# Refinement and Composition in Formal Modeling of Temporal Systems

Philip Johnson-Freyd

December 8, 2015

Human lives are increasingly dependent on digital control systems which operate everything from airplanes to power-plants. Consequently, it is crucial that we be able to verify the correctness of these systems. However, "correctness" only makes sense with respect to some *specification* of their behavior. Formal methods allow for establishing with certainty that digital systems satisfy safety and correctness properties, leaving only the reliability of the final translation of digital logic into the physical universe in doubt. Highly automated formal methods such as model checkers and common type systems allow for verifying properties of digital systems which minimal human effort but are limited in their range of applicability. Less automated techniques such as interactive theorem proving are more generally applicable but require more human intervention. In either case, however, formal verification techniques can only prove systems correct with respect to formal specifications. As such, the problems of constructing and reasoning about specification are paramount.

A key challenge for specification and formal methods is scalability. Digital systems may become too large to effectively reason about. To address this problem, it is crucial to be able to describe systems abstractly and compositionally. At its most basic, the idea of refinement is to formally relate two formal models: one more abstract and one more concrete with the goal of being able to carry reasoning performed at the more abstract (and perhaps simpler) model over to the more concrete one.

Further, refinement enables a development methodology, suggested more that 40 years ago by Wirth, whereby a system is constructed by first giving a highly abstract high level description and then by applying successive refinements deriving a final concrete system which is correct by construction [44]. In Wirth's account, each level of abstraction takes the form of an executable program but where successive refinements move from an easy to understand but inefficient program to an equivalent one which is more complicated but also much more efficient. This approach has been particularly effective in the development of functional programs as demonstrated by [10]. Often time, this refinement approach to software allows one to, in the words of Shan and Thurston while demonstrating a particularly impressive example of the technique, "first write a program to specify the problem, then interpret the program creatively to find the solution before the universe ends." [40] However, refinement is useful for more than just optimizing functional programs. In general, there is no requirement that the objects related by refinement be executable programs. Instead, they could be specifications which describe whole *classes* of programs. The act of refining a specification then could be understood as narrowing its associated class of programs. Nor should we be restricted to reasoning about *transformational* systems which merely convert inputs to outputs. Many interesting digital systems instead take the form of *reactive* systems characterized by complex patterns of interaction with their environment [23]. It is vital to be able to use refinement techniques in these areas.

In addition to refinement, scalable development calls for composition where multiple formal models at the same level of abstraction can be put together. A complete digital system will often admit multiple useful abstractions emphasizing different aspects of its behavior. Composition is connected to modular development whereby different components are constructed separately.

In this exam, we consider refinement and composition for formal models which describe how digital systems evolve over time. We focus on three general approaches. First, we consider the Event-B formalism (implemented in the Rodin tool) where models are roughly state transition systems and refinement is a form of simulation. Second, we look at the pure logic based approach in Lamport's Temporal Logic of Actions (TLA) which allows us to associate composition of models with logical conjunction and refinement of models with logical implication. Thirdly, we consider an approach based on adapting techniques from algebraic specification languages to a temporal setting which is explicitly categorical in flavor. Along the way we pay particular attention to the use of these formalism and notions of refinement as tools for verification.

## 1 Background

A specification tells us what implementations are possible. A refinement of a specification is one which admits strictly fewer implementations. A consequence of this is that if a specification implies some property about all its implementations then that same property is implied by all its refinements. What kinds of properties are implied by a specification depends on the particular specification language used. In this work, we are particularly interested in properties of reactive systems. We take the view of reactive systems as evolving through time. Various properties are of interest in this kind of system.

For example, an invariant is a property about the system state which is maintained by any evolution of the system. That is, an invariant is a property about the state which must always hold. Invariants are among the most important, but also the simplest, class of properties. Many properties considered in program verification and analysis, such as the types of variables or properties maintained by loops, take the form of invariants. Event-B, one of the specification approaches we consider in this work, focus almost exclusively on invariants. As a consequence, its accompanying notion of refinement ensures that any invariant maintained by a specification is also maintained by any refinement of that specification. However, properties other than invariants are not necessarily preserved.

Each invariant takes the form of a property on the states of the system. Beyond invariants though, we may need to consider properties of sequences of states. Specifications of reactive systems need to describe how those systems can evolve over time. Reactive systems are fundamentally dynamic: at any given time they might be in a particular state, but as time goes on that state is expected to change. Thus, we will consider specifications and properties which classify the "behaviors" or traces of such systems.

The first major class of properties about behavior are safety properties. Formally, a safety property is one which can be finitely disproven. That is, if a safety property does not hold of a behavior then it must be the case that there is some finite prefix of states in this behavior which is sufficient to disprove the safety property. Invariants are safety properties, and indeed safety properties can be seen as a generalization of invariants, since an invariant is a property which must hold always and if an invariant does not hold of a behavior than there must be some point in the trace of states that make up that behavior where that property fails. Even when they are not invariants, safety properties often take the form of statements which must "always" hold.

Properties about behaviors which are not safety properties are called liveness properties. These properties may fail to hold about an infinite behavior even if they could hold about all the finite prefixes of that behavior. The quintessential liveness property on states is that some property will *eventually* hold. If we consider as behaviors simply sequences of numbers then "contains an even number" is a liveness property. The sequence formed by the function $f(n) = 2 * n + 1$ does not satisfy this liveness property, yet the property can not be disproven by any finite prefix since any finite prefix extends to a sequence containing an even number. Conversely, the property that "every number is odd" is an invariant (and thus a safety property) while the property that the sequence is "monotonically increasing" is a safety property but not an invariant.

Composition of specification refers to the ability to construct a complete specification out of smaller component specification. Different approaches to composition differ: composition of two specifications may be satisfied by all the implementations which satisfy both component specifications, or the composition may be given in a different form. In either case, compositionality mechanisms are vital to the ability to reason about large systems. A key property of composition should therefore be that reasoning performed about the components' specifications can be used to reason about the composed specification.

As a running example we will consider the problem of specifying a vending machine. The essential features of a vending machine vary depending on your perspective: for a user of a vending machine its most important feature is that it, when demanded, can provide the user with desired products. However, from the point of view of the provider of the vending machine, which we take here, the machine's principal role is to collect money. Indeed, at a very high level of abstraction we can think of the state of a vending machine as specified by a single variable `cash` representing the value of the money it currently holds. Such a machine evolves when a user insert more money into the machine during use.

In greater detail, however, the transactions between the user and the vending machine are more complicated than this. For example, in addition to inserting money, the customer also selects an item from the machine and has that item dispensed decreasing the vending machines inventory. Further, the amount of money a user inserts into the machine matches the price for the item vended. Incorporating these additional details constitutes a refinement of the original abstract model.

However, there are other possible refinements which are of interest to us. For example, it is unlikely that the user inserts money into the machine all at once as part of a transaction. Instead, the user would insert a number of coins and bills incrementally until reaching the price of the item being dispensed. Thus, the single abstract change in the machine state may be made up of multiple incremental changes at a higher level of refinement.

These and other refinements (such as, considering the way in which money is not stored in the machine as a single variable but rather as the sum of amounts of money in different denominations) are at least partially orthogonal and can each be considered independently. Combining these multiple refinements to produce a complete vending machine model which incorporates all the relevant details and refines each abstraction appropriately thus serves as a non trivial composition problem.

## 2 Event-B

Event-B was designed by Jean-Raymond Abrial based on the earlier B-Method [5]. In Event-B, specifications take the form of models of discrete transition systems or state machines. The main component of an Event-B model is called a machine and consists of a number of variables describing the state of the system, guarded events describing the possible evolution of the system through time, and invariants which are properties which remain true throughout the evolution of the system [21]. As way of illustration, we will consider the vending machine in Event-B.

```
MACHINE
        Vending1
SEES
        VendingCtx
VARIABLES
        cash
INVARIANTS
        inv1    :   cash ∈ ℕ
EVENTS
        INITIALISATION    ≜
                BEGIN
                        act1    :  cash := starting_cash
```

```
            END

        transaction    ≜
            ANY
                    new_money
            WHERE
                    grd1   :   new_money ∈ ℕ
            THEN
                    act1   :   cash := cash + new_money
            END
END
```

This specification already includes most of the essential features of Event-B [4]. The machine, called `Vending1`, consists of a single variable `cash`. The initial value of `cash` is set to the constant `starting_cash` which is brought into scope by the fact that we specify that the machine sees the context component `VendingCtx`. For its part, `VendingCtx`

```
CONTEXT
        VendingCtx
CONSTANTS
        starting_cash
AXIOMS
        axm1   :  starting_cash ∈ ℕ
END
```

simply declares `starting_cash` to be a constant with the associated axioms that it is an integer and that it is non-negative. `Vending1` is equipped with two transitions called *events*: the initialization event [1] as well as an event specific to the vending machine. This event, `transaction` models a user purchasing something from the machine. Note that `transaction` is parameterized by a local variable `new_money` such that the `transaction` event can only fire when `new_money`'s value is a non-negative integer. This is an example of a *guard* which is actually a more general feature: an event can only fire when its guard is true and thus we might use guards which depend on the current state of the machine.

Additionally, the machine includes an *invariant* which generate *proof obligations* for our model [22]. In this case, the only invariant we consider is the type invariant that `cash` is always a natural number. The proof obligations generated require that this invariant is preserved by each event.

Event-B includes an extensive refinement mechanism allowing models to be extended as they are developed. At the highest level of abstraction we modeled a vending machine as a transition system whose state was entirely described by the amount of money it held. However, a vending machine also needs to *vend* something. In order to model a vending

[1] unlike other events, Event-B requires INITIALISATION to be capitalized and spelled with an "s".

machine which vends we can *refine* the model we gave earlier. The states of the new model will need to include not just the amount of money in the system, but also the machine's inventory. One natural way of modeling the inventory is as a mapping from some collection of `Items` that the machine could dispense to natural numbers indicating how many of that type of item the vending machine currently has available for sale. We will also need to keep track of the `price` of each item. Observe however that the price of items should not change as the vending machine evolves over time. In Event-B it is thus natural to provide our extended model with a new context which extends the previous context to provide a set of `Items` that the machine might hold, a function assigning a `price` to each item, and a description of the initial inventory of the vending machine.

```
CONTEXT
        VendingCtx2
EXTENDS
        VendingCtx
SETS
        Items
CONSTANTS
        price
        starting_supply
AXIOMS
        priceType : price ∈ Items→ ℕ
        starting_supplyType : starting_supply ∈ Items→ ℕ
END
```

With the extended context it is possible for us to define the extended version of the vending machine as a machine in Event-B. This new machine called `Vending2` refines the previous machine `Vending1` to incorporate not only the variable `cash` but also a variable `supply` representing the current inventory.

```
MACHINE
        Vending2
REFINES
        Vending1
SEES
        VendingCtx2
VARIABLES
        cash
        supply
INVARIANTS
        inv1    :    cash ∈ ℕ
        inv2    :    supply ∈ Items → ℕ
EVENTS
```

```
INITIALISATION  ≜  extended
        BEGIN
                act1   :   cash := starting_cash
                act2   :   supply := starting_supply
        END

purchase  ≜  extended
        REFINES transaction
        ANY
                new_money
                s
                f
        WHERE
                grd1   :   new_money ∈ ℤ
                grd2   :   new_money ≥ 0
                grd3   :   s ∈ Items
                grd4   :   f ∈ Items → ℤ
                grd5   :   ∀x. x ≠s ⇒ f(x) = supply(x)
                grd6   :   f(s) = supply(s)−1
        THEN
                act1   :   cash := cash + new_money
                act2   :   supply := f
        END
END
```

In the refined model each event is stated to refine a specific event in the more abstract model. In our case, `purchase` models a `transaction` where not only is money inserted but also an item is selected and dispensed. In Event-B, each refined event generates a proof obligation ensuring that the refinement is correct. In order to understand how this works, we can think of the two models as related by a simulation relation. As we used the name `cash` for a variable in both models, the simulation relation automatically relates states if and only if they have the same value for `cash`. The generated proof obligation is the standard one for simulation. Suppose we use $R$ for the relation on states in the refined and abstract models. Then our proof obligation is that it $(s_r, s_a) \in R$ and $s_r$ can step to state $s'_r$ along an event $E_r$ which refines event $E_a$ then there must exist some $s'_a$ such that $s_a$ steps to $s'_a$ along event $E_a$ such that $(s'_r, s'_a) \in R$. The Event-B language though actually allows for refinement in ways which are somewhat more general than in this example. First of all, there is no requirement that the variables in the abstract machine appear as variables in the concrete machine. Instead variables in from the abstract machine may be referenced directly in invariants of the concrete machine. Invariants that refer to variables from both machines are called *gluing* invariants and we can think of gluing invariants as representing an extension of the invariant concept from just restricting the state space of a single machine to also defining the simulation relation [6].

Further, refinements in Event-B need not be simulations: the refined machine can add *new* events which do not refine any of the events in the abstract model. Instead, these new events must correspond to a "skip" event in the abstract model, and so, in particular, cannot modify variables which correspond to aspects of the abstract model's state space [11].

The concept of "decomposition" in Event-B works by splitting a machine into multiple components, performing a refinement chain on each of those components, and then stitching the refinement components together into a single refined machine [8]. In the original approach, the events of a starting abstract machine are partitioned between the various components. Communication between components happens by way of shared variables. As each component is refined new invariants must be added, but additional proof obligations are generated indicating that these invariants are preserved by the "external events" which existed in the original machine but were partitioned into other components. Finally, the combined machine is defined as the union of the various refined components [7].

An alternative approach to composition of Event-B machines is to consider the variables of each machine to be independent of the others, but to combine events which have the same name [36]. Two descriptions of events of the same name only ever happen at the same time yielding a mechanism for describing communication [12]. Event fusion scales to parameterized events [11]. The combination of models using the shared event approach has all of the invariants of each of the component models [41] and is further compatible with the decomposition/refinement/composition methodology where the variables, rather than events, of the initial model are partitioned [42].

The use of shared names for sharing in composition has been criticized for requiring an "engineering omniscience" [17].

Event-B is a very convenient language for describing state machines. It is simple and intuitive. However, Event-B is limited as a specification language in that the only properties about systems it can express are invariants. In general, specifications can be divided roughly into two forms: system specifications describing how a system behaves and requirement specifications describing properties that must hold but not fully describing the system [35]. While, Event-B seems relatively well suited for the former, the restriction to only considering invariants makes it less well suited for the later. Further, because Event-B is focused on invariants, its notion of refinement does not necessarily preserve other properties of interest. Thus, Event-B may not be sufficient for all our specification needs.

## 3 Logics of Time

A second approach to modeling reactive systems we consider is the use of temporal logic. The use of temporal logic for specifying programs goes back to Pnueli's [34] work in the

1970s. The idea is to describe both systems and the theorems about those systems as logical statements in an underlying logic which directly incorporates a notion of time. We write specifications as formulae based on the following idea: an implementation satisfies its specification precisely if that logical statement of the specification is true about the implementation.

The view of specifications as formulae is convenient because it allows us to work with specifications using the ordinary tools of logic. For example, one specification refines another if all implementations of the refined specification are also implementations of the more abstract one. Put another way, if the more refined specification is true about an implementation then it follows that so is the more abstract one. In logical terms, this means that a specification $P$ refines a specification $Q$ precisely when the formula $P \to Q$ is true. Similarly, ordinary logical connectives provide an easy way to combine specifications: implementations which satisfy $P$ and which satisfy $Q$ are exactly those which satisfy $P \wedge Q$. Thus, just as refinement can be modeled as implication, so too can composition be modeled as conjunction.

In order to specify reactive systems, however, we will need additional logical facilities beyond the ordinary connectives because we care about additional concerns such as how systems evolve over time which are not part of the standard logical framework. Temporal logics address this by extending ordinary propositional or first order logic with additional features for reasoning about time.

We first consider a tense logic where the truth of a statement can vary in time [37]. In such a system we say "$T$ is true" if right *now* $T$ is true, but we don't necessarily assume $T$ will continue to hold. The core feature of the tense logic is modalities for considering the truth value in the future. The modality $\Box$ which is pronounced "always" is used to express properties that must continue to be true. Thus, $\Box T$ holds if $T$ holds now and at all future times. Conversely, we have an alternative modality $\Diamond$ pronounced "eventually," which is used for statements which must become true at some point. $\Diamond T$ is true if $T$ is true now or if it will be true at some point in the future. Additionally, we suppose the ability to reason about changing state in the logic. Thus, in addition to "rigid variables" $(x)$ and function symbols $(f^n)$ whose meaning remain constant over time we have "flexible variables" $(\mathbf{x})$ whose values can change over time.

$$\begin{aligned} E \in \text{Expressions} &::= \mathbf{x} \mid x \mid f^n(E_1, \ldots, E_n) \\ T \in \text{Predicates} &::= P^n(E_1, \ldots, E_n) \mid T \wedge T \mid \exists x.T \mid \neg T \mid \Box T \mid \Diamond T \end{aligned}$$

If we assume a classical conception of truth, the modalities $\Box$ and $\Diamond$ are related as De Morgan duals similar to the quantifiers $\exists$ and $\forall$. Specifically

$$\models \Diamond T \leftrightarrow \neg \Box \neg T$$

and indeed, the duality can be seen as induced by the duality between $\exists$ and $\forall$ as $\Diamond T$ asserts that there exists *some* future time where $T$ holds while $\Box T$ asserts that at *all* future times $T$ holds. Although, as a technical matter, we need only include one or the other of $\Diamond$ and $\Box$ we include both as the temporal operators are the focus of our study. By contrast, the standard logical operators of $\wedge$, $\rightarrow$, and $\forall$ are considered derived forms as they can be defined in terms of the other connectives.

