doi:10.3772/j.issn.1002-0470.2015.04.004

基于 TCSP 的实时并发系统测试方法①

郭 婧②* 徐中伟* 李丽梅**

(*同济大学电子与信息工程学院 上海 201804)

(**九江学院图书馆 九江 332005)

摘 要 基于时间通信顺序进程(TCSP)语言——一种用于建模、验证实时并发系统的形式化语言,进行了一种新型实时并发系统测试方法的研究,以提高测试的覆盖率和完整度。首先研究了实时并发系统的输入、输出一致性关系,然后在时间迹、时间拒绝两种框架下,分别定义了最小不满足时间迹、最小拒绝集合,分情况研究了其测试过程。在 TCSP 的稳定失效模型下,将输出事件加入拒绝事件集合,最后提出了基于通信顺序进程(CSP)精化关系,且辨别输入、输出事件的实时测试模型,因此测试时不仅能精确地表示系统属性及行为,而且能利用该语言的精化关系定义测试的一致性测试关系。

关键词 一致性测试,实时性,并发性,形式化方法,精化关系

0 引 言

形式化方法可用于软件测试,它是建立在严格的数学基础上的,有着精确的语法和语义,能够全面、完整地表现系统的行为和属性,有效地弥补传统测试的不足。时间通信顺序进程(timed communication sequential processes, TCSP)语言作为一种描述实时并发系统的常用形式化语言,定义了一种与一致性测试关系相似的精化关系,因此,基于该语言的测试方法是一个值得研究的方向。

文献[1]关注实时反应系统测试的自动化,通过结合软件、硬件构件来分析它们的行为,在着重于环硬件测试的基础上,提出了基于 TCSP 的测试方法。为实现变迁系统的测试算法,还可以借用 FDR 工具。但是文献[1]只研究基于 TCSP 的初步测试理论,且忽略了被测试系统与建立的 TCSP 语言模型之间的差异性。文献[2]对 TCSP 语言进行了深入研究,将事件分类为拒绝事件和被拒绝事件、高级事件和低级事件,这样可以更好地对安全组件进行建模,并更好地定义两种一致性测试关系,但这种测试方法没有应用 TCSP 的精化关系。文献[3]基于输入、输出标签变迁系统,将输入输出执行关系分类为输入关系、输出关系和静止关系,在 FDR 工具的基础上开发了产生并选择测试用例的工具。文献[4]针对通信顺序进程(CSP)模型的测试执行关系,在考虑到测试假设的前提下,介绍了一种新的测试关系,这种测试关系辨别了输入、输出表示形式的差别,讨论了输入、输出之间的执行关系。假设执行中能接受所有输入,且所有输入都能产生相应的输出,证明测试用例中根据这种测试关系,不会拒绝正确的执行关系。文献[5,6]提出了一种基于 CSP 的实时输入输出一致性关系,该关系适合分析数据流反应系统,并重用了精化检测[7]和 SMT 技术。因此,在一致性验证[8,9]的过程中,不需要执行特别的算法或处理复杂的数据结构。实时关系可以分为两种,第一种假设可以观察到延迟,第二种假设只有带限制的延迟可以被观察到。该一致性测试[10]关系

① 国家自然科学基金(61075002),国家科技支撑计划重大项目(2011BAG01B03)和863计划(2012AA112801)资助项目。

② 女,1987年生,博士生;研究方向:形式化验证,软件测试;联系人,E-mail:guo_jing163.com

(收稿日期:2014-10-14)

采用的是后一种关系,它定义了上限 k,在分析执行模型时能检测所有不正常的行为,且不需要处理操作模型[11]和建立新的算法。但该方法更适用于探索构件的性质,它没有考虑测试产生的策略。

研究中执行关系一般分为输入关系、输出关系、静态关系[12,13]。在实时理论中,静态关系被扩展加上一个限制 M,这个限制用来说明系统在到达静态状态前需要经历的时间长度。如果把静态关系看作一种特殊的输出,则它能更好地处理死锁性质。本文将输入、输出动作分割到各个频道中,对于输入频道所有输入或被接受,或被阻塞,对于输出频道静态关系都带限制常量。本文考虑了时间迹和时间拒绝,测试时间迹能发现不被允许的行为,而同时测试时间迹和时间拒绝能发现死锁行为。因此,在这两种情况下分别定义了最小不满足时间迹和最小拒绝接受集合,并定义测试的过程。测试的环境控制输入,被测试的系统控制输出,在稳定状态下输出不被允许。在 TCSP 的稳定失效模型下,将输出事件加入被拒绝事件集合,定义测试模型。

