基于实时逻辑的时间约束检测方法*)

刘瑞成　张立臣
(广东工业大学计算机学院　广州510090)

摘　要　本文针对具有严格时间要求的系统，阐述并分析了三种利用实时逻辑实现时间约束检测的方法。第一种方法通过检测系统规范和安全性断言的一致性来验证约束的满足性，非常适合于系统规范的设计与可满足性检测，算法的时间复杂度是$O(n^2)+O(n^2)+O(2^k)$。第二种方法利用实时逻辑与约束图的方法实现运行时的时间约束检测，但检测时的系统约束条件不够第三种方法简约，算法时间复杂度为$O(n^3)$，改进之后为$O(n^2)$。第三种方法通过对约束图的处理，减少运行时系统检测的约束条件，从而减少运行时的时间约束条件的搜索时间，算法的时间复杂度为$O(n)$，在实时性和检测效率明显优于前两种方法。但需要运行前优化约束规则，将会增加额外的时间和空间复杂度。

关键词　实时系统，实时逻辑，约束，图，检测

Monitoring Timing Constraints with Real-time Logic

LIU Rui-Cheng　ZHANG Li-Chen
(Faculty of Computer Science, Guangdong University of Technology, Guangzhou 510090)

Abstract　Three kinds of methods, which make use of real-time logic to realize monitoring timing constraints, are discussed and analyzed in this paper with respect to the restrict time requirement systems. The first kind of method is to prove the satisfiability for constraints by monitoring the consistency of the system specifications and safety assertions. It is very suitable for designing system specifications and detecting satisfiability. And the complexity of algorithm is $O(n^2)+O(n^2)+O(2^k)$. The second kind of method utilizes real-time lgic and constraints graphs realizing run-time constraints monitoring, but constraints in monitoring are not as simple as the third one's. And the complexity of algorithm is $O(n^3)$, after being improved it turns to be $O(n^2)$. The third kind of method reduces the constraints during the run-time monitoring, by dealing with the constraints graphs first. As a result it reduces the time of searching time-constraints in run-time. And the complexity of algorithm is $O(n)$. Obviously it is superior to the first two kinds of methods in real-time character and efficiency of measuring. However optimizing the constraints before run-time would add extra time and space complexity.

Keywords　Real-Time system, Real-Time logic, Constraints, Graph, Monitoring

1　引言

实时系统设计通常需要对系统行为和环境进行时间上的假设，例如通信延迟的上界，任务执行的截止时间，事件之间发生的最小时间间隔等等[2]。这些假设经常用于处理外部环境引起的不可预测性或复杂问题简化等问题。它们常常作为系统形式化规范的一部分，或者作为实时任务调度的需求。虽然目前已在形式化验证方法和实时调度方面做了许多研究，但还是不能完全解决时间约束的问题，因此需要在运行时进行检测。这主要就是由于大多数实时系统的执行环境并不完善，而且经常需要与外界进行交互操作，从而增加了系统的不可预测性，在运行时出现违背系统设计时的条件。另一方面，形式化技术或调度算法的应用又需要进行系统假设，在设计时用形式化方法来验证系统的特性也似乎不大可行，这也要求对系统规范的检测。除此之外，大多数实时系统的约束往往很烦琐，主要是由于没有很好的工具能够适当地把系统需求转换为系统约束，这也将导致系统的不可行或约束的不可满足，因此也需要对约束规范的满足性验证，特别是运行时的检测[11]。

运行时的时间约束检测可以通过搜集系统运行时的时间信息，并检测它们是否满足系统的时间约束假设。如果不能满足系统假设的约束条件，则触发相应的处理任务。

实时系统的时间约束检测有多种方法，例如在每个事件发生时，通过分析事件已经发生的历史来估计每个约束公式的可满足性，但这种方法在检测时没有对延迟定界，不能完全确保实时性。周期性地执行事件发生检测，由于检测周期较小，需要频繁地执行检测算法，系统开销较大，也会影响检测的实时性。

实时逻辑非常适合于实时系统约束条件的表达，并通过图形化的方法检测系统假设约束条件的可满足性，能够尽可能早地发现约束的违反。利用实时逻辑检测实时系统的时间约束，即能确保系统的实时性、正确性，又能充分应用实时逻辑本身具有的强大推理能力和众多的理论方法。

