FormalCheck Query Language Compared with CTL

Zijiang Yang
Rice Univ., Houston, TX
zijiang@cs.rice.edu

Christine Chung
Cornell Univ., Ithaca, NY

In-Ho Moon
Univ. of Colorado, Boulder, CO

January 3, 1999

Abstract

The logic CTL can express branching-time attributes that are not expressible with ω-automata and conversely the FormalCheck Query Language (FQL) whose semantics is founded on ω-automata can express sequentiality and eventuality behaviors that are not expressible with CTL (nor even its extension CTL*). Since both CTL and FQL are commonly used in model-checking, it is of interest to compare nonetheless related constructs common to both. In fact, there are subtle but important discrepancies between common FQL expressions and seemingly but not actually equivalent CTL formulae. In some cases these discrepancies illuminate common misunderstandings about the semantical meaning of the given CTL formula, and thus are of interest to CTL users irrespective of FQL. The root cause of these discrepancies is the inability of CTL to express "phase"– an unbounded succession of disjoint time intervals in which a behavior (*i.e.*, property or constraint) is required to hold. Phase is fundamental to FQL. We show that much of FQL can be expressed by a simple extension CTL$^+$ of CTL formed by conjoining "phase bits" to the atomic propositions. However, another behavior fundamental to FQL is *strong fairness*, which we prove is not expressible even in CTL$^+$. On the other hand, formulae with alternation of path quantifiers fundamental to CTL cannot be expressed in FQL. It is the purpose of this paper to clarify these relationships and distinctions.

1 Introduction

CTL has been studied extensively over the past two decades and is well-known to the model-checking community [3]. FormalCheck^{TM} [1] is a formal verification tool developed by Lucent Technologies' Bell Labs and released commercially in 1997. It uses a particular class [6] of ω-automata as its underlying semantic model. Given a system modeled by the automaton P, and a behavioral attribute that P is intended to possess modeled by the automaton T, model-checking in this context consists of the automaton language containment test: $\mathcal{L}(\mathcal{P}) \subset \mathcal{L}(\mathcal{T})$ [6]. In the context of CTL, the system model is given by a Kripke structure P, the behavioral attribute is specified by a CTL formula T and model-checking consists of the satisfiability test: $P \models T$ [3].

*This work was supported by Lucent Technologies' Bell Labs.

Today, among commercially available tools for model-checking, CTL and FQL are the two dominant paradigms for defining the attributes to be verified. CTL is associated with several commercial tools, whereas FQL is associated with FormalCheck which holds a dominant position in this market. Thus a comparison of CTL and FQL is appropriate and of general interest.

Whereas CTL can express branching-time attributes that are not expressible with ω-automata, FQL is capable of expressing sequentiality and eventuality behaviors that are not expressible with CTL (or even its extension CTL*). Thus, the expressiveness of FQL and CTL are not comparable. Since both these paradigms are commonly used in model-checking, it is of interest to compare nonetheless related constructs common to both CTL and FQL. In fact, there are subtle but important discrepancies between common FQL expressions and seemingly but not actually equivalent CTL formulae. These are easily overlooked. Moreover, the discrepancies are important and interesting. The root cause of these discrepancies is the inability of CTL to express "phase"– an unbounded succession of disjoint time intervals in which a behavior (*i.e.*, property or constraint) is required to hold. Phase is fundamental to FQL. We show that much of FQL can be expressed by a simple extension CTL$^+$ of CTL formed by conjoining "phase bits" to the atomic propositions. However, another behavior fundamental to FQL is *strong fairness*, which we prove is not expressible even in CTL$^+$. On the other hand, formulae with alternation of path quantifiers fundamental to CTL cannot be expressed in FQL.

This paper focuses upon specific CTL formulae with equivalent FQL expressions, as well as CTL formulae and FQL expressions with no respective equivalent formulations. We give a table of specific formulae (in CTL or its extensions) and equivalent FQL expressions. Some common CTL formulae which admit of no equivalent FQL expressions are cited. Conversely, we prove that all FQL expressions which express safety properties admit of equivalent CTL$^+$ formulae, but most do *not* admit of equivalent CTL formulae. In some of these cases there *are* CTL formulae which often have been presumed to be equivalent to the given FQL expressions, but in fact are distinguished by models sensitive to phase. Finally, we prove that FQL expressions which express strong fairness and its derivatives admit of no equivalent formulae even in CTL$^+$.