1　相关基本概念

定义 1　(TCSP 语言)

$P = STOP;\ SKIP;\ a \xrightarrow{d} Q;\ a@u \to Q;$

$A \to Q(x);\ Q_1 \overset{d}{\triangleright} Q_2;\ WAITd;$

$P\backslash A;\ Q_1 \square Q_2;\ Q_1 \prod Q_2;\ \prod_{i\in J} Q_i;$

$Q_1 {}_A\|_B Q_2;\ Q_1 \underset{A}{\|} Q_2;\ Q_1 \ |||\ Q_2 : f(Q);$

$Q_1 \Delta Q_2;\ Q_1\ \Delta_d Q_2;\ \mu x.\ P$

$STOP$ 表示死锁进程,时间仍然在不断增长。$SKIP$ 与 $STOP$ 相似,表示过程被终结,但是时间的增长被阻塞。$a \xrightarrow{d} Q$ 先执行 a, 经过 d 时间单位,接着执行进程 Q。$d = 0$ 时 $a \to Q$, 表示执行 a 后立即执行 Q。$a@u \to Q$ 中 u 表示时间变量,事件 a 发生的时间为 t, 在这个过程中时间变量 u 被 t 绑定,其他与 $a \to Q$ 相似。$A \to Q(x)$ 表示执行事件集合 A 中一个事件,执行后立即执行 $Q(x)$。$Q_1 \overset{d}{\triangleright} Q_2$ 在 d 个时间单位内执行 Q_1, 如果不执行 Q_1, 会执行进程 Q_2。$WAITd$ 不执行任何事件,经过 d 个时间单位,过程结束。$WAITd$ 等价于 $STOP \overset{d}{\triangleright} SKIP$。$P\backslash A$ 表示执行 P, 但事件集合 A 中所有事件被隐藏,且立即发生。$Q_1 \square Q_2$ 根据首个可见事件,执行 Q_1 或 Q_2, 且选择是确定的。$Q_1 \prod Q_2$ 非确定执行 Q_1 或 Q_2。$\prod_{i\in J} Q_i$ 则是多个进程的非确定执行。$Q_1 {}_A\|_B Q_2$ 中 Q_1 只执行集合 A 中的事件, Q_2 只执行集合 B 中的事件,两个进程都参与了集合 A、B 中的交集。两个并发进程的结合终结进程中的所有构件都终结。$Q_1 \underset{A}{\|} Q_2$ 表示两个进程除了集合 A 中的事件,其他事件都独立执行。$Q_1 \ ||| \ Q_2$ 表示两个进程 Q_1、Q_2 各自独立地执行,没有同步执行的事件。当进程 Q 执行事件 a, $f(Q)$ 则能执行 $f(a)$。而只要进程 Q 能执行, $f(Q)$ 则同时能执行内部事件。无论何时 $f(Q)$ 能执行事件 a, 则相应地 Q 能执行 $f(Q)^{-1}$。$Q_1 \triangle Q_2$ 表示 Q_2 能在任何时刻打断 Q_1 的执行, Q_2 的首个外部事件发生,控制权发生变化, Q_2 被丢弃。$Q_1\ \Delta_d Q_2$ 允许 Q_1 执行 d 个时间单位,接着执行 Q_2。$\mu x.\ P$ 表示递归程序。

定义 2　时间变迁系统(timed labelled transition system, TLTS)是一个四元数组 $\langle S, s_0, L_{\tau T}, \to\rangle$, 其中

(1) S 为非空状态集合, s_0 为初始状态。

(2) $L_{\tau T}$ 中包含外部动作、内部动作 τ、时间动作 $T \triangleq \{d \mid d \in IR \geqslant 0\}$。

(3) $\to \subseteq (S \times L_{\tau T} \times S)$ 为时间变迁关系,且满足以下性质:

如果 $s \xrightarrow{d} s_1$、$s \xrightarrow{d} s_2$, 则 $s_1 = s_2$。

$\exists s_1 : s \xrightarrow{d_1} s_1 \xrightarrow{d_2} s_2$ 成立当且仅当 $s \xrightarrow{d_1+d_2} s_2$。

