基于时序逻辑语言 XYZ/E 的实时系统应用研究*

刘珊艳，张广泉
(重庆师范大学 数学与计算机科学学院，重庆 400047)

摘 要：针对实时系统在计算机系统应用中的重要性，采用形式化方法是保证实时系统软件开发正确性的一种重要途径，而时序逻辑这种形式化方法是研究实时系统的一个重要的理论基础。本文给出了时序逻辑语言 XYZ/E 的相关介绍，并利用 XYZ/E 的子语言 XYZ/RBE 与 XYZ 图描述了煤气炉实时控制问题。

关键词：实时系统；形式化方法；时序逻辑；XYZ/E
中图分类号：TP311 **文献标识码**：A **文章编号**：1672-6693(2006)03-0063-04

The Application of Real-time System Based on Temporal Logic XYZ/E

LIU Shan-Yan, ZHANG Guang-quan
(College of Mathematics and Computer Science, Chongqing Normal University, Chongqing 400047, China)

Abstract: Formal method is an important approach to guarantee the correctness of real-time system, due to the importance when applying it into the computer system, and the temporal logic is an important theoretical foundation of real-time system research. This paper introduces the temporal logic language XYZ/E, and then describes the gasburner control problem in XYZ/RBE and XYZ diagram.

Key words: real-time systems; formal methods; temporal logic; XYZ/E

1 实时系统概述

在现代社会许多系统和设备的正确性不仅仅依赖于它产生的结果或效果，而且还依赖它产生结果所需要的时间。最近，越来越多的计算机硬件和软件嵌入到这样的对时间有要求的系统中去，监视和控制它们的操作，这样的计算机系统叫作嵌入式系统、实时计算机系统或者就叫作实时系统。和传统的非实时系统不同的是实时计算机系统是与外界环境的监视和控制联系在一起的，是一种带时间约束的计算机系统。

实时系统典型的例子有新一代的飞机和太空船上的电子设备、计划空间站的控制软件、高可靠的网络和电话转换系统、核电站控制、病人护理监控和机器人等。这些系统许多动作的完成是与时间相关，即要满足一定的时间限制，如某动作要在一秒钟内完成等，对这类系统确保其正确性和可靠性至关重要。

有两种方法可以提高系统的正确性和可靠性。一种是使用工程(包括软件和硬件)技术，比如使用结构程序原则来降低程序执行的错误，或者使用检测技术来发现执行中的错误。另一种方法是使用形式化分析和验证技术来确保执行系统满足在所有情况下给出的假设的需求安全约束。在实时系统中不仅仅要满足严格的时间上的要求，而且能够监控与预先设计假设有干扰的执行环境。第一种方法仅仅只能增加对系统正确性要求的可信度，因为测试永远不能保证系统没有错误。第二种方法就可以保证所检验系统永远可以满足所要求的安全性。故采用形式化方法对实时系统进行精确分析、设计与验证是保证其正确性和可靠性的重要途径。目前，广泛应用的一些形式语言有 Z、CSP、时序逻辑(Temporal Logic)等，它们都具有严格数学定义的语义[1]，而其中时序逻辑是实时系统研究的一个理论基础。

* 收稿日期：2006-01-11
资助项目：重庆市教委科学技术研究项目(No. 040803)；中国科学院计算机重点实验室开放科题(No. SYSKF0303)
作者简介：刘珊艳(1979-)，女，湖北潜江人，硕士研究生，研究方向为形式化方法。

2 时序逻辑 XYZ/E 的简介

近年来,随着软件工程、人工智能的产生和发展,人们逐渐认识到时序逻辑对计算机科学的重要影响,1977 年 A. Pnueli 首次将时序逻辑引入计算机科学,目前计算机科学家已开发多种时序逻辑(语言),如 Z. Manna 和 A. Pnueli 的 PLTL、Chandy 和 Misrs 的 UNITY、唐稚松的 XYZ/ E 等[2]。其中唐稚松教授提出的 XYZ/E[3] 是世界上第一个可运行的时序逻辑语言。它是一种以线性时序逻辑为基础的形式化程序设计语言,能在统一的语义框架中表示从形式规范到可执行程序的不同抽象层次的系统描述,既能表示程序的动态语义,又能表示规范的静态语义。XYZ/E 的一种基本命令形式(叫作条件元或简称 c. e.)是:

$$LB = y \wedge R \Rightarrow @\ (Q \wedge LB = z) \qquad (1)$$

式中 y 和 z 分别称为一条件元(c. e.)中的定义标号及转出标号。“ @ ”代表 $ O(下一时刻)或是◇(最终时刻);“ R ”Q ”为一阶逻辑公式,分别称为 c. e. 条件部分与动作部分;“ ⇒ ”是蕴涵词。