Returning to the vending machine, we can, assuming function and predicate symbols corresponding to ordinary mathematics, express as a temporal formula the invariant that the variable representing money in the vending machine is always a positive number:

$$\Box \mathbf{cash} \in \mathbb{N}.$$

Additionally, we can express properties which are not expressible in Event-B as Event-B provides no mechanism for stating general liveness properties. Here though, it is possible to express properties asserting that something must hold *eventually.* For example, the property that the amount of money in the vending machine will eventually exceed some fixed constant (such as 100):

$$\Diamond \mathbf{cash} \geq 100.$$

Beyond simply stating properties though, we would like to be able to specify the vending machine as a state machine [35]. However, doing so is a challenge in the simple tense logic. For example, we might attempt to construct a specification for the vending machine by stating that initially **cash** is a non-negative integer, and at any time no matter what the value of **cash** is that value will eventually become higher:

$$\text{Vending1}_{\text{tense}} \triangleq \mathbf{cash} \in \mathbb{N} \wedge \Box(\exists n.\mathbf{cash} = n \wedge \Diamond(\mathbf{cash} > n)).$$

This specification captures at least some of what we want to hold about the vending machine. For example, we should have that:

$$\models \text{Vending1}_{\text{tense}} \rightarrow \Diamond \mathbf{cash} \geq 100.$$

However, while it is clear that the specification implies this liveness property, it does not imply the safety property. That is

$$\not\models \text{Vending1}_{\text{tense}} \rightarrow \Box \mathbf{cash} \in \mathbb{N}.$$

The reason why the specification does not imply the invariant is because while $\mathbf{cash} = n \wedge \Diamond(\mathbf{cash} > n)$ means that **cash** will *eventually* be higher than $n$, in the mean time it could hold any value. For example, **cash** might start out at 3 decrease to $-7$ grow back to 4 and then monotonically increase to higher values ever there after. Or, its values might come from the sequence $0, -1, 1, -2, 2, -3, 3, -4, 4, \ldots$ or any other sequence of values so long as that sequence is unbounded.

As such, it seems that the tense logic alone is insufficient and we need something more to specify state machines.

**Remark 1.** The tense logic we consider here is based on a *linear* conception of time. What that means is that while it is possible for us to speak about a statement being true in the future, the logic does not include features to discuss the idea of a statement potentially being true in some potential future. The linear conception of time is not the only design choice, others have considered branching time logics such as CTL [13] and the extremely expressive CTL* [15] which are based on the a view of time not as a line but rather as a tree. However, while CTL has found significant use in model checking, branching time logic has also been found to be less compositional and more difficult to use in practice than logics based on a linear conception of time [43].

## 3.1 Linear Temporal Logic

Pnueli's Linear Temporal Logic (LTL) [34] extends the tense logic with one extra modality, $\circ$, pronounced "next." Here $\circ T$ is true of a system if $T$ is true in the *next* state of the system. In order to formalize this, we can adopt a discrete view of time in which the full behavior of system is understood as an infinite sequence of states. With this idea, $\circ T$ is true for some infinite sequence of states $\rho$ if $T$ is true for the *tail* (that is, all the states but the first one) of $\rho$. $\Box T$ is true of $\rho$ if $T$ is true of any sequence obtained by removing any number of the initial states from $\rho$. Conversely, $\Diamond T$ is true of $\rho$ if $\rho$ consists of some (possibly empty) prefix of states and then the sequence $\rho'$ for which $T$ is true. We write $\rho \vDash T$ to indicate that $T$ is true of $\rho$ and $\vDash T$ to indicate that $\rho \vDash T$ holds for all $\rho$.

Using LTL with its $\circ$ modality we can specify state machines quite directly. For example, rather than specifying that at any time **cash** will eventually increase, we can simply state that it will increase immediately. Thus we have

$$\mathrm{Vending1}_{\mathrm{LTL}} \triangleq \mathbf{cash} \in \mathbb{N} \land \Box(\exists n. \mathbf{cash} = n \land \circ(\mathbf{cash} > n)).$$

The updated specification of the vending machine is detailed enough that it directly implies both properties we care about:

$$\begin{aligned} &\vDash \mathrm{Vending1}_{\mathrm{LTL}} \to \Box \mathbf{cash} \in \mathbb{N} \\ &\vDash \mathrm{Vending1}_{\mathrm{LTL}} \to \Diamond \mathbf{cash} \geq 100. \end{aligned}$$

However, while this highly detailed specification guarantees both liveness and safety properties, it turns out to be less than amenable to refinement. Consider the extension to vending machine which incorporates temporary cash. Here, at any time the vending machine will either increase the variable **cash** by the amount in **temp** or it will increase **temp** and keep **cash** constant.

$$\begin{aligned} \mathrm{Vending2}_{\mathrm{LTL}} \triangleq\ & \mathbf{cash} \in \mathbb{N} \land \mathbf{temp} \in \mathbb{N} \\ & \land \Box((\exists n. \exists m. \mathbf{cash} = n \land \mathbf{temp} = m \land m > 0 \land \circ(\mathbf{cash} = n + m \land \mathbf{temp} = 0)) \\ & \qquad \lor (\exists n. \exists m. \mathbf{cash} = n \land \mathbf{temp} = m \land \circ(\mathbf{cash} = n \land \mathbf{temp} > m))) \end{aligned}$$

We want to think of Vending2 as a refinement of Vending1 where we simply implement steps of Vending1 where **cash** increases in value as sequences of steps involving increasing amounts of money held in **temp**. However, this does not work since

$$\not\models \text{Vending2}_{\text{LTL}} \to \text{Vending1}_{\text{LTL}}.$$

At its core, the issue here is that the $\circ$ modality ends up being too tight. When $\text{Vending1}_{\text{LTL}}$ specifies the value of **cash** changing in the next state it prevents implementations which increase cash by means of a series of states.

The problem here is not a problem with LTL per se, but rather with the particular specifications. Using $\circ$ to define state machines leads to specifications which are simple to read and where the liveness and safety properties we care about follow. However, it also prevents refinements of those state machines to insert intermediate states as implementations of transitions. What we would like is an approach which gives us the same sort of simple and easy to read specifications for state machine while implying the properties we care about while also allowing refinements which are not forced to respect the meaning of "next" state so exactly.

## 3.2 The Temporal Logic of Actions

The challenge with using LTL specifications of state machines with refinement is that the next operator in LTL leads to specifications which force machines to run at a certain *speed* and which can not be implemented by running slower (that is, with more steps).

For this reason Lamport suggested a variant of LTL called the Temporal Logic of Actions (TLA) [26] which ensures that the truth value of temporal formulae does not depend on how fast the system being considered runs. While other logics have been proposed to achieve the same goals by changing the semantics of the $\circ$ operator [25], TLA works by replacing it. Syntactically and semantically TLA is constructed in layers. At the base layer, we have the language of *state expressions* which consist of expressions in the ordinary language of first order logic with both rigid and flexible variables. Being completely formal, TLA is parameterized by a signature consisting of for each natural number $n$ a family of function symbols $f^n$ and a family of relation symbols $P^n$. Intuitively, functions, relations, and rigid variables correspond to the usual notions from logic: timeless and fixed entities unchanged as a reactive system evolves through time. By contrast, flexible variables are used to model time varying values and so the meaning of the flexible variables will change over time.

Above this is the language of *action expressions* which looks just like the language of state expressions except that for each flexible variable $\mathbf{x}, \mathbf{y}, \mathbf{z}$ we have the addition of a primed variable symbol $\mathbf{x}', \mathbf{y}', \mathbf{z}'$. Note that rather than simply indicating a new variable, the prime symbol is semantically significant. While an unprimed variable is used for a current value, the primed version is used for the next value of that variable. Note that while all state expressions are action expressions, the converse is not true.

$$\begin{aligned}
E \in \text{StateExpressions} &::= \mathbf{x} \mid x \mid f^n(E_1, \ldots, E_n) \\
D \in \text{ActionExpressions} &::= \mathbf{x} \mid \mathbf{x}' \mid x \mid f^n(D_1, \ldots, D_n) \\
A \in \text{ActionPredicates} &::= P^n(D_1, \ldots, D_n) \mid A \wedge A \mid \neg A \mid \exists x.A \\
T \in \text{TemporalPredicates} &::= P^n(E_1, \ldots, E_n) \mid T \wedge T \mid \neg T \mid \exists x.T \mid \Box T \mid \Diamond T \mid \Box[A]_E
\end{aligned}$$

Figure 1: Syntax of TLA

Logical formulae formed from action expressions will be called *action predicates*. An action predicate can be macro expanded into an LTL predicate which only makes a single use of the $\circ$ modality and otherwise does not incorporate any temporal modalities. More specifically, if $A$ is an action predicate that makes use of the primed flexible variables $\mathbf{x_1'} \ldots \mathbf{x_2'}$ then $A$ should correspond to the LTL formula

$$\exists y_1 . \ldots \exists y_n . A[y_1/\mathbf{x_1'}, \ldots, y_n/\mathbf{x_n'}] \wedge \circ(y_1 = \mathbf{x_1} \wedge \ldots y_n = \mathbf{x_n}).$$

In this way, the language of action predicates provides a restricted language for talking about changes in the state, capturing some of the expressive power of the $\circ$ operator in LTL.

Finally, there is the layer of *temporal predicates* formed by the usual logical connectives with state expressions as the notion of expression, and the temporal modalities for always ($\Box$) and eventually ($\Diamond$). These constructs share their meaning with LTL. Additionally, given any state expression $E$ and action predicate $A$ there is a temporal predicate, $\Box[A]_E$ whose intuitive meaning is that at any step either $A$ occurs or $E$ stays the same. In full LTL, the predicate $\Box[A]_E$ could be interpreted as syntactic sugar for

$$\Box(A \vee (\exists x . E = x \wedge \circ E = x)).$$

Observe that this means that $\Box[\bot]_E$ is true precisely if $E$ never changes [14]. The special operator $\Box[]$ in TLA is particularly well suited for expressing state machines. For example, the initial vending machine can be expressed in TLA as

$$\text{Vending1'}_{\text{TLA}} \triangleq \mathbf{cash} \in \mathbb{N} \wedge \Box[\mathbf{cash} < \mathbf{cash'}]_{\mathbf{cash}}.$$

Similarly, the specification of the refined vending machine can be expressed as

$$\begin{aligned}
\text{Vending2'}_{\text{TLA}} \triangleq &- \mathbf{cash} \in \mathbb{N} \wedge \mathbf{temp} \in \mathbb{N} \\
&\wedge \Box[(\mathbf{cash'} = \mathbf{cash} + \mathbf{temp} \wedge \mathbf{temp'} = 0) \vee (\mathbf{cash} = \mathbf{cash'} \wedge \mathbf{temp} < \mathbf{temp'})]_{(\mathbf{cash},\mathbf{temp})}.
\end{aligned}$$

With these specifications it follows immediately that Vending2'$_{\text{TLA}}$ is a refinement of Vending1'$_{\text{TLA}}$:

$$\models \text{Vending2'}_{\text{TLA}} \rightarrow \text{Vending1'}_{\text{TLA}}.$$

The issue faced with the version of the specifications constructed in LTL is resolved since in the extra possible action ($\mathbf{cash} = \mathbf{cash}' \wedge \mathbf{temp} < \mathbf{temp}'$), **cash** stays constant. Further, Vending1'$_{\text{TLA}}$ is strong enough to show all the safety properties we need, including the invariant:

$$\models \text{Vending1'}_{\text{TLA}} \rightarrow \Box \mathbf{cash} \in \mathbb{N}.$$

However, it does not imply the liveness property

$$\models \text{Vending1'}_{\text{TLA}} \rightarrow \Diamond \mathbf{cash} \geq 100$$

as Vending1'$_{\text{TLA}}$ would be satisfied by a sequence of states which just kept **cash** constant forever. Giving up this and similar liveness properties is a significant loss. However, TLA turns out to be expressive enough to write state machine style expressions which imply them.

First, we introduce a new operator for lifting action predicates into temporal predicates dual to the existing one. If $A$ is an action predicate and $E$ is a state expression, then $\Diamond \langle A \rangle_E$ is temporal predicate whose meaning can be given by macro expansion into the rest of TLA:

$$\Diamond \langle A \rangle_E \triangleq \neg\Box[\neg A]_E.$$

Taking the interpretation of TLA in terms of LTL, we can further simplify this to an LTL formula

$$\begin{aligned}
\Diamond \langle A \rangle_E &\triangleq \neg\Box[\neg A]_E \\
&\triangleq \neg\Box(\neg A) \vee \exists x.E = x \wedge \circ E = x \\
&\leftrightarrow \neg\Box\neg(A \wedge \neg\exists x.E = x \wedge \circ E = x \\
&\leftrightarrow \Diamond A \wedge \neg\exists x.E = x \wedge \circ E = x \\
&\leftrightarrow \Diamond A \wedge \forall x.E \neq x \vee \circ E \neq x
\end{aligned}$$

which suggests an intuitive meaning of $\Diamond \langle A \rangle_E$ as that eventually $A$ will happen and $E$ will change value.

Another useful operator is the *Enabled* predicate. If $A$ is an action predicate then $\mathit{Enabled}(A)$ is the temporal predicate which is true if the current state is one such that $A$ might be possible. That is, regardless of what the next state actually is, $\mathit{Enabled}(A)$ is true if there is some potential next state such that $A$ holds. The *Enabled* predicate does not add any new power not already present in TLA so we can treat it also as simply syntactic sugar. Let $\mathbf{x_1'} \ldots \mathbf{x_n'}$ be the primed variables which occur in $A$, then we simply set

$$\mathit{Enabled}(A) \triangleq \exists y_1 \ldots \exists y_n.A[y_1/\mathbf{x_1'}, \ldots, y_n/\mathbf{x_n'}.$$

From these, we can define, internally to TLA, the notion of weak fairness with respect to an action. The temporal predicate $\mathrm{WF}_E(A)$ asserts that either $A$ happens infinitely often or there are an infinite number of times when $A$ *can't* happen [2]. That is

$$\mathrm{WF}_E(A) \triangleq (\Box\Diamond \langle A \rangle_E) \vee (\Box\Diamond\neg \mathit{Enabled}(A)).$$

Similarly, TLA also allows for the assertion of a strong fairness condition. The temporal predicate $\mathrm{SF}_E(A)$ asserts that either $A$ happens infinitely often, or at some point $A$ becomes completely and permanently blocked. That is,

$$\mathrm{SF}_E(A) \triangleq (\Box\Diamond \langle A\rangle_E) \vee (\Diamond\Box\neg \mathit{Enabled}(A)).$$

Strong fairness implies weak fairness but not vice versa.

Even weak fairness, however, is strong enough to extend our state machine specifications sufficiently to prove liveness properties. We can outlaw the situation where **cash** never changes by simply adding the side condition that the action of incrementing **cash** is treated fairly (and, thus, happens sometimes). This yields to the full TLA specification of the vending machine at its greatest abstraction:

$$\text{Vending1}_{\text{TLA}} \triangleq \text{Vending1'}_{\text{TLA}} \wedge \mathrm{WF}_{\mathbf{cash}}(\mathbf{cash} < \mathbf{cash}').$$

Of course, $\text{Vending1}_{\text{TLA}}$ continues to imply all the safety properties of Vending1':

$$\vDash \text{Vending1}_{\text{TLA}} \rightarrow \Box\mathbf{cash} \in \mathbb{N}.$$

Additionally, we now add the liveness property that **cash** eventually exceeds any fixed value:

$$\vDash \text{Vending1}_{\text{TLA}} \rightarrow \Diamond\mathbf{cash} \geq 100.$$

While $\text{Vending2'}_{\text{TLA}}$ is not itself a refinement of $\text{Vending1}_{\text{TLA}}$ as it does not establish liveness, it can be extended with fairness conditions yielding the full specification of the refined machine.

$$\begin{aligned}\text{Vending1}_{\text{TLA}} \triangleq\ & \text{Vending2'}_{\text{TLA}}\\ & \wedge \mathrm{WF}_{\mathbf{cash},\mathbf{temp}}(\mathbf{cash}' = \mathbf{cash} + \mathbf{temp} \wedge \mathbf{temp}' = 0)\\ & \wedge \mathrm{WF}_{\mathbf{cash},\mathbf{temp}}(\mathbf{cash} = \mathbf{cash}' \wedge \mathbf{temp} < \mathbf{temp}')\end{aligned}$$

With fairness conditions included, we have

$$\vDash \text{Vending2}_{\text{TLA}} \rightarrow \text{Vending1}_{\text{TLA}}$$

showing that we have TLA allows us to capture all the properties of interest.

## 3.3 Formal Semantics of TLA

We have so far considered TLA as macro expanding into LTL. However, we now turn to fully specifying the semantics of TLA directly. When we later turn to the problem of variable hiding we will see that it makes sense to incorporate into TLA additional constructs which are not part of LTL. Further, it should be noted that the macro based interpretation of TLA often make use of first order features of LTL even when the original formula was quantifier free. It turns out that if we are restricted to the propositional subset of LTL (even extended with propositional variables whose truth values are functions of the state) then the goal of making propositions invariant under stuttering is achieved exactly by eliminating the $\circ$ operator [33].

In order to give a semantics for TLA we will first fix an interpretation of first order logic consisting of a pair $< \mathcal{D}, \mathcal{F} >$ where

- $\mathcal{D}$ is a set modeling the domain of discourse and
- $\mathcal{F}$ is a mapping associating each function symbol $f^n$ with a function, $\mathcal{D}^n \rightarrow \mathcal{D}$ and each relation symbol $P^n$ with a function $\mathcal{D}^n \rightarrow \{\texttt{true}, \texttt{false}\}$.

