# 三机冗余容错系统的描述和验证*

郭 亮, 唐稚松

(中国科学院 软件研究所 计算机科学重点研究实验室,北京 100080)

# Specification and Verification of the Triple-Modular Redundancy Fault-Tolerant System

GUO Liang+, TANG Zhi-Song

(Key Laboratory of Computer Science, Institute of Software, The Chinese Academy of Sciences, Beijing 100080, China)

+ Corresponding author: Phn: 86-10-62562796, Fax: 86-10-62562533, E-mail: gls@ios.ac.cn

http://lcs.ios.ac.cn

Received 2001-07-32; Accepted 2002-04-10

**Guo L, Tang ZS. Specification and verification of the triple-modular redundancy fault-tolerant system. *Journal of Software*, 2003,14(1):54~61.**

**Abstract**: XYZ/E is used to specify and verify the triple-modular redundancy fault-tolerant system. Assuming that each computer is loaded with a determined sequential program $P$ which continuously outputs data to the outer environment, the case $P$ running on single processor is illustrated by an XYZ/E program $SingleProcess_P$, and the property of program $P$ is specified by a temporal logical formula $Spec_P$. Finally, it is proved that the program $TripleProcessors_P$ obtained from the triple-modular redundancy way can still satisfy $Spec_P$ in spite of hardware errors.

**Key words**: temporal logic language XYZ/E; fault-tolerant system; triple-modular redundancy; specification; verification

**摘 要**: 使用 XYZ/E 描述和验证三机冗余容错系统.考虑每台计算机加载了一个不断向外界环境输出数据的确定性顺序程序 $P$,用 XYZ/E 程序 $SingleProcessor_P$ 刻画程序 $P$ 在单机上运行,用时序逻辑式 $Spec_P$ 刻画 $P$ 向外部环境输出的数据所满足的性质.最后证明,采用三机冗余模式所得到的程序 $TripleProcessors_P$ 即使在出现硬件错误的情况下运行,也能满足性质 $Spec_P$.

**关键词**: 时序逻辑语言 XYZ/E;容错系统;三机冗余;描述;验证

**中图法分类号**: TP311 **文献标识码**: A

容错系统(fault-tolerant system)是一种即使有错误发生,也能满足其规范的系统.常用的程序容错方法有海明码、多机冗余、断点设置、恢复这几种.将形式化理论应用于容错系统[1~5]是当前容错系统研究工作的一个重要发展方向.本文着重讨论了如何使用 XYZ/E 描述三机冗余容错系统并验证其容错性(fault-tolerance).

* Supported by the National Natural Science Foundation of China under Grant No.60073020 (国家自然科学基金); the National High Technology Development 863 Program of China under Grant No.863-306-ZT02-04-01 (国家 863 高科技发展计划)

**第一作者简介**: 郭亮(1976－),男,江西吉安人,博士,主要研究领域为软件工程.

三机冗余(triple-modular redundancy)是一种通用的容错方法,其思想核心为在使用 3 台相同的机器的同时运行相同的程序,由于两台机器硬件同时出错的概率趋近于 0,因此 3 台机器可通过交互和表决,检测并修复出错的机器,使程序正常运行.三机冗余容错系统一般假定任意时刻最多只有一台机器硬件出错,而且错误不会发生在机器交互、表决和修复过程中.

文献[3]基于 TLA[6,7](temporal logic of actions)描述并验证了一个简单的三机冗余容错系统的容错性.单机系统可抽象为由一个处理器和一块内存组成,处理器通过发出读/写命令来读/写内存数据.机器硬件可能因出错而非法修改内存数据,使得处理器从内存读出错误值.为了使系统具有容错性,在硬件出错的情况下也能使处理器对内存进行正确的读、写操作,可以使用 3 块内存.当处理器向内存发出写命令时,将 3 块内存存储的数据设置为相同值;当处理器向内存发出读命令时,3 块内存存储的数据通过表决,返回表决结果给处理器.在文献[3]中,硬件错误被建模为 TLA 中的动作(action),整个错误环境(fault environment)存在一个错误假定(fault assumption),即 3 块内存不会同时出错.系统的容错性定义为三机冗余系统在错误环境下的错误影响程序(fault-affected program)是单机系统的求精,容错性在定义两个程序之间的求精映射(refinement mapping)的基础上得以证明.

在本文中,我们考虑一种相对复杂的情况.假定单机系统同样由一个处理器和一块内存组成,但内存中装载了一个确定性的顺序程序 $P$.系统在运行时的任一时刻,或者处理器执行程序 $P$ 的一条指令,以修改 $P$ 要处理的变量的值,或者向外界环境输出这些变量的当前值.针对这种情况,我们首先分析 $P$ 的行为,然后用 XYZ/E[8,9]程序 $SingleProcessor_P$ 刻画装载了程序 $P$ 的单机系统的行为,用时序逻辑式 $Spec_P$ 抽象出单机系统向外部环境输出的数据所满足的性质.由于硬件存在不可靠性,可能因发生错误而导致数据出错,此时性质 $Spec_P$ 不能满足.因此,需要考虑采用 3 台相同的机器同时运行程序 $P$.在输出数据时,3 台机器通过交互和表决来检测及修复错误.对于这样的三机冗余系统,我们可以用一个 XYZ/E 程序 $TripleProcessor_P$ 来刻画,其错误环境 $F$ 被建模为一个状态转换集合,程序的容错性可以通过证明程序 $TripleProcessor_P$ 在错误环境 $F$ 下运行时的错误影响版本程序 $TripleProcessor_{P\text{-}F}$ 仍能满足性质 $Spec_P$ 而得到.