由于以图形表示程序可充分发挥二维显示终端的特色,增强直观性,而 XYZ/E 语言又正好便于表示图形的语义。所以对于 XYZ/E 有一与其同构的图形——XYZ 图,它可以说是 XYZ/E 时序逻辑语言的图形表示[4]。(1)式可以用图 1 表示:

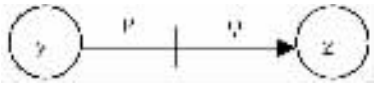

图 1 XYZ 图

图中的○对应于某个控制标号所代表的状态,——►表示有条件的转移边,条件或动作均可为空。

XYZ/E 按照不同的控制形式又分为 3 种子语言,分别是 XYZ/BE(基本 XYZ/E)、XYZ/SE(结构化 XYZ/E)、XYZ/PE(产生式型 XYZ/E)。当在进行实时系统的程序设计时,根据此系统特征,又将子语言 XYZ/BE、XYZ/SE 进行实时性扩充而成为实时 XYZ/E 的相应子语言 XYZ/RBE 及 XYZ/RSE。本文第 3 部分将利用 XYZ/RBE 语言和 XYZ 图对一个实时系统进行描述,下面给出此语言的语法规则。

将 XYZ/E 的时序算子 $ O,◇,□,$ U,$ W 加以扩充,使之表示出这些算子的实时下限(即 l)与上限(即 u)信息。这里所谓算子的实时下限是指该算子从可能执行到该算子实际执行开始所需要的最短时间;所谓算子的实时上限是指该算子从可能执行到该算子全过程实际执行完毕所允许的最长时间,故相应的实时算子具有如下的形式:$ O {l,u},◇ {l,u},□ {l,u},$ U {l,u},$ W {l,u}。令@ {l,u}表示上述前 3 种实时时序算子中任一种,则任一发如下的包含实时时序算子@ {l,u}的条件元:

$$LB = y \wedge R \Rightarrow @\ \{l,u\}\ (Q \wedge LB = z)$$

3 实时系统的一个实例

一个媒气炉装置[5] 如图 2 所示,由以下几个部分组成:控制器(controller)、气流传感器(gasflow sensor)、火焰探测器(flame detector)、点火转换器(ignition transformer)、控制开关(switch)。

控制器是整个煤气炉装置的中心,它与气流传感器与火焰探测器、点火转换器相连。控制器依据传感器传来的数据来决定要关闭或释放煤气。例如,如果气流传感器检测到煤气已经从煤气炉上的喷嘴里溢出,而火焰探测器此时又探测到没有火焰,此时说明已经发生煤气泄露,应该将煤气炉控制开关关闭,停止输出煤气,等待一段时间后去重新输出煤气重新点燃火焰。

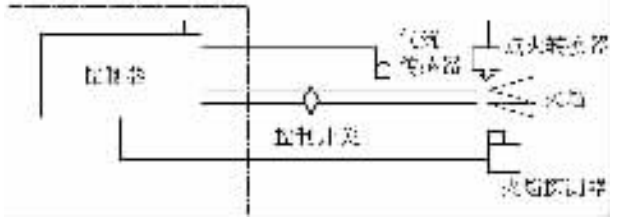

图 2 煤气炉装置的结构图

首先将整个系统划分成 5 个进程分别用于描述火焰探测器 flamede ,气流传感器 gasflowse,点火转换器 ignition,控制开关 switch,整个煤气炉的控制软件 controller。用 XYZ 图将这几个进程描述如下:

