Linear Temporal Logic and Z Refinement

John Derrick[1] and Graeme Smith[2]

[1] Computing Laboratory, University of Kent, Canterbury, CT2 7NF, UK
[2] School of Information Technology and Electrical Engineering, The University of Queensland 4072, Australia

Abstract. Since Z, being a state-based language, describes a system in terms of its state and potential state changes, it is natural to want to describe properties of a specified system also in terms of its state. One means of doing this is to use Linear Temporal Logic (LTL) in which properties about the state of a system over time can be captured. This, however, raises the question of whether these properties are preserved under refinement. Refinement is observation preserving and the state of a specified system is regarded as internal and, hence, non-observable.
In this paper, we investigate this issue by addressing the following questions. Given that a Z specification A is refined by a Z specification C, and that P is a temporal logic property which holds for A, what temporal logic property Q can we deduce holds for C? Furthermore, under what circumstances does the property Q preserve the intended meaning of the property P? The paper answers these questions for LTL, but the approach could also be applied to other temporal logics over states such as CTL and the μ-calculus.

Keywords: Z, refinement, temporal logic, LTL.

1 Introduction

Z [14], like other state-based languages such as B [1] and VDM [9], describes a system in terms of its state and the changes on this state. A specification typically comprises a state schema declaring and restricting a set of state variables, an initial state schema restricting the initial values of the state variables, and a set of operation schemas detailing possible changes to the state variables with respect to additional variables representing inputs and outputs.

While invariant properties of the system can be captured directly by the specification, more complex behavioural properties need to be proved to hold. Given the emphasis on the state while specifying using Z, it seems natural to want to also describe desired behavioural properties in terms of the system's state. Ideal for this purpose are temporal logics which define predicates over infinite sequences of states. They can be used to specify how the state of the system evolves over time, in a way that isn't possible directly in the model.

Along with their ability to capture behavioural properties in terms of state, temporal logics are also commonly used for describing properties in model checking [4]. Hence, investigating their use with Z is an important first step toward

C. Rattray et al. (Eds.): AMAST 2004, LNCS 3116, pp. 117–131, 2004.

developing model checking support for the language [13]. The most common temporal logics used in model checking are Linear Temporal Logic (LTL) [8], Computation Tree Logic (CTL) [8] and the μ-calculus [10]. In this paper, we focus on the use of LTL, although the approach we develop could also be used to similarly investigate CTL and the μ-calculus.

The purpose of the definition of refinement in Z is to formalise the development from an abstract to a more concrete specification [6]. It is therefore prudent to ask whether temporal logic properties are preserved under refinement. That is, if a property is proved to hold for a Z specification, will it also hold for a refinement of that specification? This question is complicated by the fact that refinement is defined to preserve *observable* properties. In Z, only the operations and their inputs and outputs are regarded as observable; the state of a specification, to which temporal logic properties refer, is regarded as internal and, hence, non-observable.

In this paper, we develop a general approach for investigating preservation of temporal logic properties under refinement, and apply it to LTL. In Section 2, we provide an overview of refinement in Z and motivate our work with an indicative example of where a temporal property is not preserved. Our general approach to determining temporal logic property preservation is defined in Section 3 and applied to LTL in Section 4. We conclude with a discussion of related work and future directions in Section 5.

2 Refinement

Refinement in Z is defined so that the observable behaviour of a Z specification is preserved. This behaviour is in terms of the operations that are performed and their input and output values. Values of the state variables are regarded as being internal. Hence, they are not observable and properties on them are, in general, not preserved. The reason for regarding them as internal is so that refinement can be used to change the representation of the state of a system. This is referred to as *data refinement*.

The definition of refinement in Z is derived from a relational model into which Z is embedded. The details of this are contained in [6], and if a specification C is a refinement of another specification A we write $A \sqsubseteq C$.

To prove a data refinement one needs to link the states in the abstract and concrete specifications via a relation known as the *retrieve relation*. The retrieve relation shows how a state in one specification is represented in the other. For refinement to be complete, a relation, rather than simply a function, is required [6].

Given such a retrieve relation, simulation rules relating the schemas of the specifications are used to verify the refinement. The standard simulation rules for Z assume that operations have preconditions outside of which they can occur changing the state arbitrarily. Such arbitrary state changes make it difficult to prove any interesting temporal properties however. Hence in our approach, we adopt an alternative view of operations in which they are *blocked*, i.e., cannot occur outside their preconditions.