## 1 XYZ/E 简介及对硬件错误环境建模

XYZ/E 是一种基于 Manna-Pnueli 线性时序逻辑的线性时序逻辑语言(LTLL).其最基本的元素是式(1)这种形式的可执行条件元(conditional element),它直接定义了相邻状态之间的转换关系.因此,全部由这种形式条件元组成的程序可以执行.

$$LB=y\wedge R\Rightarrow \$O(v_1,v_2,\ldots,v_n)=(e_1,e_2,\ldots,e_n)\wedge \$OLB=z. \tag{1}$$

XYZ/E 中引入了式(2)这种形式的选择语句来表示不确定性.若选择语句在某时刻存在多个分支的条件部分同时为真,则程序将在这些分支间作出不确定选择.具有相同标号且具有式(1)这种形式的一组条件元也可以用一个带相同分支数目的选择语句来表示.

$$LB=y\wedge R\Rightarrow !![Cond_1|>ExeAct_1,\ldots,Cond_k|>ExeAct_k]. \tag{2}$$

XYZ/E 中表示算法的构件为如式(3)所示的单元(unit).其中 $A_1,A_2,...,A_n$ 是条件元(选择语句),符号“;”等同于逻辑联结词合取.若单元中所有条件元都可执行,则单元可执行.

$$\Box[A_1;A_2;\ldots;A_n] \tag{3}$$

XYZ/E 可执行程序 $P$ 由一组变量集合 $Var_P$、初始条件 $Init_P$ 和一个可执行单元 $Unit_P$ 三部分组成.此外,当程序 $P$ 中出现不确定选择语句时,我们可能会希望对 $P$ 所作的选择进行约束,使得 $P$ 满足某性质 $Where_P$,此时,程序 $P$ 的语义对应于时序逻辑式 $LF_P$,

$$LF_P=_{\text{def}} Init_P\wedge Unit_P\wedge Where_P. \tag{4}$$

对于容错系统,需要刻画清楚程序 $P$ 运行所在的硬件环境可能发生的错误以及各种错误对程序运行产生的动态影响,即对错误环境(fault environment)$F$ 进行建模.在文献[3~5]中,每种可能发生的硬件错误都被建模为一个动作(action),整个错误环境 $F$ 则被建模为一个错误动作的集合.通过错误转换(fault transformation),可以给出程序 $P$ 的错误影响程序(fault-affected program)$F(P)$,$P$ 在 $F$ 下的运行等价于程序 $F(P)$在无错环境下的运行.

本文中,我们将 XYZ/E 程序 $P$ 运行所在的错误环境 $F$ 中每种可能发生的错误建模为一个形式为 $FCond \wedge \$OFExeAct$ 的状态转换,$FCond$ 和 $\$OFExeAct$ 分别表示转换的使能条件部分和动作部分,$F$ 则被建模为一个状态转换集合$\{F_1,F_2,\ldots,F_m\}$,$F$ 必须满足的错误假定可用一个时序逻辑式 $FaultAssumption_F$ 来刻画.$F$ 建模完毕后,设程序 $P$ 的可执行单元 $Unit_P=\Box[A_1;A_2;\ldots;A_n]$,其中每个 $A_i$ 都是一个选择语句,则与程序 $P$ 在 $F$ 下运行等价的错误影响程序 $P_F$ 有

$$\left.\begin{aligned} &Init_{PF}=Init_P,\\ &Var_{PF}=Var_F,\\ &Unit_{PF}=\Box[A_1\oplus F;A_2\oplus F;\ldots;A_n\oplus F],\\ &Where_{PF}=Where_P\wedge FaultAssumption_F. \end{aligned}\right\} \tag{5}$$

设 $A$ 为式(2)形式的选择语句,则等式(5)中形式为 $A\oplus F$ 的选择语句定义如下:

$$\begin{aligned} A\oplus F =_{\text{def}}\ & LB=y\wedge R \Rightarrow !![\\ & Cond_1 \rhd ExeAct_1,\ldots,Cond_k \rhd ExeAct_k,\\ & FCond_1 \rhd FExeAct_1,\ldots,FCond_m \rhd FExeAct_m\\ & ] \end{aligned} \tag{6}$$

## 2 刻画单机系统

单机系统可以看做是由一个处理器和一块内存组成,内存中装载了一个确定的顺序可执行程序 $P$(由代码段和数据段两部分组成).系统在运行的任一时刻,或者处理器执行程序 $P$ 的一条代码(由程序 $P$ 的代码段内容确定),以修改程序 $P$ 的数据段内容,或者处理器向外界环境输出这些变量的当前值.由于程序 $P$ 是确定性程序,由系统当前要执行的代码以及数据段的内容,即可得到系统下一时刻要执行的代码和数据段的内容.

因此,单机系统可以被抽象地看作存在某值域 $VD$ 上取值的两个变量:变量 $d$(对应程序 $P$ 数据段的内容和处理器将要执行的程序 $P$ 的下一条代码)和变量 $val$(当处理器向环境输出数据时,将读出变量 $d$ 的值,赋给 $val.val$ 可被外部环境看到).当然,也可以假定 $val$ 在另一值域 $VV$ 上取值,且存在一个函数 $f:VD\to VV$,有 $val=f(d)$.后面我们针对系统向环境输出数据时 $val=d$ 情况下的三机冗余系统容错性证明思路,也同样适合于这种通用情况.

