QRDChecker:一个 QRDC 模型检验工具*

裴 玉[1+], 徐启文[2], 李宣东[1], 郑国梁[1]

[1](南京大学 计算机科学与技术系,江苏 南京 210093)

[2](澳门大学 科技学院,澳门)

QRDChecker: A Model Checking Tool for QRDC

PEI Yu[1+], XU Qi-Wen[2], LI Xuan-Dong[1], ZHENG Guo-Liang[1]

[1](Department of Computer Science and Technology, Nanjing University, Nanjing 210093, China)

[2](Faculty of Science and Technology, University of Macau, Macau, China)

+ Corresponding author: Phn: +86-25-83593671, Fax: +86-25-83594683, E-mail: peiyu@must.edu.mo, http://www.nju.edu.cn

Received 2003-06-20; Accepted 2004-03-29

Pei Y, Xu QW, Li XD, Zheng GL. QRDChecker: A model checking tool for QRDC. *Journal of Software*, 2005,16(3):355−364. DOI: 10.1360/jos160355

Abstract: A reactive system does not terminate and its behaviors are typically defined as a set of infinite sequences of states. In formal verification, a requirement is usually expressed in a logic, and when the models of the logic are also defined as infinite sequences, such as the case for LTL (linear temporal logic), the satisfaction relation is simply defined by the set containment. However, this satisfaction relation does not work for interval temporal logics, where the models are finite sequences. In fact, for different interval based properties, different satisfaction relations are sensible. Two classes of finitary properties are identified, and then two satisfaction relations are defined for them, which are unified by a general relation. A model checking algorithm is proposed and implemented in a verification tool for QRDC (quantified RDC (restricted duration calculus)), which is an interval temporal logic. The tool QRDChecker can check the validity of QRDC formulae under both continuous and discrete interpretations. Moreover, for discrete QRDC, it can also translate the formulae into an automaton in the form accepted by the Spin model checking system, which can be subsequently used to verify a reactive system against properties expressed in the logic.

Key words: model checking; finitary property; reactive system; interval temporal logic

摘 要: 反应式系统通常是不终止的,其行为定义为系统状态的无限序列的集合.形式化验证时,检验需求一般使用时序逻辑给出.当使用诸如 LTL(linear temporal logic)这样的逻辑时,由于这类逻辑的模型同样是无限序列,系统与需求之间的满足性关系可以简单定义为集合的包含关系.但是,当使用时段时序逻辑(interval temporal

* Supported by the National Natural Science Foundation of China under Grant No.60233020 (国家自然科学基金); the National Grand Fundamental Research 973 Program of China under Grant No.2002CB312001 (国家重点基础研究发展规划(973))

作者简介: 裴玉(1977−),男,江苏盐城人,博士生,主要研究领域为形式方法,模型检验;徐启文(1963−),男,博士,助理教授,主要研究领域为形式方法;李宣东(1963−),男,博士,教授,博士生导师,主要研究领域为软件工程;郑国梁(1937−),男,教授,博士生导师,主要研究领域为软件工程.

logic)作为说明逻辑时,由于逻辑模型的有限性,使得上面的满足关系不再适用.称这类有限序列集合表达的性质为有限性性质.对于不同的有限性性质,它们对应的满足性关系是有区别的.针对两类有限性定义了它们各自的满足性关系,并将这两种关系统一为一个更一般的满足性关系.在此基础上,提出模型检验这两类性质的算法,并将其实现为一个针对时段时序逻辑 QRDC(quantified RDC (restricted duration calculus))的检验工具 QRDChecker.QRDChecker 可以检验 QRDC 公式在连续时间模型和离散时间模型下的有效性.在离散时间条件下,它还可以将 QRDC 公式转换成模型检验系统 Spin 能够接受的自动机的形式,从而可以检查反应式系统是否满足用 QRDC 公式表达的性质.

关键词: 模型检验;有限性性质;反应式系统;时段时序逻辑
中图法分类号: TP301 文献标识码: A

反应式系统是实时系统中的一类,这类系统通常不会终止,它们的语义一般定义为系统状态的无限序列的集合.针对这类系统,目前应用广泛的说明逻辑是线性时序逻辑 LTL(linear temporal logic),LTL 公式的模型是状态的无限序列.在这种情况下,反应式系统与 LTL 公式之间的满足性关系可以简单地定义为集合的包含关系,即系统满足公式描述的性质当且仅当系统的所有行为都是公式的模型.ITL(interval temporal logic)作为一种时段时序逻辑,虽然并不像 LTL 那样应用广泛,但是它可以更方便地表达一些特定的性质[1].在几乎所有的时段时序逻辑及其扩展中,时段都被认为是有限的,逻辑公式的模型因而也都是有限的.时段时序逻辑公式的所有模型构成一个有限状态序列的集合,称为有限性性质.这样,在使用时段时序逻辑来描述反应式系统的性质时,由于系统行为的无限性和性质模型的有限性,满足性关系将不再简单地只是集合的包含关系.一种可能的,也是常用的定义为:一个无限长的行为满足一个有限性性质当且仅当该行为所有的有限前缀都满足该性质.这样的定义对某些性质(例如 safety[2]性质)是适合的,而对另一些性质(例如 liveness[2]性质)则并不恰当[3],这与具体的性质有关.

经典的 safety 性质和 liveness 性质是针对无限模型提出来的,我们给出了这两类性质在有限模型下的版本[3],分别称为 FS 性质和 EP 性质.我们对这两类有限性性质作了研究,给出了它们各自相应的满足性关系及更为一般的,同时适用于这两类性质的统一定义.针对这两类性质,我们提出了自动检验系统是否满足该性质的方法.检验算法首先由性质出发构造相应的 Büchi 自动机,然后通过检查反应式系统行为模型的集合与这个 Büchi 自动机的语言之间的包含关系来确定系统是否满足该性质.

