一种形式化的软件可演化性特征描述方法

何 云[1] 王 炜[1,2] 李 彤[1,2]
(云南大学软件学院 昆明 650500)[1] (云南省软件工程重点实验室 昆明 650500)[2]

摘 要 软件系统的活性和安全性是判断软件可演化性的重要依据之一。现有方法多使用经典逻辑对系统的活性和安全性进行刻画。环境及涉众的复杂性使得软件的可演化性分析可能出现矛盾的输入。经典逻辑的无矛盾律导致其不能对软件系统的演化特性进行有效建模。针对该问题,提出了一种形式化的软件可演化性特征描述方法,该方法允许矛盾性输入的存在,可用于对软件可演化性等存在矛盾特性的系统进行建模和分析。该方法使用多值时序逻辑刻画软件系统的演化需求,同时提出了一种抽象软件模型对软件系统进行建模,通过抽象软件模型的活性和安全性来对软件系统的可演化特征进行描述。

关键词 活性,安全性,矛盾,软件可演化性,多值时序逻辑
中图法分类号 TP301 **文献标识码** A **DOI** 10.11896/j.issn.1002-137X.2017.07.024

Formal Method for Describing Software Evolution Ability Feature

HE Yun[1] WANG Wei[1,2] LI Tong[1,2]
(School of Software, Yunnan University, Kunming 650500, China)[1]
(Key Laboratory for Software Engineering of Yunnan Province, Kunming 650500, China)[2]

Abstract The liveness and safety are the most important basis for judging software systems evolution ability. The existing modeling methods use classical logic to describe the liveness and safety of systems. The complexity of the environment and stakeholders leads to conflicting input when analyzing software evolution ability. Because of the non-contradiction law, classical logic can't model the software system evolution ability effectively. Aiming at this problem, a formal method for describing the software evolution ability was proposed. The method allows contradictions, and can be used for modeling and analyzing systems with contradictions such as software evolution ability. The method uses multi-value temporal logic to describe software system evolution requirements, and proposes a software abstraction model to model software system. It can describe software system evolution ability by analyzing SAM's liveness and safety.

Keywords Liveness, Safety, Contradiction, Software evolution ability, Multi-value temporal logic

软件系统的活性(Liveness)和安全性(Safety)[1]是软件系统的两个基本属性。活性表示软件系统最终能够达到的预期的状态,即断言系统"好事终究会发生"。安全性表示系统不会进入异常状态,即断言系统"不会做坏事"[1]。系统可演化性分析试图给出基本演化意图实现过程是否破坏系统原有活性和安全性的判定方法。

目前,对系统活性和安全性的刻画和验证工作大多属于经典逻辑框架,常用的方法包含故障树[2]、有穷状态机[3-4]、Petri网[5-6]等。由于基于经典逻辑的方法具有许多良好的特性,如丰富的表达能力、自然而符合直觉意义的推理、行为良好的逻辑连接符,因此目前大多数对系统活性和安全性的分析均是基于经典逻辑的。然而,基于经典逻辑的模型检测方法[7-8]不允许存在矛盾的输入。

为了方便论述,给出一般意义上的矛盾概念。令 L 为一种$(T, \vdash)$类型的结构,其中 T 是一个有限集合,集合中的对象称为公式,用 $a,b,c,\cdots$ 表示,公式的集合称为理论(Theory);$\vdash$是一种理论和公式之间的关系,被称为推论关系;则 L 是一种逻辑。假设一个理论 T,假如存在公式 α 满足 $T\vdash\alpha$,$T\vdash\neg\alpha$,则 T 是矛盾的。由于我们从逻辑语言方面来定义矛盾性,因此从上面的定义可知,假如在经典逻辑框架中某软件模型或某个软件模型的集合能够同时满足某属性 p 和$\neg p$,则认为该软件模型或软件模型的集合是矛盾的。

当前的软件系统处于一个开放的、结构复杂的计算环境中,由于参与演化的各种角色对软件系统具有不同的责任和认识,因此对软件系统的安全性和活性往往存在矛盾的认识。按照经典逻辑平凡推理的观点以矛盾为前提可以推导出任何结论,这就意味着用含有矛盾的输入进行系统安全性、活性检测是毫无意义的。但矛盾的信息并不是毫无意义的,只是缺

到稿日期:2016-06-23 返修日期:2016-11-16 本文受国家自然科学基金(61462092,61262024,61379032),云南省自然科学基金重点项目(2015FA014),云南省自然科学基金(2013FB008),云南省教育厅研究生项目基金(2016YJS006)资助。
何 云(1989—),男,博士生,主要研究方向为软件工程和软件演化,E-mail:hey@ynu.edu.cn;王 炜(1979—),男,博士,副教授,主要研究方向为软件工程和软件演化;李 彤(1963—),男,博士,教授,博士生导师,主要研究方向为软件工程、软件演化过程和并行软件过程。

乏相应的检测系统来进行合理的处理。针对该问题，本文提出一种形式化的软件可演化性特征描述方法，使用多值时序逻辑(Multi-value Temporal Logic，MTL)对软件系统的演化需求信息进行刻画。在此基础上，提出一种软件系统的建模方法 SAM(Software Abstraction Model，SAM)。该方法不仅能够实现对软件系统的行为进行建模，而且能够描述系统中存在的矛盾信息。最后，给出使用 SAM 对软件演化系统的可演化性进行分析的算法。

1　时序逻辑

时序逻辑[9]是一种特殊的模态逻辑，包括线性时态逻辑、分支时态逻辑和计算树逻辑。本文所指的时序逻辑均是计算树逻辑[10-11]。由于时序逻辑提供 4 个时序算子(Xp：若谓词 p 在状态 s 的下一个状态为真，则 Xp 在状态 s 为真；Gp：若从 s 开始的将来所有状态(含 s)中谓词 p 都为真，则 Gp 在状态 s 为真；Fp：若从 s 开始的将来某个状态中谓词 p 为真，则 Fp 在状态 s 中为真；pUq：若谓词 q 在状态 s 中为真，或者它在从 s 开始的将来某个状态中为真，并且每一个中间状态谓词 p 为真，则 pUq 在 s 中为真)和 2 个路径量词(路径量词 A 表示所有路径，E 表示存在某一条路径；路径量词 A 可视为路径的全称量词，E 可视为路径的存在量词，因此可视路径量词 A 和 E 为一对对偶)，因此该系统非常适用于刻画、推理系统属性随时间的变化而变化的情况。

在时序逻辑中，时序算子的直观语义表示如图 1－图 4 所示。图中，空心圆表示状态，实心圆表示公式 p 在该状态下为真，而阴影圆表示在该状态下 q 为真，箭头表示状态转移关系。

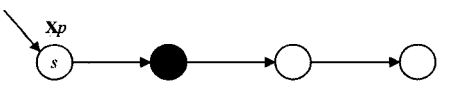

图 1　时序算子 X 的语义

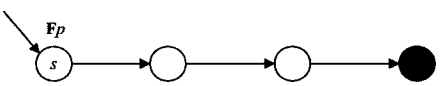

图 2　时序算子 F 的语义

图 3　时序算子 G 的语义

图 4　时序算子 U 的语义

软件系统的安全性、活性、优先权等属性非常适合使用时序逻辑进行刻画，例如系统的安全性可理解为“坏的事情从不发生”，可以用 Gp 来描述该性质；系统的活性可理解为“好的事情终将会发生”，可以用 Fp 来描述；系统的优先权可理解为描述事件发生的先后顺序，可以用 pUq 来描述。

传统的时序逻辑无法描述含有矛盾的信息，不适用于对软件系统的可演化性进行建模和分析。本文针对上述问题对时序逻辑系统进行扩展。具体而言，使时序逻辑能够最简单而又自然地描述矛盾信息的方法就是直接引入多个真值，即公式的取值除“真”、“假”以外还可以取其他真值。本文提出的多值时序逻辑通过引入不同的逻辑真值实现了对矛盾信息的刻画和推理。