在这方面已有许多人做了不少的研究。例如文[12]论述了一种嵌入式系统时间约束分析的框架。文[7]通过提出一种多项式算法来检测约束图的正环，从而检测时间约束的一致性。文[10]提出一种基于实时逻辑的统一形式化方法，说明活跃的实时数据库系统中的复杂事件的时间约束。文[13]提出一种基于实时逻辑知识，能够确保共享数据一致性的方法。文[14]说明了一种相关的实时逻辑(Relational RTL)，能够更好地描述系统规范。文[15]提出一种结合RTL和前趋图(precedence graph)优点，能够说明和验证分布式硬实时系统的方法等。

本文主要就针对具有严格时间要求的系统，阐述并分析了三种利用实时逻辑实现时间约束的检测的方法，并对它们进行分析比较，总结三种方法的算法复杂度及其应用范围。

*)本文受国家自然科学基金(No. 60474072、No. 60174050)、广东省自然科学基金(No. 04009465、No. 010059)、广东省高校自然科学研究项目(No. Z03024)基金资助。刘瑞成　硕士研究生，主要研究方向：实时系统、软件设计方法与实时软件设计方法。张立臣　博士，教授，硕士生导师，主要研究方向：并行处理、分布式处理、网络计算、实时系统等。

本文的第2节介绍了实时逻辑的基本概念；第3、4、5节分别阐述三种时间约束的检测方法；第6节对这三种方法进行分析比较；最后是对本文的总结。

2 实时逻辑

实时逻辑是一种符合实时系统时间特性推理的规范化语言，是对事件发生时刻的一种推理方式。在实时逻辑中，标识出事件发生时间点对实时系统的事件行为具有重要的意义。这里所指的事件可以分为四类[1,6]：

·开始事件：标识动作的开始，动作 A 的开始用符号 $\uparrow A$ 表示。

·停止事件：标识动作的结束，动作 A 的完成用符号 $\downarrow A$ 表示。

·外部事件：由外部激励驱动而发生的事件，按下按钮用 ΩBUTTON 表示。

·转移事件：标识系统属性的改变。

每个事件都必须用唯一的名字表示，不同类型的事件在事件名前加上相应的前缀符号来表示。每个动作都有相应的开始事件和停止事件，用大写字母表示。一个动作可以包含子动作，并把动作 B 的子动作 A 可以表示为 $B.A$。变量用小写字母表示，可以代表动作、整数或事件等。逻辑谓词由符号集 $\{<、=、>、\geqslant、\leqslant\}$ 组成，用以表示系统中时间的先后关系。逻辑函数有＋、－、×和@函数，其中加、减、乘是与常量运算，@函数如定义1。

实时逻辑公式由谓词、存在量词∃ 、全称量词∀ 以及连接词□构成。

在应用程序中，事件用于表示有关状态发生的改变。计算时，一个事件可以出现多个实例，即事件的重现。实时系统的计算也可以看作是事件发生的序列。因此事件是实时逻辑中非常重要的一个概念，一个事件的发生是在对应实例出现的时间点。用时间特性来表示计算时实例之间的关系。在实时逻辑中，事件发生函数定义为：

定义1 事件发生函数 $@(e,i)$ 表示事件 e 第 i 次实例出现的时间。其中 $i\in N^+$，是@函数的参数。@函数是一个单调递增函数，即 $\forall e,\forall i\in N^+$，有 $@(e,i+1)>@(e,i)$。

#函数定义为@函数的反函数，表示事件 e 在 t 时刻最近发生的是第几次实例。

定义2 事件 e 在 t 时刻发生的第 i 次实例，定义如下

$$\#(e,t)=\begin{cases}0, & \text{当 } t<@(e,1)\\ i, & \text{当 } i\geqslant 1\wedge @(e,i)\leqslant t\wedge @(e,i+1)>t\end{cases}$$