在此基础上,我们采用 QRDC(quantified RDC(restricted duration calculus))作为说明语言,设计并实现了模型检验工具 QRDChecker.QRDC 作为一种基于时段的逻辑,其中包括对系统停留在某一状态上的累计时间研究,并引入了作用在状态变量上的量词.QRDC 公式表达的性质由其在离散时间条件下的模型集合来确定.QRDChecker 可以作为 Spin 的前端,将表达有限性性质的 DQRDC 公式转换成时序声明(temporal claim),然后该声明可以作为 Spin 的输入用于模型检验反应式系统是否满足该性质.另外,在文献[4]工作的基础上,QRDChecker 中还实现了对 QRDC 公式在离散及连续时间条件下的有效性检验.

本文第 1 节首先回顾 QRDC 的语法、语义及相应的公式模型,然后提出针对表达两类有限性性质的 DQRDC 公式的模型检验算法.第 2 节介绍工具 QRDChecker 的设计与实现.第 3 节中使用 QRDChecker 来检验 Peterson 互斥算法的正确性.最后我们对工具的效率、意义等方面进行简要的讨论.

1 工具基础

1.1 带量词的RDC(quantified RDC)

时段演算(duration calculus[4],简称 DC)是在 ITL 基础上扩展而来的.在 DC 中,引入了对系统停留在某一状态上的累计时间的研究.QRDC 是对 DC 的一个可判定子集 RDC(restricted DC)[4]的扩展.在 QRDC 中我们引进了作用在状态变量上的量词.

本文中,我们将使用以下符号:Σ是一个有限集合,Σ^+是由Σ中元素组成的所有有限序列的集合.$\Sigma^*=\Sigma^+\cup\{\varepsilon\}$,

其中ε 是空序列.我们用Σ^{ω}表示由Σ中元素组成的所有无限序列,或者称其为ω序列的集合.$\Sigma^{\infty}=\Sigma^{*}\cup\Sigma^{\omega}$.给定$\Sigma$上的一个序列$\alpha$,$|\alpha|$表示$\alpha$的长度.具体地,当$\alpha$是$\omega$序列时,$|\alpha|=\infty$;当$\alpha=\varepsilon$时,$|\alpha|=0$.对于两个序列$\alpha_1\in\Sigma^{*}$和$\alpha_2\in\Sigma^{\infty}$,$\alpha_1\cdot\alpha_2$表示将$\alpha_2$ 连接在α_1 的后面得到的新序列;α_1 被称作α_2 的一个前缀,记作$\alpha_1\le\alpha_2$,当且仅当$\exists\alpha_3\in\Sigma^{\infty}:\alpha_1\cdot\alpha_3=\alpha_2$.$P_1$,$P_2$是两个有限序列的集合,$P_1\cdot P_2=\{\alpha_1\cdot\alpha_2|\alpha_1\in P_1\wedge\alpha_2\in P_2\}$.

1.1.1 语 法

我们用 $p,p_1,\ldots,p_n$ 表示状态变量,$SVar$ 是所有状态变量的集合.状态表达式和公式定义如下:

$$S::=0|1|p|\neg S_1|S_1\vee S_2,$$

$$\phi::=\lceil\!\lceil S\rceil\!\rceil|\neg\phi|\phi_1\vee\phi_2|\phi_1;\phi_2|\exists p.\phi.$$

1.1.2 语 义

对状态变量的解释函数 I 定义为

$$I\in SVar\to(Time\to\{0,1\}).$$

上式中的时间域 $Time$ 可以取 $R_{\ge0}$,$Q_{\ge0}$ 或者 N.对任意一个状态变量,I 给出了其在任意一个时间点上的取值.当我们限定时间只可以取值自然数,即 **N** 时,我们得到定义在离散时间上的 QRDC(DQRDC).

类似地,状态表达式 S 的解释函数也定义为从时间到$\{0,1\}$的函数.从 S 的结构出发,该函数归纳定义如下:

1. $I[\![0]\!](t)=0$;
2. $I[\![1]\!](t)=1$;
3. $I[\![p]\!](t)=I(p)(t)$;
4. $I[\![\neg S]\!](t)=1-I[\![S]\!](t)$;
5. $I[\![S_1\vee S_2]\!](t)=\max(I[\![S_1]\!](t),I[\![S_2]\!](t))$.

在给定 I 的情况下,一个 QRDC 公式ϕ的语义由定义域为时间区间集合的函数给出:

$$I[\![\phi]\!]\in\{[b,e]|b,e\in Time\wedge b\le e\}\to\{tt,ff\}.$$

$I[\![\phi]\!]$的具体定义如下:

1. $I[\![\lceil\!\lceil S\rceil\!\rceil]\!](b,e)=tt$ 当且仅当
   - 在连续时间条件下,$b<e$ 并且只存在有限个 $t(b\le t\le e)$使得 S 在 t 时刻取值为假,或者
   - 在离散时间条件下,$b<e$ 并且对于任意的 $t(b\le t<e)$,S 在 t 时刻取值为真
2. $I[\![\neg\phi]\!]([b,e])=tt$ 当且仅当 $I[\![\phi]\!]([b,e])=ff$
3. $I[\![\phi_1\vee\phi_2]\!]([b,e])=tt$ 当且仅当 $I[\![\phi_1]\!]([b,e])=tt$ 或 $I[\![\phi_2]\!]([b,e])=tt$
4. $I[\![\phi_1;\phi_2]\!]([b,e])=tt$ 当且仅当存在 $m\in([b,e]\cap Time)$使得 $I[\![\phi_1]\!]([b,m])=tt$ 并且 $I[\![\phi_2]\!]([m,e])=tt$
5. $I[\![\exists p.\phi]\!]([b,e])=tt$ 当且仅当存在 I',满足 I'是 p-等价于 I 的,并且 $I'[\![\phi]\!]([b,e])=tt$

解释函数 I'被称作 p-等价于解释函数 I 当且仅当对所有的 $p_1\ne p_2$ 和 $t\in Time$,$I(p_1)(t)=I'(p_1)(t)$均成立.