As well as facilitating the use of temporal logics, the blocking model also makes our approach applicable to variants of Z which adopt this model, such as Object-Z [12]. It is also equivalent to the standard non-blocking model of Z when operation preconditions are *totalised*, i.e., post-states are defined for all pre-states. Such totalisation of preconditions would be necessary in standard Z for the specification to exhibit interesting temporal properties.

As for standard Z refinement, there are two simulation rules for refinement under the blocking model which are together complete, i.e., all possible refinements can be proved with a combination of the rules (in fact, every refinement can be verified with one downward together with one upward simulation).

The first rule, referred to as *downward simulation*, requires that

1. the initial states of the concrete specification are related to abstract initial states,
2. the concrete operations are only enabled in states related to abstract states where the corresponding abstract operations are enabled, and vice versa (i.e., they have equivalent preconditions), and
3. whenever a concrete operation can result in the state change (t, t'), for any abstract state s related to t, the corresponding abstract operation can result in (s, s') such that s' is related to t'. That is, the effect of the concrete operation is consistent with the requirements of the corresponding abstract operation.

It is defined as follows [6]. (Note that the state variables of a schema S after each operation are denoted by S', and, in general, the Z notation is also used for expressing its own metatheory, details of this notation can be found in [6].)

Definition 1. *A Z specification with state schema CState, initial state schema CInit and operations* $COp_1 \ldots COp_n$ *is a downward simulation of a Z specification with state schema AState, initial state schema AInit and operations* $AOp_1 \ldots AOp_n$*, if there is a retrieve relation R such that the following hold for all* $i : 1..n$.

1. $\forall\, CState \bullet CInit \Rightarrow \exists\, AState \bullet AInit \wedge R$
2. $\forall\, AState;\ CState \bullet R \Rightarrow (pre\, AOp_i \Leftrightarrow pre\, COp_i)$
3. $\forall\, AState;\ CState;\ CState' \bullet R \wedge COp_i \Rightarrow (\exists\, AState' \bullet R' \wedge AOp_i)$

The second rule, referred to as *upward simulation*, requires that

1. there is an abstract state related to every concrete state,
2. the initial states of the concrete specification are only related to abstract initial states,
3. for every concrete state, there exists an abstract state, such that all operations enabled on the abstract state are also enabled on the concrete state, and
4. whenever a concrete operation can result in the state change (t, t'), for any abstract state s' related to t', the corresponding abstract operation can result in (s, s') where s is related to t.

It is defined as follows [6].

Definition 2. *A Z specification with state schema CState, initial state schema CInit and operations* $COp_1 \ldots COp_n$ *is an upward simulation of a Z specification with state schema AState, initial state schema AInit and operations* $AOp_1 \ldots AOp_n$*, if there is a retrieve relation R such that the following hold.*

1. $\forall\, CState \bullet \exists\, AState \bullet R$
2. $\forall\, AState;\ CState \bullet CInit \wedge R \Rightarrow AInit$
3. $\forall\, CState \bullet \exists\, AState \bullet \forall\, i : 1..n \bullet R \wedge (pre\, AOp_i \Rightarrow pre\, COp_i)$
4. $\forall\, i : 1..n \bullet \forall\, AState';\ CState;\ CState'$
   $\bullet\ (COp_i \wedge R' \Rightarrow (\exists\, AState \bullet R \wedge AOp_i)$

Note that we use a slightly stronger form of upward simulation, in particular, one where the quantification over the operations (that is, i) in condition 3 is after the existential quantification of the abstract state. The reasons for using this form of upward simulation are explored in [2, 7]. In particular, this form ensures that refinement corresponds to failures-divergences refinement in CSP. We need this form here since Lemma 2 does not hold for the weaker form of upward simulation.

Given these definitions, one might assume that properties involving states of the abstract system would hold for the related states in the concrete system. That is, a property referring to abstract state s would hold for concrete state t when t was related to s. This is not the case, as the following example shows.

Consider the following Z specification which, from an initial state $s = 0$, moves nondeterministically to either state $s = 1$ or $s = 2$ via operation $AOp1$ and then to state $s = 3$ or state $s = 4$ depending on its current state via operation $AOp2$.

AState

$s : 0 \ldots 4$

AInit

AState

$s = 0$

*AOp*1

$\Delta AState$

$s = 0 \wedge s' \in \{1, 2\}$

*AOp*2

$\Delta AState$

$s \in \{1, 2\}$
$s = 1 \Rightarrow s' = 3$
$s = 2 \Rightarrow s' = 4$

This is refined by the following specification which from an initial state $t = 0$ moves to a state $t = 1$ via operation $COp1$ and then nondeterministically to state $t = 2$ or $t = 3$ via operation $COp2$.

