Extending Computational Tree Logic with Relations and Undefinedness

Joey W. Coleman

TECHNICAL REPORT SERIES

No. CS-TR-1230 December 2010
Extending Computational Tree Logic with Relations and Undefinedness

J W Coleman

Abstract

This paper describes three extensions to computational tree logic: one which uses relational atomic propositions; one which can handle undefined terms; and, last, the direct combination of the first two extensions.
Bibliographical details

COLEMAN, J. W.

Extending Computational Tree Logic with Relations and Undefinedness
[By] J. W. Coleman
(University of Newcastle upon Tyne, Computing Science, Technical Report Series, No. CS-TR-1230)

Added entries

UNIVERSITY OF NEWCASTLE UPON TYNE

Abstract

This paper describes three extensions to computational tree logic: one which uses relational atomic propositions; one which can handle undefined terms; and, last, the direct combination of the first two extensions.

About the author

Joey earned a BSc (2001) in Applied Computer Science at Ryerson University in Toronto, Ontario. With that in hand he stayed on as a systems analyst in Ryerson’s network services group. Following that he took a position at a post-dot.com startup as a software engineer and systems administrator. The technical problems encountered working on a large multi-threaded application sent him back to academia in search of better ways of dealing with concurrency. At Newcastle University he has since earned a MPhil (2005) and a PhD (2008) for work on semantics and formal methods. During his time at Newcastle he has been involved with several projects including the FP7 RODIN project, working on methodology, and was associated with the EPSRC DIRC project. He is currently involved with the EPSRC "Splitting (Software) Atoms Safely" project, working on atomicity in software development methods. His interests cover a broad range of topics in computer science, though the focus has been primarily on programming language semantics and the use of formal methods to model concurrent systems. Recent work has involved rely/guarantee reasoning and structural operational semantics.

Suggested keywords

COMPUTATIONAL TREE LOGIC
TEMPORAL LOGIC
3-VALUED LOGIC
Extending Computational Tree Logic with Relations and Undefinedness

Joey W. Coleman
School of Computing Science
Newcastle University
NE1 7RU, UK
e-mail: j.w.coleman@ncl.ac.uk
20 December 2010

Abstract

This paper describes three extensions to computational tree logic: one which uses relational atomic propositions; one which can handle undefined terms; and, last, the direct combination of the first two extensions.

1 Introduction

This paper summarises the author’s explorations in using temporal logics, and should be taken in context with Technical Report CS-TR-1229 [Col10]. In particular, we give three extensions to computational tree logic (CTL*) [CES86, Eme90] that resulted from the author’s use of temporal logic as a basis on which to build rely/guarantee frameworks.1 We use a branching-time logic in particular as the structure of its models match the semantic structure of the languages we wish to reason about using a rely/guarantee framework. Ultimately we recognise that there is some debate regarding the suitability of the various forms of temporal logics [EH86] but take the position that the explicit structural similarities are useful. This paper will not go into detail about the design of rely/guarantee reasoning frameworks, though it will give justifications for some decisions as coming from them.

The starting point of this paper is CTL* and we assume that the reader is familiar with it as it is a well-understood branching-time temporal logic. We have omitted the binary until (U) operator, for reasons detailed in Section 3 relating to undefinedness. So far as we are aware, our extension using relational atomic predicates is unique.

1One such attempt is detailed in the mentioned technical report [Col10]; for rely/guarantee reasoning in general, see [Jon83].
The extension of CTL* to deal with undefinedness, however, has also been explored by Akama et al [ANY08] and Bruns & Godefroid [BG00], among others. We do not claim any new insights into this as such, but we do assert that our formulation is more to the spirit of Jones and Middelburg’s formulation of the logic of partial functions (LPF) in [JM94]. It is our aim that the formulation of CTL_{⊥}^* in Section 3 and Relational CTL_{⊥}^* in Section 4 handle undefinedness in a manner familiar to a user of LPF.

This paper is structured as follows: Section 2 presents an extension of CTL* that uses relations as atomic propositions. Section 3 presents an extension of CTL* which handles undefinedness. Section 4 presents the direct combination of the logics in the prior two sections. And, finally, Section 5 presents some conclusions and directions for future work.