在本文中使用了谓词逻辑中的标准缩写,如“$\wedge$”,“$\Rightarrow$”和“$\Leftrightarrow$”.并且,我们还使用下列关于 QRDC 公式的缩写形式:

$\lceil\!\lceil\rceil\!\rceil\triangleq\neg\lceil\!\lceil 1\rceil\!\rceil$　　$\text{false}\triangleq\lceil\!\lceil 0\rceil\!\rceil$　　$\text{true}\triangleq\neg\text{false}$

$\Diamond\phi\triangleq\text{true};\phi;\text{true}$　　$\Box\phi\triangleq\neg\Diamond\neg\phi$

以及如下的 DQRDC 缩写($k\in N$):

$l=0\triangleq\neg\lceil\!\lceil 1\rceil\!\rceil$　　$l=1\triangleq\lceil\!\lceil 1\rceil\!\rceil\wedge\neg(\lceil\!\lceil 1\rceil\!\rceil;\lceil\!\lceil 1\rceil\!\rceil)$

$\int S=0\triangleq\lceil\!\lceil\neg S\rceil\!\rceil\vee l=0$　　$\int S=1\triangleq(\int S=0);(\lceil\!\lceil S\rceil\!\rceil\wedge l=1);(\int S=0)$

$\int S=k+1\triangleq(\int S=k);(\int S=1)$　　$\int S\ge k\triangleq(\int S=k);\text{true}$

$\int S>k\triangleq\int S\ge k+1$　　$\int S\le k\triangleq\neg(\int S>k)$

$\int S<k\triangleq\int S\le k-1(k\ge1)$

其中,$\int S$ 表示状态表达式 S 在一段时间内取值为真的累计时间之和.这样,在离散时间条件下,我们可以将系统停留在特定状态上的累计时间与自然数相比较.而且,由于$\int 1$ 事实上表达了当前时段的长度 l,我们还可以在公式中加入对时段长度的约束.

1.1.3 公式模型

我们把一个 $o\in Obs=2^{SVar}$ 称作一个观察.给定状态表达式 S,$N(S)\subseteq Obs$ 表示那些 S 在其中取值为 1 的观察的集合:

$$N(S)=\{o|(\bigwedge_{p_1\in o} p_1\wedge \bigwedge_{p_2\in(SVar-o)} \neg p_2)\Rightarrow S\}.$$

给定解释函数 I 和一个时间区间$[b,e]$,二元组$(I,[b,e])$满足一个公式ϕ,表示为$(I,[b,e])\models\phi$,当且仅当 $I[\![\phi]\!](b,e)=tt$.对于公式ϕ,我们称每个满足$(I,[b,e])\models\phi$的$(I,[b,e])$为ϕ的一个模型.

在离散时间条件下,$(I,[b,e])$确定了一个有限长的观察序列 $I(b),I(b+1),\ldots,I(e-1)$.$\phi$所有模型的集合 $\boldsymbol{M}_D[\![\phi]\!]$可以构造为如下的语言[4,5],该语言的字母表为 Obs:

$\boldsymbol{M}_D[\![\lceil S\rceil]\!]=N(S)^+$ (正闭包)

$\boldsymbol{M}_D[\![\phi_1\vee\phi_2]\!]=\boldsymbol{M}_D[\![\phi_1]\!]\cup\boldsymbol{M}_D[\![\phi_2]\!]$ (交集)

$\boldsymbol{M}_D[\![\neg\phi]\!]=Obs^*-\boldsymbol{M}_D[\![\phi]\!]$ (补集)

$\boldsymbol{M}_D[\![\phi_1;\phi_2]\!]=\boldsymbol{M}_D[\![\phi_1]\!]\cdot\boldsymbol{M}_D[\![\phi_2]\!]$ (连接)

$\boldsymbol{M}_D[\![\exists p.\phi]\!]=Equival_p(\boldsymbol{M}_D[\![\phi]\!])$ (p-等价闭包)

其中 $Equival_p(\boldsymbol{M}_D[\![\phi]\!])$的定义如下:

定义 **1**. 给定两个观察的有限序列$\alpha=o_0,o_1,\ldots,o_n$和$\alpha'=o'_0,o'_1,\ldots,o'_n$及状态变量$p$,$\alpha$和$\alpha'$是$p$-等价的当且仅当对于任意的 $0\le i\le n,o_i\backslash\{p\}=o'_i\backslash\{p\}$均成立.给定状态变量 p 和观察序列的集合 $\boldsymbol{M}$,$\boldsymbol{M}$ 的 p-等价闭包,记作 $Equival_p(\boldsymbol{M})$,定义为集合

$$\{\alpha|\text{存在}\beta\in\boldsymbol{M}\text{ 使得}\alpha\text{和}\beta\text{是 }p\text{-等价的}\}.$$

由上述过程构造的 $\boldsymbol{M}_D[\![\phi]\!]$是正则语言.这是因为正则语言对前 4 种运算显然是闭合的,而 $Equival_p$ 运算实际上是通过将语言字母表映射到新的字母表来修改语言的,因而正则语言对它的运算也是闭合的.

通过应用归纳法由ϕ的结构我们可以得出以下引理:

引理 **1**. 在 DQRDC 中,对于公式ϕ和二元组$(I,[b,e])$,$(I,[b,e])\in\boldsymbol{M}_D[\![\phi]\!]$当且仅当$(I,[b,e])\models\phi$.

