基于 XYZ/E 描述和验证容错系统*

郭 亮, 唐稚松

(中国科学院 软件研究所 计算机科学重点实验室,北京 100080)
E-mail: gls@ios.ac.cn
http://www.ios.ac.cn

摘要: 研究使用 XYZ/E 描述和验证容错系统.基于 XYZ/E 中可执行程序 P 对应的状态转换系统对其错误环境 F 建模,通过错误转换给出错误影响程序 P_F;基于 P,F 和恢复算法 R,通过容错转换给出容错程序 P_{F-R};定义了程序 P,Q 之间两种求精关系:容错求精和向后恢复求精,基于这两种求精关系可直接从程序 P 的规范推导出程序 Q 满足的一些性质.

关 键 词: 容错系统;时序逻辑语言 XYZ/E;求精;容错转换;规范;验证
中图法分类号: TP311 文献标识码: A

软件系统不可靠的原因不外乎两种:硬件出错和软件设计出错.使用形式化验证方法可以有效地避免软件设计错误,然而硬件错误(fault)往往不能事先预测,需要在程序运行时采用行之有效的方法对其容忍.常用的程序容错[1,2]方法包括海明码、多机冗余、断点设置和恢复等几种.

容错系统(fault-tolerant system)是一种即使有错误发生其规范也能满足的系统.一般来说程序 P 正确,是建立在其硬件环境不出错的前提下的,这样的系统(程序及其硬件环境组成的整体)称为无错系统(fault-free system).但是,在系统运行过程中,硬件错误很可能发生,这样的系统称为易错系统(fault-prone system).有些硬件错误的发生将导致系统崩溃,不可恢复,本文中我们对此类错误不予考虑;而另外一些硬件错误系统必须容忍,设计系统时应该考虑怎样才不会因为这些错误发生而导致系统紊乱.

要建立容错系统,首先需要考虑程序 P 所运行在的硬件环境可能发生哪些错误,然后描述清楚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 的建模问题.通常的方法是把每种可能发生的硬件错误建模为一个动作[3~5](action),然后定义 P 在 F 下运行时的错误语义(failure semantics),再通过错误转换(fault transformation),给出此程序的错误影响程序(fault-affected program)$F(P)$,P 在 F 下运行等价于程序 $F(P)$在无错环境下运行.若程序运行时发生硬件错误,则需要编写一个恢复程序 R 进行补救,R 可建模为一个动作集合,然后基于 P,F,R,通过容错转换(fault-tolerant transformation),得到容错程序(fault-tolerant program)$F(R(P))$,再讨论其性质.

本文基于 XYZ/E 可执行程序 P 对应的状态转换系统(state transition system),将其错误环境 F 中每个错误建模为一组转换(transition)关系,并考虑了这些错误对程序状态转换的影响(例如有些错误将导致程序中某些状态转换被屏蔽)以及它们必须满足的错误假设(fault hypothesis)(例如三机冗余系统中一般假定两台机器不会同时出错).然后基于 P,F 定义错误影响程序 P_F 所对应的状态转换系统,并通过错误转换给出与 P_F 等价的 XYZ/E 程序.此外,恢复程序 R 也被建模为一组转换关系,基于 P,F 和 R,通过容错转换,可得到容错程序 P_{F-R}.最后,我们给出程序 P,Q 之间两种与容错有关的求精关系:容错求精(fault-tolerant refinement)和向后恢复求精

* 收稿日期: 2000-07-20; 修改日期: 2001-07-06
基金项目: 国家 863 高科技发展计划资助项目(863-306-ZT02-04-01);国家"九五"重点科技攻关项目(98-780-01-07-01)
作者简介: **郭亮**(1976－),男,江西吉安人,博士生,主要研究领域为软件工程;**唐稚松**(1925－),男,湖南长沙人,研究员,博士生导师,中国科学院院士,主要研究领域为计算机科学理论,软件工程.

(backward-recovery refinement)的定义,基于这两种求精关系可直接从程序 P 的规范出发推导出程序 Q 满足的一些性质.

1 时序逻辑语言 XYZ/E 简介

XYZ/E[6,7]语言中最基本的元素是条件元(conditional element).条件元共有 3 种形式,式(1)所示条件元直接定义了相邻状态之间的转换关系,因此全部由此种形式条件元组成的程序可以执行.其余两种形式的条件元用于表示程序的静态规范.

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

XYZ/E 中引入了如式(2)所示选择语句表示不确定性.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做出不确定选择.

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

为描述程序间求精关系,我们引入了踏步[8](stuterring)的概念,如条件元(1)可看做是如下选择语句(3)的缩写.同样,形式(2)的选择语句可认为隐含了一个分支"$\$T|>\$OLB=y$".

$$LB=y\wedge R \Rightarrow !! [\$T |> \$O(v_1,v_2,...,v_n)=(e_1,e_2,...,e_n)\wedge \$OLB = z,\ \$T |> \$OLB = y\ //踏步]. \tag{3}$$

XYZ/E 中表示算法的构件为单元(unit),形式如式(4)所示.其中 $A_1,A_2,...,A_n$ 是条件元,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.

$$\Box[A_1;A_2;...;A_n]. \tag{4}$$

XYZ/E 可执行程序 P 由一组变量集合 Var_P,初始条件 $Init_P$ 和可执行单元 $Unit_P$ 三部分组成.以后若不作特殊声明,程序都是指 XYZ/E 可执行程序.