CState

$t : 0 \ldots 3$

CInit

CState

$t = 0$

COp1

$\Delta CState$

$t = 0 \land t' = 1$

COp2

$\Delta CState$

$t = 1 \land t' \in \{2, 3\}$

This can be proved to be a refinement using an upward simulation with the following retrieve relation.

R

$AState$
$CState$

$s = 0 \Leftrightarrow t = 0$
$s \in \{1, 2\} \Leftrightarrow t = 1$
$s = 3 \Leftrightarrow t = 2$
$s = 4 \Leftrightarrow t = 3$

This can be seen more clearly in Figure 1 which depicts the behaviours of the specifications and the retrieve relation; with only the operations visible, both behaviours are identical.

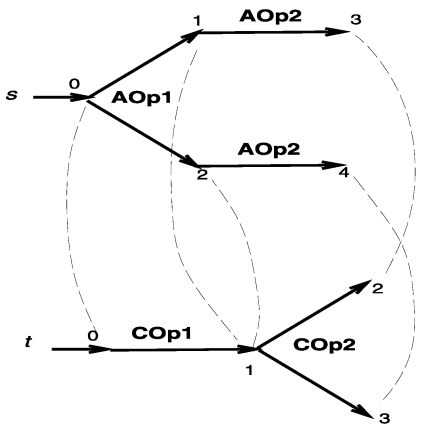

Fig. 1. An example refinement

Some temporal properties that hold for the abstract specification also hold for the related states in the concrete specification. For example, the property "it is always true that when $s = 0$, $s \in \{1, 2\}$ in the next state" holds for the abstract specification, and the corresponding property "it is always true that when $t = 0$, $t = 1$ in the next state" holds in the concrete specification.

Note, however, that while the similar property "it is always true that when $s = 1$, $s = 3$ in the next state" holds for the abstract specification, the corresponding property "it is always true that when $t = 1$, $t = 2$ in the next state"

does not hold for the concrete specification. Hence, this property is not preserved under this refinement.

This motivates the question as to when a temporal property is preserved by refinement. Does it depend on the property, the nature of the refinement or both? In the next section, we describe a general approach for answering this question for any temporal logic over states.

3 Temporal Structures

The temporal logics LTL, CTL and the μ-calculus are usually interpreted on temporal, or Kripke, structures [8]. Given the set of all atomic predicates AP, a temporal structure $(S, S_0, \mathit{Trans}, L)$ comprises

- a set of states S,
- a set of initial states $S_0 \subseteq S$,
- a transition relation $\mathit{Trans} \subseteq S \times S$ where $\forall\, s \in S. \exists\, s' \in S. (s, s') \in \mathit{Trans}$, i.e., *Trans* is total, and
- a labelling function $L : S \to \mathbb{P}\, AP$ mapping each state in S to the atomic propositions which hold in it.

The condition that *Trans* is total is necessary and, together with an assumption that some transition always occurs, ensures that temporal structures continue to progress.

To interpret temporal logic predicates on Z specifications, it is necessary to also adopt this assumption that the specified system progresses, i.e., that its environment always allows some enabled operation to eventually occur. It is also necessary to represent Z specifications as temporal structures.

In order to state temporal properties which refer to inputs and outputs, we need to embed these in the states of the specification without changing its behaviour. This can be done as detailed in [13]. In brief, the type of each input and output when embedded in the state is extended with a value $\perp$ and this value is used for the pre-state value of embedded inputs and post-state value of embedded outputs when they are not declared in a particular schema of the original specification.

The set S of the temporal structure representing a Z specification is the set of bindings of the state schema of the specification. For example, for the abstract specification of Section 2

$$S = \{s \rightsquigarrow 0, s \rightsquigarrow 1, s \rightsquigarrow 2, s \rightsquigarrow 3, s \rightsquigarrow 4\}$$

Similarly, the set S_0 is the set of bindings of the initial state schema. For the same example,

$$S_0 = \{s \rightsquigarrow 0\}$$

The transition relation *Trans* includes mappings $(s, s') : S \times S$ when there exists an operation with binding $s \cup s'$, and mappings $(s, s) : S \times S$ when there

does not exist an operation with binding $s \cup s'$ for any $s' : S$. The latter mappings represent stuttering transitions when no other transitions are available. They are necessary for *Trans* to be total. For the example,

$$Trans = \{(s \rightsquigarrow 0, s \rightsquigarrow 1), (s \rightsquigarrow 0, s \rightsquigarrow 2), (s \rightsquigarrow 1, s \rightsquigarrow 3), (s \rightsquigarrow 2, s \rightsquigarrow 4), \\ (s \rightsquigarrow 3, s \rightsquigarrow 3), (s \rightsquigarrow 4, s \rightsquigarrow 4)\}$$

where the final two pairs are stuttering transitions.