从引理 1 我们容易得到:在 DQRDC 中,为可满足当且仅当 $\boldsymbol{M}_D[\![\phi]\!]\neq\varnothing$,公式$\phi$为永真当且仅当 $\boldsymbol{M}_D[\![\phi]\!]=Obs^*$.显然,$\phi$永真当且仅当$\neg\phi$不可满足.

然而,在连续时间条件下的情况则有所不同.在连续时间模型中,任意一个非零时间区间可以分解为两个更小的非零时间区间.因此,在离散时间条件下,非永真的公式$\lceil p\rceil\Rightarrow(\lceil p\rceil;\lceil p\rceil)$在连续时间条件下是永真的.显然,上面的针对离散时间的构造方法在这里不再直接适用.但是,通过使用连接闭包的概念,我们可以借鉴前面的构造思想,得到在连续时间条件下检验公式有效性的方法[4].

一个定义在字母表Σ上的语言 L 的连接闭包$\downarrow L$ 是指包含该语言本身,并且满足条件:

$$\forall a\in\Sigma,\alpha_1,\alpha_2\in\Sigma^*:(\alpha_1\cdot a\cdot a\cdot\alpha_2\in\downarrow L\Rightarrow\alpha_1\cdot a\cdot\alpha_2\in\downarrow L)$$

的最小语言.通过下面的规则,我们可以为公式ϕ构造正则语言 $\boldsymbol{M}_C[\![\phi]\!]$:

$\boldsymbol{M}_C[\![\lceil S\rceil]\!]=N(S)^+$

$\boldsymbol{M}_C[\![\phi_1\vee\phi_2]\!]=\boldsymbol{M}_C[\![\phi_1]\!]\cup\boldsymbol{M}_C[\![\phi_2]\!]$

$\boldsymbol{M}_C[\![\neg\phi]\!]=Obs^*-\boldsymbol{M}_C[\![\phi]\!]$

$\boldsymbol{M}_C[\![\phi_1;\phi_2]\!]=\downarrow(\boldsymbol{M}_C[\![\phi_1]\!]\cdot\boldsymbol{M}_C[\![\phi_2]\!])$

$\boldsymbol{M}_C[\![\exists p.\phi]\!]=Equival_p(\boldsymbol{M}_C[\![\phi]\!])$

值得注意的是,这里构造的正则语言 $\boldsymbol{M}_C[\![\phi]\!]$中并不真正包含$\phi$在连续时间条件下的模型.$\boldsymbol{M}_C[\![\phi]\!]$中的序列是通过忽略$\phi$模型中所有时段的长度描述,而仅考虑在该时段上的状态变化得到的.从这种意义上说,ϕ的模型与 $\boldsymbol{M}_C[\![\phi]\!]$中的序列之间存在着多对一的对应关系.引理 2[4]给出了语言 $\boldsymbol{M}_C[\![\phi]\!]$与公式$\phi$在连续时间条件下可满足性之间的关系.

引理 **2**. 连续时间条件下,公式ϕ是可满足的,当且仅当语言 $\boldsymbol{M}_C[\![\phi]\!]$非空.

有效性检验.有限状态自动机和正则语言之间有着直接的对应关系.我们可以由ϕ出发,构造有限状态自动

机 FA,使得 FA 接受的语言为 $\boldsymbol{M}[\![\phi]\!]$.如果 FA 的补自动机接受的语言 $L(\overline{FA})$ 为空,则ϕ永真;否则,ϕ不永真.

1.2 模型检验

模型检验[6]是一种形式化检验技术,它构造系统的有限模型,并通过搜索模型的整个状态空间来检验该模型是否满足特定的性质.由于系统模型的有限性,检验过程一定可以结束,并且因为该检验过程可以自动进行,这种技术受到了学术界和工业界的普遍重视.本文中,我们仅考虑对离散时间下表达有限性性质的 QRDC 公式的模型检验,DQRDC 公式ϕ表达的性质由 $\boldsymbol{M}_D[\![\phi]\!]$确定.在此我们描述对表达两类有限性性质——FS 性质和 EP 性质——的 DQRDC 公式进行模型检验的算法思想,文献[7]中给出了更加具体的讨论,并提出了识别 DQRDC 公式表达何种性质的算法.

直观上讲,FS 性质要求某些(坏的)情况永远不会发生,其形式描述如下:

定义 **2**(**FS** 性质). 给定有限集合Σ和性质 $P\subseteq\Sigma^*$,P 是 FS 性质当且仅当

$$\forall\alpha\in\Sigma^*:(\alpha\notin P\Leftrightarrow\exists\beta\in\Sigma^*:(\beta\le\alpha\wedge\forall\gamma\in\Sigma^*:(\beta\le\gamma\Rightarrow\gamma\notin P))).$$

例如性质 p^*就是一个 FS 性质,一旦一个序列中出现了一个非 p 的字符,则这个序列的任何扩展都不可能满足该性质.ω序列σ满足一个 FS 性质当且仅当

$$\forall\gamma\in\Sigma^*:(\gamma\le\sigma\Rightarrow\gamma\in P).$$

而 EP 性质要求某些(希望的)情况必将发生,并且这些事情一旦发生,任何后继事件都不能改变这个事实.EP 性质可以定义为:

定义 **3**(**EP** 性质). 对于有限集合Σ及性质 $P\subseteq\Sigma^*(P\neq\varnothing)$,$P$ 是 EP 性质当且仅当

- $\forall\alpha\in\Sigma^*:\exists\beta\in\Sigma^*:(\alpha\le\beta\wedge\beta\in P)$,并且
- $\forall\alpha\in P:\forall\beta\in\Sigma^*:(\alpha\le\beta\Rightarrow\beta\in P)$

性质$\Sigma^*\cdot p\cdot\Sigma^*$就是一个 EP 性质,任何序列都可以被扩展为包含字符 p,并且一旦 p 出现了,这个事实就不可改变.ω序列σ满足一个 EP 性质 P 当且仅当

$$\exists\gamma\in\Sigma^*:(\gamma\le\sigma\wedge\gamma\in P).$$

这两类性质的满足性关系间的差别是显然的.然而,借助下面的对 FS/EP 满足性的统一定义,我们可以一致地考虑这两类性质的检验问题.