每个可执行程序 P 对应一个状态转换系统.状态转换系统是一个三元组(V,θ,Γ),其中 V 等于 Var_P 为有限变量集合,每个变量在特定值域上取值;θ等价于 $Init_P$ 为初始条件;$\Gamma=\{\tau_1,\tau_2,\ldots,\tau_n\}$为有限状态转换集合,每个元素 $\tau_i(i=1,\ldots,n)$是一个定义了两相邻状态间各变量取值关系的转换,Γ中转换与 $Unit_P$中条件元或选择语句的分支间存在一一对应关系.以后为叙述方便,我们用$\$Ox$ 表示状态转换中变量 x 的下一时刻值;用 $P_\tau\wedge\$OQ_\tau$表示状态转换$\tau$,其中 P_τ对应于τ的使能条件(enable condition)部分,$\$OQ_\tau$对应于$\tau$的动作部分;用 V_Q 表示在一阶时序逻辑公式 Q 中出现的自由变量集合.

状态转换系统的一个计算是无限状态序列:$s_0,s_1,...$,其中,(1) s_0 满足初始条件θ;(2) 对任意 $j\geq 0$,$s_{j+1}=s_j$(踏步)或存在$\tau\in\Gamma$使得 $s_{j+1}=\tau(s_j)$;两个状态转换系统等价是指它们具有相同的计算集合.这里为简化程序语义,我们没有考虑公平性的问题,但要求除非所有状态转换都被屏蔽,否则不会出现无穷踏步状态序列.

每个程序 P 的变量集合 Var_P 可划分为两个不相交的子集:内部变量集合 Int_P 和外部变量集合 Ext_P.外部变量是程序提供给环境的接口,而内部变量是程序在实现过程中引入的辅助变量.类似文献[8],设 $Int_P=\{u_1,u_2,\ldots,u_n\}$,则程序 P 的语义对应如式(5)所示公式 LF_P.

$$LF_P\equiv \exists_{u_1,u_2,...,u_n}: LF_P',\ LF_P'\equiv Init_P\wedge Unit_P. \tag{5}$$

定义 **1**. 程序 P_2 是程序 P_1 的求精,记作 $P_1\propto P_2$,当且仅当 $Ext_{p_1}=Ext_{p_2}$ 且 $LF_{p_2}\rightarrow LF_{p_1}$.

式(5)中$\exists$量词后面的内部变量是时序变量,超出了一阶时序逻辑的范畴.为证明两个程序之间的求精关系成立,文献[8]中提出了一种解决方法,设 $Int_{p_1}=\{u_1,u_2,\ldots,u_n\}$且 $Int_{p_2}=\{w_1,w_2,\ldots,w_m\}$,可先找出一个从 Int_{p_2} 到 Int_{p_1} 的映射组[9]$f=\{f_1,f_2,\ldots,f_n\}$,使得对于任意 $i=1,\ldots,n$ 有 $u_i=f_i(w_1,w_2,\ldots,w_m)$,然后证明式(6)的正确性.

$$(Init_{p_2}\wedge Unit_{p_2})\rightarrow(Init_{p_1}\wedge Unit_{p_1})[f_1/u_1,f_2/u_2,...,f_n/u_n]. \tag{6}$$

定义 **2**. 程序 P_1,P_2 等价,记作 $P_1\leftrightarrow P_2$,当且仅当 $P_1\propto P_2\wedge P_2\propto P_1$.

显然,具有相同(外部)变量集合(包括相同变量集合)的程序 P_1,P_2 等价,当且仅当它们对应的状态转换系统等价.

此外,当程序 P 中出现不确定选择语句时,我们可能希望对 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 的语义对应如式(7)所示时序逻辑公式 LF_P.

$$LF_P\equiv \exists_{u_1,u_2,...,u_n}: LF_P',\ LF_P'\equiv Init_P\wedge Unit_P\wedge Where_P. \tag{7}$$

2 硬件错误环境和恢复程序建模

在为程序硬件错误环境建模之前,我们先假定每个程序 P 存在布尔型错误指示变量集 $FV_P=\{f_1,f_2,...,f_n\}\subseteq Int_P$,每个错误指示变量代表一种可能发生的硬件错误.对于 $i=1,...,n$, f_i 为真表示其代表的硬件错误已发生且尚未修复,否则表示错误尚未发生或已修复.

为以后叙述方便,我们将 $f_1\vee f_2\vee...\vee f_n$ 简记为 f_P;$\$Of_1=f_1\wedge\$Of_2=f_2\wedge...\wedge\$Of_n=f_n$ 简记为 $\$Of_P=f_P$;$\$O\neg f_1\wedge\$O\neg f_2\wedge...\wedge\$O\neg f_n$ 简记为$\$O\neg f_P$.程序 P 开始运行时,一般假定硬件环境一切正常,即 $Init_P\rightarrow\neg f_P$.若程序 P 在一个无错环境中运行,则 $Where_P$ 满足性质$\Box\neg f_P$.

