \in [\mathbb{A}]\ n \in \mathbb{N}, n > 1 \wedge \mathbb{V}(T, n+1) \longrightarrow \text{all\_receives\_causal}(\mathbb{P}(T, n), n)$$

Lemma 13 is the least trivial and the core problem that needed to be proven to show that the larger system conforms to the smaller system. The *Receive* causality property is difficult because a *Send* action that is associated with a given *Receive* in the original system may be removed in the projection. However, if the *Send* for a *Receive* is projected out there is an equivalent *Send* from that node's predecessor that now can be matched with that *Receive*.

This lemma is proven by careful case analysis. The proof itself spans nearly 500 lines and makes use of information known about the trace of events based on the *acts_valid* assumption.

We begin by introducing an action r which is the *Receive* in question. We assume it exists in the projected list and that it is indeed a *Receive* and use the sub-lemma *in_proj_in_orig* to show that the action must also appear in the original trace. With the knowledge that the action was in the original trace the assumption of the validity of the original trace allows us to show that there must be a corresponding *Send* for that *Receive* in the original trace. We now know that there is a *Receive* that is both in the original trace and in the projected trace and that there is a corresponding *Send* in the original trace. From here we examine the possible scenarios for what happens to the corresponding *Send* in the projection.

1. Both are unchanged. Neither the *Send* nor the *Receive* are removed and neither *wrap* around in the modulus.
2. The *Send* source alone is projected out. The source of the *Send* refers to a node that is no longer in the system.
3. The *Send* destination alone is projected out and the send now *wraps*.
4. Both the *Send* source and the send destination refer to a node that is removed.

Scenario 1 is pretty straightforward, the *Send* for the *Receive* is the same *Send* as in the original trace. We just show that the send remains in the projection.

Scenario 2 is tricky. The previously associated *Send* is now projected out. But the *Receive* is not projected out. To prove this scenario we use information about the required events in the original trace. After unpacking this information we can show that there must be a *Send* that would have gone to the node that was projected out. We show that this *Send wraps* due to the change in the size which means that that send is now associated with the *Receive* in question.

Scenario 3 is the dual of scenario 2. The same problem but from the other perspective. This is the case where a node that is not projected out sends to a node that is projected out. We show that that message *wraps* to connect with a new *Receive* which is our *Receive* in question.

Scenario 4 is an interesting case because it is only possible if we project out more than one node in a single step. The way the property is defined this is not possible because we have a system that is valid for size $n+1$ that is projected to size n. It takes some mathematics and a few properties about modular arithmetic to show that this case is impossible and always leads to contradictions.

Once each of these cases were individually proven or shown to lead to contradictions the property proves out.

Lemma 14. *not_in_orig_not_in_proj*

$$\forall T \in [\mathbb{A}]\ a \in \mathbb{A}\ n \in \mathbb{N}, a \notin T \longrightarrow a \notin \mathbb{P}(T, n)$$

Lemma 15. *in_proj_in_orig*

$$\forall T \in [\mathbb{A}]\ a \in \mathbb{A}\ n \in \mathbb{N}, a \in \mathbb{P}(T, n) \longrightarrow a \in T$$

Lemma 16. *happens_ before_ independent_ of_ proj*

$$\forall T \in [\mathbb{A}]\ n \in \mathbb{N}, a\ b \in T \land a\ b \in \mathbb{P}(T,n) \land a \prec_T b \longrightarrow a \prec_{\mathbb{P}(T,n)} b$$

Lemmas 6–16 and Theorem 4 were certified correct by Coq. These results prove Lemma 2. Theorem 4 tells us that the set of possible traces of events is independent of the size of the system. Because this is true for a chain of n nodes we know that the possible traces of events in a chain of n nodes is equivalent to possible traces of events in a single node. With this and Theorem 3 we know that a chain of n Relay nodes conforms to a single node.

6 Conformance Equivalence

We would like to prove Lemma 3 and Lemma 4:

$$B \preceq R\ \land\ R \preceq B$$

By Definition 1 it is sufficient to prove:

$$\mathit{Proj}(P(B), R) \subseteq P(R) \land \mathit{Proj}(P(R), B) \subseteq P(B)$$

By design, when we construct traces for $P(R)$ we hide events that represent changes to the internal state of a node using the Hide operation described by Dill. Once this is done the alphabet of events in $P(R)$ will be equivalent to the alphabet of events in $P(B)$. This allows us to reduce the proof requirement to:

$$P(B) \subseteq P(R) \land P(R) \subseteq P(B)$$

We will consider the left side of the conjunction in Section 6.1 and the right side of the conjunction in 6.2.