2　多值时序逻辑系统的语法规则

由于多值时序逻辑(Multi-value Temporal Logic，MTL)系统是将多值逻辑和计算树逻辑相结合而产生的，因此按照传统习惯，沿用时序逻辑所提供的时序算子 X，G，F，U 以及路径量词 A，E。

设 P 为原子命题集合，按照下列规则可以定义状态公式和路径公式：

(1) $\forall p\in P$ 是一个状态公式，同时也是一个原子公式；

(2)若 p 和 q 是状态公式，则 $\neg p$，$p\vee q$ 和 $p\wedge q$ 也是状态公式；

(3)若 p 和 q 是状态公式，则 Xp，Gp，Fp 及 pUq 是路径公式；

(4)若 p 是路径公式，则 Ap 和 Ep 是状态公式。

令 Lan 为应用以上 4 条规则所产生的状态公式的最小集合，则 Lan 就是多值时序逻辑的语言，其中每一个元素称为公式。从以上规则可以看出，路径量词与时序算子必须成对出现才是状态公式，例如 A($p\vee$ Xq)就不是 MTL 公式。$\neg$，$\vee$，$\wedge$ 是 MTL 的逻辑连接词。尽管多值时序逻辑能够描述矛盾的信息，但这些逻辑符号都保留了大部分经典逻辑中良好的性质，这一点可以从语义的定义中得到。

定义 1(文字，literal)　形如 $\omega(\beta U\alpha)$，$\neg\omega(\beta U\alpha)$，$\omega\delta\alpha$ 以及 $\neg\omega\delta\alpha$ 的公式被称为文字，其中 $\beta,\alpha\in P$ 是原子命题，ω 表示路径量词 A 或者 E，而 δ 表示时序算子 X，G，U 或 F。

定义 2(析取子句，Clause)　令 $\alpha=ql_1\vee\cdots\vee ql_n$，其中 $n\geqslant 1$，且 $\forall i,1\leqslant i\leqslant n$，$ql_i$ 是文字，则称 α 为析取子句，简称子句。

多值逻辑中 De Morgan 率仍然成立，即 $\neg(p\wedge q)\equiv\neg p\vee\neg q$；$\neg(p\vee q)\equiv\neg p\wedge\neg q$。因此任意一个析取子句均可用合取子句和非进行表示。例如合取子句 $\alpha=ql_1\wedge\cdots\wedge ql_n$ 可以用析取子句表示为 $\alpha_1=\neg(\neg ql_1\vee\cdots\vee\neg ql_n)$。

3　多值时序逻辑的语义解释

多值时序逻辑是为了对含有矛盾信息的软件系统进行可演化性刻画、推理和证明而被提出来的，因此原来经典逻辑的真值观念就不合适于此。为了给出多值时序逻辑公式一个直觉上的意义，本文扩展了 Kripke 结构[12]作为解释多值时序逻辑的元结构，将通过多值时序逻辑语法规则生成的语言作为对象语言，通过明确元结构与对象语言之间的指称关系时限予以解释。

扩展 Kripke 结构是一个有向图，主要由 3 个要素组成：点、有向边和标记函数。点代表软件系统的“状态”，是系统瞬间的一个快照；有向边通常表示状态之间的转换关系，描述系统的行为、性质随时间的变化；标记函数将状态集合映射成描述状态属性的谓词集合。

元语言对对象语言进行语义解释，需要下述两个基本要素：1)需要确定解释的载体，这个载体是一个数学系统；2)需要一种确定的解释方法，从而把对象语言的符号和对象解释为载体中相应的元素的算法。

4 多值时序逻辑语义解释载体

本节从格和代数系统的角度考察多值时序逻辑语义解释载体。

定义 3 设$\langle L,<\rangle$是一个偏序集，其中 L 定义为真值集合。若对$\forall x,y\in L$，$\{x,y\}$都有最大下界和最小上界，则称偏序集$\langle L,<\rangle$构成一个多值格。

其中，真值集合 L 可以包含除了“真”、“假”两个真值元素以外的其他真值，如“可能”、“不知道”等。设任意 x,y 是格中的两个元素，由于$\{x,y\}$的最大下界和最小上界是唯一存在的，将$\{x,y\}$的最大下界记为 $x\vee y$，最小上界记为 $x\wedge y$。

当$|L|=4$时，其真值集合 L 由 4 个真值构成，即真(T)、假(F)、又真又假(B)及非真非假(N)。相应地，其论域集合可以记为真值集合 $L=\{T,F,B,N\}$，根据这些真值所表达的信息量的多少可以在该真值集合上定义一个偏序关系$<$。$<$被定义为 $B<F<N$ 以及 $B<T<N$，明显地，$\langle\{T,F,B,N\},<\rangle$是一个格，并且是完备格，其哈斯图如图 5 所示。可以在格上定义出相应的运算，例如 $\wedge$，$\vee$。表 1 列出了四值逻辑的真值表。

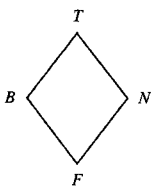

图 5 四值逻辑真值哈斯图

表 1 *Belnap* 四值逻辑的真值表

A	$\neg A$
T	F
F	T
B	B

$\wedge$	T	F	B	N
T	T	F	B	N
F	F	F	F	F
B	B	F	B	F
N	N	F	F	N

$\vee$	T	F	B	N
T	T	T	T	T
F	T	F	B	N
B	T	B	B	T
N	T	N	T	N

定义 4 多值时序逻辑的赋值关系 σ 是一个定义域为多值时序逻辑子句集合V、值域为真值集合 L 的映射，记为 $\sigma:V\to L$。赋值 σ 将使用多值时序逻辑语法规则产生的公式(子句)v映射成为真值集合 L 中的一个元素 $l\in L$，记为 $\sigma(v)=l$。

定义 5 设 P 是由多值时序逻辑语法规则产生的命题公式集合，如果将 P 中的$<$，$>$，$\wedge$，$\vee$分别替换成$>$，$<$，$\vee$，$\wedge$得到的命题集合为 P^*，称 P^* 为 P 的对偶命题，简称对偶。

例如：令 $P=\alpha\wedge\beta<\alpha$，那么 $P^*=\alpha\vee\beta>\alpha$。

定义 6 设$\langle L,<\rangle$是多值格，其中 $L=\{l_1,\cdots,l_n\}(n=1,\cdots,n,\cdots)$为非空真值集合，则绝对假定义为真值集合的最小元，即$\perp=\{l_1\wedge l_2\wedge\cdots\wedge l_n\}$。同理，绝对真定义为真值集合中的最大元，即 $\boldsymbol{T}=\{l_1\vee l_2\vee\cdots\vee l_n\}$。这里 $\wedge$，$\vee$ 分别定义为格上的求最小上界和最大下界算子。

从以上定义可知，$\perp$ 实际上是多值格$\langle L,<\rangle$上的最小元，$\boldsymbol{T}$ 是多值格上的最大元，例如在三值逻辑中 $L=\{T,M,F\}$，则$\perp=F,\boldsymbol{T}=T$；在四值逻辑系统中 $L=\{T,F,B,N\}$，则$\perp=F,\boldsymbol{T}=T$。

定义 7 设$\langle L,<\rangle$是多值格，$\perp$是真值集合 L 的绝对假，对于任意一个多值时序逻辑命题 p，若$\sigma(p)\in L-\{\perp\}$，则称命题 p 对真值集合 L 可接受，简称可接受。

定理 1 如果多值时序逻辑命题集合 P 对于一切多值格$\langle L,<\rangle$均是可接受的，则 P 的对偶命题集合 P^* 对一切多值格也是可接受的。