假定程序 P 对应状态转换系统为$<Var_P,\theta_P,\Gamma_P>$,$FV_P=\{f_1,f_2,...,f_n\}\subseteq Var_P$ 且$\theta_P\rightarrow\neg f_P$,则程序 P 的硬件错误环境 F 可建模为五元组$<Var_F,\theta_F,\Gamma_{aF},\Gamma_{dF},FH_F>$,其各元素含义如下:

Var_F:发生硬件错误时所有可能引用的变量集合,$Var_P\subseteq Var_F$.

θ_F:一阶逻辑公式,定义 Var_F 中变量满足的初始条件,$\theta_F\rightarrow\theta_P$.

Γ_{aF}:有限状态转换集合.Γ_{aF} 中每个状态转换 τ 具有形式:$P_\tau\wedge\$Of_\tau\wedge\$OQ_\tau$,其中$f_\tau\in FV_P$,$V_{P\tau}\subseteq Var_F$,$V_{Q\tau}\subseteq Var_F\backslash FV_P$,指定错误发生条件及错误对各变量取值的影响.

Γ_{dF}:从集合$\{1,...,n\}$到集合Γ_P的超集 $2^{\Gamma P}$ 的映射.我们对每个 $i=1,...,n$ 定义$\Gamma_{dFi}\equiv\Gamma_{dF}(i)$,则$\Gamma_{dFi}\subseteq\Gamma_P$,指定当硬件错误 f_i 存在时,程序 P 中哪些状态转换将被屏蔽.对于每个$\tau\in\Gamma_P$,我们定义错误屏蔽谓词$\tau_f\equiv\vee_{i=1,...,n,\,\tau\in\Gamma dFi}:f_i$,$\tau_f$ 在某状态下取值为真,表示此状态存在某种硬件错误,并且此种错误导致状态转换 τ 被屏蔽,否则,表示状态转换 τ 在此状态下使能.

FH_F:表示错误假设的一阶时序逻辑公式,$V_{FHF}\subseteq Var_F$.一般要证明程序 P 即使在有错误发生的情况下也满足其规范,都假定最终存在一时刻所有错误被修复,也即 $FH_F\rightarrow\Diamond\Box\neg f_P$.

程序 P 和相应错误环境 F 给定后,则可定义错误影响程序 P_F.设 P 对应状态转换系统$<Var_P,\theta_P,\Gamma_P>$,$F$ 建模为五元组$<Var_F,\theta_F,\Gamma_{aF},\Gamma_{dF},FH_F>$,则 P_F 所对应的状态转换系统$<Var_{PF},\theta_{PF},\Gamma_{PF}>$三元组有:$Var_{PF}=Var_F$,$\theta_{PF}=\theta_F$,$\Gamma_{PF}=\Gamma_{aF}\cup\Gamma'_{dF}$,其中 $\Gamma'_{dF}=\{\neg\tau_f\wedge\tau|\tau\in\Gamma_P\}$考虑了 Γ_P 中每个转换 τ 可能由于某种错误存在而被屏蔽的情况,从而对 τ 的使能条件合取了$\neg\tau_f$以加强.

对于 P_F,有如下错误转换定理.

定理 **1**(错误转换定理). 假定可执行程序 P'_F 的变量及初始条件与 P_F 一致,单元为将 P 中每个形式为式(1)的条件元和形式为式(2)的选择语句分别转换为如式(8)和式(9)所示选择语句而得到,则 $P_F\leftrightarrow P'_F$.

$LB=y\wedge R\Rightarrow$!! [//下面的τ_{1f},τ_{kf}等分别为与选择语句各分支对应的状态转换相关的错误屏蔽谓词

$\neg\tau_f|>\$O(v_1,v_2,...,v_n)=(e_1,e_2,...,e_n)\wedge\$OLB=z$,// τ为此条件元在其状态转换系统中对应的状态转换

$\vee_{\tau\in\Gamma aF}:P_\tau|>\$Of_\tau\wedge\$OQ_\tau$ //“$\vee_{\tau\in\Gamma aF}$:”算子对 Γ_{aF} 中所有状态转换做析取,$\$OQ_\tau$中给出了$\$OLB$ 的值]. (8)

$LB=y\wedge R\Rightarrow$!![$\neg\tau_{1f}\wedge Cond_1|>ExeAct_1,...,\neg\tau_{kf}\wedge Cond_k|>ExeAct_k,\vee_{\tau\in\Gamma aF}:P_\tau|>\$Of_\tau\wedge\$OQ_\tau$]. (9)

证明:为简化证明,我们假定程序单元全部由式(1)所示条件元组成.对于程序单元中存在选择语句的情况,证明类似,从略.

我们知道,单元中所有条件元的条件($LB=y\wedge P$ 部分)具有完整性性质,即在程序运行的任意时刻至少存在一个条件元的条件被满足.设集合 $Conds$ 是单元中所有条件的集合,则任意时刻$\vee_{c\in Conds}:c$ 为真,因此对任意一阶逻辑式 w,如下时序逻辑式(10)为真.