The labelling function L maps each binding s in S to the set comprising those atomic propositions P over the state variables where s is also a binding of the schema $[AState \mid P]$. For example, since $s \rightsquigarrow 0$ is a binding of $[AState \mid s < 10]$, the proposition $s < 10$ is a member of $L(s \rightsquigarrow 0)$.

3.1 Temporal Structures and Refinement

Given the above representation of Z specifications as temporal structures, we can express that a Z specification A meets a temporal logic property P using the standard notation $A \vDash P$. This means that the property P holds from all initial states of the specification.

To relate the temporal properties of Z specifications under a refinement, we first prove a lemma on Z refinement.

Lemma 1. *Given Z specifications, A and C, if $A \sqsubseteq C$ verified via a retrieve relation R then $\forall\, CInit \bullet (\exists\, AInit \bullet R)$.*

Proof. Since downward and upward simulation are complete, C is either a downward or upward simulation of A, or a combination of the two. It suffices to consider each case separately.

In the former case, we have from Definition 1

$$\forall\, CState \bullet CInit \Rightarrow (\exists\, AState \bullet AInit \wedge R)$$

This simplifies to the required condition.

In the latter case, we have from Definition 2

$$\forall\, CState \bullet \exists\, AState \bullet R$$

We also have

$$\forall\, AState;\ CState \bullet CInit \wedge R \Rightarrow AInit$$

Together these properties of upward simulation give us the required condition. □

For all temporal logics over states, the following general theorem holds.

Theorem 1. *Given Z specifications, A and C, and temporal logic property P, if $A \vDash P$ and $A \sqsubseteq C$ under retrieve relation R then $C \vDash \exists\, AState \bullet P \wedge R$.*

Proof. From Lemma 1, we know that for each initial state of C there exists an initial state of A related to it by R. Hence, $C \vDash \exists\, AInit \bullet R$. Also since $A \vDash P$, we know that P is true from all states satisfying $AInit$. Hence, we have $C \vDash \exists\, AInit \bullet P \wedge R$ which implies the required condition. □

This general theorem provides the basis for our investigations. If a temporal logic property holds for a specification A, we want to know whether a *corresponding property* holds for a refinement C. We define a corresponding property to be one constructed from the same logical operators and with each atomic proposition P replaced by the following representation of it in the concrete state:

$$\exists\, AState \bullet P \wedge R$$

where R is the refinement retrieve relation.

Determining whether this is true reduces to determining whether conjunction and existential quantification distribute through the operators in the temporal logic property in such a way that the resulting property is no stronger than the original. For example, suppose $A \vDash P \wedge Q$. From Theorem 1, we know that

$$C \vDash \exists\, AState \bullet (P \wedge Q) \wedge R$$

Since conjunction distributes through conjunction resulting in an equivalent property (i.e., $(P \wedge R) \wedge (Q \wedge R)$ is equivalent to $(P \wedge Q) \wedge R$), we have

$$C \vDash \exists\, AState \bullet (P \wedge R) \wedge (Q \wedge R)$$

Since existential quantification distributes through conjunction resulting in a weaker property (i.e., $(\exists\, x \bullet P) \wedge (\exists\, x \bullet Q)$ is implied by $\exists\, x \bullet P \wedge Q$), we have

$$C \vDash (\exists\, AState \bullet P \wedge R) \wedge (\exists\, AState \bullet Q \wedge R)$$

If P and Q are atomic propositions then the above property is the concrete property corresponding to the abstract property $P \wedge Q$. Hence, the abstract property is preserved by refinement. If P or Q involve additional operators then we need to repeat the above process on the relevant conjunct or conjuncts above. Note that these conjuncts are of the same form as the concrete property derived from Theorem 1 and so the distribution of conjunction and existential quantification is again required.

In the following section, we investigate whether conjunction and existential quantification distribute through the operators of Linear Temporal Logic (LTL). This shows us when LTL properties are preserved by refinement and throws light on the cases when they are not.