另外，有时需要扩展@函数，使它能够表达将来发生时间的相关实例。

定义3 相关@函数定义如下[3,4,10]：

$@_r(e,t,i)=@(e,\#(e,t)+i)$，其中 $\#(e,t)+i>0,i\in N$，为相关@函数的参数。t 称为参考时间。

3 方法一

文[1]提出了利用实时逻辑来解决时间约束规范安全性分析的方法。通过检测系统安全性断言是否与系统给定的约束规范相一致，来判断系统安全与否？安全性断言 SA 能够从系统规范 SP 通过定理推导得到，即 SP□SA，那么系统规范就是可满足性的。这种方法利用系统规范和安全性断言的否定构造一个带权有向图，然后进行有向图的正环检测，最后根据检测到的正环判断系统规范与安全性断言的一致性，从而达到检测的目的。这种方法所用到的 RTL 规则具有一些严格的限制[1,5]：

·每个算术不等式仅仅包括两个函数（或变量）和一个整数常量。

·所有算术表达式的函数不能以本身的实例作为参数。

符合上述要求的 RTL 子类，可以设计一种有效的约束图来检测它的可满足性，但这种方法是不可判定的。因此，文[5]在此基础上设计了一种新的方法。在验证实时系统时间约束时通过不断地增加系统规则公式的决定性因素，修改系统规范和安全性断，逐步实现系统的可满足性，而且还可以确定已有系统规范与安全性断言之间不一致的程度。

3.1 时间约束说明

系统规范和安全性断言都利用实时逻辑表示，下面通过例1来说明。

例1. 假设一个简单的数据采集系统，系统包括内部数据采样和信息显示两个动作，它们通过外部激励而执行。时间约束规范如下：

系统规范：当按下按钮时，动作 SAMPLE 在30个时间单元必须完成，每次采样之后就把相应信息发送到显示面板进行显示，SAMPLE 的计算时间是20个时间单元。用 RTL 表示系统规范如下：

$\forall x@(\Omega BUTTON,x)\leqslant@(\uparrow SAMPLE,x)\wedge$
$@(\downarrow SAMPLE,x)\leqslant@(\Omega BUTTON,x)+30$
$\forall y@(\uparrow SAMPLE,y)+20\leqslant@(\downarrow SAMPLE,y)$

安全性断言：如果信息必须在动作 SAMPLE 完成后10个时间单元内显示出来，那么传送的信息就能在按下按钮后的50个时间单元内显示出来。即

$\forall u\forall t@(\downarrow SAMPLE,u)\leqslant@(\Omega DISPLAY,t)\wedge$
$@(\Omega DISPLAY,t)\leqslant@(\downarrow SAMPLE,u)+10\square$
$@(\Omega BUTTON,u)<@(\Omega DISPLAY,t)\wedge$
$@(\Omega DISPLAY,t)<@(\Omega BUTTON,u)+50$

为了简单处理，把每个规则后面加减的数限定为整数常量，并用 Presburger 算法表示，每个事件用一个抽象函数代替，如下。

系统规范说明：

$\forall x\ f(x)\leqslant g_1(x)\wedge g_2(x)\leqslant f(x)+30$

$\forall y\ g_1(y)+20\leqslant g_2(y)$

安全性断言：

$\forall u\forall t\ g_2(u)\leqslant h(t)\wedge h(t)\leqslant g_2(u)+10\square f(u)<h(t)\wedge h(t)<f(u)+50$

下面证明系统规范 SP 与安全性断言 SA 的一致性，即证明公式 $F=SP\wedge\neg SA$ 是不可满足的，来确定时间约束的可满足性。

设 F 的 Presburger 算法形式为 F'，则例1写成 F' 公式为：

$$F'=f(x)\leqslant g_1(x)\wedge g_2(x)\leqslant f(x)+30\wedge g_1(x)+20\leqslant g_2(x)\wedge g_2(U)\leqslant h(T)\wedge h(T)\leqslant g_2(U)+10\wedge[h(T)\leqslant f(U)\vee f(U)+50\leqslant h(T)]$$

其中，U、T 是 Skolem 常量。下面来验证 F′的不可满足性，先利用 F′构造一个带权有向图。

3.2 构造图

构造带权有向图的方法如下，F′公式的函数用结点表示，对于变量及函数相同的就用同一个结点表示，相同函数的结点放在同一个串里；不等式用箭头表示；边上的权值是对应约束的时间偏移量。构造带权有向图的算法见文[1]。如果有 n 个函数，即构造的图有 n 个结点，则算法复杂度为 $O(n^2)$。例1构造出来的带权有向图如图1所示。