$$\Box(w\leftrightarrow\vee_{c\in Conds}:(c\wedge w)). \quad (10)$$

由转换式(8)可知,P'_F 对应的状态转换系统三元组有$\Gamma_{PF}=\Gamma'_{aF}\cup\Gamma'_{dF}$,其中 $\Gamma'_{aF}=\cup_{\tau\in\Gamma aF}\cup_{c\in Conds}:\{c\wedge\tau\}$.可以看出,$\Gamma_{aF}$ 中每个状态转换 τ 与 Γ'_{aF} 中状态转换集合$\cup_{c\in Conds}:\{c\wedge\tau\}$中所有转换的动作部分相同,而由式(10)可知,$\tau$的条件部分 $en(\tau)$成立当且仅当至少存在一个 $c\in Conds$ 使得 $en(\tau)\wedge c$ 成立.从计算的定义我们可以得到,任一状态序列是程序 P_F 的计算当且仅当它是 P'_F 的计算,因此程序 P_F 与 P'_F 所对应的状态转换系统等价,从而 $P_F\leftrightarrow P'_F$.□

若程序运行时硬件出错,则需要恢复程序 R 将程序从错误状态恢复到正确状态.一般假定错误环境 F 对程序 R 没有影响.对 R 建模比较简单,它对应有限状态转换集合 Γ_R.Γ_R 中每个状态转换 τ 形式为

P_τ∧$O\neg f_p$∧OQ_τ,$V_{P\tau}$,$V_{Q\tau}\subseteq Var_F$,指示错误如何被修复.

给定 P,F 和 R,则可定义容错程序 P_{F-R} 对应的状态转换系统$<Var_{F-R},\theta_{F-R},\Gamma_{F-R}>$三元组.其中,$Var_{F-R}=Var_F$,$\theta_{F-R}=\theta_F$,$\Gamma_{F-R}=\Gamma_{PF}\cup\Gamma_R=\Gamma_{aF}\cup\Gamma'_{dF}\cup\Gamma_R$.对于 P_{F-R},我们有如下容错转换定理,其证明与错误转换定理的证明类似,这里从略.

定理 **2**(容错转换定理). 假定可执行程序 P'_{F-R} 变量及初始条件与 P_{F-R} 一致,单元为将 P 中每个形式为式(1)的条件元和形式为式(2)的选择语句分别转换为如式(11)和式(12)所示选择语句而得到,则 $P_{F\text{-}R}\leftrightarrow P'_{F\text{-}R}$.

$LB=y\wedge R \Rightarrow$!! [$\neg\tau_f$ |>$O(v_1,v_2,...,v_n)=(e_1,e_2,...,e_n)$∧$OLB=z$,

$\vee_{\tau\in\Gamma aF}:P_\tau|>\$Of_\tau\wedge\$OQ_\tau$,$\vee_{\tau\in\Gamma R}:P_\tau|>\$O\neg f_P\wedge\$OQ_\tau$]. (11)

$LB=y\wedge R \Rightarrow$!! [$\neg\tau_{1f}\wedge Cond_1|>ExeAct_1$, ..., $\neg\tau_{kf}\wedge Cond_k|>ExeAct_k$,

$\vee_{\tau\in\Gamma aF}:P_\tau|>\$Of_\tau\wedge\$OQ_\tau$,$\vee_{\tau\in\Gamma R}:P_\tau|>\$O\neg f_P\wedge\$OQ_\tau$]. (12)

以后,我们用 P'_F 代替 P_F,用 P'_{F-R} 代替 P_{F-R}.此外,我们在为错误环境建模时给出的五元组中还存在一个元素 FH_F,指示了错误环境中错误假设约束,因此 P_{F-R} 的语义对应如式(13),其中 $Int_{PF-R}=Int_P=\{u_1,u_2,...,u_n\}$.

$$\exists_{u_1,u_2,...,u_n}:(Init_{PF-R}\wedge Unit_{PF-R}\wedge FH_F). \quad (13)$$

3 与容错有关的两种求精关系的定义

下面给出程序 P,Q 之间两种求精关系:容错求精 $P\propto_{ft}Q$ 和向后恢复求精 $P\propto_{br}Q$ 的定义.其中,P,Q 是指程序;$FV_P=\{f_1,f_2,\ldots,f_n\}$;$S,S_1$ 和 S_2 是指状态;$Comp(P)$表示程序 P 的计算集合;tr_m 表示计算 tr 的第 m 个状态;$S|_{VarSet}$ 表示 S 在变量集合 $VarSet$ 上的投影.

定义 **3**.

$Fail_P(S)\equiv\exists_{i\in 1,...,n}:S|f_i=True$ $\quad$ $Good_P(S)\equiv\neg Fail_P(S)$

$F\text{–}Free(P)\equiv\forall_{tr\in Comp(P),m\geq 0}:Good_P(tr_m)$ //P 是无错程序

$P\propto_{ft}Q\equiv F\text{–}Free(P)\wedge FV_P=FV_Q\wedge Ext_P=Ext_Q\wedge\forall_{trq\in Comp(Q)}:\exists_{trp\in Comp(P)}:\forall_{m\geq 0}:trq_m|_{ExtP}=trp_m|_{ExtP}$

$F\text{–}BrRefinable(P,Q)\equiv F\text{–}Free(P)\wedge Ext_P=Ext_Q\wedge FV_P=FV_Q\wedge Var_P\subseteq Var_Q\wedge\forall_{tr\in Comp(q)}:\exists_{k>0}:\forall_{m>k}:Good_Q(tr_m)$

//程序 P,Q 之间满足可向后恢复求精关系,当且仅当 P 是无错程序;Q 运行足够长时间后所有错误被修复从而一直处于“好”状态;P,Q 具有相同外部变量和错误指示变量集合;P 的(内部)变量集合是 Q 的(内部)变量集合的子集.