2 Relational Propositions

This section describes Relational CTL*, which is based on CTL* as formulated by Emerson, though we have omitted the until (U) binary operator; the reasons for this are described in Section 3. It should be possible to include the until operator in a version of Relational CTL*, however, its use is not necessary in this paper.

The primary motivation behind this variant of CTL* is the need, in Jones-style rely/guarantee reasoning, to directly refer to properties over pairs of states; that is, the need to directly handle relations over the system state at different points in time. The rely, guarantee, and post conditions in a Jones-style rely/guarantee system are all relational and the overall reasoning system benefits from this. Temporal logics generally are formulated on the use of a set of atomic propositions that are predicates over single states. This presents some difficulties when encoding a rely/guarantee system into a temporal logic, and it is our position that the difficulty is best avoided. To this end we have generated this variant of CTL* using a set of atomic propositions which contains relations.

In the following syntax and semantics, P and Q and their decorated variants, refer to specific relational atomic propositions; a and b and their decorated variants refer to arbitrary Relational CTL* formulae; M refers to a model; s and its decorated variants refer to specific states in a model; and x and its decorated variants refers to a temporal path. Note that superscripted variations on paths, i.e. \( x^i \), refers to the suffix of path x starting at the \( i+1 \)st state; thus, \( x^0 = x \) and \( x^1 \) is the suffix of x starting from the second state.

Two function-like notations are used in the semantic definitions which follow. First, \( x \in \text{paths}(s) \) indicates that a specific sequence of states, \( x \), is a possible path starting at \( s \) in the model; the model is taken from the context of the use. Second, \( \text{first}(x) \) is a reference to the initial state in the given sequence, \( x \). Thus, it is true that \( \forall s \cdot \forall x \in \text{paths}(s) \cdot \text{first}(x) = s \).

The syntax of Relational CTL* is given in figure 1. Formulae are comprised
Each atomic proposition $P$ is a state formula.

If $a$, $b$ are state formulae then so are $a \land b$, $\neg a$

If $a$ is a path formula then $Ea$, $Aa$ are state formulae

If $a$ is a state formula then so is $S\!a$

Every state formula is a path formula

If $a$, $b$ are path formulae then so are $a \land b$, $\neg a$

If $a$ is a path formula then so are $Xa$, $Fa$, $Ga$

If $a$ is a path formula then so is $S\!a$

A model, $M$, in Relational CTL* is defined in a similar manner as in CTL*. A model is a tuple, $(S, R, L)$: $S$ is the set of states; $R$ is the accessibility relation between states; and $L$ is the interpretation, mapping a pair of states to the set of atomic propositions which hold for that pair.

An assertion using a state formula in Relational CTL* is written

$$M, s_h, s_0 \models a$$

where $M$ is the ground model, both $s_h$ and $s_0$ are states, and $a$ is the formula that we are interested in. The second state, $s_0$, performs the same function as the single state in a regular CTL* formula; it acts as the reference from which $p$ is interpreted and from which paths are rooted by the $A$ and $E$ quantifiers. The first state, $s_h$, is a “held-aside” state and provides one of the pair of states over which atomic propositions, the usual logical connectives $\land$ and $\neg$; the CTL* path quantifiers $E$ and $A$; the CTL* path operators $X$, $F$, and $G$; and our shift operator $S$. All of these elements—with the exception of atomic propositions and the shift operator— are defined in the usual way.
propositions –relations– are checked.

Semantic rule S1 in figure 2 gives the basic case for atomic propositions. An atomic proposition, \( P \), holds in a model \( M \) given states \( s_h \) and \( s_0 \) if and only if \( P \) is in the set of atomic propositions which \( L \) designates to be true for the pair \((s_h, s_0)\).

An assertion using a path formula in Relational CTL* is written

\[ M, s_h, x \models a \]

where the difference relative to the previous state-based assertion is a the path \( x \). Path-based assertions are essentially the same as in CTL*.