The interpretation of state expressions then will be parameterized by a function called the environment and usually written as $\theta$ mapping rigid variables to $\mathcal{D}$ and function called a *state* and usually written $\sigma$ mapping flexible variables to $\mathcal{D}$.

$$\begin{aligned} [\![x]\!](\theta,\sigma) &= \theta(x) \\ [\![\mathbf{x}]\!](\theta,\sigma) &= \sigma(x) \\ [\![f^n(D_1,\ldots,D_n)]\!](\theta,\sigma) &= \mathcal{F}(f^n)([\![D_1]\!](\theta,\sigma),\ldots,[\![D_n]\!](\theta,\sigma)) \end{aligned}$$

By contrast, the semantics of action expressions is parameterized by an environment $\theta$ and *two* states $\sigma$ and $\sigma'$ referring to the current (unprimed variables) and next state (primed variables) of the system at a given time.

$$\begin{aligned} [\![x]\!](\theta,\sigma,\sigma') &= \theta(x) \\ [\![\mathbf{x}]\!](\theta,\sigma,\sigma') &= \sigma(x) \\ [\![\mathbf{x}']\!](\theta,\sigma,\sigma') &= \sigma'(x) \\ [\![f^n(E_1,\ldots,E_n)]\!](\theta,\sigma,\sigma') &= \mathcal{F}(f^n)([\![E_1]\!](\theta,\sigma,\sigma'),\ldots,[\![E_n]\!](\theta,\sigma,\sigma')) \end{aligned}$$

Just as the meaning of an action expression depends on two states so does the truth value of an action predicate.

$$\begin{array}{ll} \theta,\sigma,\sigma' \vDash P^n(D_1,\ldots,D_n) & \text{iff } \mathcal{F}(P^n)([\![D_1]\!](\theta,\sigma,\sigma'),\ldots,[\![D_n]\!](\theta,\sigma,\sigma')) = \texttt{true} \\ \theta,\sigma,\sigma' \vDash A_1 \wedge A_2 & \text{iff } \theta,\sigma,\sigma' \vDash A_1 \text{ and } \theta,\sigma,\sigma' \vDash A_2 \\ \theta,\sigma,\sigma' \vDash \neg A & \text{iff } \theta,\sigma,\sigma' \nvDash A \\ \theta,\sigma,\sigma' \vDash \exists x.A & \text{iff there is some } v \in \mathcal{D} \text{ such that } (\theta, x \mapsto v),\sigma,\sigma' \vDash A \end{array}$$

By contrast, the truth value of a temporal predicate is not parameterized by a state but rather by an infinite family of states called a *behavior* and usually written $\rho$. We treat a behavior as a function from natural numbers to states and so use $\rho(0)$ for the initial state in $\rho$ and $\rho(1)$ for the next state. The syntax $\rho^k$ is used for the shift which when written as a function in lambda notation would be $\lambda n.\rho(n+k)$. A bare state expression is treated as being an expression of the initial state of $\rho$. Otherwise the interpretation of the logical connectives is standard, while the temporal modalities are interpreted using shifts.

| | |
|---|---|
| $\theta, \rho \vDash P^n(E_1, \dots, E_n)$ | iff $\mathcal{F}(P^n)([\![E_1]\!](\theta, \rho(0)), \dots, [\![E_n]\!](\theta, \rho(0))) = \texttt{true}$ |
| $\theta, \rho \vDash T_1 \wedge T_2$ | iff $\theta, \rho \vDash T_1$ and $\theta, \rho \vDash T_2$ |
| $\theta, \rho \vDash \neg T$ | iff $\theta, \rho \nvDash T$ |
| $\theta, \rho \vDash \exists x.T$ | iff there is some $v \in \mathcal{D}$ such that $(\theta, x \mapsto v), \rho \vDash T$ |
| $\theta, \rho \vDash \Box T$ | iff for every $k \in \mathbb{N}$ such that $\theta, \rho^k \vDash T$ |
| $\theta, \rho \vDash \Diamond T$ | iff there exists some $k \in \mathbb{N}$ where $\theta, \rho^k \vDash T$ |

Finally, the operator $\Box[]$ allows us to lift an action predicate into a temporal predicate according to the following idea: $\Box[A]_E$ is true of a behavior $\rho$ if at every pair of states of $\rho$ are either related by $A$, or, have the same value of $E$.

$$\theta, \rho \vDash \Box[A]_E \text{iff for every } k \in \mathbb{N} \text{ either } \theta, \rho(k), \rho(k+1) \vDash A \text{ or } [\![E]\!](\theta, \rho(k)) = [\![E]\!](\theta, \rho(k+1))$$

The most important property about TLA's semantic interpretation is the idea of *invariance under stuttering*. Intuitively, a behavior might *stutter* by staying at the same state for several time steps. In some sense, such a stuttering behavior is equivalent to one which does not stutter, and TLA formulae should be unable to observe the difference between these behaviors.

Formally, we define $\approx$ to be the least equivalence relation on behaviors such that for every $k \in \mathbb{N}$

$$\rho \approx \lambda n.\texttt{if } n \leq k \texttt{ then } \rho(n) \texttt{ else } \rho(n-1)$$

which is to say that a behavior $\rho$ is equivalent to the behavior $\rho'$ which is defined to be identical to $\rho$ except for an extra copy of the state $\rho(k)$ inserted into $\rho$ at the position $k+1$.

**Theorem 1** (Stuttering Invariance [31])**.** *For any temporal predicate $T$ and $\rho \approx \rho'$ we have $\theta, \rho \vDash T$ if and only if $\theta, \rho' \vDash T$.*

Generally, TLA developments will use a single signature for all the various models in a project. This signature, together with some non-temporal axioms, provides the *math language* used. In Lamport's TLA+ [28] the math language is a variant of Zermelo-Fraenkel set theory with choice. Here, we follow that convention and assume $\mathcal{D}$ and $\mathcal{F}$ to be a fixed interpretation of set theory.

As we take $\mathcal{D}$ and $\mathcal{F}$ to be fixed, we will say that a temporal predicate $T$ is *valid* written $\vDash T$ if for every interpretation of rigid variables $\theta$ and behavior $\rho$ that $\theta, \rho \vDash T$.

Observe that with no consequence on the meta-theory, we can extend the syntax of TLA in various ways. One very convenient extension is to allow for the addition of the prime operator on any expression, and not just flexible variables

$$D \in \text{ActionExpressions} ::= \ldots \mid E'$$

where the semantic interpretation of $[\![E']\!](\theta, \sigma, \sigma')$ is $[\![E]\!](\theta, \sigma')$ by simply unfolding the prime in the following way: priming a rigid variable reduces to just the rigid variable, priming a flexible variable reduces to the flexible variable with the prime, and priming a function symbol reduces to that function symbol with each of its argument primed.

The proof theory for TLA is a bit more challenging, and indeed, a bit of an open question. It is relatively easy to come up with *sound* rules of inference for TLA. However, it is more to challenging to come up with a system of inference rules which is *complete*. The crux of the difficulty is in encoding the induction principle which allows for deriving an invariant from an initial condition and an action. Such an induction principle is difficult to give in full generality for TLA as presented here, but can be given for a variant of TLA called TLA* where action predicates can contain temporal modalities [31]. It is also possible to give a complete proof system for TLA extended with quantification over flexible variables as will be considered later. However, in this work we do not focus on the construction of proof systems and so instead suggest that reasoning at the level of the semantics may be used. In addition to a system for deductive proof, the TLA+ system includes a model checker which can demonstrate some properties fully automatically and which in other cases can be used to test specifications up to finite bounds [45].

## 3.4 Refinement and Composition in TLA

Reducing refinement to implication is quite convenient in a number of ways. For one thing, it means that no extra features need to be included in TLA to work with refinement. For another, it allows us to easily carry properties about the abstract model over to the concrete one. For example, we know that $\vDash \text{Vending1} \rightarrow \Box(\mathbf{cash} \in \mathbb{Z})$ and $\vDash \text{Vending2} \rightarrow \text{Vending1}$ therefore we can immediately conclude that this invariant also holds for Vending2.

$$\vDash \text{Vending2} \rightarrow \Box(\mathbf{cash} \in \mathbb{Z})$$

Moreover, that refinement is understood as implication ensures that it preserves not just invariants or other safety properties but *all* the properties expressible in TLA including liveness properties.

However, the implication based approach is implicitly taking advantage of the fact that we are using the *same* variables in both models. This seems slightly worrying and anti-modular: we cannot simply rename a variable in one model without renaming it in the other [30]. Consider for example the model Vending1[**money**/**cash**] which is just like Vending1 except that the variable **cash** has been renamed **money**. It is apparent that we do *not* get the same kind of simple refinement relationship we had above.

$$\nvDash \text{Vending2} \rightarrow \text{Vending1}[\mathbf{money}/\mathbf{cash}]$$

And yet, it seems that there is still some sort of refinement relationship between these models. In particular, we are able to combine the two models with a *gluing invariant* namely

$$\text{Glue} \triangleq \mathbf{money} = \mathbf{cash}$$

as we can show that

$$\vDash (\Box\text{Glue} \wedge \text{Vending2}) \rightarrow \text{Vending1}[\mathbf{money}/\mathbf{cash}]$$

however, it is not immediately clear that we have constructed a refinement in this case as $\Box$Glue might restrict the possible behaviors of the variables in Vending2.

To get around this problem we introduce the idea of a *refinement mapping.* Suppose $T$ is a temporal logic formula with free flexible variables $\vec{\mathbf{x}}$ and $S$ is a temporal logic specification with the set of free flexible variables $\vec{\mathbf{y}}$. A refinement mapping $h$ associates each variable in $\vec{\mathbf{y}}$ with a state expression which does not use any of the variables from $\vec{\mathbf{y}}$ (but might use variables from $\vec{\mathbf{x}}$). Then, we have the following fact

$$(\vDash (\Box\vec{\mathbf{y}} = h(\vec{\mathbf{y}}) \wedge T) \rightarrow S) \Leftrightarrow (\vDash T \rightarrow S[h])$$

where $S[h]$ is the substitution formed by replacing each variable in $S$ with its associated expression from $h$ [27].

The use of refinement mappings gives us a way to treat refinement as implication without globally agreeing on variable names. Further, because we might associate a variable with an expression and not just another variable in the refinement mapping, this notion is actually more general. The concrete model might use multiple variables to capture a single variable in the abstract model: for example, we might further refine our vending machine to store separately the number of different kinds of coins (quarters, nickels, dimes, etc): the refinement mapping would have to then compute the value of **cash** as the sum of values of these coins. Observe also that

$$(\vDash T) \Rightarrow (\vDash T[h])$$

for any TLA formula $T$ and refinement mapping $h$. This means that we can lift properties we have proven about an abstract model to be properties about a concrete model by substituting along a refinement mapping: if $\vDash S \rightarrow P$ then $\vDash S[h] \rightarrow P[h]$ so if $\vDash T \rightarrow S[h]$ then $\vDash T \rightarrow P[h]$.

The implication connective allows us to talk about refinement internally to TLA. However, allow us to step back and think about what such a refinement means semantically. If $\vDash T \rightarrow S[h]$ then for any $\Theta$ and for any $\rho$ we have $\Theta, \rho \vDash T \rightarrow S[h]$ which means that $(\Theta, \rho \vDash T) \Rightarrow (\Theta, \rho \vDash S[h])$. Put another way, this says that for any $\Theta$ that

$$\{\rho \mid \Theta, \rho \vDash T\} \subseteq \{\rho \mid \Theta, \rho \vDash S[h]\}$$

which, in English, is that the set of behaviors which satisfy $T$ is a subset of the set of behaviors which satisfy $S[h]$. Or, in other words, any sequence of states which satisfies $T$ also satisfies $S[h]$.

Composition is particularly elegant in TLA where composition is achieved using nothing more than logical conjunction [3]. Given two formulae $T_1$ and $T_2$ which each restrict the possible set of behaviors, the conjunction $T_1 \wedge T_2$ is satisfied by the intersection of the behaviors of $T_1$ and $T_2$. If $T_1$ and $T_2$ involve non-overlapping sets of variables, then the behaviors which satisfy the conjunction can be viewed as pairs of behaviors which satisfy the two components. On the other hand, if the sets of variables overlap, then the conjunction describes a combined model which is restricted by both formulae. Thus, a natural way of building large TLA specifications is as the conjunction of smaller component models where communication is achieved by variable sharing.

## 3.5 Variable Hiding

So far, we have only considered quantification in TLA over rigid variables, but it is also possible to quantify over flexible variables. Naively, we might define the meaning of existential quantification over flexible variables as being given by the existential in the meta logic. First, let $\rho \uplus (\mathbf{x} \mapsto d)$ denote the behavior which for each time step gives the same state as $\rho$ except that for every time $n \in \mathbf{N}$, $\mathbf{x}$ is assigned the value $d(n)$. Then we can interpret the existential by simply quantifying over $d$:

$$\theta, \rho \vDash \exists \mathbf{x}.T \quad \text{iff the exists } d \in \mathcal{D}^{\mathbf{N}} \text{ such that } \theta, \rho \uplus (\mathbf{x} \mapsto d) \vDash T$$

However, this definition suffers from a major flaw. Namely, it is not invariant under stuttering. To see the problem, consider the temporal formula

$$T \triangleq (\mathbf{x} = \mathbf{y}) \wedge \Box[(\mathbf{x} = \mathbf{x}' = \mathbf{y} = \mathbf{y}' - 1) \vee (\mathbf{y} = \mathbf{y}' = \mathbf{x} + 1 = \mathbf{x}')]_{\mathbf{x},\mathbf{y}}$$

where $E_1 = E_2 = E_3 = E_4$ is syntactic sugar for $E_1 = E_2 \wedge E_2 = E_3 \wedge E_3 = E_4$. The idea is that $T$ is satisfied if $\mathbf{x}$ and $\mathbf{y}$ represent a pair of numbers which start equal and advance in the following fashion: first $\mathbf{y}$ is incremented while $\mathbf{x}$ stays the same, then $\mathbf{x}$ is incremented to "catch up" while $\mathbf{y}$ stays the same bringing the two variables to the same value, at which point, the cycle repeats. Using $T$ and the existential we can construct a formula in a single variable

$$\exists \mathbf{y}.T$$

however, in so doing we have constructed a formula which is not stuttering invariant. Namely, if $\rho$ is the behavior which assigns $\mathbf{x}$ the sequence of states $1, 1, 2, 2, 3, 3, 4, 4, \ldots$ then we can use $d = 1, 2, 2, 3, 3, 4, 4, \ldots$ to prove that $\exists \mathbf{y}.T$ is satisfied by $\rho$. However, if $\rho'$ is the behavior which assigns $\mathbf{x}$ the sequence of states $1, 2, 3, 4, 5, \ldots$ then $\exists \mathbf{y}.T$ can not be satisfied by $\rho'$. This is concerning because $\rho'$ is formed simply by eliminating stutters from $\rho$ and so it should be the case that $\rho \equiv \rho'$.

As stuttering invariance is the crucial property which TLA is designed to achieve, this naive existential quantification over flexible variables is not permissible. However, we can construct an alternative semantics for the existential which *is* stuttering invariant simply by forcing stuttering invariance into the definition [26].

$$\theta, \rho \vDash \exists \mathbf{x}.T \quad \text{iff the exists } d \in \mathcal{D}^{\mathbf{N}} \text{ and } \rho' \text{ such that } \rho \equiv \rho' \text{ and } \theta, \rho' \uplus (\mathbf{x} \mapsto d) \vDash T$$

The updated definition preserves stuttering invariance by construction and so the main property of our system is met.

Having the existential satisfy stuttering invariance can be interpreted as allowing quantified flexible variables to change value *between* time steps of the behavior. Allowing such changes is precisely what we want, since, we do not want to be able to observe how quickly states change. The stuttering invariance means that a formula $\exists \mathbf{x}.T$ is satisfied by a behavior $\rho$ which can be, in some sense, implemented by a different behavior $\rho'$ that satisfies $T$ by taking multiple steps for each step of $\rho$.

However, the alternative semantics for the TLA existential is surprising because it means that the TLA semantics is not a standard Kripke semantics for modal logic. Such an outcome is worrying: how do we know that the connective we are calling an existential really behaves like a logical existential?

In order to resolve this problem, it makes sense to consider an alternative semantics of TLA with a continuous, real valued interpretation of time [24]. The ordinary TLA semantics defines a temporal logic where the time is discrete and so modeled by natural numbers. In contrast, we normally perceive time to be a continuous, and thus real valued, quantity. The use of natural numbers for time means that there is no intermediate time between two time steps and this is the reason why the naive existential does not work. By contrast, there is always a real number between two (non equal) real numbers.

However, in order to construct a real valued semantics of TLA we must be careful to ensure that we are still modeling digital systems. States of a digital do not vary continuously: they change episodically between values which they hold for positive amounts of time. To capture these we restrict temporal varying values to only certain functions of real numbers. Namely, a *non-zeno* function over a set $S$ is a function from $f$ non negative real numbers to $S$ such that

1. for every non negative real number $t$ there exists a positive number $\epsilon$ such that $\forall t', t \leq t' \leq t + \epsilon, f(t) = f(t')$ and
2. for every increasing sequence $t_0, t_1, t_2, \ldots$ such that $f(t_i) \neq f(t_{i+1})$ the set $\{t_i\}$ is unbounded.

These two conditions ensure that a non-zeno function not change too quickly: the first by guaranteeing that each state is held for positive time, while the second ensures that only a finite number of states are visited in any finite length of time. We use the notation $S^{\mathbb{R}^+}$ to refer to the set of non-zeno functions over $S$.