2 Background and Definitions

In this paper we study equivalences and the non-existence of equivalences between FQL expressions and formulae in two extensions of CTL: CTL^2 which allows two temporal operators within a path formula and CTL$^+$ which adds Boolean *phase bits* to the CTL atomic propositions.

A formula is said to be equivalent to an FQL expression if the two are both true on the same set of Kripke structures (where, in the case of FQL, the Kripke structures are interpreted as a limit-prefix-closed ω-automaton [6], *i.e.*, an automaton all of whose runs are accepting). For a general treatment of the issue of equivalence between liner-time and branching-time structures, see [5].

The syntax and semantics of CTL, its extension CTL* and of the linear-time propositional temporal logic LTL are well-known [3].

The ω-automata associated with FormalCheck comprising FQL are specified by a fixed set of parameterized macros. FQL is defined as arbitrary finite conjunctions of instantiations of various FQL macros. (Conjunction is represented by concatenation and incurs no computational overhead; disjunction, negation and nesting are not allowed.) Each macro has the same general format, embracing three qualified conditions: an optional qualified *enabling condition*, a qualified *fulfilling condition*, and an optional qualified *discharging condition*. The following shows the generic macro format [with optional components enclosed in square brackets]:

[⟨`Enabling Qualifier`⟩(*Enabling Condition*)]
⟨`Fulfilling Qualifier`⟩(*Fulfilling Condition*)
[⟨`Discharging Qualifier`⟩(*Discharging Condition*)]

The qualifier for the optional *enabling condition* is one of:

After, **IfRepeatedly**, **IfEventuallyAlways**;

the qualifier for the *fulfilling condition* is one of:

Always, **Never**, **Repeatedly**, **Eventually**, **EventuallyAlways**;

and the qualifier for the optional *discharging condition* is one of:

Unless, **UnlessAfter**, **Until**, **UntilAfter**.

For example, a common FQL expression is
After(*Enabling Condition*)
Always(*Fulfilling Condition*)
Unless(*Discharging Condition*).

The semantics of the respective FQL macros is based on a "*phase bit*" (in the underlying automaton) whose state is *ENABLED* after each time the (optional) *enabling condition* becomes true and *DISCHARGED* generally after each time that the (optional) *discharging condition* becomes true (see below for details). Optionally, the *fulfilling condition* is required at the moment when the *discharging condition* becomes true – if the qualifier **UnlessAfter** or **UntilAfter** is used.

The macros which involve **Repeatedly**, **Eventually** and **Until** in their qualifiers express liveness/ eventuality behaviors, while the remaining ones express safety behaviors. **Until** denotes *strong until* while **Unless** denotes *weak until*.

The semantics of FQL is given in [1] and the full set of macros along with their semantics is available upon request from `k@research.bell-labs.com`. Alternatively, every FQL expression can be represented by a formula in LTL^+, the linear-time propositional temporal logic LTL extended by adding phase bits to its set of atomic propositions. The equivalence of LTL^+ and FQL was proved in [2]. In order to make this paper self-contained, we give the LTL^+ formula for each FQL expression, in Tables 1 and 2. Likewise, CTL^+ is an extension of CTL derived by adding an arbitrary finite number of phase bits to its set of atomic propositions. The phase bit and its ramifications are described below.

CTL^2 is a sublogic of CTL^* which extends CTL by allowing two temporal operators within a path formula [9]. We will use it to relate FQL to CTL. The temporal operators of CTL^2 may be either nested or connected by a binary Boolean connective. Negating one or both of the temporal operators is also allowed. For example, $\mathbf{AGF}grant$ and $\mathbf{E}(\mathbf{X}request)\mathbf{U}grant$ are CTL^2 formulae.

A *phase bit* is a Boolean "history" variable p whose value at a given state of a Kripke structure is determined by the path taken from the initial state to the given state, specifically by the sequence of truth values of the atomic propositions along that path. This is a natural concept in the context of automata: p corresponds to a 2-state state machine, with states *ENABLED* and *DISCHARGED* and associated predicates expressed in terms of the atomic propositions for the respective state transitions.