定义 **4**(有限性性质的满足性定义). 对于定义在Σ上的ω序列σ和有限性性质 P,σ满足 P,表示为$\sigma\models_f P$,当且仅当$\exists\gamma\in\Sigma^*:(\gamma\le\sigma\wedge\forall\alpha\in\Sigma^*:(\gamma\le\alpha\le\sigma\Rightarrow\alpha\models P))$.

我们可以方便地得到定义 4 与上面两种满足性关系的一致性[3].我们用 $H[\![S]\!]$表示系统 S 所有行为的集合,即系统状态的ω序列的集合.在检验以 DQRDC 公式形式给出的有限性性质时,S 满足公式ϕ,当且仅当 $\forall\sigma\in H[\![S]\!]:\sigma\models_f\boldsymbol{M}_D[\![\phi]\!]$,即

$$\forall\sigma\in H[\![S]\!]:(\exists\gamma\in Obs^*:(\gamma\le\sigma\wedge\forall\alpha\in Obs^*:(\gamma\le\alpha\le\sigma\Rightarrow\alpha\in\boldsymbol{M}_D[\![\phi]\!]))).$$

亦即,S 不满足ϕ当且仅当

$$\exists\sigma\in H[\![S]\!]:(\forall\gamma\in Obs^*:(\gamma\le\sigma\Rightarrow\exists\alpha\in Obs^*:(\gamma\le\alpha\le\sigma\wedge\alpha\notin\boldsymbol{M}_D[\![\phi]\!]))).$$

我们可以构造有限状态自动机 $FA(\neg\phi)$,使得 $FA(\neg\phi)$接受的语言为 $Obs^*-\boldsymbol{M}_D[\![\phi]\!]$.这样上面的条件可以改写为

$$\exists\sigma\in H[\![S]\!]:(\forall\gamma\in Obs^*:(\gamma\le\sigma\Rightarrow\exists\alpha\in Obs^*:(\gamma\le\alpha\le\sigma\wedge\alpha\in L(FA(\neg\phi))))).$$

如果 $FA(\neg\phi)$是确定的,记为 $DFA(\neg\phi)$,引理 3 证明了此时上面的条件等价于

$$\exists\sigma\in H[\![S]\!]:\sigma\in L(DBA(\neg\phi)).$$

其中 $DBA(\neg\phi)$是一个确定的 Büchi 自动机[6],$DBA(\neg\phi)$是通过将 $DFA(\neg\phi)$中的终止状态标记为接受状态而得到的.

模型检验.要检验 S 是否满足ϕ,我们只需要检验 $H[\![S]\!]\cap L(DBA(\neg\phi))$是否等于$\varnothing$.如果交集为空,则 S 满足ϕ;否则,S 不满足ϕ.

引理 **3**. 对于ω序列σ和确定的有限自动机 DFA,通过将 DFA 中的终止状态标记为接受状态得到确定的 Büchi 自动机 DBA,则该自动机满足

$$\forall \alpha\in Obs^*:\alpha\le\sigma:(\exists \gamma\in Obs^*:\alpha\le\gamma\le\sigma.\alpha\in L(DFA)) \text{ iff } \sigma\in L(DBA).$$

证明:必要性:由 Büchi 自动机的定义,因为$\sigma\in L(DBA)$,所以σ无数次经过 *DBA* 的接受状态.因而,任意一个σ的前缀都有(无数多)扩展既是σ的前缀,又停止在接受状态上.从 Büchi 自动机的构造可知,这些扩展都是 *DFA* 的句子.

充分性:由条件可知,存在无限多个有限序列$\sigma_1,\sigma_2,\ldots$,满足$\sigma_1\le\sigma_2\le\ldots\le\sigma$,其中$\sigma_i\ne\sigma_{i+1}$且$\sigma_i\in L(DFA)(i\ge1)$.显然,$\sigma_1$停止在终止状态上.由于自动机的确定性,$\sigma_2$会首先经过$\sigma_1$经过的那些状态(特别地,那一个终止状态),然后停止在(可能是另一个)终止状态上.这样,σ_2至少会经过 Büchi 自动机的接受状态两次.由归纳法可知,σ_i至少经过 Büchi 自动机的接受状态 i 次.因为有无数多个这样的σ_i,σ一定经过了某些接受状态无数多次.从而$\sigma\in L(DBA)$. □

在上面的证明过程中,自动机的确定性是非常重要的.但是,在检验 FS 性质时,确定性却不是必需的.如果ϕ表达 FS 性质,则在 $FA(\neg\phi)$中,从任何一个终止状态出发,不论输入字符是什么,自动机都只能够转移到(另一个)终止状态.直观上看,这是因为如果一个序列不满足 FS 性质,那么该序列的任何扩展都不满足该性质.这样,如果σ的一个前缀到达了一个终止状态,那么任何输入都只能使自动机转移到(另一个)终止状态,从而σ一定可以被相应的 Büchi 自动机接受.由于自动机的确定化算法复杂度很高,在检验 FS 性质时使用非确定化的自动机将可以获得更高的效率.

模型检验工具 **Spin**.Spin[8]使用语言 Promela 对系统建模,所建立的系统模型包含一组交互的进程(process).待检验性质 P 的逆性质 $\neg P$ 也用 Promela 语言表达,称作时序声明(temporal claim).检验时,每个进程被转换成一个自动机,系统的状态空间可以通过计算所有自动机的异步积[9]得到.类似地,时序声明被转换成一个 Büchi 自动机.然后,Spin 计算该 Büchi 自动机与表示系统状态空间的自动机的同步积[9].如果该同步积自动机所接受的语言为空,系统所有的行为都满足性质 P;否则,该语言包含了系统中所有不满足 P 的行为.