The shift operator has the effect of replacing the held-aside state, as can been seen in semantic rules S4 and P4. For state assertions, the shift operator replaces the held-aside state with the reference state \( s_0 \); for path assertions the held-aside state is replaced with the initial state of the path \( x \).

We will sometimes use the abbreviation \( M, s_0 \models a \) in place of \( M, s_0, s_0 \models a \) and similarly for path formulae. Thus,

\[ M, s_0 \models p \triangle M, s_0, s_0 \models a \]

\[ M, x \models p \triangle M, \text{first}(x), x \models a \]

This shorthand is convenient for the formulae where the initial held-aside state is always the same as the initial state; i.e. those that start with the shift operator.

The full set of semantic definitions for Relational CTL* are in figure 2. Let us consider a few examples in Relational CTL* to illustrate how the shift operator works. First, consider the basic assertion

\[ M, s_h, s_0 \models P \]

where \( P \) is some relation in the set of atomic propositions. This case is simple: \( P \) holds if it is in the set designated by \( L \) for the pair of states \((s_h, s_0)\); this is as noted earlier, but also given a graphical depiction. A similar assertion using the shift operator,

\[ M, s_h, s_0 \models SP \]

holds if \( P \) is in the set designated by \( L \) for \((s_0, s_0)\). Semantic rule S4 means that this assertion is equivalent to \( M, s_0, s_0 \models P \).

For the next and eventually operators we will consider a linear example,\(^2\) with \( x \) being the path starting at \( s_0 \) and continuing with \( s_i \) for \( i \in \mathbb{N} \). An assertion such as

\[ M, s_h, x \models XP \]

holds where \( P \) is in the set designated by \( L(s_h, s_1) \), as would be expected. There are two possibilities for adding a single shift operator to the contained formula: the first is

\[ M, s_h, x \models SXP \]

\(^2\)without loss of generality...
which works out to checking $P$ against the pair $(s_0, s_1)$ (i.e. it is equivalent to $M, s_0, x \Vdash XP$); the second is

$$M, s_h, x \Vdash XSP$$

which works out to checking $P$ against the pair $(s_1, s_1)$. The difference between these last two assertions is when the held-aside state is replaced: that is, either before or after entering the context of the $next$ operator. This makes the $shift$ operator non-commutative with respect to the path operators.

The difference is similar for the $eventually$ operator: $M, s_h, x \Vdash FSP$ and $M, s_i, x \Vdash FSP$ correspond to checking $P$ against $L(s_1, s_i)$ and $L(s_i, s_i)$.\textsuperscript{3} Note that the subscript $i$ is bound per semantic rule $P3b$ for $F$.

The last linear example we will consider is representative of a pattern which occurs with some frequency in rely/guarantee reasoning. Consider the assertion

$$M, s_h, x \Vdash FSXP$$

This assertion holds if $P$ is in the set designated by $L(s_i, s_{i+1})$; that is, the assertion holds when $P$ is eventually true of some transition between states along the path.

The path quantifiers, $A$ and $E$, commute with the $shift$ operator; thus, $AS \equiv SA$ and $ES \equiv SE$. A proof of this follows trivially from the rules in figure 2.

Predicates can be encoded as relations which ignore one of the pair of states. However, if the set of atomic propositions contains only right-hand predicates\textsuperscript{4} then the $shift$ operator becomes an identity operation in Relational CTL$^*$ and the logic becomes equivalent to CTL$^*$ without the $until$ operator.

## 3 Undefinedness

This variant of CTL$^*$ introduces a way of reasoning about undefined values to the logic, giving us CTL$^*_\bot$. As noted earlier, the $until$ binary operator has been omitted: it is not entirely clear, for all cases, whether or not the operator should be considered defined.

We give the syntax of CTL$^\bot$ in Figure 3. The set of operators is similar to Relational CTL$^*$, but instead of adding $S$ to the set of operators, we add $\Delta$. The $\Delta$ operator distinguishes between formulae which are defined –and thus able to be interpreted as either true or false in a given model– and formulae which are undefined. We use the $\Delta$ operator here in a similar manner it is used in [JM94], which describes LPF as used in VDM.\textsuperscript{5}

A model, $M$, for CTL$^*_\bot$ is the usual tuple $(S, R, L)$ where $S$ is the set of possible states, $R$ is a total binary relation over $S$, and $L$ is the interpretation. In this

\textsuperscript{3} $M, s_h, x \Vdash FP$ works out to checking $P$ against $L(s_h, s_i)$.