由于 $P$ 是确定程序,则由 $P$ 可惟一确定一个函数 $fun_P^*:VD\to VD$,使得系统每执行程序 $P$ 的一条指令就把 $d$ 的值赋为 $fun_P(d)$.此外,系统只可能在执行了 $P$ 的关键指令或者 $P$ 中的数据满足特定断言的时候,才向外界环境输出数据.这些指令和断言由 $P$ 惟一确定,因此可以假定存在断言 $needWrite_P(d)$,系统向环境输出数据,当且仅当断言 $needWrite_P(d)$ 成立.系统在运行时将检查 $d$ 的值是否符合断言 $needWrite_P(d)$,若符合,则将 $d$ 的值赋给 $val$;否则,执行程序 $P$ 的下一条指令.

由上所述,单机系统的行为可用如下的 XYZ/E 程序 $SingleProcessor_P$ 来刻画.这里,我们引入了两个整数型变量 $n$ 和 $k$**,$n$ 表示处理器已经执行的程序 $P$ 的代码条数,$k$ 表示系统最近一次向环境输出数据时 $n$ 的值.程序初始条件满足 $d$ 和 $val$ 都等于同一固定值 $v_0\in VD$,且 $needWrite_P(v_0)$ 为真.此外,我们把选择语句中的两个选择分量分别命名为 $t\_cal$ 和 $t\_write$,$t\_cal$,对应于处理器执行程序 $P$ 的一条指令;$t\_write$ 对应于处理器输出程序 $P$ 的计算结果.

$\{LB=lb\_running\wedge val=d=v_0\wedge n=k=0\wedge needWrite_P(v_0)\}$ // $Init_{SingleProcessor_P}$

$SingleProcessor_P==[]$ [

---

* 每个程序对应于一个三元组$(V,\theta,\Gamma)$,$V=(v_1,v_2,\ldots,v_n)$表示变量集合,$\theta$表示变量初始条件,$\Gamma=\{\tau_1,\tau_2,\ldots,\tau_m\}$为一个有限状态转换集合,其中每个元素 $\tau_i(i\in 1,\ldots,m)$定义了下一时刻状态和当前状态各变量取值之间的函数关系,形式为 $P_i(v_1,v_2,\ldots,v_n)\to\$O(v_1,v_2,\ldots,v_n)=e_i(v_1,v_2,\ldots,v_n)$,$P_i$ 是谓词,$e_i$ 是表达式,且任一时刻存在且仅存在一个 $i(i\in 1,\ldots,m)$,有 $P_i$ 为真.因此,函数 $fun_P$ 可定义如下:

$fun_P(v_1,v_2,\ldots,v_n)=_{\text{def}}\{e_1(v_1,v_2,\ldots,v_n)$ if $P_1(v_1,v_2,\ldots,v_n),e_2(v_1,v_2,\ldots,v_n)$ if $P_2(v_1,v_2,\ldots,v_n),\ldots,e_m(v_1,v_2,\ldots,v_n)$ if $P_m(v_1,v_2,\ldots,v_n)\}$.

** 引入变量 $n,k$ 的目的在于使处理器向外界环境输出数据后能继续执行程序 $P$ 的代码,而不会由于输出语句 $t\_cal$ 不修改变量 $d$ 的值,从而导致因断言 $needWrite_P(d)$ 一直满足而始终处于输出状态.

$LB=lb\_running \Rightarrow !!$ [

$(k \geq n) \vee \neg needWrite_P(d) |> \$O d=fun_P(d) \wedge \$O n=n+1 \wedge \$O lb=lb\_running$, // $t\_cal$

$(k<n) \wedge needWrite_P(d) |> \$O val=d \wedge \$O k=n \wedge \$O lb=lb\_running$ // $t\_write$

]

]

我们可以证明程序 $SingleProcessor_P$ 满足如下定义的性质 $Spec_P$,其中 $fun_P^k$ 表示函数 $fun_P$ 的第 $k$ 次迭代且 $fun_P^0(x)=x$.这可以通过证明如下定义的逻辑式 $Inv$ 为程序的不变量得到.

$$Spec_P =_{\text{def}} \Box(needWrite_P(val) \wedge val=fun_P^k(v_0)),$$

$$Inv =_{\text{def}} d=fun_P^n(v_0) \wedge needWrite_P(val) \wedge val=fun_P^k(v_0).$$

由于硬件存在不可靠性,系统运行时可能因发生硬件错误而修改变量 $d$ 的值,使性质 $Spec_P$ 不能得到满足.这就需要采取三机冗余的模式,对此错误进行容忍及恢复.

## 3 刻画无错环境下的三机冗余系统

现在假定机器在可能出错的硬件环境下运行.当硬件出现错误时,将修改变量 $d$ 的值.为使在错误发生的情况下,整个系统还能输出正确的数据,我们采用三机冗余的方式来修正错误,让 3 台机器同时运行同样的程序,并增加一个表决组件.在需要输出的时候,表决组件将运行,对各台机器输出给此表决组件的值进行表决,决定输出给外界环境的正确值及定位,并修复运行出错的机器.表决过程的有效性是建立在两台机器同时出错的概率为 0 的假定上的.