4 Linear Temporal Logic

Linear Temporal Logic (LTL) [8] is defined on *paths*, i.e., infinite sequences of states of a temporal structure where each pair of consecutive states is related by

the transition relation of the temporal structure. In this context, $A \vDash P$ means that P holds on all paths originating from initial states of A. Given a path π of A, we also adopt the more specific notations $A, \pi \vDash P$ and $A, \pi_i \vDash P$ for some $i \geqslant 0$. The former means that property P is true on path π and the latter that the property is true on the suffix of path π starting with its ith state, i.e., if $\pi = s_0 s_1 s_2 \ldots s_i s_{i+1} s_{i+2} \ldots$ then $\pi_i = s_i s_{i+1} s_{i+2} \ldots$.

We introduce a further lemma on refinement.

Lemma 2. *Given Z specifications A and C, if $A \sqsubseteq C$ under retrieve relation R then for all paths $\pi^C = t_0 t_1 t_2 \ldots$ of C there exists a path $\pi^A = s_0 s_1 s_2 \ldots$ of A such that each state t_i of π^C is related to the corresponding state s_i of π^A by R.*

Proof. The proof follows by induction.

(i) For state t_0, we know that there exists an abstract state related to it by Lemma 1. Hence, there is a path $\pi^A = s_0 s_1 s_2 \ldots$ of A such that s_0 and t_0 are related by R.

(ii) Assume there exists a path $\pi^A = s_0 s_1 s_2 \ldots$ of A and a $j : \mathbb{N}$ such that for all $i : \mathbb{N}$ where $i \leqslant j$, s_i is related to t_i by R. Since downward and upward simulation are jointly complete, C is either a downward or upward simulation of A, or a combination of the two. It suffices to consider each case separately.

In the former case, we have from Definition 1

$$\forall\, AState;\ CState;\ CState' \bullet R \wedge COp_i \Rightarrow (\exists\, AState' \bullet R' \wedge AOp_i)$$

Hence, for any concrete transition from t_j, there will be a transition from s_j to a state s'_{j+1} such that t_{j+1} and s'_{j+1} are related by R. The definition of a path means there will always be some transition from t_j to t_{j+1}. Hence, there is a path $\pi^{A'} = s_0 s_1 s_2 \ldots s'_{j+1} s'_{j+2} \ldots$ of A such that for all $i : \mathbb{N}$ where $i \leqslant j$, s_i is related to t_i by R and s'_{j+1} is related to t_{j+1} by R.

For upward simulation, we do not require the induction assumption. We simply prove that a path $\pi^{A'} = s'_0 s'_1 s'_2 \ldots$ of A exists such that for all $i : \mathbb{N}$ where $i \leqslant j + 1$, s'_i is related to t_i by R.

We have from Definition 2

$$\forall\, CState \bullet \exists\, AState \bullet R$$

Hence, there will be an abstract state s'_{j+1} related to t_{j+1} by R. Consider a concrete transition from t_j to t_{j+1}. Either this is skip (when no concrete operations are enabled) or the concrete transition is due to a concrete operation COp_i. In the former case the upward simulation applicability condition means no abstract operations are enabled, thus there exists a skip in the abstract state and $t_j = t_{j+1}$, $s_j = s_{j+1}$. In the latter case, we use the following condition from Definition 2

$$\forall\, AState';\ CState;\ CState' \bullet COp_i \wedge R' \Rightarrow (\exists\, AState \bullet R \wedge AOp_i)$$

Hence, for the concrete transition from t_j to t_{j+1} due to COp_i, there will be a transition from a state s'_j of A to s'_{j+1} such that s'_j is related to t_j by R.

Similarly, for the concrete transition from t_{j-1} to t_j, there will be a transition from a state s'_{j-1} of A to s'_j such that s'_{j-1} is related to t_{j-1} by R. Following this line of reasoning, we can deduce that there exists a path $\pi^{A'} = s'_0 s'_1 s'_2 \ldots$ such that for all $i : \mathbb{N}$ such that $i \leqslant j + 1$, s'_i is related to t_i by R. □

We also specialise Theorem 1 of Section 3.1 for LTL as follows.

Theorem 2. *Given Z specifications, A and C, and temporal logic property P, if there exists an $i : \mathbb{N}$ such that for all abstract paths π^A, $A, \pi^A_i \vDash P$ and $A \sqsubseteq C$ under retrieve relation R then for all concrete paths π^C, $C, \pi^C_i \vDash \exists\, AState \bullet P \wedge R$.*

Proof. From Lemma 2, we know that for each concrete path $\pi^C = t_0 t_1 t_2 \ldots$ of C there exists an abstract path $\pi^A = s_0 s_1 s_2 \ldots$ of A such that for all $i : \mathbb{N}$, s_i is related to t_i by R. Hence, $C, \pi^C_i \vDash \exists\, AState \bullet R$ where one instance of $AState$ satisfying the existentially quantified predicate is s_i. Also since $A, \pi^A_i \vDash P$, we know that P is true from state s_i. Hence, we have $C, \pi^C \vDash \exists\, AState \bullet P \wedge R$ as required. □