\textsuperscript{4} i.e. those relations which only use the second, or right-hand, state in the pair.

\textsuperscript{5} Unlike LPF, however, we are treating “undefined” as a concrete value rather than as a “gap”. See JonesLovert2010.
S1 Each atomic proposition $P$ is a formula.
S2 If $a, b$ are state formulae then so are $a \land b, \neg a$
S3 If $a$ is a path formula then $Ea, Aa$ are state formulae
S4 If $a$ is a state formula then so is $\Delta a$

P1 Every state formula is a path formula
P2 If $a, b$ are path formulae then so are $a \land b, \neg a$
P3 If $a$ is a path formula then so are $Xa, Fa, Ga$

P4 If $a$ is a path formula then so is $\Delta a$

Figure 3: Syntax of $\text{CTL}^*_\bot$

variant, however, $L$ is total mapping from propositions and states to an element of the set $\mathbb{B} \cup \{\bot\}$. The symbol $\bot$ is used to denote the undefined truth value, being neither true nor false.

The semantics of $\text{CTL}^*_\bot$ are given in figure 4; the notable rules include S1, S2b/P2b, and S4/P4. The basic rule for atomic propositions, S1, means that an atomic proposition holds for the given model and state when the interpretation in $L$ is $\text{true}$. This is a change in mechanism from the $\text{CTL}^*$ and Relational $\text{CTL}^*$ semantics, where set membership is tested. However, set membership is binary — the lack of a specific proposition in the set of true propositions for a given state is not enough to indicate that the proposition is false or undefined.

The fact that not being $\text{true}$ is not the same as being $\text{false}$ leads to the pair of rules S2b and P2b, dealing with negation. A negated formula holds when the formula is defined and its un-negated does not hold. This ensures that the negation of an undefined formula is not $\text{true}$.

The rules given by S4 and P4 deal with undefinedness, and some care was needed to avoid a circular definition. Rule S4a is analogous to S1 in a sense: $\Delta P$ holds precisely when $P$ can be given a boolean interpretation. The remaining rules of S4 and P4 give the conditions for definedness of the remaining formula patterns in $\text{CTL}^*_\bot$.

The specific rules in S4 and P4 deserve further explanation. The rule for definedness of conjunction, S4b, is a direct encoding of the non-strict conjunction from LPF. Negation, S4c, is simply removed when determining definedness, as this follows from S4a.

The definedness rules path quantifiers in S4d and S4e consider the quantifiers as iterated disjunction and conjunction, and thus derive their forms from that. For example, a formula $Ea$ is defined so long as $a$ is $\text{true}$ on even a single possible path; conversely, if $a$ is not $\text{true}$ on any path, then it must be $\text{false}$ on all possible paths for $Ea$ to be $\text{false}$ (and thus defined).

A formula formed using the $\Delta$ operator at the top level is, itself, always defined. From this it is clear that S4f is valid.

The rule for definedness of the $\text{next}$ operator in P4a is straightforward, and the rule for the $\text{eventually}$ operator is clear if one considers the fixpoint definition of the
Many of the basic rules of Relational CTL* are unchanged from those of Relational CTL. The eventually operator can be defined as $\mathcal{F}a \equiv a \lor \mathcal{X}\mathcal{F}a$; this definition is simply iterated disjunction, and leads to a rule for definedness which suits.

### 4 Relations and Undefinedness

Merging the sets of syntax and semantic rules for Relational CTL* and CTL* is not difficult and results in a logic which supports reasoning about both relations and undefinedness. The syntax of this logic, Relational CTL*$_\perp$, is given in figure 5 and its semantics is in figure 6.