假定 3 台机器输出的数据分别为 $x,y,z$(包括各台机器的变量 $d$ 和 $n$ 的值),则表决后输出的数值可由如下定义的函数 $vote$ 来表示.其直观意义就是,在表决时刻,3 台机器输出的结果如果有两个相等,则表决结果为此等值;否则,等于一个无定义值$\perp$.其中,$\perp \notin Z$且$\perp \notin VD$,后面我们将证明这种情况不可能出现.

$$vote(x,y,z) =_{\text{def}} \{=x \text{ if } x=y \text{ or } x=z,\ =y \text{ else if } y=z,\ =\perp \text{ else}\}.$$

整个三机冗余系统在无错环境下的运行可用 XYZ/E 程序 $TripleProcessors_P$ 来刻画.其中,语句 $t_cal_i(i=1,2,3)$与第 $i$ 台机器的 $t\_cal$ 计算语句对应,语句 $t\_vote$ 是表决输出语句(这里假定各台机器以切换方式执行计算指令语句和表决输出语句).变量 $d_i$ 和 $n_i$ 分别表示第 $i(i=1,2,3)$台机器所执行程序的内部变量的值和已经执行的指令数.

$\{LB=lb\_running \wedge d_1=d_2=d_3=val=v_0 \wedge k=n_1=n_2=n_3=0 \wedge needWrite_P(v_0)\}$ // $Init_{TripleProcessors_P}$

$TripleProcessors_P$==[] [

$LB=lb\_running \Rightarrow !!$ [

$(k \geq n_1) \vee \neg needWrite_P(d_1) |> \$O d_1=fun_P(d_1) \wedge \$O n_1=n_1+1 \wedge \$O lb=lb\_running$, // $t\_cal_1$

$(k \geq n_2) \vee \neg needWrite_P(d_2) |> \$O d_2=fun_P(d_2) \wedge \$O n_2=n_2+1 \wedge \$O lb=lb\_running$, // $t\_cal_2$

$(k \geq n_3) \vee \neg needWrite_P(d_3) |> \$O d_3=fun_P(d_3) \wedge \$O n_3=n_3+1 \wedge \$O lb=lb\_running$, // $t\_cal_3$

$(k<n_1) \wedge (k<n_2) \wedge (k<n_3) \wedge needWrite_P(d_1) \wedge needWrite_P(d_2) \wedge needWrite_P(d_3) |>$ // $t\_vote$

$\$O val=\$O d_1=\$O d_2=\$O d_3=vote(d_1,d_2,d_3) \wedge \$O k=\$O n_1=\$O n_2=\$O n_3=vote(n_1,n_2,n_3) \wedge \$O lb=lb\_running$

]

]

## 4 错误环境建模和错误影响程序

要证明程序 $TripleProcessors_P$ 是一个容错系统,也即在错误环境下运行时仍满足性质 $Spec_P$,需要先对此程序所在的硬件错误环境建模.现在假定程序 $TripleProcessors_P$ 中存在 3 个错误指示变量 $f_1,f_2$ 和 $f_3,f_i(i=1,2,3)$为真,表示对应的处理器 $i$ 运行出错,则错误环境 $F$ 可被建模为状态转换集合$\{F_1,F_2,F_3\}$.其中,$F_i$ 表示当错误发生时,将修改第 $i$ 台机器变量 $d_i$ 的值,并将错误指示变量 $f_i$ 置为真,其定义如下:

$$F_i=\$T\wedge\$Of_i=\$T\wedge\$Od_i\in VD. \quad (7)$$

在程序开始运行时,假定一切正常,也即所有错误指示变量都为$F,则在错误环境$F$建模以后,我们可以得到错误影响程序等价 $TripleProcessors_{P\text{-}F}$ 如下:

$\{LB=lb\_running\wedge d_1=d_2=d_3=val=v_0\wedge k=n_1=n_2=n_3=0\wedge f_1=f_2=f_3=\$F\wedge needWrite_P(v_0)\}$ // $Init_{TripleProcessors_{P\text{-}F}}$

$TripleProcessors_{P\text{-}F}$==[] [

$LB=lb\_running\Rightarrow$!! [

$(k\geq n_1)\vee\neg needWrite_P(d_1)|>\$Od_1=fun_P(d_1)\wedge\$On_1=n_1+1\wedge\$Olb=lb\_running$, // $t\_cal_1$

$(k\geq n_2)\vee\neg needWrite_P(d_2)|>\$Od_2=fun_P(d_2)\wedge\$On_2=n_2+1\wedge\$Olb=lb\_running$, // $t\_cal_2$

$(k\geq n_3)\vee\neg needWrite_P(d_3)|>\$Od_3=fun_P(d_3)\wedge\$On_3=n_3+1\wedge\$Olb=lb\_running$, // $t\_cal_3$

$(k<n_1)\wedge(k<n_2)\wedge(k<n_3)\wedge needWrite_P(d_1)\wedge needWrite_P(d_2)\wedge needWrite_P(d_3)|>$

$\$Oval=\$Od_1=\$Od_2=\$Od_3=vote(d_1,d_2,d_3)\wedge\$Ok=\$On_1=\$On_2=\$On_3=vote(n_1,n_2,n_3)\wedge$

$\$Of_1=\$Of_2=\$Of_3=\$F\wedge\$Olb=lb\_running$ // $t\_vote$,在表决的时候修复错误

$\$T|>\$Of_1=\$T\wedge\$Od_1\in VD\wedge\$Olb=lb\_running$, // $t\_err_1$

$\$T|>\$Of_2=\$T\wedge\$Od_2\in VD\wedge\$Olb=lb\_running$, // $t\_err_2$

$\$T|>\$Of_3=\$T\wedge\$Od_3\in VD\wedge\$Olb=lb\_running$ // $t\_err_3$

]

] where $\Box(\neg((f_1\wedge f_2)\vee(f_1\wedge f_3)\vee(f_2\wedge f_3)))$

下一节我们将证明三机冗余系统的容错性,即程序 $TripleProcessors_{P\text{-}F}$ 满足性质 $Spec_P$.