证明略。

定理 2 设$\langle L,\leqslant\rangle$是一个多值格，若$\forall a,b,c\in L$，有：

(1)$a\wedge b\leqslant a,a\wedge b\leqslant b$；

(2)$a\leqslant a\vee b,b\leqslant a\vee b$；

(3)$a\leqslant b$ 且 $a\leqslant c\to a\leqslant b\wedge c$；

(4)$a\geqslant b$ 且 $a\geqslant c\to a\leqslant b\vee c$。

证明略。

定理 3 设$\langle L,\leqslant\rangle$是一个多值格，在$\forall a,b\in L$时，有：

$a\leqslant b\leftrightarrow a\wedge b=a\leftrightarrow a\vee b=b$

证明略。

定义 8 多值时序代数系统是一个六元组 $MT=(V,\sigma,L,\wedge,\vee,\neg)$，其中：

(1)V 是使用多值时序逻辑语法规则产生的文字或子句集合。

(2)$\sigma:V\to L$ 是一个赋值关系。

(3)L 是真值集合。$\exists C\subseteq L$，$(C,\leqslant)$构成一个有限分配格，即$\forall\alpha,\beta\in C$，若 $\alpha\leqslant\beta$ 当且仅当 $\alpha\wedge\beta=\alpha$。

(4)$\wedge$，$\vee$ 分别代表格中求最大下界和最小上界的运算。

(5)$\neg$定义为有限分配格上的求补运算，即$\neg:L\to L$。任意 $\alpha\in L,b\in L$，有唯一$\neg a\in L$ 满足如下属性：1)$\neg(a\wedge b)=\neg a\vee\neg b$；2)$\neg a=a$；3)$\neg(a\vee b)=\neg a\wedge\neg b$；4)$a\leqslant b\leftrightarrow\neg a\geqslant\neg b$。

多值时序代数割裂了经典逻辑中命题与其非之间的关系，避免了平凡推理问题。对于一个基于经典逻辑的命题集合 T，如果 $a\in T$，则$\neg a\notin T$。但在多值时序逻辑系统中，a 和 $\neg a$ 可以同时属于理论系统 T。

定理 4 设多值时序代数系统 $MT=(V,\sigma,L,\wedge,\vee,\neg)$，则

(1)$\forall a,b\in V$，有：

$\sigma(a)\wedge\sigma(b)=\sigma(b)\wedge\sigma(a)$，$\sigma(a)\vee\sigma(b)=\sigma(b)\vee\sigma(a)$；

(2)$\forall a,b,c\in V$，有：

$(\sigma(a)\wedge\sigma(b))\wedge\sigma(c)=\sigma(a)\wedge(\sigma(b)\wedge\sigma(c))$

$(\alpha(a)\vee\sigma(b))\vee\sigma(c)=\sigma(a)\vee(\sigma(b)\vee\sigma(c))$

(3)$\forall a\in V$，有：

$\sigma(a)\wedge\sigma(a)=\sigma(a),\sigma(a)\vee\sigma(a)=\sigma(a)$

(4) $\forall a,b\in V$，有：

$\sigma(a)\wedge(\sigma(a)\vee\sigma(b))=\sigma(a),\sigma(a)\vee(\sigma(a)\wedge\sigma(b))=\sigma(a)$

证明略。

定理 4 说明多值时序代数系统对运算 $\wedge$，$\vee$ 满足交换律、结合律、等幂率和吸收率。从上面的证明可知，多值时序代数系统保留了很多经典逻辑的良好属性，同时也为证明多值时序逻辑推理系统的完备性和可靠性奠定了基础。

定义 9　若存在两个多值时序代数 $MT_1=(V_1,\sigma_1,L_1,\wedge_1,\vee_1,\neg_1)$ 和 $MT_2=(V_2,\sigma_2,L_2,\wedge_2,\vee_2,\neg_2)$，则 $MT=MT_1\times MT_2=(V,\sigma,L,\wedge,\vee,\neg)$，且满足以下条件：

(1) $V=V_1\times V_2$，$\forall\langle v_1,v_2\rangle\in V,\sigma(\langle v_1,v_2\rangle)=\langle\sigma_1(v_1),\sigma_2(v_2)\rangle$

(2) $\forall\langle v_1,v_2\rangle\in V,\neg\sigma(\langle v_1,v_2\rangle)=\langle\neg\sigma_1(v_1),\neg\sigma_2(v_2)\rangle$

(3) $\forall\langle v_1,v_2\rangle,\langle v_3,v_4\rangle\in V,\sigma(\langle v_1,v_2\rangle\wedge\langle v_3,v_4\rangle)=\langle(\sigma_1(v_1)\wedge_1\sigma_1(v_3)),(\sigma_2(v_2)\wedge_2\sigma_2(v_4))\rangle$

(4) $\forall\langle v_1,v_2\rangle,\langle v_3,v_4\rangle\in V,\sigma(\langle v_1,v_2\rangle\vee\langle v_3,v_4\rangle)=\langle(\sigma_1(v_1)\vee_1\sigma_1(v_3)),(\sigma_2(v_2)\vee_2\sigma_2(v_4))\rangle$

定理 5　两个多值时序代数的笛卡尔积也是多值时序代数。

分析：显然要证明两个多值时序代数系统的笛卡尔积也是一个多值时序代数，首先证明两个多值格的笛卡尔积也是一个多值格，即存在两个多值格 $\langle L_1,\leqslant\rangle,\langle L_2,\leqslant\rangle$，其笛卡尔积 $\langle L_1\times L_2,\leqslant\rangle$ 也是一个多值格。然后为了证明两个多值时序代数的笛卡尔积是一个多值时序代数，根据定义 8 可知需要证明以下 4 条性质成立：

(1) $\forall\langle v_1,v_2\rangle\in V,\neg\neg\sigma(\langle v_1,v_2\rangle)=\sigma(\langle v_1,v_2\rangle)$

(2) $\forall\langle v_1,v_2\rangle,\langle v_3,v_4\rangle\in V,\neg\sigma(\langle v_1,v_2\rangle\wedge\langle v_3,v_4\rangle)=\neg\sigma(\langle v_1,v_2\rangle)\vee\neg\sigma(\langle v_3,v_4\rangle)$

(3) $\forall\langle v_1,v_2\rangle,\langle v_3,v_4\rangle\in V,\neg\sigma(\langle v_1,v_2\rangle\vee\langle v_3,v_4\rangle)=\neg\sigma(\langle v_1,v_2\rangle)\wedge\neg\sigma(\langle v_3,v_4\rangle)$

(4) $\forall\langle v_1,v_2\rangle,\langle v_3,v_4\rangle\in V,\sigma\langle v_1,v_2\rangle\leqslant\sigma\langle v_3,v_4\rangle\leftrightarrow\neg\sigma\langle v_1,v_2\rangle\geqslant\neg\sigma\langle v_3,v_4\rangle$

证明略。

定理 5 说明两个多值时序逻辑代数的笛卡尔积仍然是一个多值时序代数。

在经典时序逻辑中采用 Kripke 结构，并针其作为语义解释结构，即将时序逻辑作为对象语言，Kripke 结构作为元语言。时序逻辑的语义定义为对象语言与元语言之间的指称关系。本文借鉴上述思想，扩展 Kripke 结构得到 E-Kripke 结构，并将其作为多值时序逻辑的语义解释语言，即元语言。

定义 10　设一个五元组 $K=\{S,s_0,R,I,MT\}$ 是一个扩展 E-Kripke 结构(Extended Kripke Structure)，当且仅当：

