Formalization and Verification of Pointers in the Temporal Logic Language XYZ/E Programs*

LI Guang-yuan[1,2] TANG Zhi-song[1]

[1](Lab. for Computer Science Institute of Software The Chinese Academy of Sciences Beijing 100080)

[2](Institute of Computer Science Guizhou University Guiyang 550025)

E-mail: ligy@ox.ios.ac.cn

Abstract Pointer is an important data type in most programming languages. It can make programs more efficient and more elegant. Unfortunately, this important concept is always notorious for its timelessness. Until now, no proper way to formalize it in temporal logic language has been found. XYZ/E is a temporal logic system as well as a programming language. It can represent almost every kind of significant features in conventional imperative languages. This paper is devoted to the representation of pointer in language XYZ/E and the verification of XYZ/SE programs with pointers.

Key words Temporal logic, formal semantics, program verification, dynamic semantics, pointer.

Pointer is an important data type in most programming languages. It can make programs more efficient and more elegant. Unfortunately, this important concept is always notorious for its timelessness. Until now, no proper way to formalize it in temporal logic language has been found. As a result, many important conventional programming features are beyond the reach of formalization. One of the well known problems of this kind is the problem of dynamic binding of the communicating processes (as shown in telecommunication).

XYZ system is a programming support system with the goal to enhance reliability and productivity of software development[1]. It consists of a temporal logic language XYZ/E to serve as its kernel and a group of CASE tools based on XYZ/E. One of the advantages of XYZ/E is that both high level and low level specifications (i.e. the static semantics vs. the dynamic semantics) can be represented in the same framework. Owing to this special characteristic of XYZ/E, the pointers can be represented and verified in a formal way in XYZ. So far as we know, until now no other system can deal with this problem more effectively. We believe that it would have strong impact in formal-based programming.

XYZ/SE is a structured version of XYZ/E, and is selected as the object language of the verification tools in XYZ system. A set of Hoare-style rules for verification of XYZ/SE programs was given in Refs. [2,3]. A verification tool XYZ/VERI based on these rules has been implemented[3]. To verify an XYZ/SE program, the user provides the program, the pre-condition, the post-condition and the loop-invariance of the program as input and the verification system XYZ/VERI produces verification conditions which guarantee the partial correctness

* This research is supported by the National 'Ninth-Five' Sci-Tech Key Project of China("九五"国家重点科技攻关项目计划,No. 98-780-01-07-01). **LI Guang-yuan** was born in 1962. He is a Ph.D. student at the Institute of Software, the Chinese Academy of Sciences. He received the M.Sc. degree in mathematics from Guizhou University in 1990. His research interests are temporal logic, formal methods, model checking and real-time systems. **TANG Zhi-song** was born in 1925. He is a professor, a doctoral supervisor and an academician. His current research areas include temporal logic language, software engineering, CASE tools and software architecture.

Manuscript received 1998-03-27, accepted 1998-11-01.

(safety property) of the program.

But the rules given in Refs. [2,3] only work for XYZ/SE programs without pointers. In this paper, we modify these rules so as to fit the verification of XYZ/SE programs with pointers.

The rest of this paper is organized as follows. Section 1 describes the representation of pointers in temporal logic language XYZ/E. Section 2 presents the verification rules for XYZ/SE programs with pointers. Section 3 presents two examples to show how to verify an XYZ/SE program with pointers. Section 4 contains the conclusion and future work.

1 Representation of Pointers in XYZ/E

In XYZ/E, pointer is a temporal variable that contains the name of another variable. If one variable contains the name of another variable, then the first variable is said to point to the second. The base type of a pointer variable is defined to be the type of variables the pointer can point to.

We use $\&x$ to denote the name of the variable x. It can be considered as a string constant in XYZ/E programs. For example, $\&abc$ is the name of the variable abc. It can be identified with string constant "abc" in XYZ/E programs.

In XYZ/E, the equation $v=\&x$ indicates that pointer variable v points to variable x at present time, and $\$Ov=\&x$ indicates that pointer v points to x at next time. The equation $\$O*v=e$ can be used to indicate that the value of expression e will be assigned to the variable pointed to by pointer v at next time. In fact, here equation $\$O*v=e$ is used as an abbreviated form of the temporal logic formula $(v=\&x_1\rightarrow \$Ox_1=e)\wedge(v=\&x_2\rightarrow \$Ox_2=e)\wedge\ldots\wedge(v=\&x_n\rightarrow \$Ox_n=e)$, where $x_1, x_2, \ldots, x_n$ are all variables whose type is the base type of the pointer variable v.