$s \xrightarrow{0} s_1$ 成立当且仅当 $s = s_1$。

2　实时一致性输入输出测试关系

可以把动作集合分类,分成输入动作集合 L_I 和输出动作集合 L_U, 每个集合包含多个通道表示为 $L_I = \{L_I^1, \cdots, L_I^n\}$、$L_U = \{L_U^1, \cdots, L_U^m\}$。在时间变迁关系的基础上扩展拒绝动作关系和静态关系,对每个状态的拒绝动作集合为自循环变迁关系: $s \xrightarrow{A} s$, A 为状

态 s 的拒绝动作集合 $s \overset{A}{\rightarrow} s_1 \triangleq \forall \mu \in (A \cup \{\tau\}): (\neg \exists s \cdot s \overset{\mu}{\rightarrow}) \wedge s = s_1$，对输入动作 L_I^i，拒绝该动作可表示为 γ^i。静态关系为某一状态 s 在一定时间段内不执行任何输出动作 L_U^j，表示为 $\delta^j(s)$，当且仅当 $\forall \mu \in L_U^j: \forall d \in IR^+: \neg \exists s \cdot s \overset{\mu(d)}{\rightarrow}$，其中 $s \overset{\mu(d)}{\rightarrow}$ 为 $\exists s_1: s \overset{d}{\rightarrow} s_1 \overset{\mu}{\rightarrow}$ 的简化形式。将静态关系限定在某一频道 L_U^j、某一时间段 T_j，因此，静态关系被表示为 T_j-*quiescent*。由于内部动作不可见，被扩展的可见时间迹 $\sigma \in \bigcup_i \bigcup_j (T \cdot (L \cup \gamma^i \cup \delta^j))^*$，包含外部动作、拒绝动作、静态动作。可见集合分为两个部分，包含静态动作的输出动作集合和被拒绝的集合，具体定义如下：

设 S 为扩展时间输入输出变迁系统 $TIOTS(L_I, L_O)$ 的状态集合，则可见集合为

$$obs_M(S) = \bigcup_{s \in S} obs_M^o(s) \cup \bigcup_{s \in S} obs_M^r(s)$$

其中

$$obs_M^o(s) = \{\mu(d) \mid \mu \in L_U \wedge s \overset{\mu(d)}{\rightarrow}\} \cup \bigcup_j \{\delta(T_j) \mid s \overset{\delta(T_j)}{\rightarrow}\}$$

$$obs_M^r(s) = \bigcup_i \{\gamma^i(d) \mid \forall \mu \in L_I^i: \neg \exists s \cdot s \overset{\mu(d)}{\rightarrow}\}$$

设 P_1、P_2 为两个时间输入输出变迁系统，$P_1 \subseteq_{tmiorf}^{M} P_2$ 成立当且仅当任意时间迹 σ，$obs_M(P_1 after \sigma) \subseteq obs_M(P_2 after \sigma)$（系统 P_1 中的状态集合是系统 P_2 中状态集合的子集），$P after \sigma = \{p' \mid \exists p \in P: p \overset{\sigma}{\rightarrow} p'\}$。

测试用例看作实时输入输出变迁系统，测试就是运行测试用例和测试执行的并发系统，即 $TTEST(L_I, L_U) \times TMIOTS(L_I, L_U) \rightarrow TIOTS(L_I, L_U)$，具体关系如下：

由 $imp \overset{\tau}{\rightarrow} imp'$ 推出 $t \parallel imp \overset{\tau}{\rightarrow} t \parallel imp'$；

由 $t \overset{\delta_j}{\rightarrow} t'$ 推出 $t \parallel imp \overset{\delta_j}{\rightarrow} t' \parallel imp$；

由 $t \overset{\gamma_i}{\rightarrow} t'$、$\neg \exists \mu \cdot imp \overset{\mu}{\rightarrow} imp'$ 和 $\mu \in L_I^i$ 推出 $t \parallel imp \overset{\gamma_i}{\rightarrow} t' \parallel imp$；

由 $t \overset{\mu}{\rightarrow} t'$、$imp \overset{\mu}{\rightarrow} imp'$ 和 $\mu \in L$ 推出 $t \parallel imp \overset{\mu}{\rightarrow} t' \parallel imp'$；

由 $t \overset{d}{\rightarrow} t'$、$imp \overset{d}{\rightarrow} imp'$ 和 $d \in IR^{\geqslant 0}$ 推出 $t \parallel imp \overset{d}{\rightarrow} t' \parallel imp'$。