Throughout this paper, for each FQL macro, we denote by e its *enabling condition*, by f its *fulfilling condition* and by d its *discharging condition*. Thus, each FQL macro has as its parameters f and optionally

e and d. For simplicity, these will be suppressed in the given FQL expressions; which are associated with a particular macro is determined by the macro according to its format as described above. Boolean conjunction of conditions (corresponding to respective atomic propositions) will be denoted by $*$.

The phase bit associated with a given FQL macro has e as its "enabling condition" enabling the transition from its state *DISCHARGED* to *ENABLED*, and some function of f and d as its "discharging condition" enabling the transition from *ENABLED* to *DISCHARGED*. For the purposes of this paper, the precise phase bit discharging conditions do not matter. (In fact, they are one of: f, d, $f * d$, $\neg f * d$, and which may be readily inferred from the context.) For brevity, for a phase bit p, we denote its *DISCHARGED* state by p_0 and its *ENABLED* state by $\neg p_0$. Thus, the state transition $p_0 \rightarrow \neg p_0$ is enabled by e, while the reverse transition $\neg p_0 \rightarrow p_0$ is enabled by some function of f and d. The self-loops $p_0 \rightarrow p_0$ and $\neg p_0 \rightarrow \neg p_0$ are enabled by the negation of the respective non-self-loop conditions, so the phase bit is a deterministic function of e, f and d.

Phase bits may be added to LTL and CTL by including their respective states with the atomic propositions. The addition of phase bits to CTL allows CTL to capture a sense of history not otherwise possible. For example, the CTL$^+$ formula $\mathbf{AG}(e * p_0 \rightarrow \mathbf{AXAF}f)$ which is equivalent to the FQL expression $\mathbf{After}(e)\mathbf{Eventually}(f)$, gives higher precedence to f than e when the phase bit is in the *ENABLED* state (*i.e.* when $\neg p_0$ is true). Thus, once e is true, p_0 becomes false and if subsequently $e * f$ becomes true, the formula does not require a subsequent instance of f as it would without the conditioning upon p_0. (The discharging condition for p_0 is f.) This is important in common usage where e is a guard which is set by the condition e and remains set until it is cleared by the condition f. Thus, at the instant it is cleared, $e * f$ is true, but a subsequent f is not required. Similarly, e has precedence over f when the phase bit is in its *DISCHARGED* state. With this precedence the formula enforces causality: the f must *follow* the e. Again, causality is commonly required in mechanistic settings such as hardware verification: an occurrence of f "lingering" from previous actions should not satisfy the formula when it is remains true with a subsequent e (unless it survives the onset of e). By contrast, the CTL formula $\mathbf{AG}(e \rightarrow \mathbf{AF}f)$ stipulates that if f and e are simultaneously true at the outset, then f need not be true again.

CTL$^+$ is strictly more expressible than CTL: as is well-known,

$$\text{Odd(q)} = \mathbf{E}\text{"q holds at positions 1,3,5,..."}$$

is not expressible in CTL [3]. However, the equation can be expressed as the following CTL$^+$ formula:

$$(\exists p_0)(p_0 * \mathbf{AG}(p_0 \rightarrow \mathbf{AX}\neg p_0) * \mathbf{AG}(\neg p_0 \rightarrow \mathbf{AX}p_0) * \mathbf{EG}(p_0 \rightarrow q))$$

Although the addition of phase bits to CTL increases its expressiveness, we will show that *strong fairness*, expressible in FQL, cannot be expressed in CTL$^+$.

3 CTL Formulae with Equivalent FQL Expressions

Many commonly used CTL formulae without existential path quantification can be expressed in FQL. In some cases, the corresponding FQL expression is awkward. However, the awkwardness signals a source of potential confusion: in each such case, there is a simple FQL expression which is "almost" equivalent to the CTL formula, and frequently is presumed incorrectly to be exactly equivalent to the CTL formula. In these cases, it is the semantics of the FQL expression that is usually intended. This contrast thus provides a warning to CTL users, whether or not they have an interest in FQL.