In XYZ/E, you can have a pointer pointing to another pointer that points to the target value. But to simplify our discussion, this paper does not allow the expressions of the form $**v$ to occur in XYZ/E programs.

We use two forms of conditional elements (c.e.) in XYZ/E programs. They are

$$LB=l_1\wedge R\Rightarrow \$O(v_1,v_2,\ldots,v_n)=(e_1,e_2,\ldots,e_n)\wedge \$OLB=l_2 \qquad [1.1]$$

and

$$LB=l_1\wedge R\Rightarrow \$O*v=e\wedge \$OLB=l_2 \qquad [1.2]$$

Here R is a first order logic formula without temporal operators, and $e_1,e_2,\ldots,e_n$ are expressions.

In XYZ/E programs, conditional element [1.2] is used as an abbreviated form of the program block $b(l_1,l_2)$ consisting of the conditional elements of the form [1.1]. In fact,

$$
\begin{aligned}
b(l_1,l_2) == [\\
&LB = l_1 \wedge u_1 = \&x_{11} \wedge\ldots\wedge u_n = \&x_{n1} \wedge R[x_{11}/*u_1,\ldots,x_{n1}/*u_n]\\
&\quad\Rightarrow \$OLB = l_2 \wedge (\$O*v = e)[x_{11}/*u_1,\ldots,x_{n1}/*u_n];\\
&\quad LB = l_1 \wedge u_1 = \&x_{12} \wedge\ldots\wedge u_n = \&x_{n1} \wedge R[x_{12}/*u_1,\ldots,x_{n1}/*u_n]\\
&\quad\Rightarrow \$OLB = l_2 \wedge (\$O*v = e)[x_{12}/*u_1,\ldots,x_{n1}/*u_n];\\
&\ldots\ \ldots;\\
&LB = l_1 \wedge u_1 = \&x_{1m_1} \wedge\ldots\wedge u_n = \&x_{nm_n}\\
&\qquad\wedge R[x_{1m_1}/*u_1,\ldots,x_{nm_n}/*u_n]\\
&\quad\Rightarrow \$OLB = l_2 \wedge (\$O*v = e)[x_{1m_1}/*u_1,\ldots,x_{nm_n}/*u_n]\\
&],
\end{aligned}
$$

where $u_1,u_2,\ldots,u_n$ are all pointer variables occurring in conditional element [1.2], and $x_{j1},x_{j2},\ldots,x_{jm_j}$ are all variables pointed to by pointer u_j, and $R[x_{1k_1}/*u_1,\ldots,x_{nk_n}/*u_n]$ is used to indicate the result of substituting simultaneously $x_{1k_1},\ldots,x_{nk_n}$ for all occurrences of $*u_1,\ldots,*u_n$ respectively in formula R. Obviously, the program block $b(l_1,l_2)$ is composed of the conditional elements of the form [1.1]. So conditional element [1.2] is the abbreviated form of a program block consisting of the conditional elements of the form [1.1].

2 Verification of XYZ/SE Programs with Pointers

An XYZ/SE program is an XYZ/E program constructed from conditional elements, simple case blocks, loop blocks, and sequential compositions.

An XYZ/SE program block has an entry label and an exit label. These two labels uniquely determine the program block. Generally, we use $b(l_1,l_2)$ to represent the program block with entry label l_1 and exit label l_2.

(1) A conditional element is of the form

$$LB=l_1 \Rightarrow \$O(v_1,v_2,\ldots,v_n)=(e_1,e_2,\ldots,e_n) \wedge \$OLB=l_2$$

or

$$LB=l_1 \wedge R \Rightarrow \$O*v=e \wedge \$OLB=l_2$$

(2) A simple case block is of the form

$$?[LB=l_1 \wedge C \Rightarrow \$OLB=l_3; LB=l_1 \wedge \sim C \Rightarrow \$OLB=l_4; b(l_3,l_2); b(l_4,l_2)]$$

(3) A loop block is of the form

$$*[LB=l_1 \wedge C \Rightarrow \$OLB=l_3; LB=l_1 \wedge \sim C \Rightarrow \$OLB=l_2; b(l_3,l_1)]$$

(4) A sequential composition is of the form

$$[b(l_1,l_3); b(l_3,l_2)]$$

Before giving the verification rules for programs with pointers, we first give a fact that will be used in the presentation of these verification rules.