从中看出，运行测试用例必须和测试执行运行相同的动作和时间演化，测试才能执行同样的动作和时间演化，否则运行执行如果要运行，测试执行必须执行任何动作。

测试用例的产生被分为三步，即测试通过、输入动作的测试、输出动作的测试，这一过程循环反复。每种测试的不同情况用 + 号连接，具体情况如下：

（1）测试通过

$t := pass$

（2）输入动作的测试

$$\begin{aligned} t := & \sum \{o_j(d_j); t_j \mid o_j \in L_U \wedge d_j < T_B \\ & \wedge o_j(d_j) \in obs_T(S)\} \\ & + \{\mu(k); t_\mu \mid \mu \in L_I^i \wedge \exists s \in S: \\ & \gamma^i(k) \notin obs_T(S)\} \\ & + \{\mu(k); fail \mid \mu \in L_I^i \wedge \forall s \in S: \\ & \gamma^i(k) \in obs_T(S)\} \\ & + \{\gamma^i(k); fail \mid \mu \in L_I^i \wedge \gamma^i(k) \notin obs_T(S)\} \\ & + \{\gamma^i(k); t_{\gamma_i} \mid \mu \in L_I^i \wedge \gamma^i(k) \in obs_T(S)\} \\ & + \sum \{\delta^k(T_k); fail \mid T_k \in T \wedge T_k < T_B \\ & \wedge \delta^k(T_k) \notin obs_T(S)\} \\ & + \sum \{o_m(d_m); fail \mid o_m \in T \wedge o_m(d_m) \\ & \notin obs_T(S)\} \end{aligned}$$

（3）输出动作的测试

$$\begin{aligned} t := & \sum \{o_j(d_j); t_j \mid o_j \in L_U \wedge o_j(d_j) \in obs_T(S)\} \\ & + \sum \{\delta_j(T_j); t_{\delta_j} \mid \delta_j \in obs_T(SafterT_j)\} \\ & + \sum \{\delta_j(T_j); fail \mid \delta_j \notin obs_T(SafterT_j)\} \\ & + \sum \{\delta^k(T_k); fail \mid T_k \in T \wedge T_k < T_B \\ & \wedge \delta^k(T_k) \notin obs_T(S)\} \\ & + \sum \{o_m(d_m); fail \mid o_m \in T \wedge o_m(d_m) \\ & \notin obs_T(S)\} \end{aligned}$$

3 基于 TCSP 的测试

3.1 实时事件、时间迹和时间拒绝

实时事件用两部分组成，事件与事件发生的时

间,即 $R^+ \times \Sigma$。时间迹是以时间递增事件的序列,时间迹不仅需要记录执行的事件,而且需要记录进程执行的时间。内部事件无法在接口上可见,且无法作用于其他进程时间迹集合。时间迹集合包含所有有限时间事件序列和无限时间事件序列,符号表示如下:

$$TT = \{s \in (R^+ \times \Sigma^{\surd})^* \mid \forall t_1, t_2 : R^+; a_1, a_2 : \Sigma^{\surd} \cdot \langle (t_1, a_1), (t_2, a_2) \rangle \preceq s \Rightarrow t_1 \leqslant t_2 \wedge a_1 \neq \surd\} \cup \{s \in (R^+ \times \Sigma)^* \mid \forall t_1, t_2 : R^+; a_1, a_2 : \Sigma \cdot \langle (t_1, a_1), (t_2, a_2) \rangle \preceq s \Rightarrow t_1 \leqslant t_2 \wedge \forall t \in R^+ \cdot \exists t_1 > t; a_1 : \Sigma \cdot (t_1, a_1) ins\}$$

进程与时间迹之间的关系可以用执行 e 来表示,e 由进程、变迁交替构成。变迁关系有两类:如果 $\mu \in \Sigma^{\surd,\tau}$,则 $Q_i \xrightarrow{\mu_i} Q_{i+1}$,为动作变迁关系;如果 $d_i \in \mathbb{R}^+$,则 $Q_i \xrightarrow{d_i} Q_{i+1}$,为时间演化变迁关系。进程 e 中的第 i 个变迁为进程中第 $2i+1$ 个元素,用 $trans_e(i) = e@(2i+1)$ 表示。执行由时间 0 开始,当 $Q_i \xrightarrow{d_i} Q_{i+1}$ 时,时间发生变化。时间与变迁的关系如下:

$$time_e(0) = 0$$

$$time_e(i+1) = \begin{cases} time_e(i) & \text{如果 } trans_e(i) \in \Sigma^{\surd,\tau} \\ time_e(i) + trans_e(i) & \\ & \text{如果 } trans_e(i) \in \mathbb{R}^+ \cup \{\infty\} \end{cases}$$