1. $\mathbf{AG}(f) \equiv \mathbf{Always}(f)$, $\mathbf{AG}(\neg f) \equiv \mathbf{Never}(f)$

2. $\mathbf{AG}(e \rightarrow \mathbf{AXAF}(f))$
   This CTL formula is equivalent to the conjunction of the following two FQL expressions:
   $P1:$ $\mathbf{After}(e * (P2.p_0 + f * (\neg P2.p_0)))\mathbf{Eventually}(f)$
   $P2:$ $\mathbf{After}(e * f * (\neg P1.p_0)\mathbf{Eventually}(f)$
   where $P1$, $P1$ designate the respective expressions and $P1.p_0$, $P2.p_0$ denote their respective phase bits. The FQL formulation seems unsettlingly complex. In fact, it underscores the fact that the CTL formula requires a subsequent instance of the *fulfilling condition* f if the *enabling condition* e remains true at the time that f becomes true. Typically, this is *not* the semantics that is wanted. Rather, what is typically wanted for this behavior is the FQL expression and equivalent CTL$^+$ formula $\mathbf{After}(e)\mathbf{Eventually}(f) \equiv \mathbf{AG}(e * p_0 \rightarrow \mathbf{AXAF}(f))$. It can be proved that this behavior cannot be expressed in CTL.

3. $\mathbf{AG}(e \rightarrow \mathbf{AF}(f)) \equiv \mathbf{After}(e * (\neg f))\mathbf{Eventually}(f)$
   Note that this requires no causality between e and f, if f is true when e becomes true. If causality is required, then an instance of f must be true *after* e becomes true, and as in item 2, $\mathbf{After}(e)\mathbf{Eventually}(f)$ is required in lieu of the stated CTL formula. The inequivalence of the CTL formulae of items 2 and 3 with $\mathbf{After}(e)\mathbf{Eventually}(f)$ has been a source of considerable confusion among practitioners of model-checking.

4. $\mathbf{AG}(e \rightarrow \mathbf{AX}(f))$
   Analogous to item 2., this CTL formula is equivalent to the conjunction of the two FQL expressions:
   $P1:$ $\mathbf{After}(e * (P2.p_0 + f * (\neg P2.p_0)))\mathbf{Always}(f)\mathbf{UnlessAfter}(true)$
   $P2:$ $\mathbf{After}(e * f * (\neg P1.p_0)\mathbf{Always}(f)\mathbf{UnlessAfter}(true)$
   where $P1.p_0$, $P2.p_0$ denote the respective phase bits as before. If e is not possible in two successive states, the two FQL expressions may be replaced with simply $\mathbf{After}(e)\mathbf{Always}(f)\mathbf{UnlessAfter}(true)$. However, none of these is typically what is wanted in a clocked system. If a system is clocked, it is not the next event that is of interest, but rather the next *clocked* event (*i.e.*, the next event concurrent with an active clock edge or level). Expressing "next clock" is very awkward with CTL. In FQL, if *clock* denotes the predicate which defines the active clock edge or level, the expression $\mathbf{After}(e)\mathbf{Never}(clock)\mathbf{Unless}(f)$ expresses the behavior that f occurs concurrently with the next clock event following e.

5. $\mathbf{AF}(f) \equiv \mathbf{Eventually}(f)$.
   This may be used when f denotes a one-time occurrence such as initialization. More commonly, one needs to express recurring behavior triggered by a given event e: $\mathbf{After}(e)\mathbf{Eventually}(f)$.

6. $\mathbf{A}(f\mathbf{U}d) \equiv \mathbf{Always}(f)\mathbf{Until}(\mathbf{d})$
   However, as in item 5, one more commonly wants to express the recurring behavior
   $\mathbf{After}(e)\mathbf{Always}(f)\mathbf{Until}(d)$, which can be expressed in CTL$^+$ but not CTL.

4 CTL Formulae With No Equivalent FQL

FQL has no direct way to express existential path quantification. However, when there is only one existential path quantifier and no others, there is an indirect way to check the CTL using a FQL-based model-checker.

1. **EF**(f)
   FQL cannot directly express any CTL formula with existential path quantification **E**. However, if there are no other path quantifiers, the negation of the property *can* be expressed in FQL: $\neg(\mathbf{EF}(f)) = \mathbf{AG}(\neg f) \equiv \mathbf{Never}(f)$. If **Never**($f$) is false, then **EF**($f$) is true, and conversely.

2. **AFAG**(f)
   This CTL formula has no FQL counterpart. However, it also is commonly confused with the LTL formula **FG**(f) and the equivalent CTL* formula $\mathbf{AFG}(f) \equiv \mathbf{EventuallyAlways}(f)$, to which **AFAG**($f$) is *not* equivalent.