6.1 Possible traces of model checker respect the *partial order*

If $P(R) \nsubseteq P(B)$ then there must exist a trace explored by Spin that does not respect the *partial order*. By encoding the *partial order* as properties in Spin, we can verify that the rules hold for all of the traces evaluated by Spin.

To implement the properties the Spin model needed to be extended to generate a data structure that recorded the trace of events. To simplify this structure we rely on the fact that messages received by nodes are *idempotent*. If a node receives a message more than once the state will be the same as if it received the message only once. Because of this we can use Boolean flags for each type of message at a given node, and make assertions about these Booleans to verify the *partial order*. The Spin model was adjusted to set these flags when a message was sent and when a message was received.

The constraint that certain events exist is parameterized on the number of nodes in the lift cycle. This parameter is referred to as the *size* of the system.

The base system has the Originator and one Relay so the base system is of *size two*. Rather than parameterizing the properties we verify in Spin we write the properties exclusively for a system of *size two*. However, there are cases explored by Spin that when observing the trace of events appear to not include a Relay node. For example, consider the case where the Originator fails to send the first promise to the Relay. Eventually, the Originator requests the status from the referee and the referee will respond *Void* which rolls back the lift. However, this event trace does not contain any messages sent to or from the Relay. Because the Relay doesn't appear in the trace, this trace is indistinguishable from a system with no Relay, a system of *size one*.

In Theorem 2 we specify that the properties must hold for $n \geq 1$. The system of size one corresponds to $n = 0$ so this is a case we would like to ignore. But these cases where the system is effectively *size one*, cause Spin to produce some counter-examples that violate the *has_ required_ actions* rule.

To exclude these counter-examples we introduce a new fairness property that eliminates cases that represent systems smaller than the base system. This property, *size_fair* requires that each node eventually leave the *no_ lift* state ensuring that each node in the system is represented at least once in the trace. We then write each of the properties in the form: $size\_fair \longrightarrow property$. After triggering on this fairness property, all of the *partial order* for all traces evaluated by Spin.

When adding a fairness property it is prudent to check that the fairness property can be met, otherwise, we risk the properties verifying because they are vacuously true. To ensure that the new *size_fair* property is achieved we write the property: *always* $!size\_fair$. We expect this property to fail which indicates that the fairness constraint does not make our properties vacuously true.

The implementation of these properties can be found in the code repository. Spin evaluated these properties in 11K unique states with 74K edges between those states and found no counterexamples. These results verify that $P(R) \subseteq P(B)$.

6.2 Possible traces of the *partial order* are a subset of the definition

We would like to show: $P(B) \subseteq P(R)$. This work fails to produce a formal proof of this assertion. However, the following evidence provides high assurance that this assertion is true.

Consider the set of all the possible combinations of events for a single node. Because a node might emit a given event more than once, this set is infinitely large. Each of these events is associated with sending or receiving a message. Because messages are idempotent in MyCHIPs, the set of possible events we need to consider can be reduced into a small set of equivalence classes where each possible trace of events is represented by a canonical member that has only one of each type of event.

We can represent a canonical member of each of these equivalence classes as an ordered list of events. We use the following abbreviations for the alphabet of events: *Promise Send* $\equiv PS$, *Promise Receive* $\equiv PR$, *Commit Send* $\equiv CS$, *Commit Receive* $\equiv CR$, *Status Send* $\equiv SS$, and *Status Receive* $\equiv SR$.

Many of the possible orderings do not satisfy the *partial order*. After eliminating the impossible cases the remaining equivalence classes are:

1. $[PR, SS, SR]$
2. $[PR, PS, SS, SR]$
3. $[PR, PS, CR]$
4. $[PR, PS, CR, CS]$
5. $[PR, PS, SS, CR, SR]$
6. $[PR, PS, SS, SR, CR]$
7. $[PR, PS, SS, SR, CR, CS]$
8. $[PR, PS, SS, CR, SR, CS]$
9. $[PR, PS, SS, CR, CS, SR]$

We can evaluate if each of these equivalence classes is considered in the Spin model by generating a property that asserts that a trace of that class is not present. If this property fails we know that the trace is evaluated at least once in the Spin model.

The properties used to generate a witness trace for each equivalence class are defined as an automaton. The implementation of these automata can be found in the code repository. After executing the model checker to search for each of these new properties, each equivalence class was observed in the Spin model. Spin evaluated relatively few system states before finding each witness trace. The fastest search only evaluated 36 states and the longest evaluated 8K states. This allows us to conclude with high assurance that $P(B) \subseteq P(R)$.