执行 e 的长度为 $\#e$,包含 $\lfloor \#e/2 \rfloor$ 个变迁关系。时间迹与执行 e 的关系如下:

$$ttrace(e) = < (time_e(i), trans_e(i)) \mid i \leftarrow \langle 0, 1 \cdots \lfloor \#e/2 \rfloor - 1 \rangle, trans_e \in \Sigma^{\surd} >$$

从事件的角度来说,进程或运行事件,或拒绝运行事件。时间迹记录运行事件的执行,时间拒绝则记录被拒绝运行的事件,在确切的时间不可能运行的事件。时间拒绝相比不考虑时间的拒绝,最大的不同在于考虑执行的整个过程中的拒绝信息,而不是仅仅考虑执行的结束时拒绝的信息。时间拒绝集合 $\aleph$ 由时间事件组成,例如 (t,a) 表示从时间 t 开始的时间演化状态不可能发生变迁关系 a。时间拒绝集合可以表示成由时间间隔及在这个间隔内拒绝的事件集合构成的元素集合,将时间间隔表示成半开区间,时间的变化是由于时间演化变迁关系,时间演化变迁关系 $Q \xrightarrow{d} Q'$ 可表示为时间间隔 $[0,d)$ 中拒绝集合 A。拒绝集合中的元素为 $RTOK = \{[t_1, t_2) \times A \mid t1 \leqslant t < t2 \wedge A \subseteq \Sigma^{\surd}\}$,$R \in RTOK \Leftrightarrow \exists t_1 : \mathbb{R}^+; t_2 : \mathbb{R}^+ \cup \infty; A \subseteq \Sigma^{\surd} \cdot t_1 < t_2 \wedge R = \{(t,a) \mid t_1 \leqslant t < t_2 \wedge a \in A\}$,拒绝集合 $FRSET = \{\cup R \mid R \subseteq^{fin} RTOK\}$。执行与拒绝集合之间的关系为 $\forall (t, a) \in \aleph \cdot \exists i : \mathbb{N} \cdot time_e(i) \leqslant t < time_e(i+1) \wedge \neg (Q_e(i) \xrightarrow{a})$。每个执行 e 都有只有一个最大拒绝集合。时间失效 $(s, \aleph)$ 由时间迹与时间拒绝集合构成。

3.2　时间迹的测试

时间迹的精化表示 $ttrace(SUT) \subseteq ttrace(SPEC)$,测试系统的时间迹是否为规格的时间迹的子集。考虑最小不可执行的时间迹,即时间迹 $tt^{\wedge}(t,a)$,(t,a) 为时间迹中首个不属于规格时间迹的元素。测试用例产生分为三种情况:(1)时间迹测试通过,观察到的时间迹与规格中的时间迹相同,接着时间迹陷入死锁;(2)测试失效,观察到时间迹 $tt^{\wedge}(t,a)$,表示时间迹与规格中的时间迹不符;(3)测试无法判断,观察到部分时间迹 tt。时间迹的测试对于其中每个实时事件都给予判断,而最后的结果由需要测试的时间迹中最后一个判断的结果决定。

3.3　时间拒绝集合的测试

除了测试时间迹以外,在 TCSP 中,还需要测试时间拒绝。测试 SUT 是否拒绝被规格 SPEC 接收的事件集合。设规格中的时间迹为 $s \in ttraces(SPEC)$,而时间失效却 $(s, \aleph) \notin failure(SPEC)$,即 $\aleph$ 为被 SPEC 接受的事件集合。分情况对拒绝集合进行测试:(1)观察到时间迹 $s^{\wedge}a$,$(t,a) \in \aleph$,输出成功;(2)观察到时间迹 s,接着时间迹陷入死锁,输出失败即 failure;(3)观察到部分时间迹 s,接着陷入死锁,测试不完全。可接受集合 $\aleph$ 可能包含无限时间事件,上述检测需要对集合中每个事件一一测试,因此时间代价高、效率较低。因此,可以推导最小可接受集合,从而不必对整个集合进行测试,该集合定义如下:

$$\aleph_{\min} = \{\aleph \mid (t,a) \in \aleph \wedge \neg \exists (t', a') \cdot (t', a') \in \aleph' \wedge \aleph' \subset \aleph \wedge (s, \aleph) \notin failure(SPEC)\}$$