4.1 Syntax and Semantics

Formulae in LTL are generated from the following rules of syntax.

atomic propositions are formulae[1]
if P and Q are formulae, then $\neg\, P$ and $P \wedge Q$ are formulae
if P and Q are formulae, then $\mathbf{X}\ P$ and $P\ \mathbf{U}\ Q$ are formulae

The operator $\mathbf{X}$ is read as "next" and $\mathbf{U}$ as "until". The following abbreviations are also commonly used:

$$
\begin{array}{lcl}
P \vee Q & \equiv & \neg\, (\neg\, P \wedge \neg\, Q) \\
P \Rightarrow Q & \equiv & \neg\, P \vee Q \\
\mathit{true} & \equiv & P \vee \neg\, P \quad \text{for some } P \\
\mathit{false} & \equiv & P \wedge \neg\, P \quad \text{for some } P \\
\mathbf{F}\ P & \equiv & \mathit{true}\ \mathbf{U}\ P \quad \text{(read "eventually P")} \\
\mathbf{G}\ P & \equiv & \neg\, \mathbf{F}\ \neg\, P \quad \text{(read "always P")}
\end{array}
$$

The semantics of LTL is given in terms of a temporal structure A, path $\pi = s_0 s_1 s_2 \ldots$ of A, and LTL formulae P and Q as follows.

$$
\begin{array}{lll}
A, \pi \vDash P & \text{if and only if} & P \text{ is true in } s_0 \\
A, \pi \vDash \neg\, P & \text{if and only if} & A, \pi \nvDash P \\
A, \pi \vDash P \wedge Q & \text{if and only if} & A, \pi \vDash P \text{ and } A, \pi \vDash Q \\
A, \pi \vDash \mathbf{X}\ P & \text{if and only if} & A, \pi_1 \vDash P \\
A, \pi \vDash P\ \mathbf{U}\ Q & \text{if and only if} & \exists j.(A, \pi_j \vDash Q) \text{ and } \forall k < j.(A, \pi_k \vDash P)
\end{array}
$$

[1] LTL, as used in model checking, is usually restricted to atomic propositions of the form $n = v$. We do not require this restriction for our results; any proposition is allowed.

4.2 Property Preservation

Let A and C be Z specifications such that $A \sqsubseteq C$ under retrieve relation R. To determine which LTL properties of A are preserved under Z refinement, we examine the distribution of conjunction and existential quantification through each LTL operator in turn. That distribution through conjunction works was shown in Section 3.1. Of the other operators (surprisingly) negation, rather than one of the temporal operators, turns out to be the most interesting case and so we leave it until last.

Next (X). If $A \vDash \mathbf{X}\ P$ then from the semantics of the next operator, for all abstract paths π^A, we have

$$A, \pi^A_1 \vDash P$$

Hence, from Theorem 2 we have, for all concrete paths π^C,

$$C, \pi^C_1 \vDash \exists\, AState \bullet P \wedge R$$

Hence, from the semantics of the next operator, we have

$$C \vDash \mathbf{X}\ (\exists\, AState \bullet P \wedge R)$$

Hence, conjunction and existential quantification distribute through the next operator.

Until (U). If $A \vDash P\ \mathbf{U}\ Q$ then from the semantics of the until operator, we know there exists a $j : \mathbb{N}$ such that for all $k : \mathbb{N}$ such that $k < j$, for all abstract paths π^A, we have

$$A, \pi^A_k \vDash P$$

and

$$A, \pi^A_j \vDash Q$$

Hence, from Theorem 2 we have, for all concrete paths π^C,

$$C, \pi^C_k \vDash \exists\, AState \bullet P \wedge R$$

and

$$C, \pi^C_j \vDash \exists\, AState \bullet Q \wedge R$$

Hence, from the definition of the until operator, we have

$$C \vDash (\exists\, AState \bullet P \wedge R)\ \mathbf{U}\ (\exists\, AState \bullet Q \wedge R)$$

That is, conjunction and existential quantification distribute through the until operator.

Since the eventually operator (**F**) is defined in terms of the until operator, conjunction and existential quantification also distribute through it.

Negation ($\neg$)

Negation of a property distributes through conjunction with a retrieve relation. That is, $\neg\ P \wedge R$ implies $\neg\ (P \wedge R)$. However, it does not distribute through existential quantification. That is, $\exists\, x \bullet \neg\ P$ does not imply $\neg\ (\exists\, x \bullet P)$ (since there may be some values of x satisfying $\neg\ P$ and others satisfying P).