A model, $M$, for Relational CTL*$_\perp$ is the usual tuple $(S, R, L)$ where $S$ is the set of possible states, $R$ is a total binary relation over $S$, and $L$ is the interpretation. In this variant, however, $L$ is total mapping from propositions and and pairs of states to an element of the set $\mathbb{B} \cup \{\bot\}$. The symbol $\bot$ is used to denote the undefined truth value, being neither true nor false. Atomic propositions in this logic are relations.

Many of the basic rules of Relational CTL*$_\perp$ are unchanged from those of Relational CTL.
Each atomic proposition $P$ is a formula.

If $a, b$ are state formulae then so are $a \land b, \neg a$

If $a$ is a path formula then $Ea, Af, Gf$ are state formulae

If $a$ is a state formula then so is $\Delta a$

If $a$ is a state formula then so is $Sa$

Every state formula is a path formula

If $a, b$ are path formulae then so are $a \land b, \neg a$

If $a$ is a path formula then so are $Xa, Fa, Ga$

If $a$ is a path formula then so is $\Delta a$

If $a$ is a path formula then so is $Sa$

---

**Figure 5: Syntax of Relational CTL$^*$**

<table>
<thead>
<tr>
<th>Rule</th>
<th>Syntax</th>
<th>iff</th>
</tr>
</thead>
<tbody>
<tr>
<td>$S1$</td>
<td>$M, s, s_0 \models P$</td>
<td>$L(P, s, s_0)$</td>
</tr>
<tr>
<td>$S2a$</td>
<td>$M, s, s_0 \models a \land b$</td>
<td>$M, s, s_0 \models a$ and $M, s, s_0 \models b$</td>
</tr>
<tr>
<td>$S2b$</td>
<td>$M, s, s_0 \models \neg a$</td>
<td>$M, s, s_0 \models \Delta a$ and not($M, s, s_0 \models a$)</td>
</tr>
<tr>
<td>$S3a$</td>
<td>$M, s, s_0 \models Ea$</td>
<td>$\exists x \in $ paths($s_0$) $\cdot M, s_0, x \models a$</td>
</tr>
<tr>
<td>$S3b$</td>
<td>$M, s, s_0 \models Aa$</td>
<td>$\forall x \in $ paths($s_0$) $\cdot M, s_0, x \models a$</td>
</tr>
<tr>
<td>$S4a$</td>
<td>$M, s, s_0 \models \Delta P$</td>
<td>$L(P, s, s_0) \in B$</td>
</tr>
<tr>
<td>$S4b$</td>
<td>$M, s, s_0 \models \Delta(a \land b)$</td>
<td>$(M, s, s_0 \models \Delta a$ and $M, s, s_0 \models \Delta b)$ or $(M, s, s_0 \models \neg a)$ or $(M, s, s_0 \models \neg b)$</td>
</tr>
<tr>
<td>$S4c$</td>
<td>$M, s, s_0 \models \Delta(\neg a)$</td>
<td>$M, s, s_0 \models \Delta a$</td>
</tr>
<tr>
<td>$S4d$</td>
<td>$M, s, s_0 \models \Delta(Ea)$</td>
<td>$(\forall x \in $ paths($s_0$) $\cdot M, s, x \models \Delta a)$ or $(\exists x \in $ paths($s_0$) $\cdot M, s, x \models a)$</td>
</tr>
<tr>
<td>$S4e$</td>
<td>$M, s, s_0 \models \Delta(Aa)$</td>
<td>$(\forall x \in $ paths($s_0$) $\cdot M, s, x \models \Delta a)$ or $(\exists x \in $ paths($s_0$) $\cdot M, s, x \models \neg a)$</td>
</tr>
<tr>
<td>$S4f$</td>
<td>$M, s, s_0 \models \Delta(\Delta a)$</td>
<td>true</td>
</tr>
<tr>
<td>$S4g$</td>
<td>$M, s, s_0 \models \Delta(Sa)$</td>
<td>$M, s_0, s_0 \models \Delta a$</td>
</tr>
<tr>
<td>$S5$</td>
<td>$M, s, s_0 \models Sa$</td>
<td>$M, s_0, s_0 \models a$</td>
</tr>
<tr>
<td>$P1$</td>
<td>$M, s, x \models a$</td>
<td>$M, s, first(x) \models a$</td>
</tr>
<tr>
<td>$P2a$</td>
<td>$M, s, x \models a \land b$</td>
<td>$M, s, x \models a$ and $M, s, x \models b$</td>
</tr>
<tr>
<td>$P2b$</td>
<td>$M, s, x \models \neg a$</td>
<td>$M, s, x \models \Delta a$ and not($M, s, x \models a$)</td>
</tr>
<tr>
<td>$P3a$</td>
<td>$M, s, x \models Xa$</td>
<td>$M, s, x^i \models a$</td>
</tr>
<tr>
<td>$P3b$</td>
<td>$M, s, x \models Fa$</td>
<td>$\exists i \cdot M, s, x^i \models a$</td>
</tr>
<tr>
<td>$P3b$</td>
<td>$M, s, x \models \forall i \cdot M, s, x^i \models a$</td>
<td>$\forall i \cdot M, s, x^i \models a$</td>
</tr>
<tr>
<td>$P4a$</td>
<td>$M, s, x \models \Delta(Xa)$</td>
<td>$M, s, x^i \models \Delta a$</td>
</tr>
<tr>
<td>$P4b$</td>
<td>$M, s, x \models \Delta(Fa)$</td>
<td>$(\forall i \cdot M, s, x^i \models \Delta a)$ or $(\exists i \cdot M, s, x^i \models a)$</td>
</tr>
<tr>
<td>$P4c$</td>
<td>$M, s, x \models \Delta(Ga)$</td>
<td>$(\forall i \cdot M, s, x^i \models \Delta a)$ or $(\exists i \cdot M, s, x^i \models \neg a)$</td>
</tr>
<tr>
<td>$P4d$</td>
<td>$M, s, s_0 \models \Delta(\Delta a)$</td>
<td>true</td>
</tr>
<tr>
<td>$P4e$</td>
<td>$M, s, x \models \Delta(Sa)$</td>
<td>$M, first(x), x \models \Delta a$</td>
</tr>
<tr>
<td>$P5$</td>
<td>$M, s, x \models Sa$</td>
<td>$M, first(x), x \models a$</td>
</tr>
</tbody>
</table>