This together with $P(R) \subseteq P(B)$ as proven in Section 6.1 allows us to conclude:

$$B \preceq R \ \wedge \ R \preceq B$$

7 Related Work

Numerous bodies of work solve similar problems or utilize similar constructs. The following list is particularly relevant.

Dill's logic has been extended to reason about software and distributed message-passing systems [19] [10]. Rajamani and Rehof's work uses conformance to show the substitutability of a model implementation and a definition implementation similar to this work's method in Section 6. Their method uses the CCS and π-calculus along with custom notation for non-deterministic choices. This work uses a partial order which at first brush seemed easier to define and prove in the Coq theorem prover. It is possible that Rejamani and Rehof's method may be equally easy, but an exploration of that is left as future work.

Clarke, Grümberg, and Browne describe a method for verifying properties on systems with many identical subprocesses. This method relies on a manual proof of bisimulation which is difficult to prove if these subprocesses can make non-deterministic choices as the nodes in the MyCHIPs system can. The method described in this work uses some similar techniques to the method described by Clarke *et. al.* but presents it in terms of conformance and operates on an alternative representation of the system for the inductive proof.

German and Sistla build on the work of Clarke, Grümberg, and Browne, but present a fully automatic method that can reason about multiple identical processes without any manual proofs of those processes.

German and Sistla's work is often referenced when describing methods for model-checking infinite state spaces [6,15,1]. The methods adjust the state matching to weaken the equivalence relation. Two states may be unique due to some infinite variable (a natural number) being different but they are equivalent in all ways we care about for verifying a property. This allows the machinery that model checks state a system for liveness and safety properties to operate on an infinite state space. This is somewhat akin to abstract interpretation.

Fischer, Lynch, and Paterson[11], prove the impossibility of consensus on even a Boolean, with even one faulty (or malicious) process. However, this is only true if the processes don't have synchronized clocks. This proof shows the necessity of the Referee with strong reachability requirements for the lift algorithm to always eventually reach a consensus the lift's final status.

Schneider summarizes and frames many fault-tolerant distributed algorithms in the framework of state machines [20]. He shows how many common algorithms are isomorphic to, and can be derived using, the state machine approach. It is a helpful method to characterize and compare different approaches and this work is inspired by these methods.

Delzanno, Tatarek, and Traverso, model check a common consensus algorithm called Paxos in Spin. Their Spin constructs provide helpful examples of how distributed algorithms are efficiently modeled [8].

Ogles, Aldous and Mercer prove that *doesn't commute*, a weakened version of the happens-before relation, is sound for certain common classes of task parallel programs. They present a machine-checked proof that proves properties for all traces constrained by a partial order. The methods demonstrated by Ogles, Aldous and Mercer were used as inspiration for some of this work's machine-checked proofs [17].

8 Conclusion

Conformance can be an effective tool to verify that model-checking results extend to arbitrarily large systems. It provides a framework for showing equivalence between a representation of a system suitable for model checking and a representation of a system suitable for inductive proofs.

For the MyCHIPs system, this work proved that the representation used in the model checker can be safely substituted into the theorem prover. However, it failed to produce a formal proof that the representation used in the theorem prover can be safely substituted into the model checker. However, we provide evidence that gives high assurance of the conformance equivalence between the two representations.

Because the inductive proof only relies on the *partial order*, new properties of the MyCHIPs lift protocol can be verified through model checking without making adjustments to the machine-checked proof.

For some systems, the high assurance this work provides is insufficient. Future work will need to prove conformance equivalence between the two representations. We foresee three plausible paths forward:

1. Add a function in the Coq proof script to produce a complete set of traces for the base system. Then, use a similar method used in Section 6.1 to produce a witness for each of these traces.

2. Define the representation of the system used for the model checker directly in Coq. While this representation is not conducive to inductive reasoning, proving Lemma 4 may be possible.

3. Prove that a weaker equivalence than a conformance equivalence is sufficient to prove that the properties extend to a system of arbitrary size.

While the method applied in this work requires considerable manual effort, a similar method could be used to create automated approaches that combine many verification techniques.