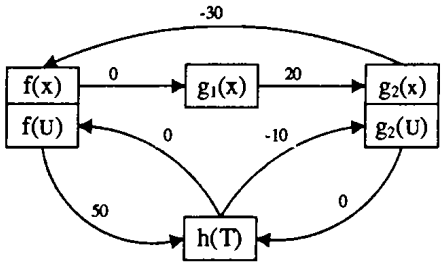

图1 构造的带权有向图

3.3 检测正环

构造出带权有向图之后就是正环检测。

定义4 在构造的带权有向图中，如果存在一条路径：$\langle v_0,v_1\rangle,\langle v'_1,v_2\rangle,\cdots\cdots,\langle v'_i,v_{i+1}\rangle,\cdots\cdots,\langle v'_n,v'_0\rangle$，其中 v_i 和 v'_i 是同一个结点或同一个串的结点，$i\in[0,n]$，那么就形成一个环。如果环上的权值之和大于0，就称这个环为正环。

定理1 如果公式 F′构成的带权有向图，存在一个正环，且 P 为构成正环边对应的规则形成的合取式，那么 P 是不可满足的。

文[1]对定理1有详细的证明过程。如果约束图中的正环表示的 EQL 程序有一个极大的反应时间[6]，那么 RTL 规则 $SP\wedge\neg SA$ 是不可满足的。正环的检测主要是进行带权有向图的结点及其相应边的删除操作，如果出现负环就将其丢弃，直到剩下正环为止，正环检测算法见文[1]。文[7]也阐述了一种在多项式时间内实现约束图正环检测的算法。下面对图1的带权有向图利用文[1]中的正环检测算法分析。

1. 删去结点 $g_1(x)$及相应的边$\langle f(x),g_1(x)\rangle$和$\langle g_1(x),g_2(x)\rangle$。增加边$\langle f(x),g_2(x)\rangle$，权值是$\langle f(x),g_1(x)\rangle$和$\langle g_1(x),g_2(x)\rangle$的权值之和，即0+20=20，结果如图2所示。

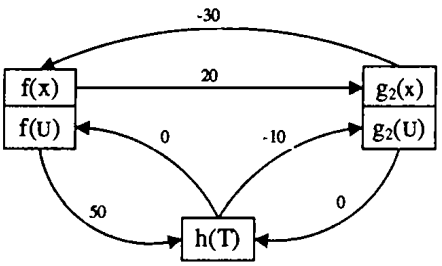

图2 删去结点 $g_1(x)$

2. 删去结点 $g_2(x)$及其相应的边$\langle f(x),g_2(x)\rangle$和$\langle g_2(x),f(x)\rangle$，增加权值为−10的环$\langle f(x),f(x)\rangle$。由于 $f(x)$与 $f(U)$，$g_2(x)$与 $g_2(U)$具有相同的函数，并且 x 是 U 的一般化，两者具有同一性，因此存在一条路径$\langle f(x),g_2(x)\rangle$，$\langle g_2(U),h(T)\rangle$。删除结点 $g_2(x)$，必然破坏这条路径，所以需要增加一条权值为20的边$\langle f(U),h(T)\rangle$，同样增加边$\langle h(T),f(U)\rangle$，其权值为−40。结果如图3所示。

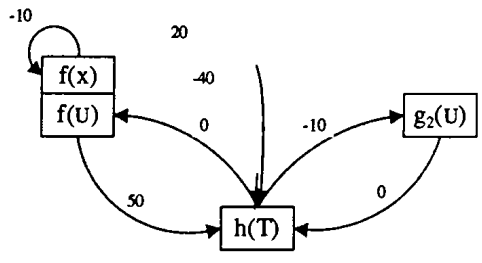

图3 删去结点 $g_2(x)$

3. 删除结点 $g_2(U)$及其边，增加边$\langle h(T),h(T)\rangle$权值为−10。删除结点 $f(x)$，丢弃负环$\langle f(x),f(x)\rangle$。如图4所示。