$Init_P(S)\equiv\exists_{tr\in Comp(P)}:tr_0=S$ //S 是 P 的初始状态

$Next_P(S_1,S_2)\equiv\exists_{tr\in Comp(P),m\geq 0}:tr_m=S_1\wedge tr_{m+1}=S_2$ //P 中某计算的两相邻状态分别为 S_1 和 S_2

$P\propto_{br}Q\equiv F\text{–}BrRefinable(P,Q)\wedge\forall_{tr\in Comp(Q)}:(Init_P(tr_{0|VarP}))\wedge\forall_{m>0}:(Fail_Q(tr_m)\vee$

$(Good_Q(tr_{m-1})\wedge Next_P(tr_{m-1|VarP},tr_{m|VarP}))\vee(Fail_Q(tr_{m-1})\wedge\exists_{k<m-1}:(Good_Q(tr_k)\wedge tr_{k|VarP}=tr_{m|VarP})))$

直观上,$P\propto_{ft}Q$ 表示 P 是一个无错程序且程序 Q 是程序 P 的求精;$P\propto_{br}Q$ 表示 P,Q 之间除满足可向后恢复求精关系 $F\text{–}BrRefinable(P,Q)$外,程序 Q 还满足:(1) 初始状态在 Var_P 上的投影是程序 P 的初始状态;(2) 在运行时任意时刻,如果处于“好”状态,则下一时刻或者由于硬件出错进入“坏”状态或者执行程序 P 的状态转换进入下一个“好”状态;(3) 如果处于坏状态,则下一时刻或者继续处于“坏”状态或者被修复为一个“好”状态,此状态与以前某个“好”状态在 Var_P 上投影的值相等.

对于程序 P,Q 之间的向后恢复求精关系,有如下定理 3 成立:

定理 **3**. 设 XYZ/AE 程序 P_{br} 变量与程序 Q 一致;初始条件与程序 P 一致;单元由将 P 中每个形式为式(2)的选择语句转换为如式(14)所示的 XYZ/AE 选择语句而得到(条件元也可看做是选择语句),则对于存在关系 $F\text{–}BrRefinable$ 的两个程序 P,Q,若 $P_{br}\propto Q$,则 $P\propto_{br}Q$.

$LB=y\wedge R \Rightarrow$!![

$\neg f_P\wedge Cond_1$ |> $O\neg f_P\wedge\$OAnyValue(Var_Q\backslash Var_P)\wedge ExeAct_1$,...,

$\neg f_P\wedge Cond_k$ |> $O\neg f_P\wedge\$OAnyValue(Var_Q\backslash Var_P)\wedge ExeAct_k$,

$\neg f_P$ |> $Of_P\wedge\$OAnyValue(Var_Q\backslash FV_P)$,

f_P|> $\$Of_P \wedge \$OAnyValue(Var_Q \backslash FV_P)$,

f_P|> $\$O\exists_{vs}:(vs|_{fP}=False \wedge Var_P=vs|_{VarP} \wedge \$E(Var_p= vs|_{VarP}))]$. (14)

在式(14)中,谓词 $AnyValue(Varset)$表示在 $Varset$ 变量集合中的所有变量都可取其值域中的任意值;时序逻辑公式$\$O\exists_{vs}:(vs|_{fP}=False \wedge Var_P=vs|_{VarP} \wedge \$E(Var_P=vs|_{VarP}))$表示下一时刻的状态与从前的某个"好"状态 vs(vs 在程序 P 的状态空间中取值)在 Var_P 上的投影相等.这里$\$E$ 为过去时序算子曾经算子,对于逻辑式 p,$\$Ep$ 在某个时刻为真当且仅当 p 在过去某时刻(不含当前时刻)为真.定理 3 的正确性可分别从式(1)P,Q 的初始状态式(2)P,Q 相邻状态之间的转换关系两个方面考虑,比较直观,这里证明从略.

若程序 P,Q 之间存在上面定义的某种求精关系,则可根据如下规则 1 从程序 P 的规范出发直接推导得出程序 Q 满足的一些性质.其中 p,q 为谓词,S 为一阶时序逻辑公式.需要说明的是,在规则 1 的(Ⅲ)和(Ⅳ)中,我们使用了 LF'_Q,而不是直接使用 Q,因为变量 f_P 引用了程序 Q 的内部变量.

规则 1.

(Ⅰ)	(Ⅱ)	(Ⅲ)	(Ⅳ)
$P \models S$	$P \models \Diamond p$	$P \models \Box p$	$P \models \Box(p \rightarrow \Diamond q)$
$P \propto_{ft} Q$	$P \propto_{br} Q$	$P \propto_{br} Q$	$P \propto_{br} Q$
$Q \models S$	$Q \models \Diamond p$	$LF_Q' \models \Box(p \vee f_P)$	$LF_Q' \models \Box((p \wedge \neg f_P) \rightarrow \Diamond q)$

4 向后恢复求精实例

实际工程中采用容错求精方式建立容错系统的典型例子是三机冗余容错系统,其思想核心为使用 3 台相同的机器同时运行相同的程序,由于同时出现两台机器硬件出错的概率趋近于 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.三机冗余容错系统假定在任意时刻最多只有一台机器硬件出错且错误不会发生在机器交互、表决和修复过程中.关于三机冗余程序是单机程序的容错求精的证明,可参见文献[3].