最小可接受集合不是唯一的,时间拒绝集合测试需要测试所有最小可接受集合,测试得到最后的结果不是 failure。测试时间失效则测试时间迹与时间拒绝,都通过则测试通过。

3.4 稳定时间失效的测试模型

测试时间迹不能分辨输入动作事件、输出动作事件,因此需要在时间失效下测试。环境控制输入事件,输出事件仅仅靠系统控制,与环境无关。因此,无法拒绝输出事件。将输出事件集合提前加入到拒绝集合,即 $TF_o(Q) = \{(s, \aleph) \mid (s, \aleph \cup \aleph_0) \in TF(Q)\}$。该定义类似于将输出集合隐藏作为内部事件集合,但不同于内部事件,在时间迹上仍然可见。重新定义 TCSP 的时间失效,具体定义如下:

$$TF_o[\![STOP]\!] = \{(\langle\rangle, \aleph) \mid \aleph \in RSET\}$$

(表示执行空集,进程停止)

$$TF_o[\![a \xrightarrow{d} Q]\!] = \{(\langle\rangle, \aleph) \mid a \notin \sigma(\aleph \cup \aleph_o)\} \cup \{(\langle (t,a)\rangle \hat{} s + t, \aleph) \mid t \in \mathbb{R}^+ \wedge a \notin \sigma((\aleph \cup \aleph_o) \uparrow [0,t)) \wedge (s, (\aleph \cup \aleph_o) - (t+d)) \in TF[\![Q]\!]\}$$

(表示不执行任何事件或经过延迟 d 个单位后执行进程 Q)

$$TF_o[\![a@u \to Q]\!] = \{(\langle\rangle, \aleph) \mid a \notin \sigma(\aleph \cup \aleph_o)\} \cup \{(\langle (t,a)\rangle \hat{} s + t, \aleph) \mid t \in \mathbb{R}^+ \wedge a \notin \sigma((\aleph \cup \aleph_o) \uparrow [0,t)) \wedge (s, (\aleph \cup \aleph_o) - t) \in TF[\![Q(t/u)]\!]\}$$

(表示与 $TF_o[\![a \xrightarrow{d} Q]\!]$ 类似,在此基础上增加了时间变量 u)

$$TF_o[\![x{:}A \to Q(x)]\!] = \{(\langle\rangle, \aleph) \mid A \cap \sigma(\aleph \cup \aleph_o) = \{\}\} \cup \{(\langle (t,a)\rangle \hat{} s + t, \aleph) \mid a \in A \wedge t \in \mathbb{R}^+ \wedge A \cap \sigma((\aleph \cup \aleph_o) \uparrow [0,t)) = \{\} \wedge (s, (\aleph \cup \aleph_o) - t) \in TF[\![Q(x)]\!]\}$$

(表示不执行 A 中任何事件或在时间 t 执行了 A 中事件,而后立即执行 $Q(x)$)

$$TF[\![Q_1 \overset{d}{\triangleright} Q_2]\!] = \{(s, \aleph) \mid begin(s) \leqslant d \wedge (s, \aleph) \in TF_o[\![Q_1]\!]\} \cup \{(s, \aleph) \mid begin(s) \geqslant d \wedge (\langle\rangle, \aleph \uparrow [0,d)) \in TF_o[\![Q_1]\!] \wedge (s, \aleph) - d \in TF_o[\![Q_2]\!]\}$$

(表示在 d 个单位内必须执行 Q_1,否则将 $(\langle\rangle, \aleph \uparrow [0,d))$ 加入 Q_1 的失效模型,表示在 d 时间单位内不执行该进程,则执行进程 Q_2。)

$$TF_O[\![WAIT\ d]\!] = \{(\langle\rangle, \aleph) \mid \surd \notin \sigma((\aleph \cup \aleph_o) \uparrow [0,d)) \cup \{(\langle (t,\surd)\rangle, \aleph) \mid t \geqslant d \wedge \surd \notin \sigma((\aleph \cup \aleph_o) \uparrow [d,t))\}$$