References

1. Abdulla, P., Haziza, F., Holík, L.: Parameterized verification through view abstraction. Int. J. Softw. Tools Technol. Transf. **18**(5), 495–516 (oct 2016). https://doi.org/10.1007/s10009-015-0406-x
2. Athalye, A.: CoqIOA : a formalization of IO automata in the Coq proof assistant. Master's thesis, Massachusetts Institute of Technology (2017)
3. Bateman, K.: Estimating the value of a CHIP (2023), https://chipcentral.net/wp-content/uploads/2023/01/chips_note-1.pdf
4. Bateman, K.: MyCHIPs digital money (2023), http://gotchoices.org/mychips/intro.html
5. Bateman, K.: MyCHIPs protocol description 1.3. MyCHIPs Protocol Description 1.3 (2023), https://github.com/gotchoices/MyCHIPs/blob/master/doc/learn-protocol.md
6. Bozga, M., Iosif, R., Sifakis, J.: Verification of component-based systems with recursive architectures. Theoretical Computer Science **940**, 146–175 (2023). https://doi.org/https://doi.org/10.1016/j.tcs.2022.10.022, https://www.sciencedirect.com/science/article/pii/S0304397522006181
7. Clarke, E.M., Grumberg, O., Browne, M.C.: Reasoning about networks with many identical finite-state processes. In: Proceedings of the Fifth Annual ACM Symposium on Principles of Distributed Computing. p. 240–248. PODC '86, Association for Computing Machinery, New York, NY, USA (1986). https://doi.org/10.1145/10590.10611
8. Delzanno, G., Tatarek, M., Traverso, R.: Model checking Paxos in Spin. Electronic Proceedings in Theoretical Computer Science **161**, 131–146 (Aug 2014). https://doi.org/10.4204/eptcs.161.13
9. Dill, D.L.: Trace theory for automatic hierarchical verification of speed-independent circuits, vol. 24. MIT press Cambridge, MA (1989)

10. Driscoll, E., Burton, A., Reps, T.: Checking conformance of a producer and a consumer. In: Proceedings of the 19th ACM SIGSOFT Symposium and the 13th European Conference on Foundations of Software Engineering. p. 113–123. ESEC/FSE '11, Association for Computing Machinery, New York, NY, USA (2011). https://doi.org/10.1145/2025113.2025132
11. Fischer, M.J., Lynch, N.A., Paterson, M.S.: Impossibility of distributed consensus with one faulty process. In: Proceedings of the 2nd ACM SIGACT-SIGMOD Symposium on Principles of Database Systems. p. 1–7. PODS '83, Association for Computing Machinery, New York, NY, USA (1983). https://doi.org/10.1145/588058.588060
12. German, S.M., Sistla, A.P.: Reasoning about systems with many processes. J. ACM **39**(3), 675–735 (jul 1992). https://doi.org/10.1145/146637.146681
13. Gray, J.: Notes on data base operating systems. In: Operating Systems, An Advanced Course. p. 393–481. Springer-Verlag, Berlin, Heidelberg (1978)
14. Huang, Y., Ogles, B., Mercer, E.: A predictive analysis for detecting deadlock in mpi programs. In: 2020 35th IEEE/ACM International Conference on Automated Software Engineering (ASE). pp. 18–28 (2020)
15. Lowe, G.: Parameterized verification of systems with component identities, using view abstraction. Int. J. Softw. Tools Technol. Transf. **24**(2), 287–324 (2022). https://doi.org/10.1007/s10009-022-00648-0
16. Nakamoto, S.: Bitcoin: A peer-to-peer electronic cash system. Cryptography Mailing list at https://metzdowd.com (03 2009)
17. Ogles, B., Aldous, P., Mercer, E.: Proving data race freedom in task parallel programs using a weaker partial order. In: 2019 Formal Methods in Computer Aided Design (FMCAD). pp. 55–63 (2019). https://doi.org/10.23919/FMCAD.2019.8894270
18. Ozanne, L.: Learning to exchange time: Benefits and obstacles to time banking. International journal of community currency research **14** (2010)
19. Rajamani, S.K., Rehof, J.: Conformance checking for models of asynchronous message passing software. In: Proceedings of the 14th International Conference on Computer Aided Verification. p. 166–179. CAV '02, Springer-Verlag, Berlin, Heidelberg (2002)
20. Schneider, F.B.: Implementing fault-tolerant services using the state machine approach: A tutorial. ACM Comput. Surv. **22**(4), 299–319 (Dec 1990). https://doi.org/10.1145/98163.98167
21. Schuppan, V., Biere, A.: Liveness checking as safety checking for infinite state spaces. Electronic Notes in Theoretical Computer Science **149**(1), 79–96 (2006). https://doi.org/https://doi.org/10.1016/j.entcs.2005.11.018, https://www.sciencedirect.com/science/article/pii/S1571066106000557, proceedings of the 7th International Workshop on Verification of Infinite-State Systems (INFINITY 2005)
22. Valek, L.: The time bank implementation and governance: Is PRINCE 2 suitable? Procedia Technology **16**, 950–956 (2014). https://doi.org/https://doi.org/10.1016/j.protcy.2014.10.048, https://www.sciencedirect.com/science/article/pii/S2212017314002758