2 模型检验工具 QRDChecker

我们已经将检验 QRDC 公式的有效性及模型检验以 DQRDC 公式形式表示的有限性性质的算法实现为工具 QRDChecker.QRDChecker 可以:

- 自动检验 QRDC 公式在离散及连续时间条件下的有效性;
- 作为 Spin 的前端,用于模型检验反应式系统是否满足以 DQRDC 公式形式给出的系统需求.QRDChecker 将表达有限性性质的 DQRDC 公式转换成时序声明,然后该声明可以作为 Spin 的输入用于检验.

本节中我们介绍工具 QRDChecker 的设计与实现,QRDChecker 的系统结构及内部处理流程如图 1 所示.待检验的公式存放在以“rdc”为扩展名的文件中.工具首先读取文件内容,分析并尝试构造与公式语法结构相对应的有向无环(DAG)图.如果输入文件中存在语法错误,则工具在给出错误定位后停止继续处理.在成功地构造了 DAG 图之后,工具根据不同的检验需求进行不同的处理.

(1) 模型检验只能对 DQRDC 公式进行,即只能在离散时间模型下进行.检验时首先构造接受语言 $\boldsymbol{M}_D[[\neg\phi]]$ 的有限自动机 *FA*,然后根据不同的性质类型,决定是否对 *FA* 进行确定化.接着将有限状态自动机中的终止状态集合作为接受状态集合,得到相应的 Büchi 自动机.最后从 Büchi 生成时序声明供 Spin 使用.

(2) 在检验公式的有效性时,工具首先根据所选择的时间模型,计算相应的 $\boldsymbol{M}_D[[\phi]]$或 $\boldsymbol{M}_C[[\phi]]$.然后,从 $\boldsymbol{M}[[\phi]]$ 出发构造相应的 *FA*.有效性判定通过检查 $\overline{FA}$ 接受的语言是否为空来进行.如果为空,则公式永真;否则,公式非永真,而该语言中的句子都是不满足公式的观察序列.

模型检验时,我们利用强大的 PROMELA 语言方便地构造系统模型,采用 DQRDC 表达需要检验的性质.接下来的从 DQRDC 公式到时序声明的转换和最终的模型检验都可以由工具自动完成.

事实上,有效性检验功能同样能够被用来检查系统是否满足特定的性质.我们可以将系统和特定的性质分别以 QRDC 公式的形式给出为ϕ_{sys}和ϕ_{spec},然后检验公式$\phi_{sys}\Rightarrow\phi_{spec}$是否永真.如果该公式永真,则说明系统是满足性质的;反之,则说明系统不满足性质.使用这种方法,我们可以摆脱工具中模型检验功能只支持离散时间模型的限制,检验连续时间模型下系统的性质.但是,对于复杂系统而言,要将系统行为正确地表达为 QRDC 公式

的形式将是一种挑战.

QRDChecker 的输入文件可以使用任意文本编辑软件进行编辑.每个说明文件可以包含以下几个部分:常量定义、宏定义和待检验公式.其中前两种定义结构的引进是为了增强说明文件的可读性和可维护性.如果说明文件中包含常量定义或者宏定义,它们必须在待检验公式之前给出.每个说明文件中有且仅有一个待检验公式.

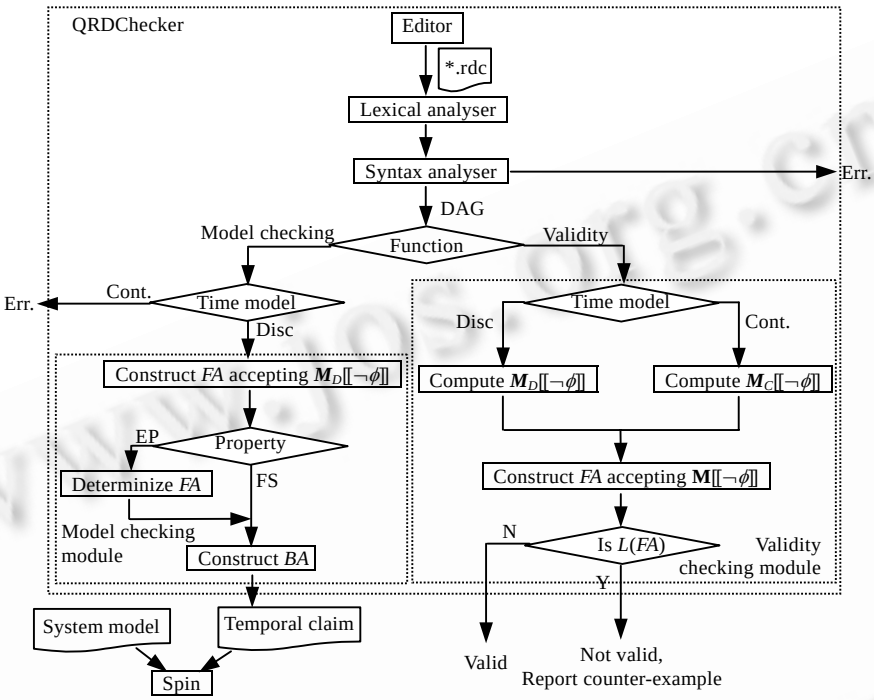

Fig.1 Architecture of QRDChecker

图 1 工具 QRDChecker 的体系结构图

常量定义以关键字“const”开始,包含一组赋值,并以“;”结束.每个赋值具有形式标识符=数字,两个赋值之间以“,”分隔.例如,我们可以通过下面的语句定义取值为 4 的常量“BOUND”.

const BOUND=4;

宏定义:说明文件中还可以加入宏定义.宏定义以关键字“define”开始,接着是用作宏的名称标识符,然后是宏的体,最后宏定义以“;”结束.下面是一个简单的例子:

define req1 exist p_1 ($\{p_1\}$>3);

待检验公式:关键字“goal”表明检验目标部分的开始.紧接其后的 QRDC 公式指出将被检验或转换的目标公式,该部分同样以“;”结束.例如:

goal []($\{p\}$>BOUND⇒req_1);

表 1 中列出了输入文件中可用操作符、其 ASCII 表示、优先级和结合性,并给出了应用实例.