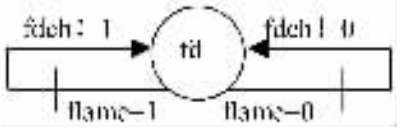

图 3 进程 flamede 的 XYZ 图

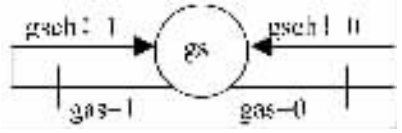

图 4 进程 gasflowse 的 XYZ 图

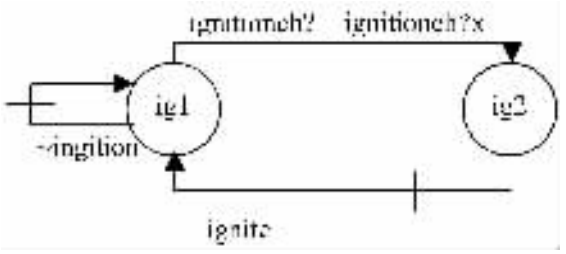

图 5 进程 ignition 的 XYZ 图

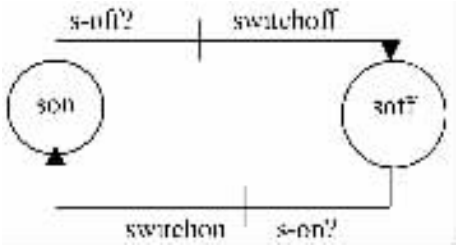

图 6　进程 switch 的 XYZ 图

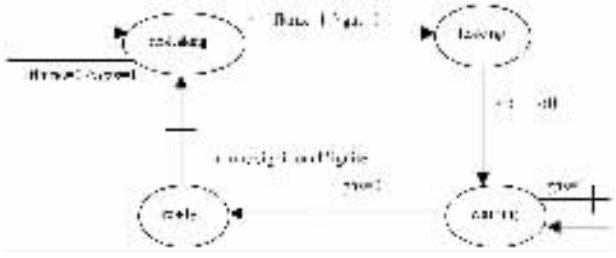

图 7　进程 controller 的 XYZ 图

用 XYZ/RBE 将上列图形转换为程序:

```
% OBPROG gasburner = = □[
% PROC[
    flamede(% CHN fdch(*,controller :NM) :INT)
= = [
        % ALG[
          LB = START-flamede⇒ $ O {0,0} (flame
= 1 ∧ LB = fd1);
          LB = fd1 ∧ flame = 1⇒ $ O {0,ts} (fdch !
1 ∧ LB = fd1);//输出 1 表示有火焰
          LB = fd1 ∧ flame = 0⇒ $ O {0,ts} (fdch !
0 ∧ LB = fd1);//输出 0 表示无火焰
            …] //ts 为输出一个值所需要的最长
时间,下同
          ]
    gasflowse (% CHN gsch (*, controller :NM) :
INT) = = [
        % ALG[
          LB = START-gasflowse⇒ $ O {0,0} (gas
= 0 ∧ LB = gs1);
          LB = gs1 ∧ gas = 0⇒ $ O {0,ts} (gsch ! 0
∧ LB = gs1);//输出 0 表示无煤气泄露
          LB = gs1 ∧ gas = 1⇒ $ O {0,ts} (gsch ! 1
∧ LB = gs1);//输出 1 表示有煤气泄露
            …]]
    ignition(% CHN ignitionch(controller,* :NM) :
INT) = = [
        % LOC[x :INT];
        % ALG[
          LB = START-ignit ⇒ $ O {0,0}
(LB = ig1);
          LB = ig1 ∧ ~ ignitionch ? ⇒ $ O {0,0}
(LB = ig1 | LB = NEXT);
          LB = ig1 ∧ ignitionch ? ⇒ $ O {0,tr}
(ignitionch ? x ∧ LB = ig2);//tr 为输入一
          LB = ig2⇒ $ O {0,td} (ignite ∧ LB =
ig1);个值所需要的最长时间,下同
          …]]//点火动作由 ignite 代替,td 为
执行动作 ignite 所需要的最长时间
    switch(% CHN s-off(controller,* :NM) :INT;
          % CHN s-on(controller,* :NM) :INT) =
= [
          % LOC[on :INT ;off :INT];
          % ALG[
            LB = START-switch ⇒ $ O {0,0}
(LB = son ∧ on = 1 ∧ off = 0);
          LB = son ∧ s-off ? ⇒ $ O {0,tr} (s-off ?
off ∧ LB = s1);
          LB = s1 ∧ off = 1⇒ $ O {0,tm} (swit-
choff ∧ LB = s2);//tm 为执行打开和关闭
          LB = s2 ∧ off = 1⇒ $ O {0,0} (on =
0 ∧ LB = soff) 操作所需要的最长时间
          LB = soff ∧ s-on ? ⇒ $ O {0,tr} (s-on ?
on ∧ LB = s3);
          LB = s3 ∧ on = 1⇒ $ O {0,tm} (switch-
on ∧ LB = s4);
          LB = s4 ∧ on = 1⇒ $ O {0,0} (off =
0 ∧ LB = son);// ON 和 OFF 为 1 时表示执行相应
的打开和关闭操作
          ]]
    controller (% CHN fdch(flamede,* :NM) :INT;
          % CHN gsch(gasflowse,* :NM) :INT;
          % CHN ignitionch(*,igition :NM) :
INT;
          % CHN s-off(*,switch :NM) :INT;
          % CHN s-on(*,switch :NM) :INT) =
= [
          % LOC[on :INT ;off :INT ;x :INT]
          % ALG[
          LB = START-controller⇒ $ O {0,0} (
flame = 1 ∧ gas = 0 ∧ LB = no_leaking);
```

```
        LB = no_leaking⇒ $ O {0,tr} (fdch ? flame ∧ gsch ? gas ∧ LB = c1) ;
        LB = c1 ∧ ~ (flame = 1 ∧ gas = 0) ⇒ $ O {0,0} (LB = no_leaking | LB = leaking) ;
        LB = leaking⇒ $ O {0,ts} (s-off ! off ∧ LB = waiting) ;
        LB = waiting⇒ $ O {0,tr} (gsch ? gas ∧ LB = c2) ;
        LB = c2 ∧ ~ (gas = 1) ⇒ $ O {0,0} ( LB = ready | LB = waiting) ;
        LB = ready⇒ $ O {0,ts} (s-on ! on ∧ ignitionch ! igite ∧ LB = c3) ;
        LB = c3⇒ $ O {0,0} (LB = no_leaking) ;
        ]]
% ALG[
    LB = START⇒ $ O {0,0} (fd = = flamede {1} ∧ gs = = gasflowse {1} ∧ ig = = ignition {1} ∧ sw = = switch {1} ∧ c = = controller {1} ∧ LB = L1) ;
    LB = L1⇒ $ O {0,tm} [fd (% CHN fdch1 (*, c) | fdch (*, controller)) ;
    gs (% CHN gsch1 (*, c) | gsch (*, controller)) ;
    ig (% CHN ignitionch1 (c, *) | ignitionch (controller, *)) ;
    sw (% CHN s-off1 (c, *) | s-off (controller, *) ;
    % CHN s-on1 (c, *) | s-on (controller, *)) ;
    c (% CHN fdch1 (fd, *) | fdch (flamede, *) ;
    % CHN gsch1 (gs, *) | gsch (gasflowse, *) ;
    % CHN ignitionch1 (*, ig) | ignitionch (*, ignition) ;
    % CHN s-off1 (*, sw) | s-off (*, switch) ;
    % CHN s-on1 (*, sw) | s-on (*, switch) ;]
    ]]
```