Fact 1. Every formula Q in an XYZ/SE program can be replaced without changing the function of the program by a formula Q' containing no expressions of the form $*v$.

In fact,

$$\begin{aligned} Q' = & (u_1=\&x_{11} \wedge \ldots \wedge u_n=\&x_{n1} \rightarrow Q[x_{11}/*u_1,\ldots,x_{n1}/*u_n]) \\ & \wedge (u_1=\&x_{12} \wedge \ldots \wedge u_n=\&x_{n2} \rightarrow Q[x_{12}/*u_1,\ldots,x_{n2}/*u_n]) \\ & \wedge \ldots\ldots \\ & \wedge (u_1=\&x_{1m_1} \wedge \ldots \wedge u_n=\&x_{nm_n} \rightarrow Q[x_{1m_1}/*u_1,\ldots,x_{nm_n}/*u_n]) \end{aligned}$$

where $u_1,u_2,\ldots,u_n$ are all pointer variables occurring in formula Q, and $x_{j1}, x_{j2}, \ldots, x_{jm_j}$ are all variables pointed to by pointer u_j in the program. Obviously, Q' contains no expressions of the form $*v$ and is equivalent to Q.

In the rest of this section, for any first order formula Q without temporal operators, we use Q' to denote the formula obtained from Q as described above.

Now we begin to present the verification rules for XYZ/SE programs with pointers. In the following rules, R, R_1, R_2 and R_3 are first order formulas without temporal operators.

Conditional element:

$$(I) \quad \vdash [\,]b(l_1,l_2) \wedge LB=l_1 \wedge (R[e_1/v_1,e_2/v_2,\ldots,e_n/v_n])' \Rightarrow R \triangle LB=l_2 \qquad [2.1.1]$$

where $b(l_1,l_2)$ is $LB=l_1 \Rightarrow \$O(v_1,v_2,\ldots,v_n)=(e_1,e_2,\ldots,e_n) \wedge \$OLB=l_2$ and "$\triangle$" is Kroger's *atnext* operator[4].

$$\begin{aligned} (II) \quad \vdash [\,]b(l_1,l_2) \wedge LB=l_1 \wedge (v=\&y_1 \rightarrow (R[e/y_1])') \wedge \ldots \\ \wedge (v=\&y_m \rightarrow (R[e/y_m])') \Rightarrow R \triangle LB=l_2 \end{aligned} \qquad [2.1.2]$$

where $b(l_1,l_2)$ is $LB=l_1\wedge R\Rightarrow \$O*v=e\wedge \$OLB=l_2$ and $y_1,y_2,\ldots,y_m$ are all variables pointed to by pointer v in the program that block $b(l_1,l_2)$ occurs in.

Simple case block:

$$\frac{\vdash[\,]b(l_3,l_2)\wedge LB=l_3\wedge R_1\wedge C'\Rightarrow R_2\triangle LB=l_2 \qquad \vdash[\,]b(l_4,l_2)\wedge LB=l_4\wedge R_1\wedge\sim C'\Rightarrow R_2\triangle LB=l_2}{\vdash[\,]b(l_1,l_2)\wedge LB=l_1\wedge R_1\Rightarrow R_2\triangle LB=l_2} \qquad [2.2]$$

where $b(l_1,l_2)$ is ? $[LB=l_1\wedge C\Rightarrow \$OLB=l_3;\ LB=l_1\wedge\sim C\Rightarrow \$OLB=l_4;\ b(l_3,l_2);\ b(l_4,l_2)]$.

Loop block:

$$\frac{\vdash[\,]b(l_3,l_1)\wedge LB=l_3\wedge R\wedge C'\Rightarrow R\triangle LB=l_1}{\vdash[\,]b(l_1,l_2)\wedge LB=l_1\wedge R\Rightarrow(R\wedge\sim C')\triangle LB=l_2} \qquad [2.3]$$

where $b(l_1,l_2)$ is $*[LB=l_1\wedge C\Rightarrow \$OLB=l_3;\ LB=l_1\wedge\sim C\Rightarrow \$OLB=l_2;\ b(l_3,l_1)]$, R is a loop-invariant of loop block $b(l_1,l_2)$.