输入文件编辑完成以后,用户可以从工具栏调用所有的功能,包括选择时间模型(连续或者离散)、检验动作(有效性检查或者模型检验)等.检查或者检验的结果将在新窗口中显示.

QRDChecker 是在 MicroSoft™ Visual C++ 6.0 环境下采用 C++语言开发的.在工具中,我们使用了用于操作有限状态机的 C++语言包 Grail+[10].另外,我们还使用了编译器构造程序 Parser Generator 来生成对输入文件的分析代码.

Table 1 Operators in QRDChecker
表 1 QRDChecker 中可用的操作符

Mathematical sign	ASCII	Priority	Association	Meaning	Example
∨	|	0	Left	Or	$p_1 \| p_2$
∧	&	1	Left	And	$p_1 \& p_2$
⇒	=>	2	Left	Imply	$[p_1]=>[p_2]$
⇔	<=>	2	Left	Iff	$[p_1]<=>[p_2]$
^	^	3	Left	Chop	$[p_1]^[p_2]$
>	>	4	None	Bigger than	$\{p_1\}>5$
<	<	4	None	Less than	$\{p_1\}<5$
=	=	4	None	Equal	$\{p_1\}=5$
≥	>=	4	None	Not less than	$\{p_1\}>=5$
≤	<=	4	None	Not bigger than	$\{p_1\}<=5$
≠	!=	4	None	Not equal	$\{p_1\}!=5$
+	+	5	Left	Plus	4+5
–	–	5	Left	Minus	4–5
×	*	6	Left	Times	4*5
÷	/	6	Left	Division	4/5
–	–	7	Right	Unary minus	–5
¬	!	8	Right	Negation	$!p_1$
◊	<>	8	Right	Sometimes	$<>[p_1]$
□	[]	8	Right	Always	$[][p_1]$
∃	Exist	8	Right	Existential quantification	Exist $p_1([][p_1])$
∀	Forall	8	Right	Universal quantification	Forall $p_1(<>[p_1])$
()	()	9	Left	Brace	$(p_1 \| p_2) \& p_3$
⌈ ⌉	[]	9	None	Duration invariant	$[p_1]$
∫	{}	9	None	Accumulated time duration	$\{p_1\}$

3 实例研究:Peterson 互斥算法

本节中,我们应用工具 QRDChecker 来检验 Peterson 互斥算法是否满足一定的性质.图 2 是以 Promela 语言程序形式给出的该算法在两个进程情况下的系统模型.

```
1.  #define true 1
2.  #define false 0
3.  #define aturn 1
4.  #define bturn 0
5.
6.  bool areq, breq, turn;
7.  bool ain;
8.  bool bin;
9.
10. proctype a()
11. {
12.         do
13.         ::      atomic{areq=true;
14.                 turn=bturn;}
15.                 (breq==false||turn==aturn);
16.                 ain=true;
17.                 /* critical section cs*/
18.                 /* assert(bin==false);*/
19.                 ain=false;
20.                 areq=false;
21.         od;
22. }
23.
24. proctype b()
25. {
26.         do
27.         ::      atomic{breq=true;
28.                 turn=aturn; }
29.                 (areq==false||turn==bturn);
30.                 bin=true;
31.                 /* critical section*/
32.                 /* assert(ain==false);*/
33.                 bin=false;
34.                 breq=false;
35.         od;
36. }
37.
38. init
39. {
40.         run a(); run b();
41. }
```

Fig.2 Promela model for Peterson algorithm
图 2 Peterson 算法的 Promela 模型

为了实现互斥,算法中使用了 3 个全局变量 turn,areq 和 breq.第 15 行和第 29 行的语句实现了同步,该语句只有当其中的布尔表达式为真的时候才可以执行.如果仅有一个进程申请访问临界区,该访问可以立即进行.当两个进程同时申请访问时,由变量 turn 决定哪一个进程可以进入临界区.变量 ain 和 bin 是为了描述性质方便而引入的,分别表示进程 *a* 或 *b* 在其临界区内.

作为一个解决互斥问题的算法,它必须能够保证互斥性,即在任何时刻,两个进程都不会同时在临界区内.

这个性质可以方便地用 DQRDC 公式表示如下,其中 l 表示当前时段的长度:

$$\mathrm{Req}_1 \triangleq \Box(l>1 \Rightarrow \lceil\!\rceil \neg(a\mathrm{in} \wedge b\mathrm{in}) \rceil\!\rceil).$$

与之相对应,Promela 时序声明如图 3 所示.其中,接受状态使用以“accept”开头的标号指出.

```
1.  never {
2.  T0_init:
3.      if
4.      :: goto T1
5.      fi;
6.  T1:
7.      if
8.      ::((true))−>goto T1
9.      ::((!ain||!bin))−>goto T2
10.     ::((ain && bin))−>goto accept_5
11.     fi;
12. T2:
13.     if
14.     ::((!ain||!bin))−>goto T2
15.     ::((ain && bin))−>goto accept_5
16.     fi;
17. accept_5:
18.     if
19.     ::((true))−>goto accept_5
20.     fi;
21. }
```