4 结语

至此用 XYZ/RBE 语言完成了对此实例的描述,由此可以看出,将 XYZ/E[6] 的子语言 XYZ/RBE 和 XYZ 图结合用来描述实时系统控制问题时,它们具有状态转换清晰、问题描述方便、易于理解等优点。只是 XYZ/RBE 在表示循环结构时不太清晰,需用多条语句才能表示此结构,但这种结构 XYZ/E 语言族中的另外一种子语言 XYZ/RSE 可以清晰表示,由此可以看出 XYZ/E 在实时系统程序设计方面的适用性和应用前景。

为了保证实时控制程序的正确性,下一步工作将利用迄今关于程序正确性验证最为完整并为计算机科学界普遍肯定的方法——Hoare 逻辑[3] 来验证本文所给程序的安全性和活性。

参考文献:

[1] 侯建民,李宣东,郑国梁. 实时系统软件开发过程中形式方法的作用[J]. 计算机应用与软件,2002,21(4):23-26.

[2] 张广泉,孙敏. 时态逻辑的比较与分析[J]. 渝州大学学报(自然科学版). 1999,16(2):15-18.

[3] 唐稚松. 时序逻辑程序设计与软件工程[M]. 北京:科学出版社,2002.

[4] 龚洁,唐若鹰,王霄,等. XYZ/CFC 与 XYZ/PAD 图形-文本程序设计环境[J]. 软件学报,1994,5(8):37-46.

[5] HANSEN K M, RAVN A P, RISCHEL H. Specifying and Verifying Requirements of Realtime Systems [J]. IEEE Transactions on Software Engineering, 1993, 19(1): 41-55.

[6] 张广泉,郑建丹,舒明. 基于时序逻辑语言 XYZ/E 软件体系结构研究(Ⅰ)[J]. 重庆师范学院学报(自然科学版),2001,18(3):1-4.

(责任编辑 游中胜)