A Credit Lift Algorithm

The MyCHIPs credit lift protocol, consists of three phases: discovery, promise, and commit. In the discovery phase, a circuit is identified where each entity is willing to participate in a lift. The correctness of the discovery phase is not critical–and will not be verified–because no transfer of value occurs during this phase. Figure 3 describes an example of a sequence of messages that would be passed in a typical lift during the *Promise* and *Commit* phases.

The Originator initiates the lift by sending Relay 1 a lift record $r = \langle id, value, ref, timeout \rangle$. This record has a unique identifier, the lift value, the network ID of a Referee that the Originator selects, and a timeout timestamp. Sending this lift record represents a *Promise* and once Relay 1 obtains the Referee's signature they can immediately add the number of chips specified in the lift record to its tally with the Originator.

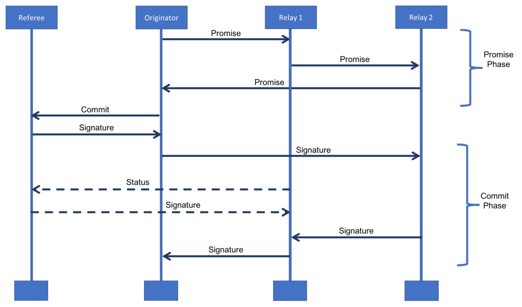


Fig. 3: Lift Protocol Sequence Example

When Relay 1 receives the lift record, it first confirms that it trusts the Referee to be fair and available, and it confirms that the timeout timestamp is acceptable. If the Referee is not trusted, or the known route to the target is no longer valid (for instance if it was used in a different lift and the node no longer has debt on that route to clear) it may choose to not continue. The node can optionally return a message indicating that it does not intend to complete the lift, or the node can simply ignore the message and wait for the lift to timeout. In our example, Relay 1 wishes to participate, so it forwards the lift record along with its own *Promise* to Relay 2. Relay 2 then similarly evaluates the lift and decides to forward a *Promise* to its predecessor which happens to be the Originator.

When the Originator receives the *Promise* from Relay 2, the lift moves into the *Commit* phase and the Originator sends the lift record to the Referee to request a signature to commit. When the Referee receives a lift record with a request to commit, it checks its clock to see if it received the *Commit* message before the timeout. If it received the *Commit* message after the timeout, the Referee signs a statement nullifying the lift. If it received the *Commit* message before the timeout, the Referee signs the lift record and returns its signature to the Originator. In either case, it records its result to provide to any node that requests the status of the lift. In the sequence diagram in Figure 3, the Referee decides the *Commit* message was received in time and sends the Originator its signature.

When the Originator receives the signature, it sends it to its successor in the circuit (Relay 2). Whenever a node in the circuit receives the signature, immediately the CHIPs it promised to its successor are valid. (i.e. it has forgiven the debt of its successor). To recover this value, it sends the signature to its predecessor.

In the sequence shown in Figure 3, once Relay 2 receives the signature, it experiences a long delay. Because of this delay, Relay 1 determines the lift is taking longer than expected and sends a message to the Referee to check the status of the lift. When the Referee receives the status request, it checks its records and recalls the signature it previously provided to the Originator. Because the lift is committed, the Referee provides this signature to Relay 1. When a node receives a signature from the Referee instead of its predecessor it should not forward this signature. This ensures there is exactly one failure point in the cycle after which every node must obtain the lift status from the Referee. However, eventually, Relay 1 receives the signature from Relay 2 which allows it to forward the signature. If the message sent by Relay 2 had arrived sooner, Relay 1 would not have needed to request the status from the Referee. Once the Originator receives the signature from Relay 1, all nodes have received exactly the amount of CHIPs they have given and the lift is complete.

B The MyCHIPs system

The MyCHIPs protocol and the methods used to verify it can be understood independently of the system it is used in. However, the curious reader will want to know more about the context in which the MyCHIPs protocol operates. This section provides an overview of the system. More details can be found on Kyle Bateman's website [4]. In particular, this section will cover the common challenges with using digital currency, how existing systems overcome those challenges, and how MyCHIPs overcomes those challenges in a unique way that provides additional benefits.