(1) S 是一个有限非空状态集合；

(2) $s_0\in S$ 定义为 *E-Kripke* 结构的初始状态；

(3) $R\subseteq S\times S$ 是一个多值变迁关系集合；

(4) $MT=(V,\sigma,L,\wedge,\vee,\neg)$ 是一个多值时序代数系统；

(5) $I:S\times 2^V\to L$ 是一个标记函数，该函数将状态集合与多值时序逻辑公式的幂集合的笛卡尔积映射成为多值集合 L。

从 E-Kripke 的定义可以看出，除了标记函数 I 外，它的其余结构与标准 Kripke 结构具有相似性。但正是因为标记函数的区别体现了该结构能够处理含有超协调信息数据的能力，避免了非平凡推理，同时该定义给出了多值时序逻辑在语义层次上的解释模型。$\forall P\in 2^V,\forall p\in P,\exists s\in S,l\in L$，则 $I(s)(p)=l$ 意味着多值时序逻辑公式 p 在状态 s 时可以取多值集合 L 中的一个真值 l。

在正式使用 E-Kripke 结构给出多值时序逻辑公式的语义前，需要首先基于定义 6 和定义 7 给出 E-Kripke 结构对多值时序逻辑公式的可满足概念。

定义 11　任意根据多值时序逻辑语法规则产生的公式 p 对一个 E-Kripke 模型 K 是可满足的，当且仅当 $\exists s\in S,I(s)(p)\in L-\{\perp\}$，记为 $(K,s)\vdash p$，否则称其为不可满足。

多值时序逻辑的可满足概念与经典逻辑的可满足概念显然存在着较大的差异。经典逻辑认为可满足性定义为：对于一个命题逻辑公式，是否存在对其变量的一个真值赋值，使得该命题逻辑公式成立。而本文中所定义的可满足性是指多值时序逻辑公式中 E-Kripke 结构的某状态上可取的真值属于 $L-\{\perp\}$。另外，从算法复杂度方面来说，经典逻辑可满足性判定算法的时间复杂度是指数级的；而在多值时序逻辑中存在时间复杂度为多项式的可满足性判定算法。

下面基于 E-Kripke 结构以递归的方式给出多值时序逻辑的语义解释。

定义 12　多值时序逻辑基于 E-Kripke 结构的递归语义解释为：

(1) 对于命题 p，当且仅当 $I(s)(p)\in L-\{\perp\}$ 时，$(K,s)\vdash p$；

(2) 对于命题 $p,q\in V$，$(K,s)\vdash p\wedge q$，当且仅当 $(K,s)\vdash p$，$(K,s)\vdash q$；

(3) $\alpha=\alpha_1\vee\alpha_2\vee\cdots\vee\alpha_n$ 是一个子句，$n\geqslant 1$。$(K,s)\vdash\alpha$ 当且仅当 $\exists i,1\leqslant i\leqslant n,\exists s\in S,(K,s)\vdash\alpha_i$；

(4) $(K,s)\vdash\mathrm{EX}\alpha$，当且仅当 $\exists t,t\in S,(s,t)\in R$ 且 $(K,t)\vdash\alpha$；

(5) $(K,s)\vdash\mathrm{AX}\alpha$，当且仅当 $\forall t,t\in S,(s,t)\in R$ 且 $(K,t)\vdash\alpha$；

(6) $(K,s)\vdash\mathrm{EF}\alpha$，当且仅当存在一条计算路径 $r=(s_0,s_1,\cdots)$，$\forall(s_i,s_{i+1})\in R$，其中 $i=\{0,1,\cdots\}$ 且 $\exists i(K,t)\vdash\alpha_i$；

(7) $(K,s)\vdash\mathrm{AF}\alpha$，当且仅当对所有从状态 S 开始的计算路径都满足 $(K,s)\vdash\mathrm{EF}\alpha$；

(8) $(K,s)\vdash\mathrm{E}(\alpha\mathrm{U}\beta)$，当且仅当 $(K,s)\vdash\beta$，或者存在一条计算路径 $r=(s_0,s_1,\cdots,s_n),n\geqslant 0,\forall i<n(K,s_i)\vdash\alpha$，且 $(K,s_n)\vdash\beta$；

(9) $(K,s)\vdash\mathrm{A}(\alpha\mathrm{U}\beta)$，当且仅当从任意状态 s 出发的计算路径均满足 $(K,s)\vdash\mathrm{E}(\alpha\mathrm{U}\beta)$；

(10) $(K,s)\vdash\mathrm{EG}\alpha$，当且仅当存在一条计算路径 $r=(s_0,s_1,\cdots,s_n),n\geqslant 0$，且 $\forall i,i\geqslant 0,(K,s_i)\vdash\alpha$；

(11) $(K,s)\vdash\mathrm{AG}\alpha$，当且仅当对所有从 s 开始的计算路径均满足 $(K,s)\vdash\mathrm{EG}\alpha$。

定义 13　$\alpha,\beta\in V,V$ 是由多值时序逻辑语法规则产生的公式集合，令 $\approx\subseteq V\times V,(\alpha,\beta)\in\approx$ 意味着对于每一个 E-Kripke 结构 K 和 $s\in S,\alpha,\beta$ 满足条件：$(K,s)\vdash\alpha$ 成立时，$(K,s)\vdash\beta$ 也成立，反之亦然。

定理 6 ≈是一个等价关系。

证明略。

5 系统抽象模型 SAM

为了讨论软件系统的活性和安全性，本节提出了一种软件系统的建模方法 SAM。该方法不仅能够实现对软件系统的行为进行建模，而且能够描述系统中存在的矛盾信息。

一般而言，软件行为是具有相对独立功能、可以明确辨识、与语境有明显依赖关系、可独立部署、可组装的状态集合。本节提出的 SAM 模型作为软件的形式化抽象元模型，不仅能够刻画软件的行为属性，而且能够刻画软件系统中的矛盾信息。

另外，还应该指出的是 SAM 模型是一个伪模型。所谓伪模型就是该模型不能作为一个真实系统的模型，而是对系统各种可能行为的模拟。它的建立是为了便于处理设计的中间过程所遇到的各类问题，同时利用该模型对目标系统进行必要的分析和验证。

定义 14 设一个七元组 $SAM=\{S,s_0,R,I,MT,\Sigma,\Delta\}$ 是一个 SAM 模型，当且仅当：

(1)S 是一个非空有限状态集合；

(2)$s_0\in S$ 定义为初始状态；

(3)Σ 是程序的输入字符集；

(4)Δ 是程序的输出字符集；

(5)$R\subseteq S\times(\Sigma\cup\Delta)\times S$ 是一个多值变迁关系集合；

(6)$MT=(V,\sigma,L,\wedge,\vee,\rightarrow)$ 是一个多值时序代数系统；其中，V 是根据多值时序逻辑语法规则产生的公式集合，σ 是赋值关系，L 是真值集合，$\wedge$，$\vee$ 分别代表格上的求最大上界和最小下界的运算，$\rightarrow$ 表示求反运算；

(7)$I:S\times 2^V\rightarrow L$ 是一个标记函数，该函数将状态集合 S 与公式集合 V 的幂集合的笛卡尔积映射成为集合 L 的多值集合。

从以上定义可以看出，SAM 模型和 E-Kripke 非常相似，都是以集合 S 作为顶点集合、R 集合作为边集合的有向图。由于 SAM 的论域是定义在多值集上的，因此其具有超协调性。也就是说一个多值时序逻辑公式 $v\in V$ 在状态 $s\in S$ 上可能同时满足 l 和 $\neg l$，$l\in L$。正如前文所述，SAM 并不是一个真实系统的反映，而是对所有可能状态的模拟。它允许目标系统含有不确定性，允许系统设计包括非协调的信息。从这个意义上来说，SAM 模型是一个伪模型。同时字符集合 Σ，Δ 分别表征了构件的输入字符集和输出字符集。利用这两个集合可对构件的接口规约行为进行定义。

定义 15 序列 $\langle s_{init},c_0,s_1,\cdots\rangle$ 是 SAM 的一个行为，当且仅当：

(1)$s_i\in S$ 表示构件的各个状态；

(2)$s_{init}=s_0$ 表示构件的初始状态；

(3)$c_i\in(\Sigma\cup\Delta)$ 表示构件的输入字符或构件内部的消息或输出字符；

(4)$\forall s_i\in\{s_{init},s_1,\cdots\},\exists s_{i+1}\in\{s_1,\cdots\},\exists c_i\in(\Sigma\cup\Delta),(s_i,c_i,s_{i+1})\in R$，其中 $i=\{0,1,2,\cdots\}$。