The challenge with using real valued behaviors for the semantics of TLA is in interpreting the primed variables $\mathbf{x}'$ as these should be given by the value of the variable at the "next" time and in general, there is no next real number. However, non-zeno functions allow us to define a `next` time function. Let $f \in S^{\mathbb{R}^+}$ and $t$ be a non-negative real number. $\texttt{next}(f,t)$ is intuitively, the least number greater than $t$ such that $f(t) \neq f(\texttt{next}(f,t))$ or $t$ if there is no $t'$ greater than $t$ such that $f(t') \neq f(t)$. Formally,

$$\begin{array}{ll} \texttt{next}(f,t) \triangleq t & \forall t' \geq t, f(t') = f(t) \\ \texttt{next}(f,t) \triangleq \mathbf{min}(\{t' \mid t' \geq t \land f(t') \neq f(t)\}) & \texttt{otherwise} \end{array}$$

However, the minimum of a set of real numbers is not well defined in general, so we must prove this function is well defined. The idea is that if there exists a $t' \geq t$ such that $f(t') \neq f(t)$ then it works out that

$$\mathbf{min}(\{t' \mid t' \geq t \land f(t') \neq f(t)\}) = \mathbf{sup}(\{t' \mid t' \geq t \land \forall t'', t \leq t'' \leq t' \Rightarrow f(t) = f(t')\})$$

which is well defined when **sup** is the notation for the supremum. The reason this works out is two fold: first there exists a positive epsilon such that

$$\begin{aligned} \forall s, (&\mathbf{sup}(\{t' \mid t' \geq t \land \forall t'', t \leq t'' \leq t' \Rightarrow f(t) = f(t')\}) \\ &\leq s \\ &\leq \mathbf{sup}(\{t' \mid t' \geq t \land \forall t'', t \leq t'' \leq t' \Rightarrow f(t) = f(t')\}) + \epsilon) \\ &\Rightarrow f(s) = f(\mathbf{sup}(\{t' \mid t' \geq t \land \forall t'', t \leq t'' \leq t' \Rightarrow f(t) = f(t')\})) \end{aligned}$$

since $f$ is non-zeno, but that means $f(\mathbf{sup}(\{t' \mid t' \geq t \land \forall t'', t \leq t'' \leq t' \Rightarrow f(t) = f(t')\})) \neq f(t)$ since if they were equal then $\mathbf{sup}(\{t' \mid t' \geq t \land \forall t'', t \leq t'' \leq t' \Rightarrow f(t) = f(t')\}) + \epsilon$ would be in the set $\{t' \mid t' \geq t \land \forall t'', t \leq t'' \leq t' \Rightarrow f(t) = f(t')\}$ which is clearly a contradiction. And, similarly, given any $s$ such that $s \geq t$ and $f(s) \neq f(t)$ we know that $s$ is an upper bound of $\{t' \mid t' \geq t \land \forall t'', t \leq t'' \leq t' \Rightarrow f(t) = f(t')\}$ and so $s \leq \mathbf{sup}(\{t' \mid t' \geq t \land \forall t'', t \leq t'' \leq t' \Rightarrow f(t) = f(t')\})$.

With the ability to select a next state, it is possible to give a real time semantics for TLA. The semantics for actions and expressions is identical to what is used with the discrete time semantics. All that differs is the interpretation of temporal predicates. In the real time semantics, a behavior $\tau \in S^{\mathbb{R}^+}$ is a non-zeno function over the set $S$ of states mapping from flexible variables to the domain $\mathcal{D}$. We use $\tau^k$ for the map $\lambda r.\tau(r+k)$. If $\theta$ is a first order interpretation and $\tau$ is such a behavior, we define the satisfiability relation $\theta, \tau \vDash_{\mathbb{R}} T$ by induction on $T$. The majority of the connectives are interpreted nearly identically to how they are in the discrete time semantics.

$$\begin{array}{ll}
\theta, \tau \models_{\mathbb{R}} P^n(E_1, \ldots, E_n) & \text{iff } \mathcal{F}(P^n)([\![E_1]\!](\theta, \tau(0)), \ldots, [\![E_n]\!](\theta, \tau(0))) = \texttt{true} \\
\theta, \tau \models_{\mathbb{R}} T_1 \wedge T_2 & \text{iff } \theta, \tau \models_{\mathbb{R}} T_1 \text{ and } \theta, \tau \models_{\mathbb{R}} T_2 \\
\theta, \tau \models_{\mathbb{R}} \neg T & \text{iff } \theta, \tau \not\models_{\mathbb{R}} T \\
\theta, \tau \models_{\mathbb{R}} \exists x.T & \text{iff there is some } v \in \mathcal{D} \text{ such that } (\theta, x \mapsto v), \tau_{\mathbb{R}} \models T \\
\theta, \tau \models_{\mathbb{R}} \Box T & \text{iff for every } k \in \mathbb{R} \text{ such that } \theta, \tau^k \models_{\mathbb{R}} T \\
\theta, \tau \models_{\mathbb{R}} \Diamond T & \text{iff there exists some } k \in \mathbb{R} \text{ where } \theta, \tau^k \models_{\mathbb{R}} T
\end{array}$$

The main difference comes in handling the action lifting connective where we use the `next` function instead of simply incrementing the time.

$$\begin{array}{ll}
\theta, \tau \models_{\mathbb{R}} \Box[A]_{\mathbf{x_i}} \text{iff for every } k \in \mathbb{R} \text{ either} & \theta, \tau(k), \tau(\texttt{next}(\tau, k)) \models A \\
& \text{or for every } i, \tau(k)(\mathbf{x_i}) = \tau(\texttt{next}(\tau, k))(\mathbf{x_i})
\end{array}$$

The other major difference is in the handling of the existential connective over flexible variables. Here, the real time semantics no longer needs to force stuttering invariance. Using $\tau \uplus (\mathbf{x} \mapsto f)$ for the continuous behavior which at any time $t$ corresponds to $\tau$ for every variable except $\mathbf{x}$ and where $(\tau \uplus (\mathbf{x} \mapsto f))(t)(\mathbf{x}) = f(t)$, the interpretation of flexible variables becomes the standard interpretation for Kripke style semantics.

$$\theta, \tau \models_{\mathbb{R}} \exists \mathbf{x}.T \quad \text{iff the exists } f \in \mathcal{D}^{\mathbb{R}^+} \text{ such that } \theta, \tau \uplus (\mathbf{x} \mapsto f) \models_{\mathbb{R}} T$$

A formula $T$ is *continuously valid* written $\models_{\mathbb{R}} T$ if for every $\theta$ and continuous behavior $\tau$ we have $\theta, \tau \models T$.

Stuttering gave us a way of relating discrete time behaviors which, intuitively, differed only in their rate. There is a similar way of relating continuous time behaviors. A *time transform* is a function on the non-negative real numbers which is one-to-one, onto, and continuous. It follows that time transforms map 0 to 0, are monotonic, and have inverses which are themselves time transforms. Two behaviors $\tau$ and $\tau'$ are *time transform equivalent*, written $\tau \sim \tau'$ if there exists a time transform $f$ such that $\tau = \tau' \cdot f$. Since time transforms are invertible we know that $\tau = \tau' \cdot f$ means $\tau \cdot f^{-1} = \tau' \cdot f \cdot f^{-1} = \tau'$ and so $\sim$ is symmetric. Similarly, since the identity function is a time transform and time transforms compose, $\sim$ is reflexive and transitive. The notion of time transform equivalence correspondence to stuttering equivalence in TLA, and, in particular, all formulae are invariant under time transformation.

**Lemma 1.** *If $\tau \sim \tau'$ and $\theta, \tau \models_{\mathbb{R}} T$ then $\theta, \tau' \models_{\mathbb{R}} T$.*

This holds even though the semantics for existential quantification of flexible variables does not force equivalence. Further, time transform equivalence is useful in the following way: any discrete behavior $\rho$ yields a continuous behavior $\lambda r.\rho(\lfloor r \rfloor)$ which is clearly non-zeno. We see at once that stuttering equivalent discrete behaviors will yield continuous behaviors which are time transform equivalent. Similarly, given a continuous behavior $\tau$ we can construct a discrete behavior $\texttt{disc}(\tau)$ which captures each change in $\tau$

$$\begin{aligned}\texttt{disc}(\tau)(n) &\triangleq \texttt{change}(\tau)(n)\\ \texttt{change}(\tau)(0) &\triangleq 0\\ \texttt{change}(\tau)(n+1) &\triangleq \texttt{next}(\tau, \texttt{change}(\tau)(n)).\end{aligned}$$

Together, these two operations define a sort of equivalence between discreet and continuous behaviors.

**Lemma 2.** *1. For any $\tau \sim \tau'$, $\texttt{disc}(\tau) = \texttt{disc}(\tau')$,*

*2. for any $\rho \equiv \rho'$, $\lambda r.\rho(\lfloor r \rfloor) \sim \lambda r.\rho'(\lfloor r \rfloor)$,*

*3. for every $\tau$, $\tau \sim \lambda r.\texttt{disc}(\tau)(\lfloor r \rfloor)$, and*

*4. and for every $\rho$, $\rho \equiv \texttt{disc}(\lambda r.\rho(\lfloor r \rfloor))$.*

Further, the ability to travel between discrete and continuous behaviors allows us to show that the two semantics are equivalent.

**Lemma 3.** *$\theta, \tau \vDash_{\mathbb{R}} T$ if and only if $\theta, \texttt{disc}(\tau) \vDash T$.*

Finally, this is enough to show that the two semantics have the same notion of validity.

**Theorem 2** ([24])**.** *$\vDash_{\mathbb{R}} T$ if and only if $\vDash T$.*

Our journey through the real time semantics is principally important for one reason: it shows that, indeed, the "existential quantifier" over flexible variables really is an existential in the sense of logic, even though its definition is unusual and is, certainly, at odds with the view of TLA as being nothing but a restriction of LTL. Additionally, a variant of TLA based on a real time semantics has proven useful for verifying cyber-physical systems such as autonomous vehicles where real valued physics interacts with discrete digital components [38].

Variable hiding allows us to have specifications which make use of pieces of state which are not externally observable. For example, we built a specification for $\text{Vending2}_{\text{TLA}}$ which made use of an extra variable **temp** which plays an important role in defining the refined vending machine, but which is perhaps not externally observable to systems (e.g. users) interacting with the vending machine. We could, therefore, instead work with the specification

$$\exists \textbf{temp}.\text{Vending2}_{\text{TLA}}$$

which hides the variable **temp**. In general, TLA specifications are likely to begin by existentially quantifying over hidden variables in this way. A convenient result is that, under certain reasonable assumptions, refinement of specifications with hidden variables implies the existence of refinement mappings between their underlying state spaces [1].

## 4 Temporal Theories

A rather different perspective on temporal logic was presented by Fiadeiro and Maibaum in their work on modular temporal theories [16]. We will refer to the specifications constructed in this style as "temporal theories" and the approach as "the temporal theory approach."

The temporal theory approach uses temporal logic for describing specifications. However, this is done in a very different way than what was seen with logics like TLA. The difference is that while a specification in TLA was encoded as a logical formula, here we represent specifications as logical theories (hence the name "temporal theory"). This means that composition and refinement cannot simply be accounted for as logical connectives. However, it also enables a potentially richer account of specifications and how they are interrelated.

Further, while TLA was constructed as an extension of *single sorted* first order logic, and both Event-B and TLA+ are based on untyped set theoretic reasoning, temporal theories are based on multi-sorted or *typed* first order logic where equality is the basic relation of interest.

In order to see temporal theories in action, let us construct a temporal theory corresponding to the vending machine at the highest level of abstraction. As before, we model the amount of money in the system with a variable **cash**, called an attribute, indicating the amount of money in the system. Because we work in a typed system, we give this variable the sort $\mathbb{N}$. Unlike what we saw in Event-B, that **cash** has type $\mathbb{N}$ is not a theorem which derives from the specification, but intrinsically part of the definition since every expression must have a type.

We need now axioms for reasoning about how **cash** evolves over time. We could have an axiom which asserts that at any time the next value of **cash** is greater than its current value. However, we already witnessed how that created problems for refinement. Instead, let us theorize the existence of an action which the system can take called `transaction`. We then have an axiom, which is always true, which asserts the liveness property that this action keeps happening:

$$\Diamond \texttt{transaction}.$$

Further, we should have an axiom that anytime `transaction` happens **cash** increases:

$$\texttt{transaction} \rightarrow \mathbf{cash} < \circ\mathbf{cash}.$$

Here we, in a slight change of notation from TLA, use $\circ$ to indicate the next value of an expression.

There is an additional detail we must consider: in Event-B and TLA we assumed the existence of a flexible math language which includes collections such as $\mathbb{N}$ and operations like $<$. However, in temporal theories we make no such assumption. Instead, we must include $\mathbb{N}$ as a sort symbol in the theory and account for $<$ ourselves. As [16] did not utilize relation symbols except for indexed action symbols (which, as action symbols, have truth values that vary with time), we will instead introduce a new sort `bool` together with a function symbol `lt` which takes two natural numbers as inputs and outputs a `bool`, and rewrite our axiom as

$$\mathtt{transaction} \rightarrow \mathtt{lt}(\mathbf{cash}, \circ\mathbf{cash}) = \mathtt{true}.$$

Further, the constant `true` must be included as a function symbol with zero arguments (that is, a constant) of sort `bool` as part of the theory. Similarly, we would have a constant `false` of sort `bool`, a constant `Z` or sort $\mathbb{N}$, and a function `S` which takes an $\mathbb{N}$ as an argument and outputs an $\mathbb{N}$. With these, we still need to axiomatize $\mathbb{N}$, `bool`, and `lt` (note that rigid variables are implicitly universally quantified)

$$\begin{aligned}
&\neg(\mathtt{true} = \mathtt{false}) \\
&\neg(\mathtt{Z} = \mathtt{S}(n)) \\
&t = \mathtt{true} \vee t = \mathtt{false} \\
&n = \mathtt{Z} \vee \exists m, n = \mathtt{S}(m) \\
&\mathtt{lt}(n, m) = \mathtt{true} \rightarrow \mathtt{lt}(m, n) = \mathtt{false} \\
&\mathtt{lt}(n, m) = \mathtt{true} \vee n = m \vee \mathtt{lt}(m, n) = \mathtt{true} \\
&\mathtt{lt}(\mathtt{Z}, \mathtt{S}(n)) = \mathtt{true} \\
&\mathtt{lt}(n, m) = \mathtt{true} \leftrightarrow \mathtt{lt}(\mathtt{S}(n), \mathtt{S}(m)) = \mathtt{true}.
\end{aligned}$$

A *temporal theory signature* (called an *object signature* in Fiadeiro and Maibaum's original paper) consists of a a four tuple $< \mathcal{S}, \mathcal{O}, \mathcal{A}, \mathcal{G} >$ where

- $\mathcal{S}$ is a set of *sorts*,
- $\mathcal{O}$ is a $\mathcal{S}^\star \times \mathcal{S}$ indexed collection of function symbols,
- $\mathcal{A}$ is a $\mathcal{S}^\star \times \mathcal{S}$ indexed collection of attribute symbols and
- $\mathcal{G}$ is a $\mathcal{S}^\star$ indexed collection of action symbols.

Intuitively, $\mathcal{S}$ and $\mathcal{O}$ will yield the (standard) notion of a many sorted algebra signature. By contrast $\mathcal{A}$ contains the evolving or programmatic variables; these variables can have parameters similar to function and array variables in programming languages. The action symbols are a form of relation symbols and will refer to transitions which may be taken in a particular time step.

Each temporal theory signature defines a syntax for expressions and logical formulae. Because we are working in a typed setting only well typed expressions and formulae are interpretable. For clarity, we present the syntax in two steps. First we give the syntax of raw terms without considering types in Figure 2. Here we use $f$ as the meta variable for symbols in $\mathcal{O}$, $\mathbf{a}$ for symbols in $\mathcal{A}$, and $g$ for symbols in $\mathcal{G}$.

$$
\begin{aligned}
E \in \text{Expressions} &::= f(E_1, \ldots, E_n) \mid \mathbf{a}(E_1, \ldots, E_n) \mid x \mid \circ E \\
P, Q \in \text{Formulae} &::= P \wedge Q \mid \neg P \mid \exists x : S.P \mid g(E_1, \ldots, E_n) \mid \text{BEG} \mid \circ P \mid \Box P \mid \Diamond P
\end{aligned}
$$

Figure 2: Syntax for Temporal Theories

The special formula BEG serves as a marker for the begging of time: it is true at the first time and at no other. As before, we consider the other logical connectives such as $\vee$, $\rightarrow$ and $\forall$ to be derived from $\neg$, $\wedge$ and $\exists$. The type system, Figure 3, is given in two judgments: $\Gamma \vdash E : S$ says that $E$ is a well formed expression of sort $S \in \mathcal{S}$ while $\Gamma \vdash P$ **wff** says that $P$ is a well formed formula; in either case $\Gamma$ is interpreted as a set of pairs $x : S$ associating a variable $x$ with a sort $S$

$$
\frac{x : S \in \Gamma}{\Gamma \vdash x : S} \qquad \frac{\Gamma \vdash E : S}{\Gamma \vdash \circ E : S}
$$

$$
\frac{f \in \mathcal{O}([S_1, \ldots, S_n], S) \qquad \Gamma \vdash E_1 : S_1 \ldots \Gamma \vdash E_n : S_n}{\Gamma \vdash f(E_1, \ldots, E_n) : S}
$$

$$
\frac{\mathbf{a} \in \mathcal{A}([S_1, \ldots, S_n], S) \qquad \Gamma \vdash E_1 : S_1 \ldots \Gamma \vdash E_n : S_n}{\Gamma \vdash \mathbf{a}(E_1, \ldots, E_n) : S}
$$

$$
\frac{\Gamma \vdash E_1 : S \qquad \Gamma \vdash E_2 : S}{\Gamma \vdash E_1 = E_2 \ \mathbf{wff}} \qquad \frac{\Gamma \vdash P \ \mathbf{wff} \qquad \Gamma \vdash Q \ \mathbf{wff}}{\Gamma \vdash P \wedge Q \ \mathbf{wff}}
$$

$$
\frac{\Gamma, x : S \vdash P \ \mathbf{wff} \qquad x : S' \notin \Gamma}{\Gamma \vdash \exists x : S.P \ \mathbf{wff}} \qquad \frac{\Gamma \vdash P \ \mathbf{wff}}{\Gamma \vdash \neg P \ \mathbf{wff}} \qquad \frac{}{\Gamma \vdash \text{BEG} \ \mathbf{wff}}
$$

$$
\frac{\Gamma \vdash P \ \mathbf{wff}}{\Gamma \vdash \circ P \ \mathbf{wff}} \qquad \frac{\Gamma \vdash P \ \mathbf{wff}}{\Gamma \vdash \Box P \ \mathbf{wff}} \qquad \frac{\Gamma \vdash P \ \mathbf{wff}}{\Gamma \vdash \Diamond P \ \mathbf{wff}}
$$

$$
\frac{g \in \mathcal{G}([S_1, \ldots, S_n], S) \qquad \Gamma \vdash E_1 : S_1 \ldots \Gamma \vdash E_n : S_n}{\Gamma \vdash g(E_1, \ldots, E_n) \ \ \mathbf{wff}}
$$

Figure 3: Typing Rules for Temporal Theories

For $< \mathcal{S}, \mathcal{O}, \mathcal{A}, \mathcal{G} >$ a temporal theory signature, a $< \mathcal{S}, \mathcal{O}, \mathcal{A}, \mathcal{G} >$ *interpretation* is a four tuple $< \mathbf{S}, \mathbf{O}, \mathbf{A}, \mathbf{G} >$ where

- $\mathbf{S}$ is a map assigning to each symbol in $\mathcal{S}$ a set,
- $\mathbf{O}$ is a map which assigns each $f \in \mathcal{O}([S_1, \ldots, S_n], S)$ a function $\mathbf{S}(S) \times \ldots \times \mathbf{S}(S_n) \rightarrow \mathbf{S}(S)$,
- $\mathbf{A}$ is a map which assigns each $\mathbf{a} \in \mathcal{A}([S_1, \ldots, S_n], S)$ a function $\mathbf{S}(S) \times \ldots \times \mathbf{S}(S_n) \times \mathbb{N} \rightarrow \mathbf{S}(S)$ and
- $\mathbf{G}$ is a map which assigns each $g \in \mathcal{G}([S_1, \ldots, S_n], S)$ a function $\mathbf{S}(S) \times \ldots \times \mathbf{S}(S_n) \times \mathbb{N} \rightarrow \{\mathtt{true}, \mathtt{false}\}$.