Sequential composition:

$$\frac{\vdash[\,]b(l_1,l_3)\wedge(LB=l_1)\wedge R_1\Rightarrow R_3\triangle LB=l_3 \qquad \vdash[\,]b(l_3,l_2)\wedge(LB=l_3)\wedge R_3\Rightarrow R_2\triangle LB=l_2}{\vdash[\,]b(l_1,l_2)\wedge(LB=l_1)\wedge R_1\Rightarrow R_2\triangle LB=l_2} \qquad [2.4]$$

where $b(l_1,l_2)$ is $[b(l_1,l_3);\ b(l_3,l_2)]$.

Rule for procedure call is the same as that in Ref. [3]. The only restriction on procedure call is that pointer variable can not be used as an input or output parameter of a procedure. So we omit the verification rule for procedure call here.

3 Examples

Example 1. The following procedure exchanges the values of two variables x and y.

```
{x = #x ∧ y = #y}
%PROC f(%IOP/x:INT; %IOP/y:INT) —— [
%LOC[t:INT;p1,p2:POINT(INT)];
      %STM[
      LB = START⇒ $O(p1,p2) = (&x,&y) ∧ $OLB = l1;
      LB = l1⇒$Ot = *p1 ∧ $OLB = l2;
      LB = l2⇒$O*p1 = *p2 ∧ $OLB = l3;
      LB = l3⇒$O*p2 = t ∧ $OLB = RETURN
      ]
   ]
{x = #y ∧ y = #x}
```

In the above program, $\&x$ and $\&y$ are the names of variables x and y. They are two constants in the procedure f, and satisfy the properties: $\&x=\&x$, $\&y=\&y$, and $\&x\neq\&y$. The terms $\#x$ and $\#y$ that occur in the pre-condition and the post-condition are used here to denote the initial values of variables x and y.

Now we verify the partial correctness of procedure f.

(1) $\vdash[\,]b(l3,RETURN)\wedge LB=l3$

$\wedge(p2=\&x\rightarrow(t=\#y\wedge y=\#x))\wedge(p2=\&y\rightarrow(x=\#y\wedge t=\#x))\Rightarrow(x=\#y\wedge y=\#x)$

$\triangle LB=RETURN$

by rule [2.1.2].

(2) $\vdash[\,]b(l2,l3)\wedge LB=l2$

$\wedge(p1=\&x\wedge p2=\&x\rightarrow(t=\#y\wedge y=\#x))$

$\wedge(p1=\&x\wedge p2=\&y\rightarrow(y=\#y\wedge t=\#x))$
$\wedge(p1=\&y\wedge p2=\&x\rightarrow(t=\#y\wedge x=\#x))$
$\wedge(p1=\&y\wedge p2=\&y\rightarrow(x=\#y\wedge t=\#x))$
$\Rightarrow(p2=\&x\rightarrow(t=\#y\wedge y=\#x))$
$\wedge(p2=\&y\rightarrow(x=\#y\wedge t=\#x))\triangle LB=l3$

by rule [2.1.2] and simplifying.

(3) $\vdash[\,]b(l1,l2)\wedge LB=l1$
$\wedge(p1=\&x\wedge p2=\&x\rightarrow(x=\#y\wedge y=\#x))$
$\wedge(p1=\&x\wedge p2=\&y\rightarrow(y=\#y\wedge x=\#x))$
$\wedge(p1=\&y\wedge p2=\&x\rightarrow(y=\#y\wedge x=\#x))$
$\wedge(p1=\&y\wedge p2=\&y\rightarrow(x=\#y\wedge y=\#x))$
$\Rightarrow(p1=\&x\wedge p2=\&x\rightarrow(t=\#y\wedge y=\#x))$
$\wedge(p1=\&x\wedge p2=\&y\rightarrow(y=\#y\wedge t=\#x))$
$\wedge(p1=\&y\wedge p2=\&x\rightarrow(t=\#y\wedge x=\#x))$
$\wedge(p1=\&y\wedge p2=\&y\rightarrow(x=\#y\wedge t=\#x))\triangle LB=l2$

by rule [2.1.1] and simplifying.

(4) $\vdash[\,]b(START,l1)\wedge LB=START$
$\wedge(\&x=\&x\wedge\&y=\&x\rightarrow(x=\#y\wedge y=\#x))$
$\wedge(\&x=\&x\wedge\&y=\&y\rightarrow(y=\#y\wedge x=\#x))$
$\wedge(\&x=\&y\wedge\&y=\&x\rightarrow(y=\#y\wedge x=\#x))$
$\wedge(\&x=\&y\wedge\&y=\&y\rightarrow(x=\#y\wedge y=\#x))$
$\Rightarrow(p1=\&x\wedge p2=\&x\rightarrow(x=\#y\wedge y=\#x))$
$\wedge(p1=\&x\wedge p2=\&y\rightarrow(y=\#y\wedge x=\#x))$
$\wedge(p1=\&y\wedge p2=\&x\rightarrow(y=\#y\wedge x=\#x))$
$\wedge(p1=\&y\wedge p2=\&y\rightarrow(x=\#y\wedge y=\#x))\triangle LB=l1$

by rule [2.1.1].