SAM 行为的定义通过状态间的转换关系，不仅刻画了软件系统的实际运行状况，也回答了任意一个状态在 SAM 模型中是否可达。对任意状态 $s\in S$，我们说状态 s 在系统 SAM 是可达的，当且仅当存在一个有穷序列 $\langle s_{init},c_0,s_1,\cdots,s_n\rangle$，其中 $s_n=s$。

图 6 是一个 SAM 模型，其中黑色方框表示状态，有向箭头表示状态转移关系，$s_0=s_{init}$，$\{Msg_1,Msg_2,Msg_3,Msg_4\}=\Sigma\cup\Delta$。而关于 R,I,MT 的定义被省略，但并不影响内容的表达。通过图 6 可知，SAM 模型的行为是 $\langle s_0,Msg_1,s_1,Msg_2,s_2,Msg_3,s_3,Msg_4\rangle^*$，其中 $*$ 表示闭包运算。

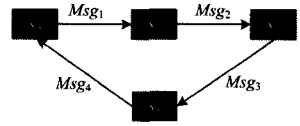

图 6 SAM 模型

由定义 14 和定义 15 可知，系统的状态转移关系构成了描述构件行为的工具，即系统状态和输入构成的输出字符集合是构件行为的形式化刻画。下面给出判定一个输入序偶是否为系统行为的算法。

算法 1 输入序偶判定算法

```
PROCEDURRE BehTest(α:MTL formula representing behavior,M:
SAM model)
BEGIN
  SEND "reset" to M;
  FOR each i in the length of α DO
    BEGIN
      IF ⟨s_i,c_i,s_{i+1}⟩∉R THEN
      RETURN False
    END
  RETURN Ture
END
```

算法 1 将行为 α 和 SAM 模型 M 作为输入，将 α 是否是 M 的行为的判断结果作为输出。若是，则算法返回 True；否则，返回 False。首先对构件发送“*rest*”信号，使构件模型 SAM 的当前状态处于初始状态，若每一个序偶均属于状态转移关系，则该行为属于 SAM 模型，否则不属于。

若行为 α 的长度为 n，即 $|\alpha|=n$。状态转移关系集合 R 的元素个数为 m，即 $|R|=m$，则算法 1 的时间复杂度为 $O(n*m)$。

6 SAM 模型的属性

通过扩展 E-Kripke 结构，本文提出了 SAM 模型作为构件系统的形式化抽象元模型，并且取得了超协调性。下面对 SAM 模型的属性进行讨论。

定义 16 给定一个公式 $\alpha\in V$，SAM 模型 $M=\{S,s_0,R,I,MT,\Sigma,\Delta\}$。对于 $s\in S$，$(M,s_0)\models\alpha$，当且仅当从初始状态 s_0 开始，状态 s 可达，且与 SAM 模型 M 相对应的 E-kripke 结构 EK 在状态 s_0 上语义满足公式 α，即 $(EK,s_0)\vdash\alpha$。其中 $\models$[1] 表示 SAM 模型上的可满足关系。

判定一个 SAM 模型是否是属性 α 的模型，需要两个步骤：1)判断状态 s_0 在 SAM 模型中是否可达，即是否存在一个

初始状态为 s_0 的行为，对状态 s 来说是可达的；2）判断 $(EK, s) \vdash \alpha$ 是否成立。根据经验，这样构造模型检测算法是不可接受的，不可能通过一个更加复杂的形式推演去实现一个模型检测问题。需要对 SAM 模型的性质进行更加细致的研究，以便为设计可行的模型检测算法奠定基础。

命题 1　令 $\alpha \in V, \alpha_1 \wedge \alpha_2 \wedge \cdots \wedge \alpha_n$ 是 α 的 CNF，其中 $n \geqslant 1$。对于 $s \in S, (M,s) \vdash \alpha$，当且仅当 $\forall i.\ 1 \leqslant i \leqslant n, (M,s) \vdash \alpha_i$。

证明：设 $\exists i.\ 1 \leqslant i \leqslant n, (M,s) \vdash \alpha_i$ 不成立，根据定义 16 可知，存在与 M 相对应的 E-kripke 结构 EK 在状态 s 上语义不满足 α_i，即 $(EK,s) \vdash \alpha_i$ 不成立。根据定义 10 和定义 12 可知，$l(s)(\alpha_i) = \perp$，由于 $\alpha = \alpha_1 \wedge \alpha_2 \wedge \cdots \wedge \alpha_n$ 可知 $\alpha = \perp$。与已知矛盾。

由于 $\alpha = \alpha_1 \wedge \alpha_2 \wedge \cdots \wedge \alpha_n$ 以及 $(M,s) \vdash \alpha$ 已知，根据定义 15 可知，存在与 M 相对应的 E-kripke 结构 EK 在状态 s 上满足 α，则可得 $(EK,s) \vdash \alpha_1 \wedge \alpha_2 \wedge \cdots \wedge \alpha_n$，即 $(EK,s) \vdash \alpha_i, i = \{1, 2, \cdots, n\}$，因此 $(M,s) \vdash \alpha_i$。

证毕。

命题 2　令 $c = l_1 \vee l_2 \vee \cdots \vee l_n$ 是一个子句，其中 $n \geqslant 1$。对于 $s \in S, (M,s) \vdash c$ 当且仅当 $\exists i.\ 1 \leqslant i \leqslant n, (M,s) \vdash l_i$。

证明：如果 $\exists i.\ 1 \leqslant i \leqslant n, (M,s) \vdash l_i$，容易得到 $(M,s) \vdash c$。下面用反证法给出必要性证明。

假设 $(M,s) \vdash c$，但是 $\forall i.\ 1 \leqslant i \leqslant n, (M,s) \vdash l_i$ 均不成立，这意味着存在与 M 相对应的 E-kripke 结构 EK 在状态 s 上语义不满足 l_i，即 $(EK,s) \vdash l_i$ 不成立，即 $l(s)(l_i) = \perp$，也就意味着 $c = l_1 \vee l_2 \vee \cdots \vee l_n = \perp$，这与已知相矛盾，故上述假设不成立，必要性得证。

证毕。

命题 3　令 $\alpha \in V$。对于 $s \in S$，可以得出以下结论：

(1) $(M,s) \vdash \mathrm{EX}\alpha$ 当且仅当 $\exists t, t \in S, c \in \{\Sigma \cup \Delta\}, (s,c,t) \in R, (M,t) \vdash \alpha$；

(2) $(M,s) \vdash \mathrm{EF}\alpha$ 当且仅当有一条计算路径 $(s_0, s_1, \cdots)$，其初始状态 s_0 为 s，满足 $\exists i, i \geqslant 1, (M,s_i) \vdash \alpha$；

(3) $(M,s) \vdash \mathrm{AX}\alpha$ 当且仅当任意一条计算路径 $(s_0, s_1, \cdots)$，其初始状态 s_0 为 s，满足 $\exists t, t \in S, c \in \{\Sigma \cup \Delta\}, (s,c,t) \in R, (M,t) \vdash \alpha$；

(4) $(M,s) \vdash \mathrm{AF}\alpha$ 当且仅当任意一条计算路径 $(s_0, s_1, \cdots)$，其初始状态 s_0 为 s，满足 $\exists i, i \geqslant 1, (M,s_i) \vdash \alpha$；

(5) $(M,s) \vdash \mathrm{E}(\alpha \mathrm{U} \beta)$ 当且仅当 $(M,s) \vdash \alpha$，或者存在一条计算路径 $(s_0, s_1, \cdots, s_n)$，其初始状态 s_0 为 s，满足 $\forall i, i < n, (M,s_i) \vdash \alpha$，且 $(M,s_n) \vdash \beta$；