3. **AGEF**(f).
   This very important CTL formula is the "famous" *reset* formula, which expresses the behavior that from every reachable state there exists a path to a state where f is true (commonly, f denotes "reset"). It has no counterpart in FQL. (This is not to be confused with the fact that FormalCheck has a built-in algorithm which in effect checks **AGEF**(*reset*) as a part of a broader check that the design model is not over-constrained.)

5 FQL With Equivalent CTL$^+$ Formulae

It can be proved that *no* FQL expression with both enabling and discharging conditions admits of an equivalent CTL or LTL formula; on the other hand, every FQL expression does have an equivalent LTL$^+$ formula [2]. Table 1 gives a representative listing of those FQL expressions which admit of equivalent LTL formulae. Table 2 is a representative listing of those FQL macros which admit of no equivalent LTL formulae, and their equivalent LTL$^+$ formulae. We will determine which of those FQL macros from Tables 1 and 2 admit of equivalent CTL or CTL$^+$ Formulae.

It is not always easy to determine which of these can be expressed in CTL or CTL$^+$. Because LTL and CTL formulae are interpreted over different structures, we cannot compare them directly. A straightforward approach is as follows: given a transition system M and an LTL formula ψ to be checked with respect to M, try to translate the CTL* formula $\mathbf{A}\psi$ to an equivalent CTL formula φ, and then check M with respect to φ. But the problem of how to decide whether $\mathbf{A}\psi$ has an equivalent CTL formula φ is open. Kupferman and Vardi [8] studied a more modest approach: instead of looking for some equivalent CTL formula to $\mathbf{A}\psi$, they restricted themselves to the specific candidate ψ_A, which precedes each temporal operator in ψ by the path quantifier **A**. Although $\mathbf{A}\psi$ may have an equivalent CTL formula and still not be equivalent to ψ_A, their method provides a practical means of translation which we will use for some cases.

Translation for some LTL formulae are obvious. For each of the LTL formulae: $\mathbf{G}f$, $\mathbf{G}(\neg f)$, $\mathbf{F}f$, $f\mathbf{U}d$, and $f\mathbf{U}(f * d)$ ψ, $\mathbf{A}\psi$ is already a CTL formula, and thus the corresponding FQL macro (Table 1) has an equivalent CTL formula.

Kupferman and Grunberg [9] showed that the CTL^2 formula $\mathbf{EG}(\phi_1\mathbf{U}\phi_2)$ embodies all the expressive superiority of CTL^2 over CTL. They provided rules for translating to equivalent CTL formulae all CTL^2 formulae which do not contain $\mathbf{EG}(\phi_1\mathbf{U}\phi_2)$. Table 3 gives those LTL formulae form Table 1 whose corresponding CTL* formulae $\mathbf{A}\psi$'s belong to CTL^2, and thus the above result applies, giving rise to the listed CTL formulae.

6 FQL With No equivalent CTL$^+$ formulae

The deepest part of the taxonomy is proving that certain FQL expressions admit of no equivalent CTL$^+$ formulae. In this section we show *strong fairness* is not expressible in CTL$^+$ (Table 6). We also list those FQL macros for which it is presumed (but not proved) that they are not expressible in CTL$^+$ (Table 5); for these, we do know that they are not expressible in CTL. This extended abstract does not allow space for the proofs which are available from the authors.

We use a theorem of Clark and Draghicescu [4] to conclude that the CTL* formula **AFG**f admits of no equivalent CTL formula. Thus, neither does $\mathbf{AG}(\mathbf{GF}e \rightarrow \mathbf{FG}f)$. Indeed, suppose f is always false; then

$$\mathbf{AG}(\mathbf{GF}e \rightarrow \mathbf{FG}f) = \mathbf{AG}(\mathbf{GF}e \rightarrow False) = \mathbf{AGFG}(\neg e) = \mathbf{A}(\mathbf{FG}(\neg e)),$$

proving that the FQL macro **IfRepeatedly_EventuallyAlways** admits of no equivalent CTL formula. Similarly, one can prove that the other FQL expressions in Table 5 admit of no equivalent CTL formulae.

Finally, Immerman [7] has proved that *strong fairness*

$$\mathbf{IfRepeatedly\_Repeatedly\_} \equiv \mathbf{G}(\mathbf{GF}e \rightarrow \mathbf{GF}f)$$

is not expressible in CTL$^+$, using Ehrenfeucht-Fraissé games. It follows that

$$\mathbf{IfRepeatedly\_Eventually\_} \equiv \mathbf{IfRepeatedly\_Repeatedly\_}$$

also is not expressible: see Table 6.