Fig.3 Temporal claim for Req_1

图 3 用于检验互斥性的时段声明

互斥性是较简单的性质,我们可以通过在模型中增加断言(第 18 行和第 32 行)来进行检验.下面我们考虑一个更为复杂的性质,该性质要求两个进程在竞争访问临界区时遵循某种公平性,具体地,它要求如果一个进程试图访问其临界区,而此时另一个进程正在其临界区内,那么在第 2 个进程再次进入临界区之前,第 1 个进程一定已经进入过临界区.这个性质可以用公式表达如下:

$$\mathrm{Req}_2 \triangleq \Box(\lceil\!\rceil a\mathrm{req} \wedge b\mathrm{in}) \rceil\!\rceil; \lceil\!\rceil \neg b\mathrm{in} \rceil\!\rceil; \lceil\!\rceil b\mathrm{in} \rceil\!\rceil \Rightarrow \Diamond \lceil\!\rceil a\mathrm{in} \rceil\!\rceil).$$

从这个公式出发,我们可以得到如图 4 所示的时序声明.

```
1.  never {
2.  T0_init:
3.      if
4.      ::goto T0
5.      fi;
6.  T0:
7.      if
8.      ::((true))−>goto T0
9.      ::((areq && bin && !ain))−>goto T2
10.     fi;
11. T2:
12.     if
13.     ::((!bin && !ain))−>goto T6
14.     ::((areq && bin && !ain))−>goto T2
15.     fi;
16. T6:
17.     if
18.     ::((!bin && !ain))−>goto T6
19.     ::((bin && !ain))−>goto accept_14
20.     fi;
21. accept_14:
22.     if
23.     ::((true))−>goto accept_19
24.     ::((bin && !ain))−>goto accept_14
25.     fi;
26. accept_19:
27.     if
28.     ::((true))−>goto accept_19
29.     fi;
30. }
```

Fig.4 Temporal claim for Req_2

图 4 用于检验性质 Req_2 的时序声明

将上面的系统模型和生成的时段声明作为输入,Spin 显示算法满足该性质.

4 讨 论

本文介绍了模型检验工具 QRDChecker 的理论基础及其设计与实现,并针对工具在模型检验方面的具体应用给出了应用实例.相对于在 LTL 方面的大量的研究而言,在时段时序逻辑自动检验方面的工作并不多见.QRDChecker 主要面向那些以使用时段时序逻辑为主的研究者,但人们可以在主要使用 LTL 的同时,用 DQRDC 来表达那些用 LTL 不方便描述的性质.事实上,Peterson 互斥算法中的性质 Req_2,就是这样一个适合使用时段时序逻辑来描述,而用 LTL 表达则很困难的性质.

虽然将时段时序逻辑的公式转换成有限状态自动机的算法并不复杂且已经提出了一段时间,但相关的实现则很少.Skakkebaek 和 Sestoft[11]的实现作为证明助手的一部分,是基于一个不再维护的低版本的 PVS 的.Pandya[5]在系统 DCValid 中借助 Mona[12]系统将时段逻辑转换为 Mona 的语言,进而进行检验.虽然 Mona 的效率非常高,但这也使得它难于使用.另外,在 DCValid 中并没有针对不同的有限性性质进行相应的处理.在 QRDChecker 中,我们区分不同的有限性性质,检验算法更加适应不同性质描述的实际要求.工具根据不同的性

质更好地控制检验过程,并进行相应的优化.例如,由于对有限状态自动机进行确定化的算法的复杂性是非常高的,所以我们在 QRDChecker 中根据公式表达的不同性质,决定是否需要进行相关的确定化运算,尽可能地提高工具的效率.

致谢 本文的工作很大一部分是在第一作者作为 fellow 在 UNU/IIST 访问学习期间完成的,在此对 UNU/IIST 提供的资助表示衷心的感谢.

References:

[1] Ramakrishna YS, Melliar-Smith PM, Moser LE, Dillon LK, Kuttys G. Interval logics and their decision procedures, Part I: An interval logic. Theoretical Computer Science, 1996,166(1&2):1−47.

[2] Alpern B, Schneider FB. Recognizing safety and liveness. Technical Report TR86-727, New York: Cornell University, 1986.

[3] Pei Y, Xu QW. Checking interval based properties for reactive systems. In: Steffen B, Levi G, eds. Proc. of the 5th Int'l Conf. on Verification, Model Checking, and Abstract Interpretation. Venice: Springer-Verlag, 2004. 122−134.

[4] Hansen MR, Zhou CC. Duration calculus: Logical foundations. Formal Aspects of Computing, 1997,9(3):283−330.

[5] Pandya PK. Specifying and deciding quantified discrete-time duration calculus formulae using DCVALID. Technical Report, No.TCS00-PKP-1, Mumbai: Tata Institute of Fundamental Research, 2000.

[6] Clarke EM, Grusmberg O, Peled DA. Model Checking. Cambridge: MIT Press, 1999.

[7] Pei Y, Xu QW. Model checking reactive systems for finitary properties. Technical Report, No. 283, Macau: UNU/IIST, 2003. http://www.iist.unu.edu/newrh/III/1/page.html

[8] Holzmann G. The model checker SPIN. IEEE Trans. on Software Engineering, 1997,23(5):279−295.

[9] Arnold A. Finite Transition Systems: Semantics of Communicating Systems. Hemel Hempstead: Prentice-Hall, 1994.

[10] The Grail+ Project. Department of Computer Science, University of Western Ontario. 2002. http://www.csd.uwo.ca/research/grail/

[11] Skakkebaek JU. A verification assistant for real-time logic [Ph.D. Thesis]. Department of Computer Science, Technical University of Denmark, 1994.

[12] Henriksen JG, Jensen J, Jorgensen M, Klarlund N, Paige B, Rauhe T, Sandholm A. Mona: Monadic second-order logic in practice. In: Brinksma E, Cleaveland R, Larsen KG, Margaria T, Steffen B, eds. Proc. of the Tools and Algorithms for the Construction and Analysis of Systems'95. Aarhus: Springer-Verlag, 1996. 89−110.