So, in general, LTL properties involving negation are not preserved by refinement. Consider, for example, the following Z specification which performs $Op1$ once and then $Op2$ an infinite number of times.

AState

$s : \mathbb{N}$

AInit

AState

$s = 0$

AOp1

$\Delta AState$

$s = 0 \wedge s' = 1$

AOp2

$\Delta AState$

$s \neq 0 \wedge s' = s + 1$

It is refined by the following specification with identical behaviour.

CState

$t : \{0, 1\}$

CInit

$t = 0$

COp1

$\Delta CState$

$t = 0 \wedge t' = 1$

COp2

$\Delta CState$

$t = 1 \wedge t' = 1$

The refinement is a downward simulation under the retrieve relation

R

AState
CState

$s = 0 \Leftrightarrow t = 0$
$s \neq 0 \Leftrightarrow t = 1$

However, while the LTL property $\mathbf{G}\ (s = 1 \Rightarrow \mathbf{X}\ (\neg\ s = 1))$ holds for the abstract specification, the corresponding property $\mathbf{G}\ (t = 1 \Rightarrow \mathbf{X}\ (\neg\ t = 1))$ does not hold for the concrete specification.

This result does not mean that LTL properties involving negation are never preserved by refinement. For example, the property $\mathbf{G}\ (s = 0 \Rightarrow \mathbf{X}\ (\neg\ s = 0))$ holds for the abstract specification above, and the corresponding property $\mathbf{G}\ (t = 0 \Rightarrow \mathbf{X}\ (\neg\ t = 0))$ holds for the concrete specification.

The reason, in this case, is that the retrieve relation R is functional for the state $t = 0$, i.e., there is only one related abstract state. When this is so, $\exists\, AState \bullet \neg\, P \wedge R$ implies that $\neg\, P$ is true from the single instance of $AState$ related to the concrete state. Hence, it is equivalent to $\neg\, (\exists\, AState \bullet P \wedge R)$. That is, negation distributes through existential quantification.

Also, even without the retrieve relation being functional on negated states, some LTL properties involving negation are preserved. For example, disjunction which is defined in terms of negation does distribute through existential quantification as follows.

$$
\begin{array}{ll}
 & \exists\, x \bullet P \vee Q \\
\equiv & \exists\, x \bullet \neg\, (\neg\, P \wedge \neg\, Q) \\
\equiv & \neg\, (\forall\, x \bullet \neg\, P \wedge \neg\, Q) \\
\Rrightarrow & \neg\, ((\forall\, x \bullet \neg\, P) \wedge (\forall\, x \bullet \neg\, Q)) \\
\equiv & \neg\, ((\neg\, (\exists\, x \bullet P)) \wedge (\neg\, (\exists\, x \bullet Q))) \\
\equiv & (\exists\, x \bullet P) \vee (\exists\, x \bullet Q)
\end{array}
$$

In fact, whenever we have an even number of successive negations to distribute, as above, then they will distribute through existential quantification. The proof follows the reasoning of that above. Therefore, as well as the disjunction operator, we have that conjunction and existential quantification distribute through the always operator which is defined by $\mathbf{G}\ P = \neg\, \mathbf{F}\, \neg\, P$.

Of the operators defined as abbreviations, the only one we haven't considered is implication. Since this is defined by $P \Rightarrow Q = \neg\, P \vee Q$, we have an odd number of successive negations preceding predicate P. For this reason, the operator does not distribute through existential quantification, and hence temporal logic properties involving implication are not, in general, preserved by refinement.

This explains the example at the end of Section 2. The first concrete property can be formulated in LTL as

$$\mathbf{G}\ (s = 0 \Rightarrow \mathbf{X}\ (s \in \{1, 2\}))$$

which despite involving implication is preserved by the refinement since the retrieve relation is functional on $t = 0$ (the concrete state related to s_0).

The second property can be formulated in LTL as

$$\mathbf{G}\ ((s = 1 \Rightarrow (\mathbf{X}\ s = 3))$$

In this case, the retrieve relation is not functional on state $t = 1$ (the concrete state related to $s = 1$) and so the property is not preserved.

5 Conclusion

In this paper, we have shown that all temporal properties P over the state $AState$ of an abstract Z specification A are transformed to properties $\exists\, AState \bullet P \wedge R$ over a concrete Z specification C which is a refinement of A under the retrieve relation R.