While

$$\mathbf{After\_EventuallyAlways\_Unless\_} \equiv \mathbf{G}(e * p_0 \rightarrow \mathbf{X}(\mathbf{FG}f + \mathbf{F}d))$$

is based upon strong fairness, and is presumed not expressible in CTL$^+$, its expressibility in CTL$^+$ is not known.

Acknowledgements: We thank Neil Immerman for his game-theoretic proof that strong fairness cannot be expressed in CTL$^+$, Fabio Somenzi and Thomas Wilke for their guidance in translating LTL formulae to equivalent CTL formulae and Bob Kurshan for suggesting the framework of this paper and general guidance.

References

[1] Bell Labs Design Automation. Formalcheck user's guide. Lucent Technologies, May 1998.

[2] A. Dershowitz, K. Fisler, S. K. Shukla, G. Holzmann, R. P. Kurshan, and D. Peled. Testing the FormalCheck Query Library. In *Proc. LCET'96*, volume 14, pages 173–176. Lucent Technologies, 1997.

[3] E.A.Emerson. Temporal and modal logic. *Handbook of Theoretical Computer Science*, 1991.

[4] E.M.Clarke and I.A.Draghicescu. Expressibility results for linear-time and branching-time logics. *Springer Lecture Notes in Computer Science*, 354:428–438, 1988.

[5] O. Grumberg and R. P. Kurshan. How linear can branching-time be? In *Proc. Int'l. Conf. on Temporal Logic,* (ICTL'94), Bonn, Germany, 1994.

[6] Robert P. Kurshan. *Computer-Aided Verification of Coordinating Processes.* Princeton University Press, 1994.

[7] N.Immerman. Strong fairness not expressible in ctl$^+$. *To be published.*

[8] O.Kupferman and M.Y.Vardi. Relating linear and branching model checking. *IFIP*, 1996.

[9] O.Kupferman and O.Grumberg. Buy one, get one free!!! *Journal of Logic and Computation*, 6(4):523–539, 1996.

FQL expressions	LTL formulae
Always_	$\mathbf{G}f$
Never_	$\mathbf{G}(\neg f)$
After_Always_	$\mathbf{G}(e \rightarrow \mathbf{X}\mathbf{G}f)$
Always_Unless_	$\mathbf{G}f + f\mathbf{U}d$
Always_Until_	$f\mathbf{U}d$
Always_UnlessAfter_	$\mathbf{G}f + f\mathbf{U}(f * d)$
Always_UntilAfter_	$f\mathbf{U}(f * d)$
Eventually_	$\mathbf{F}f$
Eventually_Unless_	$\mathbf{F}f + \mathbf{F}d$
IfRepeatedly_Repeatedly_	$\mathbf{G}(\mathbf{GF}e \rightarrow \mathbf{GF}f)$
IfRepeatedly_Eventually_	$\mathbf{G}(\mathbf{GF}e \rightarrow \mathbf{F}f)$
Repeatedly_	$\mathbf{GF}f$
Repeatedly_Unless_	$\mathbf{GF}f + \mathbf{F}d$
EventuallyAlways_	$\mathbf{FG}(f)$
EventuallyAlways_Unless_	$\mathbf{F}(\mathbf{G}f + d)$
After_EventuallyAlways_	$\mathbf{G}(e \rightarrow \mathbf{FG}f)$
IfEventuallyAlways_Eventually_	$\mathbf{G}(\mathbf{FG}e \rightarrow \mathbf{GF}f)$
IfRepeatedly_EventuallyAlways_	$\mathbf{G}(\mathbf{GF}e \rightarrow \mathbf{FG}f)$
IfEventuallyAlways_EventuallyAlways_	$\mathbf{G}(\mathbf{FG}e \rightarrow \mathbf{FG}f)$