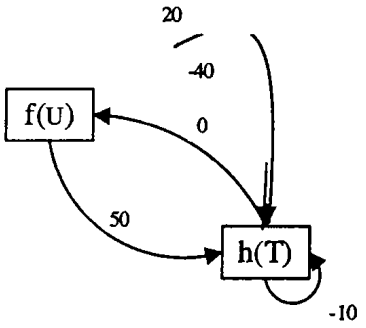

图4 删除结点 $g_2(U)$

4. 删除结点 $f(U)$及其边，增加四条边$\langle h(T),h(T)\rangle$，权值分别为20、50、10、−20。丢弃权值为负的两个环，结果为三个正环。最终结果如图5所示。

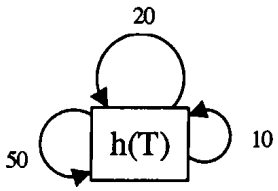

图5 删除结点 $f(U)$

3.4 验证满足性

根据正环来验证 F'的不可满足性。

令 $X_{i,j}$为检测出的第 i 个正环包含的第 j 条边对应的公式，P_i 为 $X_{i,j}$的合取式，即

$P_i=X_{i,1}\wedge X_{i,2}\wedge\cdots\cdots\wedge X_{i,m}, j\in[1,m]$

从定理1可知 P_i 是不可满足的，那么$\neg P_i$必定是可满足的。因此，F'可满足的充要条件是 $F'\wedge\neg P_i$ 是可满足的，即 $SP\square SA$ 为真[6]。所以只要证明 $SP\square SA$ 的否定不可满足，即 $F'\wedge\neg P_i$ 的不可满足性就可以得到 F'的不可满足性。

如果所有的 $X_{i,j}$是单元子句，根据上面的分析可以推出 F'的不可满足性。如果 $X_{i,j}$是非单元子句，例如图5权值为50的环，在图1中包括规则 $h(T)\leqslant f(U)\vee f(U)+50\leqslant h(T)$，它是两个单元子句的析取式，这就需要复杂的分析计算。如果正环中包括许多非单元子句时，问题就与命题逻辑合取范式的可满足性问题一样变得非常复杂了。另外，命题逻辑的 CNF 可满足性问题是一个 NP 完全问题[1]。

为了简化计算，利用构造搜索树的方法来证明 $F'\wedge\neg P_i$ 的不可满足性。用一个例子来说明构造搜索树的步骤。

例2. 设 S_1为 F'中的约束集，A、B、C、D、E 和 F 分别表示不同的公式，例如 A 表示不等式 $f(x)\leqslant g_1(x)$。$C_1\sim C_6$表示6个约束规则。

$C_1:A\vee D$

$C_2:C\vee B\vee D$

$C_3:A\vee F$

$C_4:B\vee D\vee E$

$C_5:B\vee F$

$C_6:A\vee B$

S_2是所有的$\neg P_i$构成的公式集。

$\neg A\vee\neg B$

$\neg D\vee\neg F$

$\neg A\vee\neg C$

$\neg B\vee\neg E$

利用深度优先搜索构造搜索树的步骤如下：

1. 取 S_2 的第一个公式 $\neg A \vee \neg B$，作为搜索树第一层结点，首先构造第一个结点 $\neg A$，如图6(a)所示。由于 $\neg A$ 结点没有否定 S_1 的任一公式，因此取 S_2 的第二个规则继续构造第二层结点。

2. 取 S_2 中的 $\neg D \vee \neg F$ 构造第二层结点，在 $\neg A$ 结点生产孩子结点 $(\neg A, \neg D)$，此时 $(\neg A, \neg D)$ 否定 S_1 的 C_1 公式，把它作为叶子，如图6(b)所示。接下来生成 $\neg A$ 结点的第二个分枝，孩子结点为 $(\neg A, \neg F)$，如图6(c)所示。由于 $(\neg A, \neg F)$ 也否定 S_1 的 C_3，因此它也作为叶子结点，$\neg A$ 子树构造完成。