(表示在 d 个单位内执行空集,接着结束进程)

$$TF_o[\![Q_1 \prod Q_2]\!] = TF_o[\![Q_1]\!] \cup TF_o[\![Q_2]\!]$$

(表示执行 Q_1 或 Q_2)

$$TF_o[\![\prod_{i \in J} Q_i]\!] = \cup_{i \in J} TF_o[\![Q_i]\!]$$

(表示与 $TF_o[\![Q_1 \prod Q_2]\!]$ 类似,执行多个进程中的一个)

$$TF_o[\![Q_1 {}_A\|_B Q_2]\!] = \{(s, \aleph) \mid \exists \aleph_1, \aleph_2 \cdot (\aleph \cup \aleph_o) \uparrow (A \cup B)^{\surd} = ((\aleph_1 \cup \aleph_o) \uparrow A^{\surd}) \cup ((\aleph_2 \cup \aleph_o) \uparrow A^{\surd}) \wedge s = s \uparrow (A \cup B)^{\surd} \wedge (s \uparrow A^{\surd}, \aleph_1) \in TF_o[\![Q_1]\!] \wedge (s \uparrow B^{\surd}, \aleph_2) \in TF_o[\![Q_2]\!]\}$$

(表示只执行 A、B 中的事件,即 Q_1 只执行 $\boldsymbol{A}$ 中的事件, Q_2 只执行 $\boldsymbol{B}$ 中的事件)

$$TF_o[\![f(Q)]\!] = \{(f(s), \aleph) \mid (s, f^{-1}(\aleph)) \in TF_o[\![Q]\!]\}$$

(表示 $\boldsymbol{Q}$ 执行事件的映射 $\boldsymbol{f}$, $\aleph$ 只能执行 $f^{-1}(\aleph)$)

$$TF_o[\![Q_1 \triangle Q_2]\!] = \{(s_1 \hat{} s_2) \mid (s_1, \aleph \uparrow [0, begin(s_2))) \in TF_o[\![Q_1]\!] \wedge (s_2, \aleph \uparrow [0, begin(s_1 \uparrow \{\surd\}))) \in TF_o[\![Q_2]\!]\}$$

(表示在 Q_2 执行前的时间段执行 Q_1, Q_2 在任何时间内打断执行)

$$TF_o[\![Q_1 \triangle_d Q_2]\!] = \{(s_1 \hat{} (s_2 + d), \aleph) \mid end(s_1) \leqslant d \wedge (s_1, \aleph \uparrow [0,d)) \in TF_o[\![Q_1]\!]$$

$\wedge \surd \notin \sigma(s_1) \Rightarrow (s_2, \aleph - d) \in TF_o[\![Q_2]\!]\}$

(表示在 d 单位内都能执行 Q_1, 如若进程没结束,则执行 Q_2)

$TF_o[\![Q_1 \;|||\; Q_2]\!] =$

$\{(s, \aleph_1 \cup \aleph_2) \mid \exists s_1, s_2 \cdot s \; interleaves \; s_1, s_2 \wedge$

$(s_1, \aleph_1) \in TF_o[\![Q_1]\!] \wedge (s_2, \aleph_2) \in TF_o[\![Q_2]\!]$

$\wedge (\aleph_1 \cup \aleph_o) \uparrow \Sigma = (\aleph_2 \cup \aleph_o) \uparrow \Sigma\}$

(表示 Q_1、Q_2 都能执行所有的事件,独立执行没有交集)

$s \; intleaves \; s_1, s_2 =$

其中 $\forall t : \mathbb{R}^+ \cdot strip(s \uparrow [0,t)) \; interleaves$

$strip(s_1 \uparrow [0,t)), \; strip(s_2 \uparrow [0,t))$

$TF_o[\![Q_1 \underset{A}{\|} Q_2]\!] =$

$\{(s, \aleph_1 \cup \aleph_2) \mid \exists s_1, s_2 \cdot ssynch_A \; s_1, s_2 \wedge$

$(s_1, \aleph_1) \in TF_o[\![Q_1]\!] \wedge (s_2, \aleph_2) \in TF_o[\![Q_2]\!]$

$\wedge (\aleph_1 \cup \aleph_o) \backslash A^{\surd} = (\aleph_2 \cup \aleph_o) \backslash A^{\surd}\}$

(表示将集合 A 及结束事件从失效事件集合中去除, Q_1、Q_2 执行除集合 A 中的所有事件)