Intuitively, temporal theory interpretations are the natural extension of algebras to accommodate the temporally dependent symbols in $\mathcal{A}$ and $\mathcal{G}$ where, as was the case in TLA, time is interpreted as simply a natural number. Specifically, we can interpret symbols in $\mathcal{A}$ as time varying *attributes* of a system and symbols in $\mathcal{G}$ as *events* which either happen at a specific time or do not. However, we use a slightly more restricted notion of model than temporal theory interpretations to allow for a modular approach to temporal logic. The problem with the notion of interpretation given above is that it is possible for an attribute to change at a time when no events in $\mathcal{G}$ are interpreted to occur. This is a problem because it means we can not use the event structure of the signature to constrain our understanding of when things change.

To get around this, we define the notion of a *locus* of a signature $< \mathcal{S}, \mathcal{O}, \mathcal{A}, \mathcal{G} >$. A locus is a $< \mathcal{S}, \mathcal{O}, \mathcal{A}, \mathcal{G} >$-interpretation $< \mathbf{S}, \mathbf{O}, \mathbf{A}, \mathbf{G} >$ such that attributes only change at times *witnessed* by an event. Formally, a time $t \in \mathbb{N}$ is witnessed by an event if there is some $g \in \mathcal{G}([S_1, \ldots, S_n])$ and $v_1 \in \mathbf{S}(S_1), \ldots, v_n \in \mathbf{S}(S_n)$ such that $\mathbf{G}(g)(v_1, \ldots, v_n, t) = \mathtt{true}$. $< \mathbf{S}, \mathbf{O}, \mathbf{A}, \mathbf{G} >$ is a locus if for every $t \in \mathbb{N}$ such that $t$ is not witnessed then $\mathbf{A}(\mathbf{a})(v_1, \ldots, v_n, t) = \mathbf{A}(\mathbf{a})(v_1, \ldots, v_n, t+1)$ for every $\mathbf{a} \in \mathcal{A}([S_1, \ldots, S_n], S)$ and $v_1 \in \mathbf{S}(S_1), \ldots, v_n \in \mathbf{S}(S_n)$.

Although we will only care about loci in the definition of validity, it is possible to interpret a statement in the language of a temporal theory signature using any interpretation. Specifically, we can define the semantics for expressions and formulae as in Figure 4.

We say a formula $\Gamma \vdash P$ **wff** is *true* with respect to an interpretation $< \mathbf{S}, \mathbf{O}, \mathbf{A}, \mathbf{G} >$ if for every $t \in \mathbf{N}$ and $\theta$ which is well typed in the sense that $x : S \in \Gamma$ implies $\theta(x) \in \mathbf{S}(S)$ then $\mathbf{S}, \mathbf{O}, \mathbf{A}, \mathbf{G}, \theta, t \vDash P$.

A *temporal theory* (also called an "object description") is a pair of a temporal theory signature $\Theta =< \mathcal{S}, \mathcal{O}, \mathcal{A}, \mathcal{G} >$ and a set $\Phi$ of formulae such that $P \in \Phi$ implies $\vdash P$ in the signature $< \mathcal{S}, \mathcal{O}, \mathcal{A}, \mathcal{G} >$. A locus $l$ for $\Theta$ is a *model* for $< \Theta, \Phi >$ if every every formula in $\Phi$ is true with respect to $l$. A well formed formula is valid for $< \Theta, \Phi >$ if it is true with respect to all of $< \Theta, \Phi >$'s models.

Because we use loci instead of general interpretations, any temporal theory with signature $< \mathcal{S}, \mathcal{O}, \mathcal{A}, \mathcal{G} >$ has a valid formula which encodes the locality principle that either

$$
\begin{aligned}
[\![x]\!]_{\mathbf{S},\mathbf{O},\mathbf{A},\mathbf{G},\theta,t} &= \theta(x)\\
[\![f(E_1,\ldots,E_n)]\!]_{\mathbf{S},\mathbf{O},\mathbf{A},\mathbf{G},\theta,t} &= \mathbf{O}(f)([\![E_1]\!]_{\mathbf{S},\mathbf{O},\mathbf{A},\mathbf{G},\theta,t},\ldots,[\![E_n]\!]_{\mathbf{S},\mathbf{O},\mathbf{A},\mathbf{G},\theta,t})\\
[\![\mathbf{a}(E_1,\ldots,E_n)]\!]_{\mathbf{S},\mathbf{O},\mathbf{A},\mathbf{G},\theta,t} &= \mathbf{A}(\mathbf{A})([\![E_1]\!]_{\mathbf{S},\mathbf{O},\mathbf{A},\mathbf{G},\theta,t},\ldots,[\![E_n]\!]_{\mathbf{S},\mathbf{O},\mathbf{A},\mathbf{G},\theta,t},t)\\
[\![\circ E]\!]_{\mathbf{S},\mathbf{O},\mathbf{A},\mathbf{G},\theta,t} &= [\![E]\!]_{\mathbf{S},\mathbf{O},\mathbf{A},\mathbf{G},\theta,t+1}\\
\mathbf{S},\mathbf{O},\mathbf{A},\mathbf{G},\theta,t \vDash P\wedge Q &\text{ iff } (\mathbf{S},\mathbf{O},\mathbf{A},\mathbf{G},\theta,t\vDash P) \text{ and } (\mathbf{S},\mathbf{O},\mathbf{A},\mathbf{G},\theta,t\vDash Q)\\
\mathbf{S},\mathbf{O},\mathbf{A},\mathbf{G},\theta,t \vDash \neg P &\text{ iff } \mathbf{S},\mathbf{O},\mathbf{A},\mathbf{G},\theta,t \nvDash P\\
\mathbf{S},\mathbf{O},\mathbf{A},\mathbf{G},\theta,t \vDash \exists x:S.P &\text{ iff there exists } v\in\mathbf{S}(S) \text{ s.t. } \mathbf{S},\mathbf{O},\mathbf{A},\mathbf{G},(\theta,x\mapsto v),t\vDash P\\
\mathbf{S},\mathbf{O},\mathbf{A},\mathbf{G},\theta,t \vDash g(E_1,\ldots,E_n) &\text{ iff } \mathbf{G}(g)([\![E_1]\!]_{\mathbf{S},\mathbf{O},\mathbf{A},\mathbf{G},\theta,t},\ldots,[\![E_n]\!]_{\mathbf{S},\mathbf{O},\mathbf{A},\mathbf{G},\theta,t},t)=\texttt{true}\\
\mathbf{S},\mathbf{O},\mathbf{A},\mathbf{G},\theta,t \vDash \text{BEG} &\text{ iff } t=0\\
\mathbf{S},\mathbf{O},\mathbf{A},\mathbf{G},\theta,t \vDash \circ P &\text{ iff } \mathbf{S},\mathbf{O},\mathbf{A},\mathbf{G},\theta,t+1\vDash P\\
\mathbf{S},\mathbf{O},\mathbf{A},\mathbf{G},\theta,t \vDash \Box P &\text{ iff for all } k\geq t\ \mathbf{S},\mathbf{O},\mathbf{A},\mathbf{G},\theta,k\vDash P\\
\mathbf{S},\mathbf{O},\mathbf{A},\mathbf{G},\theta,t \vDash \Diamond P &\text{ iff there existsl } k\geq t \text{ s.t. } \mathbf{S},\mathbf{O},\mathbf{A},\mathbf{G},\theta,k\vDash P
\end{aligned}
$$

Figure 4: Semantics of Temporal Theory Expressions and Formulae

no attribute changes or there is an action. Specifically, if we use the syntax $\bigvee_{v\in T}$ for the disjunction of formulae parameterized by $T$ and $\bigwedge_{v\in T}$ for the conjunction, and use $\exists x_i : S_i$ as shorthand for $\exists x_1 : S_1 .\ldots\exists x_n : S_n$ (and similarly for $\forall$) we get a locality principle of the form

$$(\bigvee_{g\in\mathcal{G}([S_1,\ldots,S_n])} \exists x_i : S_i.g(x_1,\ldots,x_n)) \vee \bigwedge_{\mathbf{a}\in\mathcal{A}([S_1,\ldots,S_n],S)} \forall x_i : S_i.\mathbf{a}(x_1,\ldots,x_n) = \circ\mathbf{a}(x_1,\ldots,x_n)$$

which together with the formulae in $\Phi$ forms the axioms specific to a temporal theory.

The locality property in temporal theories and the stuttering invariance property in TLA are of fundamentally different characters. Stuttering invariance is a limitation on what questions can be asked, while locality is a limitation on what interpretations are models. However, they seem to be both attempts to ensure that local reasoning about a specification can be used as that specification is embedded into a larger system.

For temporal theories the relevant notion of refinement will come from what is called a *temporal theory morphism*. Before formally defining these morphisms, however, let us consider a refinement of the vending machine to include an extra attribute **temp** to model temporarily inserted cash as a temporal theory. As before, we would need to include sorts for naturals and booleans, together with constants for truth values, numbers, and comparison. The axioms associated with these operations would be identical to what we saw in the first vending machine temporal theory. In addition, it would be helpful to have a function symbol `plus` taking two naturals and producing a natural. A potential set of axioms about $\texttt{plus}$ would be:

$$\begin{aligned}
&\texttt{plus}(\texttt{Z}, n) = n\\
&\texttt{plus}(\texttt{S}(n), m) = \texttt{S}(\texttt{plus}(n, m))\\
&\texttt{plus}(n, m) = \texttt{plus}(m, n)\\
&\texttt{plus}(n, \texttt{plus}(m, r)) = \texttt{plus}(\texttt{plus}(n, m), r)\\
&\texttt{lt}(n, m) = \texttt{true} \to \texttt{lt}(\texttt{plus}(n, r), \texttt{plus}(m, r)) = \texttt{true}\\
&\texttt{lt}(n, \texttt{plus}(n, \texttt{S}(m))) = \texttt{true}.
\end{aligned}$$

With the help of the plus function, we can describe the temporal behavior of the updated vending machine. Here, instead of using the single action symbol $\texttt{transaction}$ we will have two: $\texttt{insert\_money}$ and $\texttt{vend}$. The final axioms we need are:

$$\begin{aligned}
&\texttt{insert\_money} \to \texttt{lt}(\mathbf{temp}, \circ\mathbf{temp}) = \texttt{true}\\
&\texttt{insert\_money} \to \mathbf{cash} = \circ\mathbf{cash}\\
&\texttt{vend} \to (\circ\mathbf{cash}) = \texttt{plus}(\mathbf{cash}, \mathbf{temp})\\
&\texttt{vend} \to \texttt{Z} = \circ\mathbf{temp}\\
&\texttt{vend} \to \neg(\mathbf{temp} = \texttt{Z})\\
&\Diamond\texttt{vend} \land \Diamond\texttt{insert\_money}.
\end{aligned}$$

Now, it what sense can we say that this second theory is a refinement of the first? Well, the idea is that we can embed the symbols of the first theory into the second: most of the symbols are interpreted as themselves, but we also map $\texttt{transaction}$ to $\texttt{vend}$. Then, under this embedding all of the true statements of the first theory are also true in the second. We see this by checking all of the axioms: in most cases the result is automatic. The one that is not is the fact that

$$\texttt{vend} \to \texttt{lt}(\mathbf{cash}, \circ\mathbf{cash}) = \texttt{true}.$$

But, here we know that $\texttt{vend}$ implies that $\circ\mathbf{cash} = \texttt{plus}(\mathbf{cash}, \mathbf{temp})$ so we only need to show that $\texttt{lt}(\mathbf{cash}, \texttt{plus}(\mathbf{cash}, \mathbf{temp}) = \texttt{true}$. However we know that $\texttt{vend}$ also implies $\mathbf{temp}$ is not $\texttt{Z}$ and so it must be the case that $\mathbf{temp} = \texttt{S}(m)$ for some $m$ and, because of how we axiomatized $\texttt{plus}$ the property we are after follows.

This is the essential idea of a temporal theory morphism: we embed one theory into another showing that the target (the refined theory) is strong enough to validate all the true statements of the initial theory.

Given two temporal theory signatures $< \mathcal{S}, \mathcal{O}, \mathcal{A}, \mathcal{G} >$ and $< \mathcal{S}', \mathcal{O}', \mathcal{A}', \mathcal{G}' >$ a morphism of temporal theory signatures $\phi :< \mathcal{S}, \mathcal{O}, \mathcal{A}, \mathcal{G} > \to < \mathcal{S}', \mathcal{O}', \mathcal{A}', \mathcal{G}' >$ consists of a function mapping symbols in $\mathcal{S}$ to symbols in $\mathcal{S}'$, symbols in $\mathcal{O}$ to symbols in $\mathcal{O}'$ and so on, which is well typed. Specifically,

- $\forall f \in \mathcal{O}([S_1, \ldots, S_n], S)$, $\phi(f) \in \mathcal{O}'([\phi(S_1), \ldots, \phi(S_n)], \phi(S))$,
- $\forall \mathbf{a} \in \mathcal{A}([S_1, \ldots, S_n], S)$, $\phi(\mathbf{a}) \in \mathcal{A}'([\phi(S_1), \ldots, \phi(S_n)], \phi(S))$, and
- $\forall g \in \mathcal{G}([S_1, \ldots, S_n])$, $\phi(\mathbf{a}) \in \mathcal{G}'([\phi(S_1), \ldots, \phi(S_n)])$.

A morphism of signatures naturally induces a translation function on expressions and formulae. We will, in a slight abuse of notation, use $\phi(E)$ for the translation of expression $E$ using $\phi$, and $\phi(P)$ for the translation of formula $P$ using $\phi$ (the definition is given in Figure 5).

$$\begin{aligned}
\phi(f(E_1, \ldots, E_n) &= \phi(f)(\phi(E_1), \ldots, \phi(E_n)) \\
\phi(\mathbf{a}(E_1, \ldots, E_n) &= \phi(\mathbf{a})(\phi(E_1), \ldots, \phi(E_n)) \\
\phi(x) &= x \\
\phi(\circ E) &= \circ\phi(E) \\
\phi(P \wedge Q) &= \phi(P) \wedge \phi(Q) \\
\phi(\neg P) &= \neg\phi(P) \\
\phi(\exists x : S.P) &= \exists x : \phi(S).\phi(P) \\
\phi(g(E_1, \ldots, E_n) &= \phi(g)(\phi(E_1), \ldots, \phi(E_n)) \\
\phi(\mathrm{BEG}) &= \mathrm{BEG} \\
\phi(\circ P) &= \circ\phi(P) \\
\phi(\Box P) &= \Box\phi(P) \\
\phi(\Diamond P) &= \Diamond\phi(P)
\end{aligned}$$

Figure 5: Translation Function Extension

**Theorem 3.** *For any theory signature morphism* $\phi :< \mathcal{S}, \mathcal{O}, \mathcal{A}, \mathcal{G} > \rightarrow < \mathcal{S}', \mathcal{O}', \mathcal{A}', \mathcal{G}' >$

1. *if* $\Gamma \vdash E : S$ *is true according to* $< \mathcal{S}, \mathcal{O}, \mathcal{A}, \mathcal{G} >$ *then* $\phi(\Gamma) \vdash \phi(E) : \phi(S)$ *according to* $< \mathcal{S}', \mathcal{O}', \mathcal{A}', \mathcal{G}' >$ *and*
2. *if* $\Gamma \vdash P$ ***wff*** *is true according to* $< \mathcal{S}, \mathcal{O}, \mathcal{A}, \mathcal{G} >$ *then* $\phi(\Gamma) \vdash \phi(P)$ ***wff*** *according to* $< \mathcal{S}', \mathcal{O}', \mathcal{A}', \mathcal{G}' >$.

The notion of theory signature morphism also induces a translation on interpretations. Suppose $\phi :< \mathcal{S}, \mathcal{O}, \mathcal{A}, \mathcal{G} > \rightarrow < \mathcal{S}', \mathcal{O}', \mathcal{A}', \mathcal{G}' >$ is a signature morphism and and $\Psi =< \mathbf{S}', \mathbf{O}', \mathbf{A}', \mathbf{G}' >$ is a $< \mathcal{S}', \mathcal{O}', \mathcal{A}', \mathcal{G}' >$-interpretation. Then, we define $\Psi|_\phi$ called $\Theta$'s *reduct* along $\phi$ to be the interpretation $< \mathbf{S}, \mathbf{O}, \mathbf{A}, \mathbf{G} >$ formed by composing $\Psi$ with $\phi$, that is

- $\mathbf{S}(S) = \mathbf{S'}(\phi(S))$,
- $\mathbf{O}(f) = \mathbf{O'}(\phi(f)))$,
- $\mathbf{A}(\mathbf{a}) = \mathbf{A'}(\phi(\mathbf{a})))$, and
- $\mathbf{G}(g) = \mathbf{G'}(\phi(g)))$.

It is immediate that the reduct operation and the extension of a morphism of signatures to formulae are related by Goguen and Burstall's satisfaction condition for institutions [20].

