VIS CTL Syntax

Yuji Kukimoto† Jae-Young Jang‡

† The VIS Group
University of California, Berkeley
vis@ic.eecs.berkeley.edu

‡ The VIS Group
University of Colorado, Boulder
vis@ic.eecs.berkeley.edu

February 27, 1997

CTL (Computation Tree Logic) is a language used to describe properties of systems. This document describes the CTL syntax used in VIS. For the semantics of CTL, the reader should refer to the following paper.

> E. M. Clarke, E. A. Emerson and A. P. Sistla, *Automatic Verification of Finite-State Concurrent Systems Using Temporal Logic Specifications*, ACM Transactions on Programming Languages and Systems, vol 8-2, pages 244-263, April, 1986

This syntax should be followed when VIS users create CTL files and fairness constraint files for the commands `model_check`, `approximate_model_check`, and `read_fairness`, respectively.

The syntax for CTL is:

> TRUE, FALSE, and *var-name*=*value* are CTL formulas, where *var-name* is the full hierarchical name of a variable, and *value* is a legal value in the domain of the variable. *var-name1* == *var-name2* is the atomic formula that is true if *var-name1* has the same value as *var-name2*. Currently it can be used only in the Boolean domain. (It cannot be used for variables of enumerated types.) *var-name1*[i:j] == *var-name2*[k:l] can be used if the lengths of vectors are the same. Vector variables, the syntax of hierarchical names, and macro definition are described later in this document. The following character set may be used for variable names and values:

```
A-Z a-z 0-9 ^ ? | / [ ] + * $ < > ~ @ _ # % : " ' .
```

> If f and g are CTL formulas, then so are the following:

```
(f), f * g, f + g, f ^ g, !f, f -> g, f <-> g, AG f,
AF f, AX f, EG f, EF f, EX f, A(f U g) and E(f U g).
```

`AX:n(f)` is allowed as a shorthand for `AX(AX(...AX(f)...))`, where `n` is the number of invocations of `AX`. `EX:n(f)` is defined similarly.

Binary operators must be surrounded by spaces, i.e. `f + g` is a CTL formula while `f+g` is not. The same is true for `U` in until formulas. Once parentheses are inserted, the spaces can be omitted, i.e. `(f)+(g)` is a valid formula. Unary temporal operators and their arguments must be separated by spaces unless parentheses are used.

The symbols have the following meanings[1].

```
* -- AND, + -- OR, ^ -- XOR, ! -- NOT, -> -- IMPLY,
<-> -- EQUIV
```

Operator Precedence:

High

```
!

AG, AF, AX, EG, EF, EX

*

+

^

<->

->

U
```

Low

An entire formula should be followed by a semicolon. All text from `#` to the end of a line is treated as a comment. The model checker (`mc`) package is used to decide whether or not a given FSM satisfies a given CTL formula. See the help files for the `model_check` and `approximate_model_check` commands for more details.

Hierarchical Names

If a variable *var* belongs to the root model, the hierarchical name of the variable is *var*. If the variable belongs to a subcircuit in the root model, the hierarchical name of the

[1] && and || are also supported for AND and OR respectively.

variable is *subcircuit_name.var*, where *subcircuit_name* is the name of the subcircuit. In general, the hierarchical name of a variable is the names of the subcircuits concatenated with '.' followed by its name.

Vector Variables

Vector variables have the form *var-name*[i:j]; each bit can be written as either *var-name*<i> or *var-name*[i]. To compare a *value* to a *vector variable*, the following can be used.

```
var-name[i:j] = n
              = bxxx
              = {value1,value2,...,valueN}
```

```
n: integer, bxxx: binary string, valueN: either n or bxxx
```

For instance, counter[3:0] = 10, counter[3:0] = b1010, counter[3:0] = {1,2,10} are valid formulae. For variables of enumerated types, one can write *var-name* = {RED, YEL-LOW, BLUE} without vector indices. *var-name*[i:j] = {value1,value2,...,valueN} represents set membership; that is, it is equivalent to (*var-name*[i:j] = value1) $\vee$ (*var-name*[i:j] = value2) $\vee$ ... $\vee$ (*var-name*[i:j] = valueN).

Macro Definition

Macros can be defined and used in CTL formulae for valid sub-formulae. You cannot use a macro such as "\define MSB 7" since "7" is not a valid CTL formula. The macro definition must precede its use.

```
{\DEFINE | \Define | \define} MACRO formula
```

For instance,

```
\define Red t_sig = RED
\define Blue t_sig = BLUE
```

defines `\Red` as a shorthand for `t_sig = RED`. Similarly for `\Blue`. One can then write:

```
AG ( \Red -> AX ( \Blue ) );
```