B.1 The two types of Money: Commodity and Credit

The purpose of money is to be a medium of exchange, and it does so by storing value. When someone provides a good or service, the seller is usually given money in exchange. This money stores the value they provided until the seller is ready to redeem that value to get something they need. For this to work, whatever is used as money needs to be considered valuable. There are two prevailing methods for accomplishing this: commodity money, and credit money.

In this context, a commodity is anything that can be owned and that can have its right to ownership transferred from one party to another. A commodity derives its value from the economic laws of supply and demand. Because some commodities are consistently desired and are sufficiently scarce, their value is stable enough to be used as money. For example, gold has historically seen widespread use as commodity money. As long as a commodity is in demand and its supply is reasonably limited, it can likely be used as money.

In contrast, credit money exists as a promise from one party to another of future value–likely a commodity or service. If the party giving the promise is trustworthy, the credit money has a value equal to what was promised.

B.2 Digital Currencies

Most digital currencies, including Bitcoin and Ethereum [16], take the commodity money approach. This decision led to two key technical challenges: achieving scarcity, and community consensus on ownership.

Achieving Scarcity It can be difficult to make a digital object that can be kept scarce. Physical commodities, such as gold, are difficult to locate and refine. To mirror this difficulty of discovery, a finite set of digital tokens can be mathematically defined in such a way that they were not known specifically in advance. However, any token, once discovered, could be easily checked to determine if it belonged to the set. Furthermore, discovering tokens is often computationally complex enough that it requires significant time and energy to discover a new token. This allows for a commodity that is both digital and scarce. Many possible sets could be used, however, hash functions and blockchains became the prevailing method because using this method simultaneously solves the community consensus problem.

Community Consensus For someone to transfer ownership of a commodity, they need to be able to prove that they own the commodity. Because of this, using commodity-based money requires community consensus about who owns what commodities.

Community consensus can be difficult to achieve, especially when many malicious actors may be present. One frequently discussed example where community consensus is not successfully achieved is the *double-spending* problem. *Double spending* is a difficult challenge and efforts to solve the problem lead to the development of blockchain algorithms.

B.3 Double-Spending

Double spending is a challenge that arises when using digital commodities as currency. To spend a commodity-based digital token, an entity must send a transmission that convinces a peer that it owns a digital token and is transferring ownership of that token to the peer. However, if the entity can send one transmission that fulfills this, it can easily send the same transmission to two peers simultaneously. If the peers have no means of communicating with each other (or some third party) then they each have no reason not to accept the digital currency. This allows a malicious actor to “double-spend” a digital token.

B.4 Overcoming Double-Spending with a Public Ledger

Most commodity-based digital currencies overcome the double-spending problem is using a public ledger. With a public ledger, each transaction between any entities using the digital currency must be stored in a single public record. By using a public record, double-spending can be identified and prevented. This public ledger may be implemented by assigning a central trusted authority to manage the ledger or it may use a decentralized ledger with a consensus algorithm.

While using a central trusted authority to manage the ledger is simple, its centralized nature leads to certain risks. First, the central authority is a single point of failure. If the central authority experiences some error and is not able to process transactions for a period of time, all trade using the digital currency would halt. Second, giving the central authority control over all the transactions of a digital currency requires great trust in the authority. The central authority could easily falsify records to benefit itself or a third party. In practice, there is not a single authority that every potential user of a digital currency will consider worthy of that amount of trust.

Using a decentralized ledger removes the need for a single trusted authority, but steps must be taken to ensure that the ledger cannot be falsified by an attacker and that a quorum of users eventually agrees on the state of the public ledger. There are two common consensus algorithms used to achieve this: proof-of-work, and proof-of-stake.

Proof-of-Work Proof-of-work consensus algorithms allow for a form of democratic consensus where the group that holds more than 50% of the computational resources decides the state of the public ledger. The blockchain method accomplishes this by requiring each block to include a partial hash collision. Such a collision will include a certain number of leading zeros in the result of the hash. Because cryptographic hash functions are designed to be unpredictable and non-reversible, there is no inverse function to compute what data must be added to a record to get the required hash. To add a block of data to the ledger an entity must compute many billions of hashes until it finds a partial collision. These blocks are then chained together so the hash of the previous block is included in the next block.

By chaining these blocks together, to make a change to a previous block that appears legitimate an attacker must recompute the hash collision for all subsequent blocks. If the attacker does not have more than 50% of the computation power they won't be able to compute the required hashes faster than the blockchain grows and will never be able to make a falsification appear legitimate.