(5) $\vdash(\&x=\&x\wedge\&y=\&x\rightarrow(x=\#y\wedge y=\#x))$
$\wedge(\&x=\&x\wedge\&y=\&y\rightarrow(y=\#y\wedge x=\#x))$
$\wedge(\&x=\&y\wedge\&y=\&x\rightarrow(y=\#y\wedge x=\#x))$
$\wedge(\&x=\&y\wedge\&y=\&y\rightarrow(x=\#y\wedge y=\#x))\equiv(y=\#y\wedge x=\#x)$

by simplifying.

(6) $\vdash[\,]b(START,l1)\wedge LB=START\wedge(y=\#y\wedge x=\#x)$
$\Rightarrow(p1=\&x\wedge p2=\&x\rightarrow(x=\#y\wedge y=\#x))$
$\wedge(p1=\&x\wedge p2=\&y\rightarrow(y=\#y\wedge x=\#x))$
$\wedge(p1=\&y\wedge p2=\&x\rightarrow(y=\#y\wedge x=\#x))$
$\wedge(p1=\&y\wedge p2=\&y\rightarrow(x=\#y\wedge y=\#x))\triangle LB=l1$

by (4) and (5).

(7) $\vdash[\,]b(START,l2)\wedge LB=START\wedge(y=\#y\wedge x=\#x)$
$\Rightarrow(p1=\&x\wedge p2=\&x\rightarrow(t=\#y\wedge y=\#x))$
$\wedge(p1=\&x\wedge p2=\&y\rightarrow(y=\#y\wedge t=\#x))$
$\wedge(p1=\&y\wedge p2=\&x\rightarrow(t=\#y\wedge x=\#x))$
$\wedge(p1=\&y\wedge p2=\&y\rightarrow(x=\#y\wedge t=\#x))\triangle LB=l2$

by (6), (3) and rule [2.4].

(8) $\vdash [\,]b(START, l3) \wedge LB=START \wedge (y=\#y \wedge x=\#x)$

$\Rightarrow (p2=\&x \to (t=\#y \wedge y=\#x))$

$\wedge (p2=\&y \to (x=\#y \wedge t=\#x)) \triangle LB=l3$

by (7), (2) and rule [2.4].

(9) $\vdash [\,]b(START, RETURN)$

$\wedge (LB=START \wedge (y=\#y \wedge x=\#x)$

$\Rightarrow (x=\#y \wedge y=\#x) \triangle LB=RETURN$

by (8), (1) and rule [2.4].

Example 2. Verify the partial correctness of procedure g.

```
{|a-b|>h}
%PROC g(%IOP/a:INT; %IOP/b:INT)==[
%LOC[p:POINT(INT)];
%STM[
        ?[LB=START∧(a<b)⇒$OLB=l1;
        LB=START∧~(a<b)⇒$OLB=l2;
        LB=l1⇒$Op=&b∧$OLB=l3;
        LB=l2⇒$Op=&a∧$OLB=l3
        ]
        LB=l3⇒$O*p=*p+1∧$OLB=RETURN
        ]
]
{|a-b|>h+1}
```

The verification process runs as follows:

(1) $\vdash [\,]b(l3, RETURN) \wedge LB=l3$

$\wedge (p=\&a \to (|a+1-b|>h-1))$

$\wedge (p=\&b \to (|a-(b+1)|>h+1))$

$\Rightarrow (|a-b|>h+1) \triangle LB=RETURN.$

by rule [2.1.2].

(2) $\vdash [\,]b(l1, l3) \wedge LB=l1 \wedge (|a-(b+1)|>h+1)$

$\Rightarrow (p=\&a \to (|a+1-b|>h+1))$

$\wedge (p=\&b \to (|a-(b+1)|>h+1)) \triangle LB=l3$

by rule [2.1.1] and simplifying.

(3) $\vdash [\,]b(l2, l3) \wedge LB=l2 \wedge (|a+1-b|>h+1)$

$\Rightarrow (p=\&a \to (|a+1-b|>h+1))$

$\wedge (p=\&b \to (|a-(b+1)|>h+1)) \triangle LB=l3$

by rule [2.1.1] and simplifying.