## 5 容错性证明

要证明程序 $TripleProcessors_{P\text{-}F}$ 满足性质 $Spec_P$,关键在于证明在执行表决语句时不可能出现 $d_1,d_2,d_3$ 的值两两不等的情况.虽然这个命题比较直观,但若要严格证明则需要一定的技巧.

我们将通过证明一系列命题的正确性来证明系统容错性.在证明之前,我们需要先给出一些辅助谓词:$NormalValues$,$Envote$,$A_{i,j}$ 和 $B_{i,j}(i,j=1,2,3,i\neq j)$,$A$,$B$,$C$ 和 $C_i(i=1,2,3)$的定义.

**定义**.

$NormalValues=_{\text{def}} k\neq\perp\wedge n_1\neq\perp\wedge n_2\neq\perp\wedge n_3\neq\perp\wedge val\neq\perp\wedge d_1\neq\perp\wedge d_2\neq\perp\wedge d_3\neq\perp$,

$Envote(kx,nx,dx)=_{\text{def}} (kx<nx)\wedge needWrite_P(dx)$,

$A_{i,j}=_{\text{def}} (\neg f_i\wedge\neg f_j\wedge n_i\leq n_j)\rightarrow(d_j=fun_P^{n_j-n_i}(d_i)\wedge\forall_m:((n_i\leq m\wedge m<n_j)\rightarrow\neg Envote(k,n_i,fun_P^{m-n_i}(d_i))))$,

$B_{i,j}=_{\text{def}} ((\neg f_i\wedge\neg f_j\wedge n_i\leq n_j)\wedge Envote(k,n_i,d_i)\wedge Envote(k,n_j,d_j))\rightarrow(d_i=d_j\wedge n_i=n_j)$,

$A=_{\text{def}}\Lambda_{i,j=1,2,3\ i\neq j}:A_{i,j}$, $B=_{\text{def}}\Lambda_{i,j=1,2,3\ i\neq j}:B_{i,j}$, $C_i=_{\text{def}}(\neg f_i\rightarrow(d_i=fun_P^{n_i}(v_0)))$, $C=_{\text{def}}\Lambda_{i=1,2,3}:C_i$.

在程序运行的任一时刻,谓词 $NormalValues$ 为真,当且仅当各变量值有效(即不等于$\perp$).如果当前系统 $k$ 值为 $kx$ 且机器 $i$ 的 $n_i$ 值为 $nx$,$d_i$ 值为 $dx$,则谓词 $Envote(kx,nx,dx)$为真,当且仅当机器 $i$ 处于等待表决状态(机器已停止计算,正准备输出数据).谓词 $A_{i,j}$ 的直观含义为:如果机器 $i$ 和 $j$ 都运行正常,且机器 $i$ 执行的代码条数 $n_i$ 不大于机器 $j$ 执行的代码条数 $n_j$,则 $d_j=fun_P^{n_j-n_i}(d_i)$,且机器 $i$ 从当前状态运行任意小于 $n_j-n_i$ 的步数以后(若 $n_i<n_j$,则包括当前状态)都不可能处于等待表决状态.谓词 $B_{i,j}$ 的直观含义为:如果机器 $i$ 和 $j$ 运行都没有出错,且两台机器都处于等待表决状态,则 $n_i=n_j$ 且 $d_i=d_j$.谓词 $C_i$ 的直观含义为:如果机器 $i$ 当前运行正常,则 $d_i=fun_P^{n_i}(v_0)$.

**命题 1**. $(NormalValues\wedge A_{i,j})\rightarrow(NormalValues\wedge B_{i,j})$.

整个证明过程由以下推理过程得到.这里,使用谓词 $NormalValues$ 是使变量之间的比较以及函数 $fun_P$ 有意义.下面的推理过程的正确性都是建立在谓词 $NormalValues$ 成立的前提下的.在每个逻辑式的后面,我们都加上了注释,说明其推导过程.

证明:

(1) $A_{i,j}\wedge(\neg f_i\wedge\neg f_j\wedge n_i\leq n_j)\wedge Envote(k,n_i,d_i)\wedge Envote(k,n_j,d_j)\rightarrow\forall_m:((n_i\leq m\wedge m<n_j)\rightarrow\neg Envote(k,n_i,fun_P^{m-n_i}(d_i)))$.