3. 重复上面步骤，生成规则 $\neg A \vee \neg B$ 中对应的 $\neg B$ 子树，如图6(d)所示。生成结点 $(\neg B, \neg D, \neg A)$ 时，$(\neg B, \neg D, \neg A)$ 否定 S_1 的 C_6，它必须作为叶子结点。但这个结点中的 $\neg D$ 在否定 C_6 时没有起任何作用，因此可以修剪生成树，减少它的复杂性。重新调整构造 $\neg B$ 子树的顺序，先利用规则 $\neg A \vee \neg C$ 生成 $\neg B$ 的子结点，结果如图6(e)所示。

4. 反复执行上面三个步骤构造搜索树的其它结点，直到用完 S_2 中所有规则或生成的搜索树的叶子结点都否定 S_1 的规则，且不能再扩展搜索树为止。例2生成的搜索树最终结果如图6(f)所示。

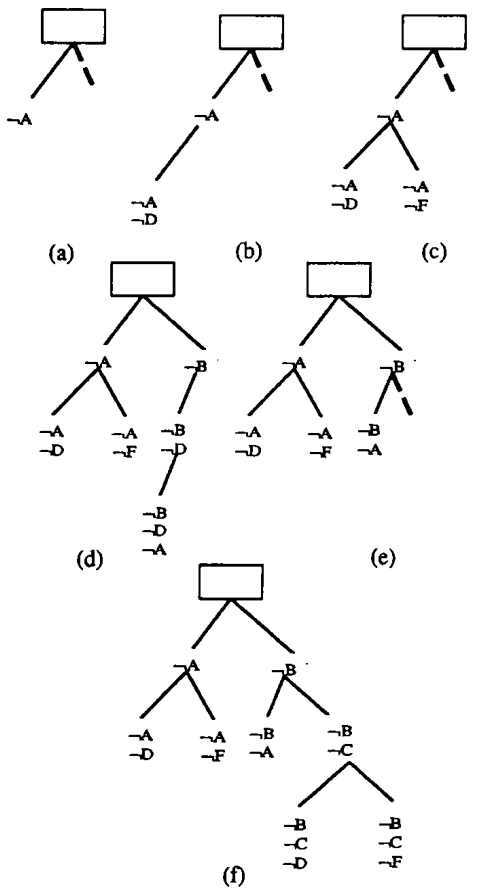

图6 构造搜索树的过程图

最坏情况下，生成的搜索树是一棵完全树。如果生成的搜索树的叶子结点都否定 S_1 的某一规则，并且搜索树不能再扩展了，则 $F' \wedge \neg P_i$ 是不可满足的。文[1]描述了不可满足性的具体算法。如果检测到的正环的个数为 k，S_2 中 P_i 包含2个约束条件，则算法时间复杂度为 $O(2^k)$。算法的空间复杂度与树的深度有关，与检测到的正环的个数成比例。

这种方法适合于设计系统对系统规范的验证，即通过判断安全性断言来验证规范的可满足性。文[6]通过这种方法来检测系统的反应时间的可满足性，从而达到对嵌入式系统安全性的验证。但约束图是不可判定的。因此，文[5]对此进行了改进。在验证实时系统时间约束时通过不断地增加系统规则公式的决定性因素，修改系统规范和安全性断言，逐步实现系统的可满足性，而且可以确定已有的系统规范与安全性断言之间不一致的程度。但是它并不能很好地实现在运行时实时的检测系统规范的可满足性，下面介绍两种方法就实现了运行时的系统时间规范的检测。

4 方法二

文[2,8]论述了在实时系统运行环境的时间特性检测模型及其实现，包括同步和异步检测时间约束两种方法。同步检测方法是把系统约束嵌入在一个程序里，因此只需要在实时任务执行的某个特定时间点检测系统约束即可。异步检测方法用一个单独的约束检测任务实现，约束检测贯穿于整个实时任务执行期间。

在检测运行时的约束时，需要记录系统计算过程中事件发生时前一次发生的实例。由于运行时检查系统特性可能涉及到同一个事件的多个实例，需要记录一个事件的多个实例以便于发现不满足的时间特性。因此利用事件历史(event histories)来存储先前许多实例发生的时间。文[17]更详细地阐述了事件历史在实时编程中的应用，并定义了相应的支持语言和系统。