This method has been quite successful at accomplishing its goals, but it has received criticism for the economic and environmental impacts it causes. In order to maintain the security of the public ledger the computation work required needs to be great enough to prevent attacks, however computational work is not free. There is a very large overhead to purchase and maintain equipment and supply power to be able to complete the computational puzzles required by proof-of-work. This means that transaction costs for blockchain currencies are much higher than traditional transactions. Additionally, the power used for proof-of-work often is generated by burning fossil fuels, which contributes to greenhouse gas emissions and depletes limited resources. Many argue that proof-of-work consensus algorithms are not sustainable.

Proof-of-Stake In response to these criticisms, algorithms have been implemented that rely on *proof-of-stake* rather than proof of work. Proof-of-stake algorithms rely on similar principles as proof-of-work but rather than allowing anyone to complete the computational puzzle, validators are randomly selected from a pool. To become a validator an entity must place a sufficient stake as collateral. If they are found to behave maliciously, their collateral is forfeit. This allows the computation puzzles to be set to be much easier to solve which in turn reduces the overhead and environmental impact of transactions.

Public Ledgers and Privacy Both proof-of-work and proof-of-stake algorithms operate to create consensus on a single, public ledger. The requirement for a public ledger means that no transactions can be made privately. Steps can be taken to attempt to anonymize transactions but if the identifiers used in the transaction are connected with the individuals they represent, anyone can see the entire history of that person's transactions. Additionally, requiring a consensus on a single public record limits how decentralized the digital currency can be. Because of the privacy concerns, the economic and environmental impact of blockchain transactions, and to make a more decentralized monetary system, a non-blockchain digital currency is desirable. MyCHIPs aims to serve as a credit-based digital currency that avoids these fundamental problems.

B.5 Private Credit Digital Currency

Transferability is intrinsic to commodity money. If you cannot transfer ownership of the commodity you can't transfer the value and it doesn't work as money. Credit money however can be made non-transferable and still maintain value. The promise for goods or services connected with the credit money can be valuable even if that promise is valid only for you. Making tokens non-transferable solves many security problems but leads to other practical problems.

B.6 Solving the double-spending problem with non-transferability

To prevent double spending we must ensure that a malicious actor gains no benefit from transferring a digital token to two different peers. Rather than preventing duplication, MyCHIPs instead ensures that tokens are only transferred once. To meet this goal, tokens in the MyCHIPs system are designed to be non-fungible and non-transferable. A token is created for a single recipient and is valid for only that recipient. However, this method has an obvious problem: how does one spend a non-fungible, non-transferable token? MyCHIPs has a method to make CHIPs *virtually fungible* that involves giving the CHIPs back to the issuer in a distributed algorithm called a credit lift [4].

The value of a CHIP As mentioned above, blockchain-based cryptocurrencies rely primarily on scarcity to drive their value. Experience has shown how this results in highly volatile valuations over time. Volatility may be useful in certain investments, but it is generally not desirable in a currency.

In the same way that bonds tend to be less volatile than stocks, a credit-based currency will be more stable than one based on a purely demand-driven commodity. However, the credit lift algorithm does rely on the assumption that at each stage of the lift, parties are trading tokens that have a uniform nominal value.

This unit of measure is called the CHIP–a clever backronym for Credit Hour in Pool. The value of a CHIP is defined to be the value of one hour of unskilled labor. Parties agree to this definition in a digital contract when they issue CHIPs. Recent research has placed this value at $2.53 [3].

However, if a laborer is providing tools, or has some expertise in the work they may be able to negotiate receiving more than one CHIP for an hour's worth of work. That is up to the market to decide. The CHIP definition only quantifies what is being promised with the issuance of each new CHIP.

Unfortunately, it is not very common for the seller of a good to want something directly from the buyer in exchange. To make MyCHIPs useful to facilitate trade the seller needs to be able to use the CHIPs to get goods or services from other entities besides the buyer. This is accomplished through a *credit lift* (hereafter, *lift*).

B.7 Chits and Tallies

Because a Token in the MyCHIPs is only valid between exactly two entities, these tokens only need to be tracked by those two entities. These tokens are tracked on a consensed record called a *Tally*. A *Chit* is an atomic transaction where one or more CHIPs are given to the partner. A Tally is an ordered collection of Chits with additional information to note the terms of the two entities' agreement.

Whenever a pair of entities desire to establish a trade relationship with each other, they create a new Tally to track that relationship. Most trade relationships are fairly consistent about who is selling goods or services and who is buying. When creating a Tally the entities establish this normal flow by assigning the entity that is usually paid and provides goods the *Stock* role in the Tally, and the one who usually pays the *Foil* role. These names hearken to a traditional method of tracking such credit relationships called a *Split Tally*. If the entity that takes the *Stock* role wishes to purchase goods or services from the *Foil* entity, the entities create a new Tally with the roles reversed.