Table 1: FQL expressions admitting of equivalent LTL Formulae

FQL expressions	LTL^+ formulae
After_Always_Unless_	$\mathbf{G}(e * p_0 \rightarrow \mathbf{X}(\mathbf{G}f + f\mathbf{U}d))$
After_Always_Until_	$\mathbf{G}(e * p_0 \rightarrow \mathbf{X}(f\mathbf{U}d))$
After_Eventually_	$\mathbf{G}(e * p_0 \rightarrow \mathbf{XF}f)$
After_Repeatedly_Unless_	$\mathbf{G}(e * p_0 \rightarrow \mathbf{X}(\mathbf{GF}f + \mathbf{F}d)$
After_Eventually_Unless_	$\mathbf{G}(e * p_0 \rightarrow \mathbf{XF}(f + d))$
After_EventuallyAlways_Unless_	$\mathbf{G}(e * p_0 \rightarrow \mathbf{X}(\mathbf{FG}f + \mathbf{F}d))$

Table 2: FQL expressions with equivalent LTL^+ Formulae

LTL formulae	CTL formulae
$\mathbf{F}f + \mathbf{F}d$	$\mathbf{AF}(f + d)$
$\mathbf{G}f + f\mathbf{U}d$	$\neg\mathbf{E}(f * \neg d)\mathbf{U}(\neg f * \neg d)$
$\mathbf{G}f + f\mathbf{U}(f * d)$	$\neg\mathbf{E}(f * \neg d)\mathbf{U}\neg f$

Table 3: FQL LTL formulae with CTL^2 and hence CTL counterparts.

FQL LTL$^+$ formulae	CTL$^+$ formulae
$\mathbf{GF}f$	$\mathbf{AGAF}f$
$\mathbf{GF}f + \mathbf{F}d$	$\neg\mathbf{E}(\neg d\mathbf{UEG}(\neg f * \neg d))$
$\mathbf{G}(\mathbf{FG}e \rightarrow \mathbf{GF}f)$	$\neg\mathbf{EFEG}(e * \neg f)$
$\mathbf{G}(e \rightarrow \mathbf{XG}f)$	$\mathbf{AG}(e \rightarrow \mathbf{AXAG}f)$
$\mathbf{G}(e * p_0 \rightarrow \mathbf{X}(\mathbf{G}f + f\mathbf{U}d))$	$\mathbf{AG}(e * p_0 \rightarrow \mathbf{AX}(\neg\mathbf{E}(f * \neg d)\mathbf{U}(\neg f * \neg d)))$
$\mathbf{G}(e * p_0 \rightarrow \mathbf{X}(f\mathbf{U}d))$	$\mathbf{AG}(e * p_0 \rightarrow \mathbf{AXA}(f\mathbf{U}d))$
$\mathbf{G}(e * p_0 \rightarrow \mathbf{XF}f)$	$\mathbf{AG}(e * p_0 \rightarrow \mathbf{AXAF}f)$
$\mathbf{G}(e * p_0 \rightarrow \mathbf{XF}(f + d))$	$\mathbf{AG}(e * p_0 \rightarrow \mathbf{AXAF}(f + d))$
$\mathbf{G}(e * p_0 \rightarrow \mathbf{X}(\mathbf{GF}f + \mathbf{F}d))$	$\mathbf{AG}(e * p_0 \rightarrow \mathbf{AX}(\neg\mathbf{E}(\neg d\mathbf{UEG}(\neg f * \neg d))))$

Table 4: FQL LTL$^+$ formulae with CTL$^+$ counterparts.

FQL expressions	LTL formulae
EventuallyAlways_	$\mathbf{FG}(f)$
IfRepeatedly_EventuallyAlways_	$\mathbf{G}(\mathbf{GF}e \rightarrow \mathbf{FG}f)$
IfEventuallyAlways_EventuallyAlways_	$\mathbf{G}(\mathbf{FG}e \rightarrow \mathbf{FG}f)$
After_EventuallyAlways_	$\mathbf{G}(e \rightarrow \mathbf{FG}f)$
EventuallyAlways_Unless_	$\mathbf{F}(\mathbf{G}f + d)$
After_EventuallyAlways_Unless_	$\mathbf{G}(e * p_0 \rightarrow \mathbf{X}(\mathbf{FG}f + \mathbf{F}d))$

Table 5: FQL expressions with no equivalent CTL formulae

FQL expressions	LTL$^+$ formulae
IfRepeatedly_Repeatedly_	$\mathbf{G}(\mathbf{GF}e \rightarrow \mathbf{GF}f)$
IfRepeatedly_Eventually_	$\mathbf{G}(\mathbf{GF}e \rightarrow \mathbf{F}f)$

Table 6: *strong fairness* not expressible in CTL$^+$.