$ssynch_A s_1, s_2 =$

其中 $\forall t : \mathbb{R}^+ \cdot strip(s \uparrow [0,t)) \; synch_A$

$strip(s_1 \uparrow [0,t)), \; strip(s_2 \uparrow [0,t))$

4　结　论

目前,尚没有较成熟针对时间通信顺序进程(TCSP)语言测试的理论研究,本文以该语言为基础,研究实时系统的输入输出一致性测试关系,将带限制的静态关系看成是一种特殊的输出关系,在时间迹、时间拒绝的两种情况下分别研究它的测试过程。在假设环境不会阻塞输出的前提下,在输入、输出稳定失效模型的框架下,定义了一种新的实时测试模型。这种测试模型的理论基础是通信顺序进程(CSP)语言的精化关系,能够很好地辨别输入事件和输出事件,因此,实时并发系统的特性在模型中能够得到精确的体现,最终使测试结果更加全面。下一步任务是考虑测试模型与被测试系统的差异性,在安全性的框架下,不考虑系统的一些功能,模型必须能够将构件之间的关系表示出来,从而能够将构件失效造成系统失效这一关键过程完整体现,改进系统的测试模型。对 TCSP 语言的测试作出全面测试假设,并符号化,测试假设与测试模型最好能用统一符号体系表示,若不是同一符号体系,必须建立两者之间的转换规则,提高测试的覆盖率,并进一步研究 TCSP 的测试理论。

参考文献

[1] Peleska J, Siegel M. Test Automation of safety-critical reactive systems. *South African Computer Journal*, 1997, (19):53-77

[2] Schneider S. Concurrent and Real-time Systems: The CSP Approach. USA: Wiley Press, 2000. 334-350

[3] Nogueira S. Automatic Test Case Generation from CSP Specifications. Portuguese, 2005. 20-26

[4] Sidney N, Augusto S, Alexandre M. Guided test generation from CSP models. *Lecture Notes in Computer Science*, 2008,(5160): 258-273

[5] Gustavo C, Augusto S, Alexandre M. A CSP timed input-output relation and a strategy for mechanised conformance verification. *Lecture Notes in Computer Science*, 2013, (8144): 148-164

[6] Laura B, Ed B. Testing real-time multi input-output systems. In: Proceedings of the 7th International Conference on Formal Engineering Methods, Manchester, UK, 2005. 264-279

[7] Tomasz Mazur, Gavin Lowe. Counter abstraction in the CSP/FDR setting. *Electronic Notes in Theoretical Computer Science*. 2009, (250):171-186

[8] Moez K, Stavros T. Conformance testing for real-time systems. *Form Methods Syst Des*,2009, 34(3): 238-304

[9] Moez K, Stavros T. Black-box conformance testing for real-time systems. In: Proceedings of the 11th International SPIN Workshop, Barcelona, Spain, 2004. 109-126

[10] Ana C, Marie C. Testing for refinement in CSP. In: Proceedings of the 9th International Conference on Formal Engineering Methods, Boca Raton, FL, USA, 2007. 151-170

[11] Thomas G, Sabine G. An approach for machine-assisted verification of Timed CSP Specifications. *Innovations System Software Eng*, 2010,6(3):181-193

[12] Robert M, Kirill B. Using Formal Specifications to Support Testing. *ACM Computing Surveys*, 2009,41(2):9:

1-9:76

[13] Steve S. Abstraction and Testing in CSP. *Formal Aspects of Computing*, 2000,12(3): 165-181

Novel method for test of real-time concurrent systems based on TCSP

Guo Jing*, Xu Zhongwei*, Li Limei**

(*School of Electronics & Information Engineering, Tongji University, Shanghai 201804)

(**Library, Jiujiang University, Jiujiang 332005)

Abstract

Based on the timed communication sequential process (TCSP) language, a formal verification language for modeling and testing of real-time concurrent systems, the study was conducted, and a novel approach to testing real-time concurrent systems was proposed to improve the coverage and integrity of the testing. Firstly, the conformance relations of real-time systems between input and output were studied, and then the minimum unsatisfied timed trace and the minimum refused set were defined under the framework of timed trace and timed refuse. The next step, was to deal with the testing process in terms of different situations. Finally, under the stable failure model of TCSP, a real-time testing model based on CSP refinement was put forward by combining output events and refused events to identify input and output events. Thus the system properties and behaviors can be accurately presented when testing, and the conformance test relations can be defined by using the language's refinement.

Key words: conformance test, real-time, concurrency, formal method, refinement