**Theorem 4.** *Given any signature morphism $\phi :< \mathcal{S}, \mathcal{O}, \mathcal{A}, \mathcal{G} > \to < \mathcal{S}', \mathcal{O}', \mathcal{A}', \mathcal{G}' >$, $< \mathcal{S}', \mathcal{O}', \mathcal{A}', \mathcal{G}' >$-interpretation $\Theta$ and formula $P$ such that $\vdash P$ **wff** according to $< \mathcal{S}, \mathcal{O}, \mathcal{A}, \mathcal{G} >$ we have that*

$$(\Theta|_{\phi}, \emptyset, t \vDash P) \Leftrightarrow (\Theta, \emptyset, t \vDash \phi(P))$$

*for any time $t$.*

Given temporal theories $< \Theta_1, \Phi_1 >$ and $< \Theta_2, \Phi_2 >$ a *temporal theory morphism* $\phi :< \Theta_1, \Phi_1 > \to < \Theta_2, \Phi_2 >$ is a morphism of signatures $\phi : \Theta_1 \to \Theta_2$ such that for any valid $P$ which is valid for $< \Theta_1, \Phi_1 >$ we have $\phi(P)$ valid for $< \Theta_2, \Phi_2 >$.

We can further characterize temporal theory morphisms in terms of their behavior with respect to loci. Observe specifically that if $\phi :< \Theta_1, \Phi_1 > \to < \Theta_2, \Phi_2 >$ is a temporal theory morphism and $\Psi$ is a $< \Theta_2, \Phi_2 >$ locus then for any formula $P$ which is valid for $< \Theta_1, \Phi_1 >$ we know that $\phi(P)$ is valid for $< \Theta_2, \Phi_2 >$ and so $\Psi \vDash \phi(P)$ which means that $\Psi|_{\phi} \vDash P$. Since $\Psi|_{\phi}$ models all true statements of $< \Theta_1, \Phi_1 >$ it must in particular model all of $\Phi_1$ and the locality axiom and so therefore be a $< \Theta_1, \Phi_1 >$-locus. Going the other way, suppose that $< \Theta_1, \Phi_1 >$ and $< \Theta_2, \Phi_2 >$ are temporal theories and that $\phi : \Theta_1 \to \Theta_2$ is a morphism of signatures such that for every $< \Theta_2, \Phi_2 >$ locus $\Psi$, $\Psi|_{\phi}$ is a $< \Theta_1, \Phi_1 >$ locus. Then, given any valid formula of $P$ of $< \Theta_1, \Phi_1 >$ and $\Psi|_{\phi} \vDash P$ and so $\Psi \vDash \phi(P)$ for every $< \Theta_2, \Phi_2 >$-locus which means that $\phi(P)$ is valid for $< \Theta_2, \Phi_2 >$. Thus, $\phi$ must be a morphism of temporal theories $< \Theta_1, \Phi_1 > \to < \Theta_2, \Phi_2 >$.

**Theorem 5.** *If $\phi$ is a temporal theory morphism $< \Theta_1, \Phi_1 > \to < \Theta_2, \Phi_2 >$ if and only if it is a morphism of temporal theory signatures $\Theta_1 \to \Theta_2$ and for every $< \Theta_2, \Phi_2 >$-locus $\Psi$ the reduct $\Psi|_{\phi}$ is a $< \Theta_1, \Phi_1 >$-locus.*

The above theorem gives a characterization of temporal theory morphisms in terms of models, but we can also give a characterization which is about the axioms of the models and thus may be more amenable to proving that a given signature morphism is a theory morphism. By definition a temporal theory morphism preserves all true statements. So a morphism $< \Theta_1, \Phi_1 > \to < \Theta_2, \Phi_2 >$ must map all of the axioms in $\Phi_1$ to valid statements in $< \Theta_2, \Phi_2 >$. However, preserving the axioms $\Phi_1$ alone is not sufficient because the axioms in $\Phi_1$ all the true statements of $< \Theta_1, \Phi_1 >$ as we work only with loci. We have noted however that we can derive the set of true formulae for $< \Theta_1, \Phi_1 >$ from $\Phi_1$ together with a locality axiom. This axiom must be preserved by any theory morphism (which preserve all valid statements) but, moreover, preserving locality and the axioms of $\Phi_1$ is sufficient since those axioms are enough to derive all the true statements in $< \Theta_1, \Phi_1 >$ using only temporal reasoning principles which are preserved by any morphism of signatures.

Theory morphisms induce a notion of refinement for temporal theories. Namely, $< \Theta_2, \Phi_2 >$ refines $< \Theta_1, \Phi_1 >$ if there exists a temporal theory morphism $\phi :< \Theta_1, \Phi_1 > \rightarrow < \Theta_2, \Phi_2 >$. We see this is an appropriate notion of refinement because any formula $P$ which is valid in $< \Theta_1, \Phi_1 >$ induces a valid formula $\phi(P)$ in $< \Theta_2, \Phi_2 >$.

## 4.1 Background on Category Theory

Category theory provides a set of tools and terminology for mathematical reasoning at a very high level of abstraction.

A category is consists of the following data

- a collection of "objects",
- for every pair of objects $A$ and $B$ a collection of morphisms $hom(A, B)$,
- for every object $A$ a morphism $id_A \in hom(A, A)$ and
- for any three objects $A$ $B$ and $C$, an operation $\cdot : hom(B, C) \times hom(A, B) \rightarrow hom(A, C)$.

Where,

1. given any objects $A, B, C, D$ and morphisms $f \in hom(D, C)$, $g \in hom(C, B)$ and $h \in hom(A, B)$ were have that $(f \cdot g) \cdot h = f \cdot (g \cdot h)$, and
2. for and objects $A$ and $B$ and any morphism $f \in hom(A, B)$, $f \cdot id_A = f = id_B \cdot f$.

In any category we write $f : A \rightarrow B$ to indicate that the morphism $f$ is in $hom(A, B)$. Categories provide a common abstraction for reason about a great variety of mathematical objects. Perhaps the quintessential category is the category of SET whose objects are sets and whose morphisms are functions such that $f : A \rightarrow B$ has the usual meaning. Here $id_A$ is the identity function on $A$ and $\cdot$ is just function composition. SET motivates the use of the term "collection" instead of "set" for the objects and morphisms of a category, as the "set of all sets" presents foundational problems. However, here we will not dwell on these foundational issues, and simply assume that large collections of some sort (be they proper classes, types, or sets of unreachable cardinality) are available as we require little in the way of details about the behavior of these collections.

Similar to SET is the category GROUP whose objects are groups and where $hom(A, B)$ is the set of group homomorphisms from $A$ to $B$. Of course, $id_A$ is once again the identity function and $\cdot$ is function composition as the identity function is always a homomorphism and composition of two group homomorphisms yields a new group homomorphism. Similar categories can be constructed whose objects are other sorts of algebraic object such as monoids or rings and whose morphisms are functions which preserve the algebraic structure we are interested. Similarly, the category TOP has as objects topological spaces and as morphisms continuous functions.

All of these are examples of categories whose underlying objects are sets with structure and whose morphisms are functions which preserve that structure. However, there are other categories. The category REL has as objects, like the category SET, sets, but has relations as morphisms, specifically $hom(A,B) = P(A \times B)$. In REL, the identity map $1_A$ is the least reflexive relation on $A$, namely $\{(x,x) \mid x \in A\}$ while the composition is the composition of relations: for $f : B \to C$ and $g : A \to B$, $f \cdot g = \{(x,y) \mid \exists z \in B, (x,z) \in f \wedge (z,y) \in g\}$. One can check that the associativity and identity laws hold for REL just as they do for the various categories of sets with structure. REL is another large category in that its objects do not form a set, however, this is not a requirement either. Indeed, many interesting categories are small.

For example, let $S$ be a set. The discrete category on $S$ has as objects just the elements of $S$ and for every pair of objects $A, B \in S$, the set of morphisms $hom(A,B)$ is the empty set if $A \neq B$ and the set with one element $\{\emptyset\}$ when $A = B$. Here, there is only one possibility for composition (if $f \cdot g$ type checks then $f \cdot g = \{\emptyset\}$) and identity ($1_A = \emptyset$). Indeed, we could define the notion of "set" (modulo issues of size) as a category where the only morphisms are identity morphisms.

Another example comes when $(S, \otimes, 1)$ is a monoid. Then, a category is formed consisting of a single object $A$ where $hom(A,A) = S$ The identity morphism $1_A$ is just the unit of the monoid, namely, 1 and the composition operator is just defined as the composition from the monoid, namely $\otimes$. The identity and associativity properties for the category flow from those properties for the monoid. An alternative definition of a monoid therefore would be a category with exactly one object.

A third way to construct categories generally is to start with a pre-ordered set $(S, \leq)$. A category is produced whose objects are the elements of $S$ and where $hom(A,B) = \{\emptyset\}$ whenever $A \leq B$ and is the empty set otherwise. We see immediately that $1_A = \emptyset$ which is well defined since $A \leq B$ by reflexivity of the preorder. Similarly, $f \cdot g = \emptyset$ which is well defined since if $f : B \to C$ and $g : A \to B$ then $A \leq B$ and $B \leq C$ which by transitivity of the pre-order implies that $A \leq C$. The identity and associativity axioms are of course automatic. Any category which has at most one morphism between any pair of objects is called a "pre-order category" or, just a "pre-order."

Thus, categories generalize common mathematical constructs such as monoids, pre-orders, and sets. Moreover, "large categories" generalize the basic algebraic properties of sets and functions. The utility of this is that various important ideas can be expressed in a general way by framing them in terms purely of objects and morphisms. One such general definition concerns combining objects. For example, two objects $A$ and $B$ are said to be isomorphic if there are a pair of morphisms $f : A \to B$ and $g : B \to A$ such that $f \cdot g = id_A$ and $g \cdot f = id_B$. The morphisms $f$ and $g$ are referred to as an isomorphism. Isomorphic objects are interesting because they can be said to "behave the same" in the sense that when $A$ and $B$ are isomorphic any property true about $A$ which can be expressed just in terms of morphisms and other objects must must also be true about $B$. In the category SET two sets are isomorphic if they have the same cardinality while in the category GROUP isomorphic objects share all group theoretic properties.

Another example of a common definition which works across all categories is a co-span, which describes the the situation when two objects both are equipped with morphisms into a common target. More precisely, if $A$ and $B$ are objects in a category, a co-span of $A$ and $B$ consists of some other object $C$ together with morphisms $f : A \to C$ and $g : B \to C$. Diagrammatically:

$$\begin{tikzcd} A \arrow[dr, "f"] & & B \arrow[dl, "g"] \\ & C & \end{tikzcd}$$

An interesting case occurs when we have not just any co-span but the *best* co-span. This is called a *co-product*. If $A$ and $B$ are objects the co-product of $A$ and $B$ is, if it exists, and object denoted $A + B$ such that there exists maps $\iota_1 : A \to A + B$ and $\iota_2 : B \to A + B$

$$\begin{tikzcd} A \arrow[dr, "\iota_1"] & & B \arrow[dl, "\iota_2"] \\ & A+B & \end{tikzcd}$$

and where, further, given any other object $C$ and maps $f : A \to C$ and $g : B \to C$ there exists a unique map $u : A + B \to C$ such that $u \cdot \iota_1 = f$ and $u \cdot \iota_2 = g$.

$$\begin{tikzcd} A \arrow[dr, "\iota_1"] \arrow[ddr, "f"'] & & B \arrow[dl, "\iota_2"'] \arrow[ddl, "g"] \\ & A+B \arrow[d, dotted, "\exists! u"] & \\ & C & \end{tikzcd}$$

In the category of sets the co-product of $A$ and $B$ is, up to isomorphism, the set $\{(\emptyset, x) \mid x \in A\} \cup \{(\{\emptyset\}, y) \mid y \in B\}$ with $\iota_1$ the map $x \mapsto (\emptyset, x)$ and $\iota_2$ the map $y \mapsto (\{\emptyset\}, y)$. If $C$ is another set with $f : A \to C$ and $g : B \to C$ then the universal map $u$ is given by

$$\begin{aligned} (\emptyset, x) &\mapsto f(x) \\ (\{\emptyset\}, y) &\mapsto g(y). \end{aligned}$$

In a pre-order interpreted as a category, the co-product $A + B$ (if it exists) is the least upper bound of $A$ and $B$.

Beyond co-products, there are other interesting concepts that can be expressed purely in terms of diagrams. For example, the dual notion to a co-span, a span from $A$ to $B$ consists of some object $C$ and morphisms $f : C \to A$ and $g : C \to B$.

Now, suppose $A$ and $B$ are objects and there exists a span with morphisms $r_1 : S \to A$ and $r_2 : S \to B$ then the push-out of $r_1$ and $r_2$ consists of an object $A + r_1, r_2B$ and morphisms $\iota_1 : A \to A + r_1, r_2B$ and $\iota_2 : B \to A + r_1, r_2B$ which commutes with $r_1$ and $r_2$ in the sense that $\iota_1 \cdot r_1 = \iota_2 \cdot r_2$,

$$\begin{tikzcd}
& S \arrow[dl, "r_1"'] \arrow[dr, "r_2"] & \\
A \arrow[dr, "\iota_1"] & & B \arrow[dl, "\iota_1"] \\
& A +_{r_1,r_2} B &
\end{tikzcd}$$

and, moreover, this is the best such span in that given any other object $C$ and morphisms $f : A \to C$ and $g : B \to C$ where $f \cdot r_1 = g \cdot r_2$ then there exists a unique morphism $u$ such that $f = u \cdot \iota_1$ and $g = u \cdot \iota_2$.

$$\begin{tikzcd}
& S \arrow[dl, "r_1"'] \arrow[dr, "r_2"] & \\
A \arrow[r, "\iota_1"] \arrow[dr, "f"'] & A +_{r_1,r_2} B \arrow[d, dotted, "\exists! u"] & B \arrow[l, "\iota_1"] \arrow[dl, "g"] \\
& C &
\end{tikzcd}$$

In SET, the push-out corresponds to taking the quotient of the co-product under the equivalence relation generated by the common source. Specifically given $r_1 : S \to A$ and $r_2 : S \to B$ let $R$ be the least equivalence relation on $\{(\emptyset, x) \mid x \in A\} \cup \{(\{\emptyset\}, y) \mid y \in B\}$ such that given any $s \in S$ we have $((\emptyset, r_1(s)), (\{\emptyset\}, r_2(s))) \in R$. Then $A +_{r_1,r_2} B = (\{(\emptyset, x) \mid x \in A\} \cup \{(\{\emptyset\}, y) \mid y \in B\})/R$.

As with the co-product, the definition of a push-out describes an object which, if it exists, is unique up to isomorphism.

Another general definition that works in any category is of an initial object. The initial object, which we will denote as 0, is, if it exists, an object such that for every object $A$ there exists a unique morphism $0 \to A$. In SET the empty set is initial. In a pre-order interpreted as a category the initial object is the least element of the pre-order.

The notions of co-product, push-out, and initial object are all generalized by the concept of co-limit. A diagram consists of a collection of objects and morphisms between those objects in a category which is closed under identity and composition. A co-cone of a diagram consists of an object $C$ and for each object $A$ in the diagram a morphism $c_A : A \to C$ such that given any morphism $f : A_1 \to A_2$ in the diagram $c_{A_2} \cdot f = c_{A_1}$. A co-limit of a diagram is a co-cone $(C, \{c_i\})$ such that for any other co-cone $(C', \{c'_i\})$ there exists a unique morphism $u : C \to C'$ such that for all objects $A$ in the diagram $u \cdot c_A = c'_A$.

The co-product of $A$ and $B$ is the co-limit of the diagram containing $A$ and $B$ as objects and only $id_A$ and $id_B$ as morphisms. Similarly, if $r_1 : S \to A$ and $r_2 : S \to B$ the push-out $A +_{r_1,r_2} B$ is the co-limit of the diagram containing $A$, $B$, and $S$ as objects and $r_1$, $r_2$, and the identity morphisms as morphisms. The initial object of a category is the co-limit of the empty diagram.

A diagram is finite if it contains only a finite number of objects and morphisms. It turns out that a category has an initial object and all push-outs if and only if it has a co-limit for each of its finite diagrams. Such a category is said to have all finite co-limits.

## 4.2 Categorical Account of Composition

Temporal theories leverage refinement to form a basis for composition. The idea, is to specify composition in terms of refinement. In this way, refinement structure can serve as a higher level specification language for complex systems. Specifically, by viewing refinement in categorical terms we can leverage general definitions from category theory to account for composition.

There exists a category whose objects are temporal theory signatures and whose morphisms are, naturally, morphisms of temporal theory signatures. In this category the identity morphism on a signature simply maps each symbol to itself while composition of morphisms is just composition of the underlying functions. More importantly, temporal theories and temporal theory morphisms form a category by taking the categorical structure on their underlying signatures.

If $A$ and $B$ are two models then the composition $A \oplus B$ should be a model which refines both $A$ and $B$. Using the style of modular temporal theories where refinement is captured by morphisms going from more abstract to more concrete, this relation can be represented diagrammatically.

$$\begin{tikzcd}
A \arrow[dr] & & B \arrow[dl] \\
& A + B &
\end{tikzcd}$$

Here it is immediately the case that $A \oplus B$ refines anything which $A$ refines (and similarly for $B$) since refinements compose in the "vertical" direction. In categorical language, however, this diagram simply states that the composition $A \oplus B$ must form a co-span of $A$ and $B$.