在第2节中，可知 $@(e,i)$ 表示事件 e 第 i 次事例发生的时间。同样，用 $@val(v,i)$ 表示变量 v 在迁移事件第 i 次实例发生的值。当 $i<0$ 时，$@(e,i)$ 表示比当前实例早 i 个实例发生的时间。

为了区分同步和异步检测时间约束，用嵌入式约束(embedded constraints)和监控式约束(monitored constraints)两种方法来表示时间特性。嵌入式约束允许程序员在程序执行时的特定点来检测时间特性的可满足性，并做相应的修改，即允许程序员直接地处理@函数和存取事件历史，是与系统程序相关的。监控式约束与嵌入式约束不同，它表示的时间特性是与程序无关的，它利用一个独立的监控程序与实时应用任务同时的运行，并检测约束的可满足性。但这两种方法都必须满足下列的条件[2]：

- 一个事件不能同时出现两个实例；
- 在整个系统任务里，事件必须用唯一的名字表示；
- 必须有一个对所有任务都有效的单调递增时钟。

4.1 嵌入式约束

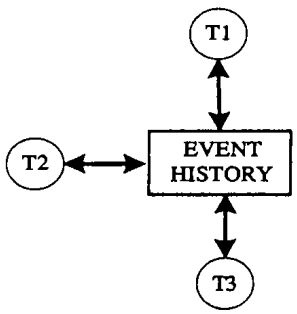

图7 嵌入式模型

嵌入式约束模型的时间特性是设置在执行序列中的一个特定的点，能够对约束进行同步的监测，模型如图7所示。它由一系列应用任务和共享的事件历史构成。任务之间事件的实例通信都通过事件历史实现，所有事件的实例通过任务记录到共享的事件历史里。事件发生实例的可满足性通过事件历

史返回的近似值来检验。

4.2 监控式约束

这种模型将时间特性设置在整个执行过程中，提供异步的约束监控方式。异步监测需要正确设置截止时间和延迟时间。模型如图8所示，它是由应用任务产生的一系列事件和一个监控任务(监控器)组成。通过一个队列实现每个任务和监控器进程间的通信。只有监控器才有权操作事件历史，其它任务通过监控器实现对历史的操作。在任务执行过程中，事件实例发生时，就被送到监控器，监控器处理每个事件发生实例，并把它记录到历史，然后监控器检测相应约束规则的可满足性。监控器维护事件历史，并对同步发生实例做出判断。

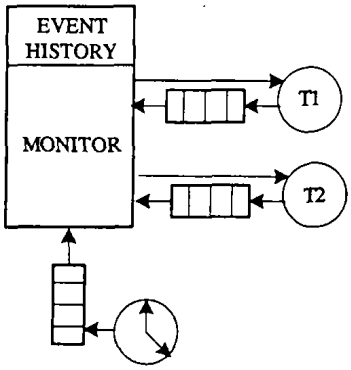

图8 监控式模型

4.3 可满足性检测

当确定了利用同步还是异步方式来检测之后，就必须实现如何检测约束的可满足性。用实时逻辑的方法来定义系统的约束规范，每个约束都由下面的不等式表示：

$@(e,i)\leqslant@(f,j)+C$，其中 i、j 和 C 都是整数常量，i、j 为事件第 i 或 j 次发生的实例，C 为相应的偏移时间值，表示延迟或截止时间。

用约束图表示系统时间约束，图中的每个结点表示不等式的一个谓词，每条边表示一个约束规范公式 $@(e,i)\leqslant@(f,j)+C$。把所有的约束都表示为约束图，然后通过检测约束图来检测系统规范的可满足性。

定理2 在约束图中，一个约束最早被违背的时间定义如下[8,9]：

·对于延迟约束，如果结点 e 到0结点的路径长度为 $-T$ ($T\geqslant0$)，且结点 e 相应的事件在时间 T 之前发生，那么此延迟约束将不可满足。

·对于截止时间约束，如果0结点到结点 e 的最短路径为 T($T\geqslant0$)，T 是0结点到其它所有结点最短路径的最小值，且结点 e 相应的事件在 T 时刻或者 T 之前还没发生，那么此截止时间约束将是不可满足的。