//由 $A_{i,j}$ 定义得到

(2) $A_{i,j}\wedge(\neg f_i\wedge\neg f_j\wedge n_i\leq n_j)\wedge Envote(k,n_i,d_i)\wedge Envote(k,n_j,d_j)\wedge(n_i<n_j)\rightarrow\neg Envote(k,n_j,d_j)$. //将 $m=n_i$ 代入(1)得到

(3) $A_{i,j}\wedge(\neg f_i\wedge\neg f_j\wedge n_i\leq n_j)\wedge Envote(k,n_i,d_i)\wedge Envote(k,n_j,d_j)\rightarrow n_i=n_j$. //由(2)及永真式

// $((A\wedge B\wedge C)\rightarrow\neg A)\leftrightarrow((A\wedge C)\rightarrow\neg B)$直接得到

(4) $A_{i,j}\wedge(\neg f_i\wedge\neg f_j\wedge n_i\leq n_j)\wedge Envote(k,n_i,d_i)\wedge Envote(k,n_j,d_j)\wedge n_i=n_j\rightarrow d_i=d_j$.

//将 $n_i=n_j$ 带入 $A_{i,j}$ 中可以得到 $d_j=fun_P^{n_j-n_i}(d_i)=d_i$,再由(3)可以得到

(5) $A_{i,j}\wedge(\neg f_i\wedge\neg f_j\wedge n_i\leq n_j)\wedge Envote(k,n_i,d_i)\wedge Envote(k,n_j,d_j)\rightarrow(d_i=d_j\wedge n_i=n_j)$. //由(3)、(4)得到

(6) $A_{i,j}\rightarrow B_{i,j}$. //由(5)得到 □

在后面的证明中,我们要用到形如$\{P\}\tau\{Q\}$的 Hoare 三元组.其含义为:若程序当前状态满足断言 $P$ 且语句$\tau$的使能条件满足,则执行语句$\tau$以后,程序状态满足断言 $Q$.这是一个完全正确性的概念.如果语句$\tau$是一条赋值语句 $en(\tau)\wedge\$Ox=e,en(\tau)$为语句$\tau$的使能条件,则证明$\{P\}\tau\{Q\}$成立等价于证明逻辑式 $P\wedge en(\tau)\rightarrow Q(e/x)$成立,其中$Q(e/x)$为将 $Q$ 中所有自由变量 $x$ 用 $e(x)$代替以后得到的逻辑式.

命题 2~命题 8 用于证明谓词 $A\wedge NormalValues$ 是程序 $TripleProcessors_{P\text{-}F}$ 的不变量.

**命题 2**. $\{A\wedge NormalValues\}t\_vote\{A\wedge NormalValues\}$.

证明:

(1) $\exists_{i,j=1,2,3\ i\neq j}:(\neg f_i\wedge\neg f_j)$. //由于$\neg((f_1\wedge f_2)\vee(f_1\wedge f_3)\vee(f_2\wedge f_3))$是程序的不变量

(2) $NormalValues\wedge\neg f_i\wedge\neg f_j\wedge Envote(k,n_i,d_i)\wedge Envote(k,n_j,d_j)\rightarrow(n_i=n_j\wedge d_i=d_j)$.

//由 $A\rightarrow B_{i,j}$ 和 $B_{i,j}$ 得到,这里和下面的 $i,j$ 可为任意一对满足$\neg f_i\wedge\neg f_j$的数值

(3) $NormalValues\wedge\neg f_i\wedge\neg f_j\wedge Envote(k,n_i,d_i)\wedge Envote(k,n_j,d_j)\rightarrow(vote(d_1,d_2,d_3)\neq\perp\wedge vote(n_1,n_2,n_3)\neq\perp)$.

//由(2)及 *vote* 函数的定义得到

(4) $\{\neg f_i\wedge\neg f_j\wedge NormalValues\wedge A_{i,j}\}\ t\_vote\ \{d_1=d_2=d_3=val\wedge n_1=n_2=n_3=n\wedge NormalValues\}$.

//由(3),$A_{i,j}\rightarrow B_{i,j}$ 和 *t_vote* 的使能条件蕴涵了 $Envote(k,n_i,d_i)\wedge Envote(k,n_j,d_j)$得到

(5) $d_1=d_2=d_3=val\wedge n_1=n_2=n_3=n\wedge NormalValues\rightarrow A\wedge NormalValues$.

//由对任意 $i,j=1,2,3,i\neq j$,有$(d_i=d_j\wedge n_i=n_j)\rightarrow A_{i,j}$ 得到

(6) $\{A\wedge NormalValues\}t\_vote\{A\wedge NormalValues\}$. //由(1)、(4)和(5)得到 □

**命题 3**. 若 $i,j,k=1,2,3$,且两两不等,则$\{A_{i,j}\wedge NormalValues\}t_err_k\{A_{i,j}\wedge NormalValues\}$.

由于 $t\_err_k$ 的执行不会改变谓词 $A_{i,j}$ 中出现的任意变量,因此命题 3 显然是正确的.

**命题 4**. 若 $i,j=1,2,3$,且 $i\neq j$,则$\{A_{i,j}\wedge NormalValues\}t_err_i\{A_{i,j}\wedge NormalValues\}$.

由于执行 $t\_err_i$ 将使 $f_i$ 置为真,而 $f_i\rightarrow A_{i,j}$,因此命题 4 显然是正确的.

**命题 5**. 若 $i,j=1,2,3$,且 $i\neq j$,则$\{A_{i,j}\wedge NormalValues\}t_err_j\{A_{i,j}\wedge NormalValues\}$.

由于执行 $t\_err_j$ 将使 $f_j$ 置为真,而 $f_j\rightarrow A_{i,j}$,因此命题 5 显然是正确的.