However, the co-span diagram does not fully specify the meaning of composition in terms of refinement as it does not tell us when a theory $C$ should refine $A \oplus B$. An idea then would be to model the composition of $A$ and $B$ as the *co-product* $A + B$. Then, not only does $A + B$ refine $A$ as well as $B$ but that $A + B$ is *the most abstract* model which refines $A$ and refines $B$. That is, by the definition of the co-product, if we are provided some other theory $C$ where $C$ refines $A$ and $C$ refines $B$ it must be the case that $C$ also refines $A+B$. Since in temporal theories we do not expect refinements to form a pre-order, we are additionally interested in *which* refinement we are given. In particular, we should have a refinement $\iota_1$ by which $A + B$ refines $A$ and a refinement $\iota_2$ by which $A + B$ refines $B$

$$\begin{tikzcd}
A \arrow[dr, "\iota_1"] & & B \arrow[dl, "\iota_2"'] \\
& A+B &
\end{tikzcd}$$

such that given *any* temporal theory $C$ which refines $A$ using refinement $f$ and which refines $B$ using refinement $g$ there exists a unique refinement $u$ by which $C$ refines $A + B$ such that the composition of $\iota_1$ and $u$ yields *the same* refinement as $f$ and the composition of $\iota_2$ and $g$ is the *the same* refinement as $g$.

$$\begin{tikzcd}
A \arrow[dr, "\iota_1"] \arrow[ddr, "f"'] & & B \arrow[dl, "\iota_2"'] \arrow[ddl, "g"] \\
& A+B \arrow[d, dotted, "\exists! u"] & \\
& C &
\end{tikzcd}$$

The notion of a co-product gives a general characterization of the composition of specifications applicable to nearly any specification language which has refinement. However, this characterization is abstract and does not directly tell us what composition of specifications should like in the particular case. Further, it is a *characterization* of composition in terms of how composition interacts with refinement, not a recipe for combining specifications. Yet, all is not lost. Just as a general formula for co-products exists in the category of sets, so too does so a formula exist in the category of temporal theories.

We defined temporal theories by first defining temporal theory signatures. We defined temporal theory morphisms by first defining temporal theory signature morphisms. So, before figuring out how to combine temporal theories we should discuss how to combine temporal theory signatures. The co-product $< \mathcal{S}, \mathcal{O}, \mathcal{A}, \mathcal{G} > + < \mathcal{S}', \mathcal{O}', \mathcal{A}', \mathcal{G}' >$ of temporal theory signatures $< \mathcal{S}, \mathcal{O}, \mathcal{A}, \mathcal{G} >$ and $< \mathcal{S}', \mathcal{O}', \mathcal{A}', \mathcal{G}' >$ consists of the point wise co-product $< \mathcal{S}+, \mathcal{S}', \mathcal{O} + \mathcal{O}', \mathcal{A} + \mathcal{A}', \mathcal{G} + \mathcal{G}' >$ where $\mathcal{S}+, \mathcal{S}'$ is the disjoint union of $\mathcal{S}$ and $\mathcal{S}$

$$\{\iota_1(S) \mid S \in \mathcal{S}\} \cup \{\iota_2(S) \mid S \in \mathcal{S}'\}$$

where we use $\iota_i$ as a formal constant and the other combinations such as $\mathcal{O} + \mathcal{O}'$ are constructed by taking the disjoint union of symbols in $\mathcal{O}$ and $\mathcal{O}'$ where a symbol $O \in \mathcal{O}([S_1, \ldots, S_n], S)$ leads to a symbol $\iota_1(O) \in (\mathcal{O} + \mathcal{O}')([\iota_1(S_1), \ldots, \iota_1(S_n)], \iota_1(S))$. We can check that this is indeed a co-product: the formal constants $\iota_1$ and $\iota_2$ can be treated as functions which preserve typing and thus are theory signatures morphisms.

$$\iota_1 :< \mathcal{S}, \mathcal{O}, \mathcal{A}, \mathcal{G} > \to (< \mathcal{S}, \mathcal{O}, \mathcal{A}, \mathcal{G} > + < \mathcal{S}', \mathcal{O}', \mathcal{A}', \mathcal{G}' >)$$
$$\iota_2 :< \mathcal{S}', \mathcal{O}', \mathcal{A}', \mathcal{G}' > \to (< \mathcal{S}, \mathcal{O}, \mathcal{A}, \mathcal{G} > + < \mathcal{S}', \mathcal{O}', \mathcal{A}', \mathcal{G}' >)$$

Now, suppose $< \Theta_1, \Phi_1 >$ and $< \Theta_2, \Phi_2 >$ are temporal theories. The co-product of these two theories should be given as the co-product of the signatures, together with axioms making the diagram work. In particular, all of the axioms of $\Phi_1$ and $\Phi_2$ will be needed to be carried over to $< \Theta_1, \Phi_1 > + < \Theta_2, \Phi_2 >$. Beyond these, however, we also need axiomatic support for locality. In particular, let $L_1$ be the locality law associated with $\Theta_1$ and $L_2$ the locality law associated with $\Phi_2$. Then the axioms of the co-product will also contain $\iota_1(L_1)$ and $\iota_2(L_2)$ indicating that a change to an attributed from $\Theta_1$ only happens at an action from $\Theta_1$.

$$< \Theta_1, \Phi_1 > + < \Theta_2, \Phi_2 > \triangleq < \Theta_1 + \Theta_2, \{\iota_1(A) | A \in \Phi_1\} \cup \{\iota_2(A) | A \in \Phi_2\} \cup \{\iota_1(L_1), \iota_2(L_2)\} >$$

The co-product of temporal theories can additionally be understood in terms of its loci. A locus for $< \Theta_1, \Phi_1 > + < \Theta_2, \Phi_2 >$ is exactly equivalent to a pair of a locus $l_1$ for $< \Theta_1, \Phi_1 >$ and a locus $l_2$ for $< \Theta_2, \Phi_2 >$ where symbols which come from $\Theta_1$ are interpreted by using $l_1$ and symbols which come from $\Theta_2$ are interpreted by using $l_2$. Any such pair of loci yields a $< \Theta_1, \Phi_1 > + < \Theta_2, \Phi_2 >$ locus, and vice versa.

The co-product ensures the absence of spurious sharing and synchronization, however, sometimes, indeed often, we *want* components to communicate. The idea of specifications and refinements forming a category fits with a view advanced by Gougen as his first dogma for categories in computer science that "to each species of mathematical structure, there corresponds a a category whose objects have that structure, and whose morphisms preserve it." [19] However, it is Gougen's fifth dogma that address composition of specifications: "Given a category of widgets, the operation of putting a system of widgets together to form some super-widget corresponds to taking the co-limit of the diagram of widgets that shows how to interconnect them." Categorically, binary co-products are a kind of co-limit but they are not the only one. Indeed, temporal theories have all finite co-limits and more general co-limits support forms of composition with sharing. In order to construct general co-limits, we first observe that there is an initial temporal theory. Indeed, the initial temporal theory simply has no symbols and no axioms [16].

Now suppose $r_1 : \Theta_A \to \Theta_1$ and $r_2 : \Theta_A \to \Theta_1$ are morphisms of theory signatures. Then, then push-out $\Theta_1 +_{r_1,r_2} \Theta_2$ is given by simply taking the disjoint unions of each set of symbols (as with the co-product) and then quotienting out by the equivalence relation where $\iota_1(S_1) = \iota_2(S_2)$ whenever $r_1(S_1) = r_2(S_2)$. Extending this to theories requires no additional work beyond what was needed for co-products. Namely, if $r_1 :< \Theta_A, \Phi_A > \to < \Theta_1, \Phi_1 >$ and $r_2 :< \Theta_A, \Phi_A > \to < \Theta_2, \Phi_2 >$ are theory morphisms then the push-out $<$

$\Theta_1, \Phi_1 > +_{r_1,r_2} < \Theta_2, \Phi_2 >$ is given as the push-out of signatures together with $\Theta_1 +_{r_1,r_2} \Theta_2$ and a the set of axioms as in the co-product:

$$\{\iota_1(A) | A \in \Phi_1\} \cup \{\iota_2(A) | A \in \Phi_2\} \cup \{\iota_1(L_1), \iota_2(L_2)\}.$$

Push-outs allow for constructing specifications as components while enabling sharing. For example, if we want to specify a system consisting of two components communicating over a shared wire we might first construct a model of the wire and then model each component as a refinement of the wire and take the push-out to construct a complete system.

We have already seen two versions of the vending machine as temporal theories. The first, which we will here refer to as `Vending1` described the machine in terms of a single attribute **cash** and a single action `transaction`. The second machine which we will call `VendingTemp` added an additional variable **temp** and had two transactions `insert_money` and `vend`. There was also a canonical inclusion morphism $r_1 : \texttt{Vending1} \to \texttt{VendingTemp}$ which simply mapped every symbol to the symbol with the same name except for `transaction` which mapped to `vend`.

However, we can extend the vending machine in other ways. If we wish to model the dispensing of goods, we might build a machine `VendingInventory` with would be similar to `Vending1` except that in addition to $\mathbb{N}$ and `bool` it would have a new sort `item`, a function symbol `price` taking an `item` and valued in $\mathbb{N}$, a attribute symbol **supply** taking an `item` as an argument and yielding an $\mathbb{N}$, and an an attribute symbol **selection** of sort `item`. In addition to the action symbol `transaction` we would add a new attribute symbol for making a new selection `select` which could be parameterized by an `item`. As a transaction now will decrease the inventory, we also add a new action symbol `service` which allows the inventory to increase. Here, we would retain the axioms for numbers, booleans, and comparison, and, indeed, include the symbol `plus` and axioms about it used in `VendingTemp`. The remaining axioms would be:

$$\begin{aligned}
&\exists n, \texttt{price}(i) = \texttt{S}(n) \\
&\texttt{select}(i) \to i = \circ\textbf{selection} \\
&\texttt{select}(i) \to \textbf{cash} = \circ\textbf{cash} \\
&\texttt{select}(i) \to \textbf{supply}(j) = \circ\textbf{supply}(j) \\
&\texttt{transaction} \wedge i = \textbf{selection} \to \texttt{plus}(\textbf{cash}, \texttt{price}(i)) = \circ\textbf{cash} \\
&\texttt{transaction} \wedge i = \textbf{selection} \to \texttt{S}(\textbf{supply}(i)) = \circ\textbf{supply}(i) \\
&\texttt{transaction} \wedge \neg(\textbf{selection} = j) \to \textbf{supply}(j) = \circ\textbf{supply}(j) \\
&\texttt{service} \to \textbf{cash} = \circ\textbf{cash} \\
&\Diamond\texttt{service} \wedge \exists i, \texttt{lt}(\textbf{supply}(j), \circ\textbf{supply}(j)) = \texttt{true} \\
&\Diamond\texttt{transaction}
\end{aligned}$$

The canonical morphism $r_1$ from `Vending1` to `VendingInventory` simply maps each symbol to the symbol with the same name. Each of the axioms of `Vending1` become derivable from the axioms of `VendingInventory`. Additionally, the locality property holds by construction since we explicitly include that `service` and `select`$(i)$ do not change the value of **cash**.

Now we have two different refinements of `Vending1` incorporating different features of interest. Simply taking the co-product of `VendingInventory` and `VendingTemp` would yield a combined specification, but not one we would find useful because it would include two versions of the attribute **cash** and both the action `transaction` and the action `vend` having nothing to do with each other. Instead, however, we can combine the two refinements using the push-out construction. The specification `VendingTemp` $+_{r_1,r_2}$ `VendingInventory` is, modulo renaming of symbols, more akin to what we want. Namely, it has four action symbols `select`, `vend`, `insert_cash` and `service` and only a single attribute **cash** (as well as the attributes **temp**, **supply** and **selection**). The axioms of the push-out are approximately the union of the axioms of `VendingInventory` (with `transaction` renamed `vend`) and `VendingTemp`.

However, there is still a small problem with this characterization. We incorporated the function symbol `plus` into both `VendingInventory` and `VendingTemp`, however, this symbol did not originate in `Vending1` and so, as a consequence must appear twice in the push-out. As a consequence, the push-out contains an unfortunately complex set of axioms including:

$$\texttt{vend} \wedge i = \textbf{selection} \rightarrow \iota_1(\texttt{plus})(\textbf{cash}, \texttt{price}(i)) = \circ\textbf{cash}$$
$$\texttt{vend} \wedge i = \textbf{selection} \rightarrow \iota_2(\texttt{plus})(\textbf{cash}, \textbf{temp}) = \circ\textbf{cash}$$

and other such redundancies. Even if our axioms for `plus` were strong enough to show that $(\texttt{plus}(n,k) = \texttt{plus}(m,k)) \rightarrow n = m$ it is unlikely that we could show from these axioms that $\texttt{vend} \rightarrow \textbf{temp} = \texttt{price}(\textbf{selection})$, a property we would like to hold, as we have two different `plus` symbols.

A solution is to simply define a new specification which only describes the sorts and function symbols associated with $\mathbb{N}$ and `bool` and which are shared between both `VendingInventory` and `VendingTemp`. Let us call this shared specification, containing no attributes or actions, `PlusRules`. It is apparent, that there are canonical morphisms $p_1$ : `PlusRules` $\rightarrow$ `VendingTemp` and $p_2$ : `PlusRules` $\rightarrow$ `VendingInventory`. With this in place, we can construct a more complicated diagram demonstrating the full sharing structure.

$$\begin{tikzcd}
\texttt{PlusRules} \arrow[d, "p_1"] \arrow[dr, "p_2"] & \texttt{Vending1} \arrow[d, "r_2"] \arrow[dl, "r_1"'] \\
\texttt{VendingTemp} & \texttt{VendingInventory}
\end{tikzcd}$$

Taking the co-limit of this diagram yields a vending machine incorporating both the temporary money and the inventory which we call `VendingTempInventory`.

$$\begin{tikzcd}
\texttt{PlusRules} \arrow[d, "p_1"] \arrow[rr, "r_1"] & & \texttt{Vending1} \arrow[d, "r_2"] \arrow[ll, "p_2"'] \\
\texttt{VendingTemp} \arrow[dr] & & \texttt{VendingInventory} \arrow[dl] \\
& \texttt{VendingTempInventory} &
\end{tikzcd}$$

Note that, because we have general finite co-limits, this is guaranteed to exist. However, it may be informative to observe in detail how it can be constructed using co-products and push-outs. The idea is that if `VendingTemp` and `VendingInventory` refine *both* `PlusRules` and `Vending1` then we know they must also both refine $\texttt{PlusRules} + \texttt{Vending1}$, and, in particular there must be unique morphisms $u_1 : (\texttt{PlusRules} + \texttt{Vending1}) \to \texttt{VendingTemp}$ and $u_2 : (\texttt{PlusRules} + \texttt{Vending1}) \to \texttt{VendingInventory}$ such that the following diagram commutes:

$$\begin{tikzcd}
\texttt{PlusRules} \arrow[dd, "p_1"] \arrow[dr, "\iota_1"] \arrow[ddrr, bend left, "r_1"] & & \texttt{Vending1} \arrow[dd, "r_2"] \arrow[dl, "\iota_1"] \arrow[ddll, bend right, "p_2"'] \\
& \texttt{PlusRules} + \texttt{Vending1} \arrow[dl, "u_1"] \arrow[dr, "u_2"] & \\
\texttt{VendingTemp} & & \texttt{VendingInventory}.
\end{tikzcd}$$

We then set `VendingTempInventory` as the push-out of the span formed from by $u_1$ and $u_2$.

$$\texttt{VendingTempInventory} \triangleq \texttt{VendingTemp} +_{u_1,u_2} \texttt{VendingInventory}$$

As, then there exists a pair of morphism $\kappa_1 : \texttt{VendingTemp} \to \texttt{VendingTempInventory}$ and $\kappa_2 : \texttt{VendingInventory} \to \texttt{VendingTempInventory}$ such that the diagram

$$\begin{tikzcd}
\texttt{PlusRules} \arrow[dd, "p_1"] \arrow[dr, "\iota_1"] \arrow[ddrr, bend left, "r_1"] & & \texttt{Vending1} \arrow[dd, "r_2"] \arrow[dl, "\iota_1"] \arrow[ddll, bend right, "p_2"'] \\
& \texttt{PlusRules} + \texttt{Vending1} \arrow[dl, "u_1"] \arrow[dr, "u_2"] & \\
\texttt{VendingTemp} \arrow[dr, "\kappa_1"] & & \texttt{VendingInventory} \arrow[dl, "\kappa_1"] \\
& \texttt{VendingTempInventory} &
\end{tikzcd}$$

commutes. All that is left to show that this co-cone is a co-limit, which is equivalent to showing the existence of a universal arrow. Suppose that there was some object $T$ and morphisms $f : \texttt{VendingTemp} \to T$ and $g : \texttt{VendingInventory} \to T$ where $f \cdot p_1 = g \cdot p_2$ and $f \cdot r_1 = g \cdot r_2$. Now, observe that $f \cdot p_1 : \texttt{PlusRules} \to T$ and $g \cdot r_2 : \texttt{Vending1} \to T$, therefore, there exists a unique morphism $u_T : (\texttt{PlusRules} + \texttt{Vending1}) \to T$ where $f \cdot p_1 = u_T \cdot \iota_1$ and where $g \cdot r_2 = u_T \cdot \iota_2$. But, we know already that $u_1 \cdot \iota_1 = p_1$ and $u_1 \cdot \iota_2 = r_1$ and so therefore $f \cdot p_1 = f \cdot u_1 \cdot \iota_1$ and $g \cdot r_2 = f \cdot r_1 = f \cdot u_1 \cdot \iota_2$. Thus, $u_T = f \cdot u_1$. Similarly, $u_2 \cdot \iota_1 = p_2$ and $u_2 \cdot \iota_2 = r_2$ so $f \cdot p_1 = g \cdot p_2 = g \cdot u_2 \cdot \iota_1$ and $g \cdot r_2 = g \cdot u_2 \cdot \iota_2$ and so $u_T = g \cdot u_2$. Therefore, $f \cdot u_1 = g \cdot u_2$ which means, that since $\texttt{VendingTempInventory}$ is the push-out, there exists a unique morphism $u : \texttt{VendingTempInventory} \to T$ where $f = u \cdot \kappa_1$ and $g = u \cdot \kappa_2$.

## 5 Conclusions and Directions for Future Work

We have seen three formalisms for specifying reactive systems. Event-B provides a convenient notation for describing system specifications of state machines and supports an approach to iterative development of specifications by way of refinement. However, Event-B is limited when it comes to property specification. While Event-B can be used to express invariants it does not support liveness. Moreover, Event-B's notion of refinement does not preserve liveness properties, so even if an abstract specification seems to imply a liveness property of interest, its refinements and implementations may not.

By contrast, in supporting both the always and eventually modalities, temporal logics are well suited for expressing a broad range of safety and liveness properties. Further, TLA turns out to be well equipped for expressing system specifications as well as property specifications. By using implication for refinement and conjunction for composition, TLA allows us to express everything in a single framework. The guarantee that TLA formulae are stuttering invariant keeps TLA specifications from being too tight to effectively refine and fits well with philosophical intuition that time is relative and so how fast a system evolves should not be observable.