(6) $(M,s) \vdash \mathrm{A}(\alpha \mathrm{U} \beta)$ 当且仅当 $(M,s) \vdash \alpha$，或者任意一条计算路径 $(s_0, s_1, \cdots, s_n)$，其初始状态 s_0 为 s，满足 $\forall i, i < n, (M,s_i) \vdash \alpha$，且 $(M,s_n) \vdash \beta$；

(7) $(M,s) \vdash \mathrm{EG}\alpha$ 当且仅当存在一条计算路径 $(s_0, s_1, \cdots)$，其初始状态 s_0 为 s，满足 $\forall i, i \geqslant 0, (M,s_i) \vdash \alpha$；

(8) $(M,s) \vdash \mathrm{AG}\alpha$ 当且仅当任意一条计算路径 $(s_0, s_1, \cdots)$，其初始状态 s_0 为 s，满足 $\forall i, i \geqslant 0, (M,s_i) \vdash \alpha$。

证明：对于结论(1)，根据定义 16 可知，$(M,s) \vdash \mathrm{EX}\alpha$ 意味着按照定义存在与 M 相对应的 E-kripke 结构 EK 在状态 s 上语义满足 α，即 $(EK,s) \vdash \mathrm{EX}\alpha$。显然由于存在 $\exists t, t \in S, c \in \{\Sigma \cup \Delta\}, (s,c,t) \in R, (M,t) \vdash \alpha$，因此意味着存在 $\exists t, t \in S, (s,t) \in R, (EK,t) \vdash \alpha$，可知 $(EK,s) \vdash \mathrm{EX}\alpha$ 成立。故 $(M,s) \vdash \mathrm{EX}\alpha$ 成立，即结论(1)的必要性得到了证明。

由于上述推理思想的反向证明也是成立的，即充分性也是成立的。因此结论(1)得证。

对于结论(2)，可以采用类似的方法给出相应的证明。

对于结论(3)，根据定义 16 可知，$(M,s) \vdash \mathrm{AX}\alpha$ 意味着存在与 M 相对应的 E-kripke 结构 EK 在状态 s 上语义满足 $\mathrm{AX}\alpha$，即 $(EK,s) \vdash \mathrm{AX}\alpha$。显然由于任意一条计算路径 $(s_0, s_1, \cdots)$，其初始状态 s_0 为 s，满足 $\exists t, t \in S, c \in \{\Sigma \cup \Delta\}, (s,c,t) \in R, (M,t) \vdash \alpha$，则意味着存在 $(s,t) \in R_{sk}, (EK,t) \vdash \alpha$，可知 $(EK,s) \vdash \mathrm{AX}\alpha$ 成立。故 $(M,s) \vdash \mathrm{AX}\alpha$ 成立，即结论(3)的必要性得到证明。

上述推理思想的反向证明也是成立的，即充分性也是成立的。因此结论(3)得证。

对于结论(4)，可以采用类似的方法给出相应的证明。

对于结论(5)，根据定义 16 可知，$(M,s) \vdash \mathrm{E}(\alpha \mathrm{U} \beta)$ 意味着存在与 M 相对应的 E-kripke 结构 EK 在状态 s 上语义满足 $\mathrm{E}(\alpha \mathrm{U} \beta)$，即 $(EK,s) \vdash \mathrm{E}(\alpha \mathrm{U} \beta)$。对于当前状态 s，其初始状态 s_0 为 s，$(M,s) \vdash \alpha$，则 $(EK,s) \vdash \alpha$；或者存在一条计算路径 $(s_0, s_1, \cdots, s_n)$，其初始状态 s_0 为 s，满足 $\forall i, i < n, (M,s_i) \vdash \alpha$，且 $(M,s_n) \vdash \beta$，则 $\forall i, i < n, (EK,s_i) \vdash \alpha$，且 $(EK,s_n) \vdash \beta$，则 $(EK,s) \vdash \mathrm{E}(\alpha \mathrm{U} \beta)$。故 $(M,s) \vdash \mathrm{E}(\alpha \mathrm{U} \beta)$ 成立，即结论(5)的必要性得到证明。

上述推理思想的反向证明也是成立的，即充分性也是成立的。因此结论(5)得证。

对于结论(6)，可以采用类似的方法给出相应的证明。

对于结论(7)，根据定义 16 可知，$(M,s) \vdash \mathrm{EG}\alpha$ 意味着存在与 M 相对应的 E-kripke 结构 EK 在状态 s 上语义满足 $\mathrm{EG}\alpha$，即 $(EK,s) \vdash \mathrm{EG}\alpha$。存在一条计算路径 $(s_0, s_1, \cdots, s_n)$，其初始状态 s_0 为 s，满足 $\forall i, i \geqslant 0, (M,s_i) \vdash \alpha$，则 $\forall i, i \geqslant 0, (EK,s_i) \vdash \alpha$，则 $(EK,s) \vdash \mathrm{EG}\alpha$。故 $(M,s) \vdash \mathrm{E}(\alpha \mathrm{U} \beta)$ 成立，结论(7)的必要性得到证明。

上述推理思想的反向证明也是成立的，即充分性也是成立的。因此结论(7)得证。

对于结论(8)，可以采用类似的方法给出相应的证明。

证毕。

上述的一些命题显示了 SAM 模型具有一些我们所期待的性质，这些性质显然对于经典的标注 Kripke 结构也是成立的，这使我们能最大限度地利用过去的各种模型检测技术和工具，从而设计出行为良好的 MTL 模型检测算法。同时，通过命题 1－命题 3 的证明可以清晰地认识到 SAM 模型不但能对构件系统进行建模，而且取得了超协调性，为下一步开展模型检测提供了基本框架。

从另一个角度来说，SAM 模型是对 E-Kripke 结构的良好继承。从定义 16 可以看出，一个多值时序逻辑公式 α 中

[1] 在定义 16 中符号 $\vdash$ 表示 SAM 模型的可满足关系。

SAM 模型 M 可满足的先决条件是：1)判断是否存在一个初始状态为 s_0 的行为，对状态 s 来说是可达的；2)判断 α 对于与 SAM 模型相对应的 E-Kripke 结构 EK 是否成立，即 $(EK, s) \vdash \alpha$ 是否成立。因此，定义 16 与定义 15、定义 14 和定义 12 有着密切的关联。从这一点也可从侧面再次证明本文所提出的理论是一脉相承的，具有较高的内聚度。

7 演化系统活性、安全性分析

可演化性是针对系统的活性和安全性进行判定的。活性表征了系统中的某特定状态 m 从初始状态 m_0 开始存在一条计算路径，使状态 m 可达。安全性表征了系统中某个特定状态 m 在任意计算路径中出现的次数不得高于某正整数 $k>0$。具有安全性的系统不会出现死锁现象。

本节围绕上述两个问题在提出相应定义的基础上，给出了各自的判定算法。

定义 17 称一个 SAM 模型 $M=\{S, s_0, R, I, MT, \Sigma, \Delta\}$ 是活的，当且仅当 $\forall s \in S, \exists \sigma \in \{\Sigma \cup \Delta\}^*, 1 \leqslant i \leqslant n-1, \sigma=\sigma_1 \circ \sigma_2 \circ \cdots \circ \sigma_n$ 使得从 s_0 开始，$\forall (s_i, \sigma_i, s_{i+1}) \in R$ 且 $s_n=s_0$。

系统的活性保证了任何一个状态都是可达的。

算法 2 系统活性判定算法

```
PROCEDURE CheckL(M:SAM model)
BEGIN
  FOR each n in |S| DO
  BEGIN
    result =FALSE
    FOR each k in |{Σ∪Δ}*| DO
    BEGIN
      str=σk
      IF(s0,str,sn)∈R THEN
      BEGIN
        result =TRUE
        BREAK
      END
    END
    IF result =FALSE THEN
    BEGIN
      RETURN FALSE
    END
  END
  RETURN TRUE
END
```