**命题 6**. 若 $i,j,k=1,2,3$,且两两不等,则$\{A_{i,j}\wedge NormalValues\}t_cal_k\{A_{i,j}\wedge NormalValues\}$.

由于 $t\_cal_k$ 的执行不会改变谓词 $A_{i,j}$ 中出现的任意变量,因此命题 6 显然是正确的.

**命题 7**. 若 $i,j=1,2,3$,且 $i\neq j$,则$\{A_{i,j}\wedge NormalValues\}t_cal_i\{A_{i,j}\wedge NormalValues\}$.

证明:

(1) $((\neg f_i\wedge\neg f_j\wedge n_i+1\leq n_j)\wedge NormalValues\wedge A_{i,j}\rightarrow d_j=fun_P^{n_j-n_i}(d_i))$.

//由 $fun_P^{n_j-n_i}(d_i)=fun_P^{n_j-n_i-1}(fun_P(d_i))$和 $A_{i,j}$ 的定义得到

(2) $((\neg f_i\wedge\neg f_j\wedge n_i+1\leq n_j)\wedge NormalValues\wedge A_{i,j}\rightarrow\forall_m:((n_i\leq m\wedge m<n_j)\rightarrow\neg Envote(k,n_i,fun_P^{m-n_i}(d_i))))$.

//由 $A_{i,j}$ 的定义得到

(3) $\forall_m:((n_i\leq m\wedge m<n_j)\rightarrow\neg Envote(k,n_i,fun_P^{m-n_i}(d_i)))\rightarrow$

$\forall_m:((n_i+1\leq m\wedge m<n_j)\rightarrow\neg Envote(k,n_i,fun_P^{m-n_i-1}(fun_P(d_i))))$. //由$(n_i+1\leq m\rightarrow n_i\leq m)$得到

(4) $\neg Envote(k,n_i,d_i)\wedge A_{i,j}\wedge NormalValues\rightarrow A_{i,j}(fun_P(d_i)/d_i,n_i+1/n_i)$.

//由 $A_{i,j}(fun_P(d_i)/d_i,n_i+1/n_i)=(((\neg f_i\wedge\neg f_j\wedge n_i+1\le n_j)\rightarrow(d_j=fun_P^{n_j-n_i-1}(fun_P(d_i)))\wedge\forall_m:((n_i+1\le m\wedge m<n_j)\rightarrow$

//$\neg Envote(k,n_i,fun_P^{m-n_i-1}(fun_P(d_i)))))$及(1)、(2)和(3)得到

(5) $\{A_{i,j}\wedge NormalValues\}t\_cal_i\{A_{i,j}\wedge NormalValues\}$.

//由(4)及 $t\_cal_i$ 使能条件满足$\neg Envote(k,n_i,d_i)$得到 □

**命题 8**. 若 $i,j$=1,2,3 且 $i\neq j$,则$\{A_{i,j}\wedge A_{j,i}\wedge NormalValues\}t_cal_j\{A_{i,j}\wedge NormalValues\}$.

证明:

(1) $(A_{i,j}\wedge NormalValues\wedge n_i\le n_j)\rightarrow(A_{i,j}(fun_P(d_j)/d_j,n_j+1/n_j)\wedge n_i\le n_j)$. //直接代入得到

(2) $(\neg Envote(k,n_j,d_j)\wedge A_{j,i}\wedge NormalValues\wedge n_i=n_j+1)\rightarrow(A_{i,j}(fun_P(d_j)/d_j,n_j+1/n_j)\wedge n_i=n_j+1)$. //直接代入得到

(3) $n_i>n_j+1\rightarrow(A_{i,j}(fun_P(d_j)/d_j,n_j+1/n_j)\wedge n_i>n_j+1)$.

(4) $\{A_{i,j}\wedge A_{j,i}\wedge NormalValues\}t\_cal_j\{A_{i,j}\wedge NormalValues\}$.

//结合$\{P\}R\{Q\}$的含义,(1)~(3)分别表示在 $n_i$ 小于、等于、大于 $n_j$+1 这 3 种情况,命题 8 都成立,

//从而命题得证 □

由命题 2~命题 8 以及 $Init_{TripleProcessors_{P\text{-}F}}\rightarrow(A\wedge NormalValues)$显然成立,我们即可得到谓词 $A\wedge NormalValues$ 是程序 $TripleProcessors_{P\text{-}F}$ 的一个不变量,从而谓词 $A$ 也是程序的不变量.此外,由命题 1 还可得到谓词 $B$ 也是程序的不变量.

下面的命题 9 证明谓词 $C$ 是程序 $TripleProcessors_F$ 的不变量,证明过程中我们用到了谓词 $NormalValues$ 和谓词 $B$ 是程序的不变量这个已经证明的结果.

**命题 9**. $TripleProcessors_{P\text{-}F}\rightarrow\Box C$.

证明:

(1) $Init_{TripleProcessors_F}\rightarrow C$.

(2) 对任意 $j$=1,2,3,$\{C\}t\_cal_j\{C\}$.

(3) 对任意 $j$=1,2,3,$\{C\}t\_err_j\{C\}$.

(4) 存在 $i,j$=1,2,3,$i\neq j$,使得$\neg f_i\wedge\neg f_j$.