This result allowed us to reduce the problem of determining when properties of a given temporal logic are preserved by refinement to an investigation of the distribution of the various operators of the temporal logic through conjunction and existential quantification.

We carried out such an investigation for Linear Temporal Logic (LTL) and discovered that while properties containing only conjunction and the temporal operators "next" and "until" (and, hence, also the derived operator "eventually") are preserved, those involving negation are, in general, not preserved.

We also pointed out two important cases when properties involving negation are preserved. The first is when any negated concrete state is related to only one abstract state. The second is when the number of successive negations in any part of the property is even. The first case is important as it shows that all LTL properties are preserved when the refinement retrieve relation is functional. The second is important as it shows that properties involving certain operators derived using negation, namely, disjunction and the temporal operator "always", are preserved.

The preservation of all LTL properties under a functional retrieve relation agrees with the recent result by Darlot et al. [5]. Our work extends this result by considering non-functional retrieve relations and also a complete definition of refinement (Darlot et al. consider only a variant of downward simulation in their approach).

We used a form of upward simulation that is compatible with CSP refinement, and is slightly stronger than the form often used in Z. The extension of our results to the weaker form of upward simulation is left for future work.

Our work is also closely related to that on abstraction, relevant in model checking as a means to reduce the size of the state space of a specification to be checked. Such techniques are in essence the inverse of refinement. Clarke et al. [3] proved that all properties in the universal fragment of CTL* (i.e., the fragment of CTL*, an extension of CTL, which excludes existential quantification over paths) are true of a specification when they are true of an abstraction of that specification. Loiseaux et al. [11] proved a similar result for the universal fragment of the *mu*-calculus. Both of these fragments subsume LTL, but there is no contradiction with our result as only functional abstraction relations were considered.

The abstraction results, as well as being limited to functional relations, also only consider variants of downward simulation. It would be interesting, therefore, to extend our results to CTL and the μ-calculus. Not only would this make our approach of using temporal properties with Z more general, and compatible with more model checking techniques, it would also open the possibility of developing more general abstraction techniques for model checking. This extension of our results could readily be achieved using the approach we have presented in this paper.

Acknowledgement

This work was carried out while Graeme Smith was on a visit to the University of Kent funded by a University of Queensland External Support Enabling Grant.

References

1. J.R. Abrial. *The B Book: Assigning Programs to Meaning*. Cambridge University Press, 1996.
2. C. Bolton and J. Davies. A Singleton Failures Semantics for Communicating Sequential Processes. *Formal Aspects of Computing*, 2002. Under consideration.
3. E. Clarke, O. Grumberg, and D. Long. Model checking and abstraction. *ACM Transactions on Programming Languages and Systems*, 16(5):1512–1542, 1994.
4. E. Clarke, O. Grumberg, and D. Peled. *Model Checking*. MIT Press, 2000.
5. C. Darlot, J. Julliand, and O. Kouchnarenko. Refinement preserves PLTL properties. In D. Bert, J.P. Bowen, S. King, and M. Waldén, editors, *International Conference of Z and B Users (ZB2003)*, volume 2651 of *Lecture Notes in Computer Science*, pages 408–420. Springer Verlag, 2003.
6. J. Derrick and E. Boiten. *Refinement in Z and Object-Z, Foundations and Advanced Applications*. Springer-Verlag, 2001.
7. J. Derrick and E.A. Boiten. Relational concurrent refinement. *Formal Aspects of Computing*, 15(1):182–214, November 2003.
8. E. A. Emerson. Temporal and modal logic. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science*, volume B, pages 996–1072. Elsevier Science Publishers, 1990.
9. C.B. Jones. *Systematic Software Development using VDM*. Prentice Hall, 1986.
10. D. Kozen. Results on the propositional μ-calculus. *Theoretical Computer Science*, 27:333–354, 1983.
11. C. Loiseaux, S. Graf, J. Sifakis, A. Bouajjani, and S. Bensalem. Property preserving abstractions for the verification of concurrent systems. *Formal Methods in System Design*, 6(1), 1995.
12. G. Smith. *The Object-Z Specification Language*. Advances in Formal Methods. Kluwer Academic Publishers, 2000.
13. G. Smith and K. Winter. Proving temporal properties of z specifications using abstraction. In D. Bert, J.P. Bowen, S. King, and M. Waldén, editors, *International Conference of Z and B Users (ZB2003)*, volume 2651 of *Lecture Notes in Computer Science*, pages 408–420. Springer Verlag, 2003.
14. J.M. Spivey. *The Z Notation: A Reference Manual*. Prentice Hall, 2nd edition, 1992.