算法 2 判定了系统任意一个状态是否是可达的。系统是否可达表示为从初始状态开始是否存在一个计算路径，使得该状态可达。若每个状态可达则布尔变量 *result* 被置为 TRUE，若存在一个或多个状态不可达则 *result* 被置为 FALSE。显然算法 2 的时间复杂度由两个循环式来确定，即 $O(|S| \times |\{\Sigma \cup \Delta\}^*|)$。假设 $|S|$ 等于 n，$|\{\Sigma \cup \Delta\}|=m$，则 $|\{\Sigma \cup \Delta\}^*|=2^m-1$。故 $O(|S| \times |\{\Sigma \cup \Delta\}^*|)= O(n \times 2^m)$。

定义 18 SAM 模型 $M=\{S, s_0, R, I, MT, \Sigma, \Delta\}$，若 $\sigma \in L(M)$，记 $\#(t/\sigma)$ 为 t 在 σ 中出现的个数。

定义 19 称一个 SAM 模型 $M=\{S, s_0, R, I, MT, \Sigma, \Delta\}$ 是安全的，当且仅当 $\exists k>0, \forall \sigma \in L(M), \forall \sigma_i \in S, \#(\sigma_i/\sigma)<k$。

安全性保证了系统无死锁现象。

算法 3 系统安全性判定算法

```
PROCEDURE CheckFair(M:SAM model,k:int)
BEGIN
  lan=Language(M)
  FOR each n in |lan| DO
  BEGIN
    str= lani
    FOR each m in length(m) DO
    BEGIN
      IF #(strm/str)≥k THEN
      BEGIN
        RETURN FALSE
      END
    END
  END
  RETURN TRUE
END
```

算法 3 判定了系统是否是活的。系统是否是活的可以转化为是否存在一个状态在任意计算路径中出现的次数大于或等于某正整数 k。若存在这样的状态，则算法返回 FALSE，否则返回 TRUE。显然算法 3 的时间复杂度由两个循环式来确定，即 $O(|lan| \times length(m))$。假设 $|l|$ 等于 n，$length(m)=m$，则 $O(|lan| \times length(m))=O(n \times m)$。

8 可演化性分析

识别软件的可演化性是根据参与演化的各个角色对软件系统提出的演化需求或系统属性，使用某种形式化验证方法判定软件系统演化后的活性和安全性。该过程的核心问题有两个：1)如何对演化需求进行忠实地反映；2)如何构造一种形式化验证方法来判定软件系统演化后的活性和安全性。对于第一个问题，第 2－4 节提出的多值时序逻辑能较好地完成任务。对于第二个问题，使用第 5－6 节提出的 SAM 构建软件系统的模型，通过第 7 节的算法检索软件系统模型的状态空间，验证从当前状态开始是否存在一个状态满足使用 MTL 所刻画的系统属性。常见的软件演化行为包括构件的添加、删除和替换等，本节将分别对这 3 种演化行为的可演化性进行讨论。

8.1 构件可添加性

构件添加意图的目标是向正在运行的系统添加一个新构件，从而实现容错、负载均衡或新功能的加入。由于新构件的加入可能导致待演化子系统活性、公平性等性质的变化。因此，判定构件的可添加性实际上回答了添加一个子构件后系统的活性、安全性是否受到影响的问题，若受到影响，则该子构件对于待演化子系统来说是不可添加的。

算法 4 构件可添加性算法

```
PROCEDURE CheckADD(M1,M2:SAM model,k:int)
BEGIN
  SAM M={S,s0,R,I,MT,Σ,Δ}
  S=S1∪S2
```

$s_0 \subseteq \{s_{10} \cup s_{20}\}$
$\Sigma = \Sigma_1 \cup \Sigma_2$
$\Delta = \Delta_1 \cup \Delta_2$
$R \subseteq S \times (\Sigma \cup \Delta) \times S$
$I: S \times 2^V \rightarrow L$
$MT = (V, \sigma, L, \wedge, \vee, \rightarrow)$
$V = V_1 \cup V_2$
IF not CheckL(M) THEN
BEGIN
  RETURN FALSE
END
IF not CheckF(M,k) THEN
BEGIN
  RETURN FALSE
END
RETURN TRUE
END

8.2 构件可删除性

构件删除意图的目标是从正在运行的系统中删除一个构件。被删除的构件成为目标构件。目标构件的删除可能导致事务被中断与环境引用错误等问题。因此,判定构件的可删除性实际上回答了删除一个构件后系统的活性、安全性是否受到影响的问题,若受到影响,则该构件对于待演化子系统来说是不可删除的。

算法 5　构件可删除性算法

PROCEDURE CheckDelete(M,M_1:SAM models,k:int)
BEGIN
  $S = S - S_1$
  $s_0 = s_0 - s_{10}$
  $\Sigma = \Sigma - \Sigma_1$
  $\Delta = \Delta - \Delta_1$
  $R \subseteq S \times (\Sigma \cup \Delta) \times S$
  $I: S \rightarrow V \rightarrow L$
  $MT = (V, \sigma, L, \wedge, \vee, \rightarrow)$
  $V = V - V_1$
  IF not CheckL(M) THEN
  BEGIN
    RETURN FALSE
  END
  IF not CheckFair(M,k) THEN
  BEGIN
    RETURN FALSE
  END
  RETURN TRUE
END

8.3 构件可替换性

构件替换意图的目标是在系统运行时刻用一个新创建的构件替换系统中原有的构件。新旧构件具有相同的接口,但可能具有不同的实现或物理位置。系统中原有构件被称为旧目标构件,新创建的构件被称为新目标构件。构件替换是构件添加和删除操作的结合,即将旧目标构件从系统中删除后添加新目标构件进入原系统。因此,构件可替换性判定实际上包含了两部分内容,即可删除性和可添加性判定。判定构件的旧目标构件可替换性实际上回答了替换一个构件后系统的活性、安全性是否受到影响的问题,若受到影响,则该构件对于待演化子系统来说是不可替换的。

算法 6　构件可替换性算法

PROCEDURE CheckSubstitute(M,M_1,M_1:SAM models,k:int)
BEGIN
  IF not CheckDelete(M,M_1,k) THEN
  BEGIN
  RETURN FALSE
  END
  SAM M=$\{S, s_0, R, I, MT, \Sigma, \Delta\}$
  $S = \{S - S_1\} \cup S_2$
  $s_0 = \{s_0 - \{s_{10}\}\} \cup \{s_{20}\}$
  $\Sigma = \{\Sigma - \Sigma_1\} \cup \Sigma_2$
  $\Delta = \{\Delta - \Delta_1\} \cup \Delta_2$
  $R \subseteq S \times (\Sigma \cup \Delta) \times S$
  $I: S \times 2^V \rightarrow L$
  $MT = (V, \sigma, L, \wedge, \vee, \rightarrow)$
  $V = \{V - V_1\} \cup V_2$
  IF not CheckADD(M,M_1,k) THEN
  BEGIN
    RETURN FALSE
  END
  RETURN TRUE
END

8.4 案例分析

本节将给出一个具体例子来说明本文提出的软件可演化性分析方法的可行性。为节省篇幅,以构件可添加性为例。

8.4.1 待演化系统

一个汽车租赁系统包含 3 个典型构件:汽车构件(Car)、租车业务构件(CarRental)和维修服务构件(CarRepair)。

8.4.2 待演化系统的 SAM 模型

汽车构件 $Car = \{S_{car}, s_{car0}, R_{car}, I_{car}, MT_{car}, \Sigma_{car}, \Delta_{car}\}$ 可定义为:

(1)S_{car}=\{库存,预定,提供服务\}。库存状态表示汽车存放在仓库中没有被租赁出去;预定状态表示由于客户发出租赁请求,满足用户需求的汽车进入的预留状态;提供服务状态表示汽车按照租赁合同为客户提供的服务。

(2)S_{car0}=库存。在本案例中将库存状态设置为系统的初始状态。

(3)R_{car}=\{〈库存,用户订车请求,预定〉,〈预定,提车单,提供服务〉,〈提供服务,付款证明〉\}表示汽车构件各状态之间的转换关系。

(4)$MT_{car} = (V_{car}, \sigma_{car}, L, \wedge, \vee, \rightarrow)$,其中 V_{car}=\{车况好,是否预定,定金已付,提车\},$L = \{T, F, M\}$ 可以看出 L 是一个典型的三值真值集合。$\wedge$,$\vee$ 分别代表三值逻辑上的求最大下界和最小上界运算符。其中 $T \wedge F = F$, $T \vee F = T$, $T \wedge M = M$, $T \vee M = T$, $F \wedge M = F$, $F \vee M = M$, $\rightarrow: L \rightarrow L$。其中 $\rightarrow T = F$, $\rightarrow F = T$, $\rightarrow M = M$。

(5)$I: S \times 2^V \rightarrow L$ 是一个标记函数,其中 I(库存)(车况好)$= T$, I(库存)(是否预定)$= F$, I(预定)(车况好)$= T$, I(预

定)(定金已付)=M,I(预定)(提车)=F,I(提供服务)(是否预定)=T,I(提供服务)(定金已付)=M,I(提供服务)(提车)=T。

(6)Σ_{car}=｛用户订车请求｝输入信息。

(7)Δ_{car}=｛提车单,付款证明｝输出信息。

8.4.3 可演化性分析

演化需求:需要在预定状态后增加一个租车历史记录查询状态。通过对该状态实施查询可以得到用户的信用等级,依据信用等级来决定是否租车给客户或是决定租金的多少。

分析:增加一个与汽车构件连接的构件"用户历史信息构件(CustomerInformation)"。该构件接收"预定申请表"作为构件接收信息,发布"用户历史信息"作为发送信息。由于构件"汽车"的输出字符集合与"用户历史信息"构件的输入字符集合的交集不为空,因此可以进行合成操作。同理,"用户历史信息"构件与"租车业务"构件可进行合成操作。合成操作实际上模拟了构件的添加过程。

$Car \oplus CustomerInformation \oplus CarRental \oplus CarRepair = \{S, s_0, R, I, MT, \Sigma, \Delta\}$,其中 S=｛库存,预定,查询用户信息,提供服务,租车,还车,修车｝,s_0=库存,R=｛〈库存,用户订车请求,预定〉,〈预定,提车单,提供服务〉,〈提供服务,付款证明,库存〉,〈租车,｛车况报告,费用明细｝,还车〉,〈修车,费用明细,还车〉｝,I(库存)(车况好)=T,I(库存)(是否预定)=F,I(预定)(车况好)=T,I(预定)(定金已付)=M,I(预定)(提车)=F,I(提供服务)(是否预定)=T,I(提供服务)(定金已付)=M,I(提供服务)(提车)=T,I(租车)(费用已付)=F,I(租车)(正在使用)=T,I(租车)(产生车况报告)=T,I(还车)(费用已付)=T,I(还车)(正在使用)=F,I(修车)(费用已付)=F,I(修车)(车况好)=F,I(修车)(进行维修)=T,Σ=｛用户订车说明｝,Δ=｛提车单,付款证明,车况报告,费用明细,交款证明｝。

调用算法 4 分析构件的可添加性。根据图 7 可知,系统任意一个状态均是可达的,因此系统的活性得到了保证。同时,系统不存在死锁状态,因此系统的安全性得到保证。因此构件"用户历史信息"是可添加的。

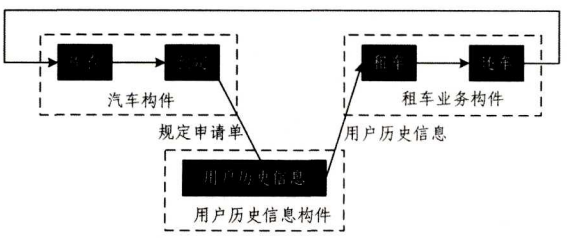

图 7 系统演化后的功能描述

结束语 本文构建了一种形式化的软件可演化性分析特征描述方法。该方法使用多值时序逻辑刻画软件系统的演化需求,同时将软件系统以 SAM 的形式进行抽象。通过实施演化行为后 SAM 模型的活性和安全性的变化情况来分析软件的可演化性。简而言之,本文方法可表述为:对于软件系统 M,在对其进行某个演化活动后其演化成为软件系统 M',对 M' 的活性和安全性进行判定,若 M' 保持了系统的活性和安全性,则认为该演化活动是可行的,反之则是不可行的。

参 考 文 献

[1] LAMPORT L. Proving the correctness of multiprocess programs[J]. IEEE Transactions on Software Engineering, 1977, SE-3(2): 125-143.

[2] WANG S Q, HUANG Z Q, HUANG C L, et al. Method Based on State/Even Fault Tree for Safety Analysis of Software[J]. Journal of Chinese Computer Systems, 2016, 37(1): 12-17. (in Chinese)
王思琪,黄志球,黄传林,等. 一种基于状态事件故障树的软件安全性分析方法研究[J]. 小型微型计算机系统,2016,37(1):12-17.

[3] SKELIN M, WOGNSEN E R, OLESEN M C, et al. Model checking of finite-state machine-based scenario-aware dataflow using timed automata[C]// 10th IEEE International Symposium on Industrial Embedded Systems (SIES), 2015. Piscataway, NJ: IEEE, 2015: 1-10.

[4] BELLI F, BEYAZLT M, ENDO A T, et al. Fault domain-based testing in imperfect situations: a heuristic approach and case studies[J]. Software Quality Control, 2015, 23(3): 423-452.

[5] UZAM M, LI Z, ZHOU M C. Identification and elimination of redundant control places in petri net based liveness enforcing supervisors of FMS[J]. International Journal of Advanced Manufacturing Technology, 2007, 35(1): 150-168.

[6] BOUDI Z, KOURSI E, COLLART-DUTILLEUL S. Colored Petri Nets formal transformation to B machines for safety critical software development[C]// International Conference on Industrial Engineering and Systems Management, 2015. Piscataway, NJ: IEEE, 2015: 12-18.

[7] WANG X B, DUAN Z H. Model Checking for Temporal Logic Programs[J]. Computer Science, 2009, 36(10): 164-167. (in Chinese)
王小兵,段振华. 时序逻辑程序的模型检测[J]. 计算机科学,2009,36(10):164-167.

[8] XU C, LIU J F, SUN J G. Security Protocol Model Checking Based on Classcial Logic[J]. Computer Science, 2008, 35(6): 20-24. (in Chinese)
徐畅,刘吉锋,孙吉贵. 基于经典逻辑的安全协议模型检测方法[J]. 计算机科学,2008,35(6):20-24.

[9] PNUELI A. The temporal logic of programs[C]// 18th Annual Symposium on Foundations of Computer Science, 1977. Piscataway, NJ: IEEE, 1977: 46-57.

[10] EMERSON E A, HALPERN J Y. Sometimes and not never revisited: on branching versus linear time temporal logic[J]. Journal of the ACM, 1986, 33(1): 151-178.

[11] QIAN J Y, XU B W. Model Checking CTL Based on Complete Abstraction Interpretation[J]. Chinese Journal of Computers, 2009, 32(5): 992-1001. (in Chinese)
钱俊彦,徐宝文. 基于完备抽象解释的模型检验 CTL 公式研究[J]. 计算机学报,2009,32(5):992-1001.

[12] KRIPKE S. Semantic considerations on modal logic[J]. Acta Philosophica Fennica, 1963(16): 83-94.