向后恢复求精方式对应断点设置和恢复容错方法,实际工程中采用此方法建立容错系统的一般步骤为:对于给定无错程序 P,通过添加一些不被错误环境 F 修改的冗余变量不定期地备份程序 P 中关键数据,把程序 P 求精至无错程序 Q.此时 Q 的错误环境实际是 P 的错误环境 F 的一个求精[5],这里为方便叙述也记作 F.在检测到硬件错误后,启动恢复程序 R,从备份数据中提取一个相对比较近的过去的"好"状态作为当前状态恢复.程序 Q_{F-R} 满足的一些性质可通过先证明 $P \propto_{\mathrm{br}} Q_{F-R}$ 再根据规则 1 推导得出.下面介绍一个采用这种方式建立容错系统的简单例子.

XYZ/E 程序(函数)Cal 功能为计算大小为 $N \in \mathcal{N}$的数组 A 中各元素的平方和,并把结果存储在变量 Sum 中.其中 $Ext_{Cal}=\{A,Sum\}$,$Int_{Cal}=\{lb,i,f\}$,$FV_{Cal}=\{f\}$.显然,程序 Cal 满足规范 $Spec$:$\Box(lb=\mathrm{RETURN} \rightarrow Sum=\Sigma_{k \in 1,\ldots,N}: A^2[k])$.

```
{lb=START∧Sum=0∧i=0∧f=$F /*Init_Cal*/}
%Proc Cal(%INP/A:array[int;N];%IOP/Sum:int)== [
  %loc [i:int; f:bool]
  %alg [
    lb=START => $olb=l0;   //C1
    lb=l0∧!! [
      i<N   => $oSum=Sum+A[i]*A[i]∧$oi=i+1∧$olb=l0; //C21
      i>=N => $olb=l1; //C22] //C2
    lb=l1 => $olb=RETURN //C3
  ]
]
```

现假定程序 *Cal* 运行时可能发生硬件错误 *f* 改变变量 *Sum* 的值.错误环境 *F* 可建模为五元组 $<Var_F,\theta_F,\Gamma_{aF},\Gamma_{dF},FH_F>$, 其中 $Var_F=Var_P,\theta_F=\theta_P,\Gamma_{aF}=\{lb=10\wedge f=\$F\wedge\$of=\$T\wedge\$oSum\neq Sum\wedge\$olb=10\},\Gamma_{dF}=\{\{C_{22}\}\}$, $FH_F=\Diamond\Box\neg f.\Gamma_{aF}$ 指定错误 *f* 只可能在 *lb* 等于 10 时发生,错误发生将修改变量 *Sum* 的值;Γ_{dF} 指定,若错误 *f* 存在,则条件元 C_{22} 被屏蔽.FH_F 表示最终存在一时刻所有错误被修复.通过错误转换可得到其错误影响程序 Cal_F,显然,Cal_F 不满足规范 *Spec*.

为建立容错系统,需要引入两个内部冗余变量 *bi* 和 *bSum* 在计算过程中不定期记录下 *i* 和 *Sum* 的值,从而把程序 *Cal* 求精到下面的程序 *BCal*.其中,$Ext_{BCal}=\{A,Sum\}$,$Int_{BCal}=\{lb,i,bi,bSum,f\}$,$FV_{BCal}=\{f\}$,显然程序 *BCal* 是 *Cal* 的求精.

```
{lb=START∧Sum=0∧i=0∧bi=0∧bSum=0∧f=$F /*Init_BCal*/}
%Proc BCal(%INP/A:array[int;n];%IOP/Sum:int)== [
  %loc [i,bi,bSum:int; f:bool]
  %alg [
    lb=START => $olb=l0;
    lb=l0 => !![
      i<N   |> $oSum=Sum+A[i]*A[i]∧$oi=i+1∧$olb=l0,
      i>=N |> $olb=l1,
      $T    |> $obSum=Sum∧$obi=i∧$olb=l0 //不定期备份 ]
    lb=l1 => $olb=RETURN   ]
     ]
```

BCal 程序的错误环境是 *Cal* 的错误环境 *F* 的求精,与 *F* 的不同主要在于 $Var_F=Var_{Bcal},\theta_F=\theta_{BCal}$ 且错误 *f* 发生后将屏蔽 *BCal* 程序中备份语句.为方便叙述,我们将 *F'* 记作 *F*.由 *BCal* 及其错误环境 *F*,我们可以得到错误影响程序 $BCal_F$.显然,$BCal_F$ 也不满足规范 *Spec*.