Each time a transaction is made, the entity with the Foil role creates a Chit that includes:

1. The number of CHIPs being sent
2. A verifying digital signature
3. The date and time of the transaction
4. Other metadata about the transaction

The verifying signature could be either the signature of the entity with the Foil role or as described in Section A the signature of the Referee if the transaction is part of a credit lift. The verifying signature is all that is needed for one entity to prove the other owes some debt, however, it is helpful for both entities to have assurances that they agree on the balance of their tally.

To ensure that both entities agree on the state of the Tally, they exchange hashes of their Tally. If the hashes match they can be assured they are in agreement on both the Chits in the tally as well as the order of those Chits. If there is a mismatch, the consensus algorithm is simple: the Tally of the entity with the Foil role is considered correct. If there are still missing Chits, the entity with the Stock re-transmits those to the Foil. The Foil adds those Chits to the Tally and re-transmits its new hash.

The MyCHIPs Credit Lift A lift makes CHIPs *virtually fungible* by identifying a circuit where every entity holds CHIPs issued by their predecessor in the circuit. The lift algorithm arranges for each entity in the cycle to forgive the debt of their predecessor to get the same amount forgiven by their successor.

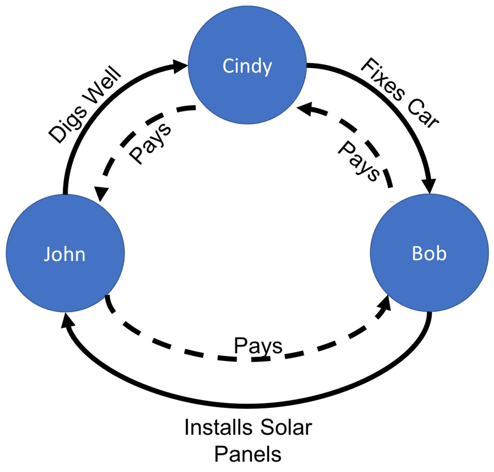


Fig. 4: Example Circuit

How this facilitates trade can be better understood by considering a barter system that involves more than two parties. For example, Bob needs his car fixed, John wants solar panels installed on his roof, and Cindy wants a well dug. Cindy knows how to fix cars, Bob knows how to install solar panels, and John can dig a well. See Figure 4 for a graphical representation of this arrangement. If they get together, they can work out a trade where John digs the well for Cindy, Cindy fixes Bob's Car, and Bob installs John's solar panels. Everyone would be happy.

However, it is difficult to identify these cycles before every transaction. This is where money comes in. Using government-backed money like dollars or euros, the process for enabling this transaction might go as follows:

1. Bob gets a loan from a bank
2. Bob uses the money from the loan to pay Cindy to fix his car
3. Cindy gives that money to John to dig the well
4. John gives Bob the money to install the solar panels
5. Bob uses the money to repay his loan (but he may need to pay a little interest).

MyCHIPs solves the same problem but instead does so by explicitly finding these cycles of value. Each person providing goods or services accepts CHIPs as an IOU for the exchange. Then a successful lift discovers this cycle of value and clears the debts of all involved.

If John, Bob, and Cindy were to use a credit lift to facilitate the arrangement it would go as follows:

1. Bob asks Cindy to fix his car and agrees to give Cindy 10 CHIPs
2. Cindy gives John 10 CHIPs to dig the well
3. John gives Bob 10 CHIPs to install the solar panels
4. Cindy talks with Bob and learns that John gave Bob 10 CHIPs
5. Cindy realizes that Bob gave her 10 CHIPs, she gave John 10 CHIPS and John gave Bob 10 CHIPs. There is a cycle!
6. Cindy initiates a lift and *promises* Bob that she will forgive his debt if he can get John to do the same for her. By doing this she takes on the role of the *Originator* of the lift.
7. Bob gives John a similar promise.
8. John gives Cindy a similar promise and Cindy knows the deal is on.
9. Cindy *commits* and gives Bob 10 CHIPs to cancel his debt
10. Bob gives John 10 CHIPs
11. John gives Cindy 10 CHIPs

Everyone's tally for how many CHIPs they have given each other now totals to 0. We can see this exchange allows for virtual fungibility: each person trades some CHIPs they have received to get back some CHIPs they have given out. This resets their balances so they can then give more CHIPs out again in exchange for future goods they will need.