(5) $C\rightarrow\exists_{i,j=1,2,3\ i\neq j}:(\neg f_i\wedge\neg f_j\wedge d_i=fun_P^{n_i}(v_0)\wedge d_j=fun_P^{n_j}(v_0))$. //由(4)得到

(6) $\{\neg f_i\wedge\neg f_j\wedge d_i=fun_P^{n_i}(v_0)\wedge d_j=fun_P^{n_j}(v_0)\}t\_vote\{C\}$. //由 $t\_vote$ 使能条件及 $B$ 为程序不变量得到

(7) $\{C\}$ $t\_vote$ $\{C\}$. //由(5)、(6)得到

(8) $TripleProcessors_F\rightarrow\Box C$. //由(1)~(3)和(7)得到 □

在命题 10 的证明过程中,我们用到了谓词 $NormalValues$,$C$ 和$\exists_{i=1,2,3}:\neg f_i$ 是程序的不变量这几个已经证明的结果.

**命题 10**. $TripleProcessors_{P\text{-}F}\rightarrow\Box(val=fun_P^k(v_0))$.

证明:

(1) $Init_{TripleProcessors_{P\text{-}F}}\rightarrow val=fun_P^k(v_0)$.

(2) 对任意 $j$=1,2,3,$\{val=fun_P^k(v_0)\}t\_cal_j\{val=fun_P^k(v_0)\}$.

(3) 对任意 $j$=1,2,3,$\{val=fun_P^k(v_0)\}t\_err_j\{val=fun_P^k(v_0)\}$.

(4) $\{NormalValues\wedge val=fun_P^k(v_0)\wedge C\wedge\exists_{i=1,2,3}:\neg f_i\}t\_vote\{val=fun_P^k(v_0)\}$.

(5) $TripleProcessors_{P\text{-}F}\rightarrow\Box(val=fun_P^k(v_0))$. //由(1)~(4)得到 □

由命题 10,我们得到 $TripleProcessors_{P\text{-}F}$ 满足性质 $Spec_P$.因此,我们采用三机冗余模式得到的程序 $TripleProcessors_P$ 在错误环境下 $F$ 运行时仍满足性质 $Spec_P$.

## 6 结 语

我们在 XYZ/E 框架下对三机冗余容错系统进行了描述和验证.本文与文献[3]的主要区别在于,我们将各台

机器抽象成为由一个处理器和一块加载了一个不断向外界环境输出数据的确定性顺序程序 $P$ 的内存组成,处理器既可以向外界环境输出内存数据的值,也可以执行程序 $P$ 的一条指令,从而修改内存数据的值,而不只是简单地对内存中存储的数据进行读写操作.这种抽象更符合三机冗余系统采用 3 台相同的机器运行相同的程序这一思想,因此更具有代表性.

在本文中,我们只考虑了处理器向外界环境输出内存中存储的数据而没有考虑从环境读入数据的情况,这主要是为了简化我们从程序中抽象出来的性质 $Spec_P$.此外,虽然 $Spec_P$ 刻画出程序 $SingleProcessor_P$ 的重要性质,但并非完全等价.实际上,假定程序 $SingleProcess_P'$,$TripleProcessors_P'$ 和 $TripleProcessors_{P\text{-}F}'$ 分别为考虑了内存中加载的程序 $P$ 从外部环境读入数据的单机程序、三机冗余程序和三机冗余程序在错误环境 $F$ 下的错误影响程序,如果不考虑公平性的问题,那么两个程序之间的求精关系即 $TripleProcessors_{P\text{-}F}' \rightarrow SingleProcess_P'$(这个结果比我们现在得到的结果更强)也是能够证明的.整个证明过程需要引入一些辅助的历史变量(history variables)和预言变量(prophecy variables)来建立两个程序之间的求精映射,相对比较复杂,我们将在今后的工作中尝试对此问题进行更深一步的研究.

**References:**

[1] Schepers H. Terminology and paradigms for fault tolerance. In: Vytopil J, ed. Formal Techniques in Real-Time and Fault Tolerant Systems. Boston: Kluwer Academic Publishers, 1993. 3~31.

[2] Doug GW. Fault tolerance as self-similarity. In: Vytopil J, ed. Formal Techniques in Real-Time and Fault Tolerant Systems. Boston: Kluwer Academic Publishers, 1993. 33~49.

[3] Liu ZM, Joseph M. Specification and verification of fault-tolerance, timing and scheduling. ACM Transaction on Programming Languages and Systems, 1998,21(1):46~89.

[4] Liu ZM, Joseph M. Transformation of programs for fault-tolerance. Formal Aspects of Computing, 1992,4(5):442~469.

[5] Liu ZM. Fault-Tolerant programming by transformations [Ph.D. Thesis]. Department of Computer Science, University of Warwick, 1991.

[6] Lamport L. The temporal logic of actions. ACM Transactions on Programming Languages and Systems, 1994,16(3):872~923.

[7] Abadi M, Lamport L. The existence of refinement mapping. Theoretical Computer Science, 1991,83(2):253~284.

[8] Tang ZS. An introduction to XYZ system. Technical Report, ISCAS-XYZ-88-1, Beijing: Institute of Software, the Chinese Academy of Sciences, 1988.

[9] Tang ZS. Temporal Logical Programming and Software Engineering. Beijing: Science Press, 1999 (in Chinese).

**附中文参考文献**:

[9] 唐稚松.时序逻辑程序设计与软件工程.北京:科学出版社,1999.