现在我们将恢复算法 *R* 建模为$\{lb=10\wedge f=\$T\wedge\$of=\$F\wedge\$o(i,Sum)=(bi,bSum)\wedge\$olb=10\}$,指定错误修复时设置 *i* 和 *Sum* 当前值为 *bi* 和 *bSum* 存储的值.通过容错转换,我们得到容错程序 $BCal_{F-R}$ 如下,其中 $Ext_{Bcal-F}=\{A,Sum\}$, $Int_{Bcal-F}=\{lb,i,bi,bSum,f\}$,$FV_{Bcal-F}=\{f\}$.

```
{lb=START∧Sum=0∧i=0∧bi=0∧bSum=0∧f=$F /*Init_BCALF-R*/}
%Proc BCal_F-R(%INP/A:array[int;n];%IOP/Sum:int) == [
  %loc [i,bi,bSum:int; f:bool]
  %alg [
    lb=START=>$olb=10;//BC1 由于只在 lb=10 时可能出错或修复错误,所以 C1 容错转换后等于自身
    lb=10 => !![
      i<N   |> $oSum=Sum+A[i]*A[i]∧$oi=i+1∧$olb=10, //BC21
      f=$F∧i>=N |> $olb=11, //BC22  由于被错误 f 屏蔽,所以只可能在无错时执行
      f=$F |> $obSum=Sum∧$obi=i∧$olb=l0, //BC23 不定期备份,由于被 f 屏蔽,所以只在无错时发生
      f=$F |> $of=$T∧$oSum≠Sum∧$olb=l0, //BC24 错误发生修改 f 和 Sum 值
      f=$T |> $of=$F∧$oi=bi∧$oSum=bSum∧$olb=10, //BC25 设置 i 和 Sum 值为 bi 和 bSum 值
      $T   |> $olb=10 //BC26  踏步所对应的转换] //BC2
    lb=11 => $olb=RETURN //BC3  ]
     ]
```

下面证明程序 $BCal_{F-R}$ 满足规范 *Spec*:$\Box(lb=\text{RETURN}\rightarrow Sum=\Sigma_{k\in 1,\ldots,N:}A^2[k])$.

证明:首先证明 $Cal\propto_{br}BCal_{F-R}$.显然,程序 Cal_{br},$BCal_{F-R}$ 之间满足可向后恢复求精关系 *F–BrRefinable*,则由定理 3 可知,我们只需证明 $Cal_{br}\propto BCal_{F-R}$,也即证明逻辑式 $LF_{BCalF-R}\rightarrow LF_{Calbr}$ 成立.由于程序 Cal_{br} 和 $BCal_{F-R}$ 具有相同的外部变量集合$\{A,Sum\}$和内部变量集合$\{lb,i,bSum,bi,f\}$,因此只需证明 $Init_{BCalF-R}\wedge Unit_{BCalF-R}\wedge\Diamond\Box$

$\neg f \rightarrow Init_{Calbr} \wedge Unit_{Calbr}$,这里我们为两个程序内部变量建立的映射关系为同名映射.

在证明程序 Cal_{br},$BCal_{F-R}$ 间求精关系成立之前,我们先定义过去时序逻辑公式 p 如下,p 中$E′也为过去时序算子,对于逻辑式 L,$E′L 在某个时刻为真当且仅当 L 在当前时刻或过去的某个时刻为真,显然$E′L\leftrightarrow$EL\vee L$.

$$p \equiv LB=10 \rightarrow (\exists_{(i',s')}:(i'=bi \wedge s'=bSum) \wedge \$E'(i=i' \wedge Sum=s' \wedge f=\$F \wedge LB=10)). \quad (15)$$

p 的直观含义为,如果当前时刻有 lb=10,则此状态或过去某个状态为“好”状态,且其中变量 i,Sum,lb 的取值分别等于在当前状态中变量 bi,$bSum$ 和 lb 的取值.显然$\Box p$ 是程序 $BCal_{F-R}$ 满足的安全性[10]性质,因此,逻辑式 $Init_{BCalF-R} \wedge Unit_{BCalF-R} \rightarrow \Box p$ 成立.

此外,$Init_{BCalF-R} \rightarrow Init_{Calbr}$ 显然,由 $Init_{BCalF-R} \wedge Unit_{BCalF-R} \rightarrow \Box p$ 可知,要证明命题 $Init_{BCalF-R} \wedge Unit_{BCalF-R} \wedge \Diamond \Box \neg f \rightarrow Init_{Calbr} \wedge Unit_{Calbr}$ 成立,只需证明 $Unit_{BCalF-R} \wedge \Box p \rightarrow Unit_{Calbr}$.由于单元中各条件元间为合取关系,因此只需证明$\wedge_{i=1,...,3}:(Unit_{BCalF-R} \wedge \Box p \rightarrow \Box BR(C_i))$,这里 $BR(C_i)$为程序 Cal 中条件元或选择语句 C_i 在进行定理 3 中所述转换得到的选择语句.下面我们以 i=2 为例证明命题 $Unit_{BCalF-R} \wedge \Box p \rightarrow \Box BR(C_2)$成立,其中 $BR(C_2)$为如式(16)所示的选择语句.对于 i=1 或 3,证明类似,这里从略.