Temporal theories leverage temporal logic in a different way. Instead of treating specifications as formulae, specifications are given as logical theories. Temporal theories are typed, and no pre-set mathematics language is assumed. Instead, mathematical types and operators are included directly as part of the specification of a theory. Because different specifications are different theories instead of different formulae, refinement in temporal theories corresponds not to implication but to theory morphisms. However, the move to theory morphisms yields benefits as temporal theories and their morphisms can be viewed from a higher level using the perspective of category theory. As composition is modeled via co-limits, complex specifications can be built out of component parts where sharing is specified using refinement of component specifications. In this way, refinement takes center stage and serves as the basis of composition.

Temporal theories and TLA each have their benefits. There is a great conceptual elegance to modeling refinement as implication and composition as conjunction. These connectives are simple and easy to understand and mean that everything in TLA is included as part of a single logic. At the same time, the categorical approach used by temporal theories is interesting in providing a higher level, diagrammatic, language for constructing modular specifications. Temporal theories emphasize the importance of not simply knowing that a refinement between two specifications exists, but also knowing what the refinement is. A concrete specification might refine an abstract one in multiple ways, and knowing when two refinements are the same is crucial to understanding how specifications interrelate.

Further, while TLA is based on an untyped single-sorted first order logic, temporal theories are based on typed multi-sorted logic. While, in general, my personal preference is for multi-sorted logics, both designs have benefits. Interestingly, recent work has examined adding types to TLA for the purpose of improved automation [32]. Moreover, reworking TLA as a typed logic may lead to new insights.

For example, while the most obvious role for types in TLA is to fill the normal mathematical purpose of differentiating different sorts of mathematical objects (e.g. sets and natural numbers) a different dimension of distinction already appears even in untyped TLA. That is, TLA includes both rigid variables which do not change over time and flexible variables which do. Instead of handling rigid and flexible variables in an ad hoc manner where each receives its own form of quantification, they could instead be handled in a uniform way where there are simply variables and where "rigid integer" and "flexible integer" are simply different types. In this case, types do not simply classify the possible values of a variable at any given time, but also, how the values of those variables change over time.

Taking this idea further, we could imagine a whole host of types which classify the temporal evolution of variables more precisely. We might even allow for comprehension akin to set theory. Suppose $P$ is a (temporal) proposition over a single flexible variable $\mathbf{x}$ of type $T$, then we would have a new type $\{\mathbf{x} : T \mid P\}$ inhabited by objects of type $T$ whose behaviours satisfy $P$. In particular we would expect there to be an operator *proj* where if $E$ has type $\{\mathbf{x} : T \mid P\}$ then $proj(E)$ would have type $T$ and where it would always be the case that $\models P[proj(E)/\mathbf{x}]$. Using the comprehension type the rigid version of a type $T$ could be encoded as $\{\mathbf{x} : T \mid \Box[False]_{\mathbf{x}}\}$. Further, using comprehension could allow us to unify the diagrammatic perspective on refinement used with temporal theories with the interpretation of refinement as implication in TLA. If $P$ and $Q$ are both TLA formulae over a single variable $\mathbf{x}$ of type $T$ then $P$ refines $Q$ if it implies it. Just as in set theory, however, the implication $P \Rightarrow Q$ should correspond to a type inclusion $\{\mathbf{x} : T \mid P\} \subseteq \{\mathbf{x} : T \mid Q\}$.

Set inclusion is a kind of function, so by analogy we would expect there to be some "function" in TLA $r : \{\mathbf{x} : T \mid P\} \to \{\mathbf{x} : T \mid Q\}$. However, this cant just be any function:

we need $r$ to be a subset inclusion which means that $\mathit{proj}(r(E)) = \mathit{proj}(E)$ for all $E$.

$$\begin{tikzcd}
& T & \\
\{\mathbf{x} : T \mid P\} \arrow[ur, "\mathit{proj}"] \arrow[rr, "r"] & & \{\mathbf{x} : T \mid Q\} \arrow[ul, "\mathit{proj}"']
\end{tikzcd}$$

In order to make this work, of course, we would need to come up with a notion of "morphism" or "function" internal to an extended and typed TLA. But, in doing so we would end up in a situation where refinement as implication would correspond exactly to morphisms in a category (these commuting triangles are exactly morphism in the slice category over $T$). Of course, we oriented the arrow in the opposite direction from what we used with temporal theories, but by simply dualizing everything we did with temporal theories this becomes a non issue.

Further, a slight extension of the picture handles even the case where refinement requires a refinement mapping. Suppose $P$ is a property over a variable $\mathbf{x}$ of type $T$ but $Q$ is a property over a variable $\mathbf{y}$ of type $S$ and that $P \Rightarrow Q[h(\mathbf{x})/\mathbf{y}]$ then we would have some $r : \{\mathbf{x} : T \mid P\} \to \{\mathbf{y} : S \mid Q\}$ such that the diagram

$$\begin{tikzcd}
T \arrow[r, "h"] & S \\
\{\mathbf{x} : T \mid P\} \arrow[u, "\mathit{proj}"] \arrow[r, "r"] & \{\mathbf{y} : S \mid Q\} \arrow[u, "\mathit{proj}"]
\end{tikzcd}$$

commutes.

The natural logical setting for such a system is not simply multi-sorted first order logic but actually higher-order logic where we can have typing judgments like $P : T \to \mathit{Prop}$ to indicate that $P$ is a predicate over the type $T$. However, building a higher-order version of TLA seems surprisingly tricky. To get there first we would need an interpretation of the sort of propositions $\mathit{Prop}$. Propositions are semantically interpreted as functions from behaviors to truth values, so it might make sense for propositional variables to range over such functions. However, various problems emerge with that idea. For example, consider a formula of the form $\forall p : \mathit{Prop}.\exists \mathbf{x} : T.p \wedge Q$. Here, $p$ would range over propositions which would be semantically modeled as functions $\mathtt{state}^\star \to \mathtt{bool}$. However, the proposition $p \wedge Q$ happens under the existential and so would have to be modeled as a function $(\mathtt{state} \times S)^\star \to \mathtt{bool}$ for some set $S$ to account for the extra variable $\mathbf{x}$. It is also not immediately obvious if proposition variables should range over all functions $\mathtt{state}^\star \to \mathtt{bool}$ or only those which are stuttering invariant. An alternative design therefore would be to equate proposition variables with flexible variables of boolean type. A greater challenge comes then in interpreting the function type constructor. Any temporal logic formula $P$ with a free variable of type $T$ must yield an expression whose type is $T \to \mathit{Prop}$. Worryingly, if propositions are simply time varying booleans then some such functions will be anti-causal in that the value of their output at the initial time will depend on the value of their input at future times. Conversely, we do not want there to be functions of type $T \to Prop$ which are not stuttering invariant, however, what stuttering invariant means in this context is non-obvious. The main insight which could provide a way out of these challenges is that the function type in a higher-order logic is the internal representation of its notion of morphisms which we have seen should correspond to refinement. Thus, a deep understanding of refinement in TLA and the possibility of a higher-order extension go hand in hand.

That said, is a higher-order version of TLA really that important? The current single sorted variant of TLA is well understood, while the meaning of higher-order TLA is not clear. Perhaps then higher-order TLA is not worth pursuing. In response to this, let us observe some natural places where a higher-order variant of TLA would come in handy.

The most immediate use for combining higher-order reasoning and temporal logic is in order to build specifications which incorporate both higher-order mathematics and temporal components. We actually saw this very situation in this exam although we did not dwell on it at the time. In formalizing the vending machine as a temporal theory we needed a type of natural numbers and an axiomatization of those numbers. In that regard we defined constants for zero and the successor and a number of axioms, however, he did not give an induction axiom. That is because, in a first order setting, induction is not an axiom but rather an axiom schema. As such, the ability to reason about natural numbers in these theories is quite limited. Of course, this problem could easily be avoided using higher-order quantification. As currently construed, the temporal theory formalism, which lacks a powerful math language, is significantly limited in its inability to support higher-order reasoning. The traditional solution when it comes to TLA is to incorporate a separate math language as part of the logic such as set theory, yet it seems concerning that this is forced on us. Further, even ZF set theory is not finitely axiomatizable in single sorted first order logic.

A higher-order logic is also appealing as a tool for improved modularity. In this work we have considered two classes of techniques for scalable specification engineering. But beyond refinement and composition, there are other approaches to modular development. One which is particularly important is the ability to design specifications which are parameterized by other specifications. For example, we might want to parameterize a specification of a vending machine by a specification of the products it vends or the currency system it accepts. Systems like ML modules put higher-order quantification to great use [39] and it would be desirable to be able to use such higher-order modular techniques with TLA as well.

Beyond modularity of specification, though, digital systems are often inherently higher-order. For example, specifying a function which receives and invokes a function pointer seems to require higher order quantification to abstract over the specification of the argument [9]. In this way, the lack of higher-order features fundamentally limits the problem domains to which TLA is currently applicable. We would like to be able to view an implementation of a specification as a refinement: along these lines recent work ha looked at translating imperative programs into TLA to check them against more abstract specifications [29]. However, these techniques are currently limited in that they can not handle the pervasive higher-order features of programming languages such as function pointers. It is for this reason that order extensions of temporal logic—but not stuttering invariant TLA—have in the past been developed for handling object oriented software [18].

## References

[1] Abadi, M., and Lamport, L. The existence of refinement mappings. *Theor. Comput. Sci. 82*, 2 (May 1991), 253–284.

[2] Abadi, M., and Lamport, L. Conjoining specifications. *ACM Trans. Program. Lang. Syst. 17*, 3 (May 1995), 507–535.

[3] Abadi, M., and Merz, S. On tla as a logic. In *Proceedings of the NATO Advanced Study Institute on Deductive Program Design* (Secaucus, NJ, USA, 1996), Springer-Verlag New York, Inc., pp. 235–271.

[4] Abrial, J. The event-b modelling notation, 2007.

[5] Abrial, J., and Voisin, L. Event-b language, May 2005.

[6] Abrial, J.-R. Event driven sequential program construction. *Approches Formelles dans l'Assistance au Développement des Logiciels* (2001).

[7] Abrial, J.-R. Event model decomposition. `http://wiki.event-b.org/images/Event_Model_Decomposition-1.3.pdf`, 2009.

[8] Abrial, J.-R., and Hallerstede, S. Refinement, decomposition, and instantiation of discrete models: Application to event-b. *Fundam. Inf. 77*, 1-2 (Jan. 2007), 1–28.

[9] Biering, B., Birkedal, L., and Torp-Smith, N. Bi-hyperdoctrines, higher-order separation logic, and abstraction. *ACM Trans. Program. Lang. Syst. 29*, 5 (Aug. 2007).

[10] Bird, R. *Pearls of Functional Algorithm Design*, 1st ed. Cambridge University Press, New York, NY, USA, 2010.

[11] Butler, M. Decomposition structures for event-b. In *Proceedings of the 7th International Conference on Integrated Formal Methods* (Berlin, Heidelberg, 2009), IFM '09, Springer-Verlag, pp. 20–38.

[12] Butler, M. Incremental design of distributed systems with event-b. In *Engineering Methods and Tools for Software Safety and Security - Marktoberdorf Summer School*

*2008*, M. Broy, W. Sitou, and T. Hoare, Eds. IOS Press, 2009, pp. 131–160. Chapter: 4.

[13] Clarke, E. M., and Emerson, E. A. Design and synthesis of synchronization skeletons using branching-time temporal logic. In *Logic of Programs, Workshop* (London, UK, UK, 1982), Springer-Verlag, pp. 52–71.

[14] Cohen, E., and Lamport, L. Reduction in tla. In *CONCUR'98 Concurrency Theory*, D. Sangiorgi and R. de Simone, Eds., vol. 1466 of *Lecture Notes in Computer Science*. Springer Berlin Heidelberg, 1998, pp. 317–331.

[15] Emerson, E. A., and Halpern, J. Y. &ldquo;sometimes&rdquo; and &ldquo;not never&rdquo; revisited: On branching versus linear time temporal logic. *J. ACM 33*, 1 (Jan. 1986), 151–178.

[16] Fiadeiro, J., and Maibaum, T. Temporal theories as modularisation units for concurrent system specification. *Formal Aspects of Computing 4*, 3 (1992), 239–272.

[17] Fiadeiro, J., and Maibaum, T. Categorical semantics of parallel program design. *Science of Computer Programming 28*, 2–3 (1997), 111 – 138. Formal Specifications: Foundations, Methods, Tools and Applications.

[18] Furia, C. A., M, D., Morzenti, A., Pradella, M., Rossi, M., and Pietro, P. S. Pietro: "higher-order trio. Tech. rep., Dipartimento di Elettronica ed Informazione Politecnico di Milano, 2004.

[19] Goguen, J. A. A categorical manifesto. *Mathematical structures in computer science 1*, 01 (1991), 49–67.

[20] Goguen, J. A., and Burstall, R. M. Institutions: Abstract model theory for specification and programming. *J. ACM 39*, 1 (Jan. 1992), 95–146.

[21] Hallerstede, S. Justifications for the event-b modelling notation. In *B 2007: Formal Specification and Development in B*, J. Julliand and O. Kouchnarenko, Eds., vol. 4355 of *Lecture Notes in Computer Science*. Springer Berlin Heidelberg, 2006, pp. 49–63.

[22] Hallerstede, S. On the purpose of event-b proof obligations. *Formal Aspects of Computing 23*, 1 (2009), 133–150.

[23] Harel, D., and Pnueli, A. Logics and models of concurrent systems. Springer-Verlag New York, Inc., New York, NY, USA, 1985, ch. On the Development of Reactive Systems, pp. 477–498.

[24] Kaminski, M., and Yariv, Y. A real-time semantics of temporal logic of actions. *J. Log. Comput. 13*, 6 (2003), 921–937.

[25] Kesten, Y., Manna, Z., and Pnueli, A. Temporal verification of simulation and refinement. In *A Decade of Concurrency, Reflections and Perspectives, REX School/Symposium* (London, UK, UK, 1994), Springer-Verlag, pp. 273–346.

[26] Lamport, L. The temporal logic of actions. *ACM Trans. Program. Lang. Syst. 16*, 3 (May 1994), 872–923.

[27] Lamport, L. Refinement in state-based formalisms. Tech. rep., DEC Systems Research Center, 1996.

[28] Lamport, L. The specification language tla+. In *Logics of Specification Languages*, D. Bjørner and M. Henson, Eds. Springer Berlin Heidelberg, 2008.

[29] Lamport, L. The pluscal algorithm language. In *Proceedings of the 6th International Colloquium on Theoretical Aspects of Computing* (Berlin, Heidelberg, 2009), ICTAC '09, Springer-Verlag, pp. 36–60.

[30] Lamport, L., Roever, D., Langmaack, H., and Pnueli, A. Composition: A way to make proofs harder, 1997.

[31] Merz, S. *Logic-based analysis of reactive systems: hiding, composition and abstraction.* PhD thesis.

[32] Merz, S., and Vanzetto, H. Refinement types for tla +. In *NASA Formal Methods - 6th International Symposium, NFM 2014, Houston, TX, USA, April 29 - May 1, 2014. Proceedings* (2014), pp. 143–157.

[33] Peled, D., and Wilke, T. Stutter-invariant temporal properties are expressible without the next-time operator. *Inf. Process. Lett. 63*, 5 (Sept. 1997), 243–246.

[34] Pnueli, A. The temporal logic of programs. In *Proceedings of the 18th Annual Symposium on Foundations of Computer Science* (Washington, DC, USA, 1977), SFCS '77, IEEE Computer Society, pp. 46–57.

[35] Pnueli, A. System specification and refinement in temporal logic. In *Proceedings of Foundations of Software Technology and Theoretical Computer Science, volume 652 of LNCS* (1995), Springer-Verlag, pp. 1–38.

[36] Poppleton, M. The composition of event-b models. In *Proceedings of the 1st International Conference on Abstract State Machines, B and Z* (Berlin, Heidelberg, 2008), ABZ '08, Springer-Verlag, pp. 209–222.

[37] Prior, A. N. Diodoran modalities. *The Philosophical Quarterly (1950-) 5*, 20 (1955), 205–213.

[38] Ricketts, D., Malecha, G., Alvarez, M. M., Gowda, V., and Lerner, S. Towards Verification of Hybrid Systems in a Foundational Proof Assistant. In *13th ACM-IEEE International Conference on Formal Methods and Models for System Design* (2015).

[39] Rossberg, A., Russo, C. V., and Dreyer, D. F-ing modules. In *Proceedings of the 5th ACM SIGPLAN Workshop on Types in Language Design and Implementation* (New York, NY, USA, 2010), TLDI '10, ACM, pp. 89–102.

[40] Shan, C.-c., and Thurston, D. Word numbers, part 1: Billion approaches. `http://conway.rutgers.edu/~ccshan/wiki/blog/posts/WordNumbers1/`, 2008.

[41] Silva, R. Towards the composition of specifications in event-b. *Electronic Notes in Theoretical Computer Science 280* (2011), 81 – 93. Proceedings of the B 2011 Workshop, a satellite event of the 17th International Symposium on Formal Methods (FM 2011).

[42] Silva, R., and Butler, M. Shared event composition/decomposition in event-b. In *Formal Methods for Components and Objects*, B. Aichernig, F. de Boer, and M. Bonsangue, Eds., vol. 6957 of *Lecture Notes in Computer Science*. Springer Berlin Heidelberg, 2012, pp. 122–141.

[43] Vardi, M. Branching vs. linear time: Final showdown. In *Proceedings of the 2001 Conference on Tools and Algorithms for the Construction and Analysis of Systems, TACAS 2001 (LNCS Volume 2031* (2001), Springer-Verlag, pp. 1–22.

[44] Wirth, N. Program development by stepwise refinement. *Commun. ACM 14*, 4 (Apr. 1971), 221–227.

[45] Yu, Y., Manolios, P., and Lamport, L. Model checking tla+ specifications. In *Correct Hardware Design and Verification Methods* (1999), Springer-Verlag, pp. 54–66.