(4) $\vdash [\,]b(l1, l3) \wedge LB=l1 \wedge ((a<b) \to (|a-(b+1)|>h+1))$

$\wedge (\sim(a<b) \to (|a+1-b|>h+1)) \wedge (a<b)$

$\Rightarrow (p=\&a \to (|a+1-b|>h+1))$

$\wedge (p=\&b \to (|a-(b+1)|>h+1)) \triangle LB=l3$

by (2).

(5) $\vdash [\,]b(l2, l3) \wedge LB=l2 \wedge ((a<b) \to (|a-(b+1)|>h+1))$

$\wedge (\sim(a<b) \to (|a+1-b|>h+1)) \wedge \sim(a<b)$

$$\Rightarrow (p=\&a \rightarrow (|a+1-b|>h+1))$$
$$\wedge (p=\&b \rightarrow (|a-(b+1)|>h+1)) \triangle LB=l3$$

by (3).

(6) $\vdash [\,]b\,(START, l3) \wedge LB=START$
$$\wedge ((a<b) \rightarrow (|a-(b+1)|>h+1))$$
$$\wedge (\sim (a<b) \rightarrow (|a+1-b|>h+1))$$
$$\Rightarrow (p=\&a \rightarrow (|a-1-b|>h+1))$$
$$\wedge (p=\&b \rightarrow (|a-(b+1)|>h+1)) \triangle LB=l3$$

by (4), (5) and rule [2.2].

(7) $\vdash [\,]b\,(START, RETURN) \wedge LB=START$
$$\wedge ((a<b) \rightarrow (|a-(b+1)|>h+1))$$
$$\wedge (\sim (a<b) \rightarrow (|a+1-b|>h+1))$$
$$\Rightarrow (|a-b|>h+1) \triangle LB=RETURN$$

by (1), (6) and rule [2.4].

(8) $\vdash ((a<b) \rightarrow (|a-(b+1)|>h+1))$
$$\wedge (\sim (a<b) \rightarrow (|a+1-b|>h+1)) \equiv (|a-b|>h)$$

by simplifying.

(9) $\vdash [\,]b\,(START, RETURN) \wedge LB=START \wedge (|a-b|>h)$
$$\Rightarrow (|a-b|>h+1) \triangle LB=RETURN$$

by (7) and (8).

4 Conclusion

This paper has discussed the representation of pointers in temporal logic language XYZ/E, and has presented the rules to verify XYZ/SE programs with pointers. It turns out that XYZ/E is a suitable language for representing such dynamic mechanism as pointers.

In the future work, we plan to remove some restrictions on the application of pointers in XYZ/SE programs (for example, pointers can not be used as input or output parameter of procedures). We also plan to develop some methods or strategies to simplify the process of program verification.

Acknowledgment Li Guang-yuan would like to thank Dr. Zhao Chen and Dr. Shen Wu-wei for their many helps during the time he stays in the Institute of Software, the Chinese Academy of Sciences.

References

1 Tang C S. A temporal logic oriented toward software engineering——an introduction to XYZ system (I). Chinese Journal of Advanced Software Research, 1994,1(1):1~29

2 Xie Hongliang, Gong Jie, Tang C S. A structured temporal logic language: XYZ/SE. Journal of Computer Science and Technology, 1991,6(1):1~10

3 Zhang Wenhui. Verification of XYZ/SE programs. Chinese Journal of Advanced Software Research, 1995,2(4):364~373

4 Kroger F. Temporal Logic of Programs. Berlin: Springer-Verlag, 1987

时序逻辑语言 XYZ/E 中指针的形式化表示与验证

李广元[1,2]　唐稚松[1]

[1](中国科学院软件研究所计算机科学开放研究实验室　北京　100080)

[2](贵州大学计算机理论研究所　贵阳　550025)

摘要　指针是一种重要的数据类型,使用指针能使程序更加有效和优美.可是指针却以不易驾御而闻名,至今在时序逻辑语言中未见到对它的形式化工作.XYZ/E 既是一个时序逻辑系统也是一个程序设计语言,它能表示普通高级语言中几乎所有的重要机制.本文主要讨论在时序逻辑语言 XYZ/E 中指针的形式化表示问题以及在结构化 XYZ/SE 程序中指针的验证问题.

关键词　时序逻辑,形式语义,程序验证,动态语义,指针.

中图法分类号　TP312