---

**Figure 6: Semantics of Relational CTL$^*$**

It is easy to see this logic as the addition of rules for reasoning about undefinedness to Relational CTL$^*$ (rather than as the addition of rules for reasoning about relations to CTL$^*$).
The only genuinely new rules in the semantics are $S_4g$ and $P_4e$: they describe when formulae using the *shift* operator are defined. Like rule $P_4a$—which deals with definedness of the *next* operator—rules $S_4g$ and $P_4e$ depend upon the definedness of the contained formula after the operator’s effect is felt. That is, a formula using the *shift* operator at the top level is defined if the contained formula is defined after the shift of the held-aside state.

5 Conclusions and Future Work

An immediate area for future work that suggests itself is to provide an axiomatization of the logics given here, as well as full soundness proofs. Related to this, we feel that it would be a useful exercise to provide a semantics of CTL* and these extensions in an inference rule-based format. There are two advantages to such a format: first, it would place the logical axioms in the same format as those used in the proofs of Coleman and Jones’ joint paper [CJ07], allowing a tighter integration into proofs of that style; and, second, it would allow for the creation of a logical frame in the style of mural [JJLM91].

It would also be useful to provide tool support for these logics. Some work on implementing three-valued temporal logics in tools has been done; Chechik et al [CED01] is one example.

As far as we know, the use of relational atomic propositions in a temporal logic is unique, and is worth further exploration. A relational extension to a linear-time temporal logic may provide further insight into this sort of structure.

Acknowledgements Thanks are due to Cliff Jones and Ben Moszkowski for discussions about this work; and to Edsko de Vries for pointing out that the $S$ operator is orthogonal to the $F$ and $X$ operators, thereby simplifying the structure of Relational CTL* immensely.

The author would also like to gratefully acknowledge support from the EPSRC “TrAmS” platform grant.

References