$lb=l0$ => !![

$f=\$F \wedge i<N$ |> $\$of=\$F \wedge \$oSum=Sum+A[i]*A[i] \wedge \$oi=i+1 \wedge \$oAnyValue(\{bi,bSum\}) \wedge \$olb=l0$, //$BR_{21}$

$f=\$F \wedge i>=N$ |> $\$of=\$F \wedge \$oAnyValue(\{bi,bSum\}) \wedge \$olb=l1$, //$BR_{22}$

$f=\$F$ |> $\$of=\$T \wedge \$OAnyValue$ ($\{A,i,Sum,lb,bi,bSum\}$),//BR_{23}

$f=\$T$ |> $\$of=\$T \wedge \$OAnyValue$ ($\{A,i,Sum,lb,bi,bSum\}$), //BR_{24}

$f=\$T$ |> $\$of=\$F \wedge \$O\exists_{vs}:(vs|_{fP}=False \wedge Var_P=vs \wedge \$E(Var_p=vs))$, //$BR_{25}$

$\$T$ |> $\$olb=l0 \wedge \$oAnyValue(\{bi,bSum\})$ //BR_{26} 踏步对应的转换] (16)

我们知道,选择语句中各个条件元之间为析取关系,因此要证明命题 $Unit_{BCalF-R} \wedge \Box p \rightarrow \Box BR(C_2)$成立,只需要依次证明如下命题成立.这些命题比较简单,其正确性显然,具体证明这里从略.

(1) $BC_{21} \rightarrow (BR_{21} \vee BR_{24})$

(2) $BC_{22} \rightarrow BR_{22}$

(3) $BC_{23} \rightarrow BR_{26}$

(4) $BC_{24} \rightarrow BR_{23}$

(5) $BC_{25} \wedge p \rightarrow BR_{25}$ //变量 A 在函数 $BCal_{F-R}$ 运行过程中值不会改变,所以不需要恢复

(6) $BC_{26} \rightarrow BR_{26}$

因此 $Cal \propto_{br} BCal_{F-R}$.由规则 1 中的(III)和 $Cal|=Spec$,我们可以得到 $LF_{BCalF-R}'|=\Box((lb=\text{RETURN} \rightarrow Sum=\Sigma_{k\in 1,...,N}:A^2[k]) \vee (f=\$T)))$.此外,显然有 $LF'_{BCalF-R}|=\Box((lb=\text{RETURN} \rightarrow f=\$F)$,从而有 $LF'_{BCalF-R}|=\Box(lb=\text{RETURN} \rightarrow Sum=\Sigma_{k\in 1,...,N}:A^2[k])$,因此程序 $BCal_{F-R}$ 满足规范 $Spec$. □

5 结 语

本文基于 XYZ/E 对程序错误环境和恢复算法建模,通过容错转换给出程序在错误环境和恢复算法作用下对应的容错程序,从而可在 XYZ/E 框架下研究整个容错系统的性质.定义了两种与容错有关的程序求精关系:容错求精关系和向后恢复求精关系.基于这两种求精关系,可直接从无错程序满足的规范推导得出整个容错系统满足的一些性质.

与文献[4,5]所做工作相比,我们为错误环境建模采用的五元组模型不仅考虑了硬件错误对程序状态产生的影响,还考虑了错误对程序中各状态转换的屏蔽关系以及错误假设问题,因此模型更具有普遍性.此外,本文提出的向后恢复求精关系对应于文献[4,5]中采用的向后恢复程序(backward-recovery program)作用于顺序程序的特殊情况.相比而言,文献[4,5]基于程序的观察序列(observations)给出向后恢复程序的定义和证明框架;而本文通过引入过去时序算子$E,将定义在程序计算集合上的向后恢复求精关系转化为 XYZ/AE 程序的求精关系,从而可在统一的时序逻辑框架下进行验证.

在进一步的工作中,我们将对 XYZ/E 适当扩充以研究实时容错系统和分布式容错系统的描述和验证问题,

研究如何定义错误单调概念以及如何采用逐步求精方式设计容错系统.

References:

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

[2] Richard, K., Sam, T. Checkpointing and rollback-recovery for distributed systems. IEEE Transactions on Software Engineering, 1987,SE-13(1):23~31.

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

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

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

[6] Tang, Zhi-song. An introduction to XYZ system. ISCAS-XYZ-88-1, Institute of Software, The Chinese Academy of Sciences, 1988.

[7] Tang, Zhi-song. Temporal logical programming and software engineering. Beijing: Science Press, 1999(in Chinese).

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

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

[10] Manna, Z., Pnueli, A. Completing the temporal picture. Theoretical Computer Science, 1991,83:97~130.

附中文参考文献:

[7] 唐稚松.时序逻辑程序设计与软件工程.北京:科学出版社,1999.

To Specify and Verify Fault-Tolerant Systems in XYZ/E*

GUO Liang, TANG Zhi-song

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

E-mail: gls@ios.ac.cn

http://www.ios.ac.cn

Abstract: To specify and verify fault-tolerant systems in XYZ/E is discussed in this paper. Based on the corresponding state transition system of an XYZ/E executable program P, how to model its fault environment and obtain its fault affected program P_F by fault transformation is illustrated. With P, F, P_F and a recovery algorithm R, how to obtain the fault-tolerant program P_{F-R} by fault tolerant transformation is also illustrated. Furthermore, two kinds of refinement relationships between programs P and Q: fault-tolerant refinement and backward-recovery refinement are defined. Based on these two refinement relationships, some properties satisfied by program Q can be directly deduced from the specification of program P.

Key words: fault-tolerant system; temporal logic language XYZ/E; refinement; fault-tolerant transformation; specification; verification

* Received July 20, 2000; accepted July 6, 2001

Supported by the National High Technology Development 863 Program of China under Grant No.863-306-ZT02-04-01; the Key Sci-Tech Project of the National 'Ninth Five-Year-Plan' of China under Grant No.98-780-01-07-01