约束图的检测是利用事件历史中的事件实例来实例化约束图的结点，不断地把被实例化的结点合并为0结点。下面以一个例子来说明系统约束规范可满足性的检测方法。

例3. 假设系统的一个反应动作为 RESPONSE，外部信号为 SIGNAL，当系统接收到信号 SIGNAL 时，就执行动作 RESPONSE。如果在动作 RESPONSE 还没完成时，再次收到信号 SIGNAL，则不满足系统约束规范，系统就会发生异常。如果动作 RESPONSE 在信号 SIGNAL 到达之前已经发生了，则 RESPONSE 必须先于 SIGNAL 事件完成。用实时逻辑公式表示如下：

$@(\uparrow RESPONSE,-1)\leqslant@(\Omega SIGNAL,-1)\square$

$@(\uparrow RESPONSE,-1)\leqslant@(\downarrow RESPONSE,-1)\wedge$

$@(\downarrow RESPONSE,-1)<@(\Omega SIGNAL,-1)$

假设偏移时间值为1，转化为析取范式形式为：

$@(\Omega SIGNAL,-1)+1\leqslant@(\uparrow RESPONSE,-1)\vee$

$[@(\uparrow RESPONSE,-1)\leqslant@(\downarrow RESPONSE,-1)\vee$

$@(\downarrow RESPONSE,-1)+1\leqslant@(\Omega SIGNAL,-1)]$

下面用分解树来表示上述公式，树根为'∨'，每片叶子是一个不等式，中间结点为'∧'。上式公式分解为如图9所示的树状。树根下的每棵子树都可转换为一个等价的带权有向图，公式中每个@函数对应图的一个结点。如果是 $@(e,i)\leqslant C$ 这种形式，则增加一个0结点。图9的分解树转换为带权有向图如图10所示。

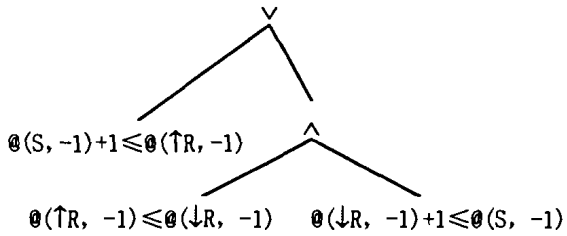

图9 分解树

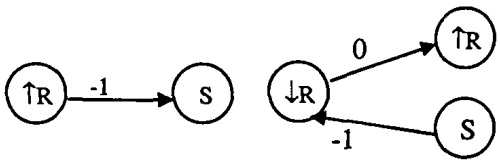

图10 为实例化的约束图

然后，利用带权有向图来进行可满足性检测。调用事件历史来实例化每个图，如果出现负环(形成环，权值小于0)，则对应的约束子句是不可满足的。如果所有的析取范式子句都不可满足，则这个约束条件是不可满足的。

定义5 在构造出来的带权有向图 $G=(V,E)$ 中，如果存在一条路径 $p:\langle v_0,v_1\rangle,\langle v_1,v_2\rangle,\cdots\cdots,\langle v_i,v_{i+1}\rangle,\cdots\cdots,\langle v_n,v_0\rangle,v_i\in V,i\in N^+$，则称 p 构成一个环。如果环上的权值之和小于0，则称为负环。

定理3 如果约束图出现负环，那么负环对应的约束是不可满足的[9]。

假设例1中事件的执行顺序如图11所示。

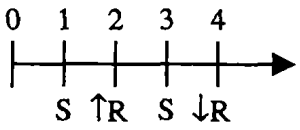

图11 执行顺序图

当时间 $t=1$时，事件 S 发生，把图10中相应的 S 结点换为0结点，然后在对应的带权边加上(或减去)t。如果当前转换的0结点相连的边箭头指向本身的，那么边上的权值减去 t；如果边的箭头指向其它结点的，那么边上的权值加上 t。重复转换直到每个带权有向图都出现负环或已到达当前时间为止。图10在时间 $t=1$时，如图12(a)所示。当 $t=2$时，如图12(b)所示，此时其中一个子图中已经出现负环，但整个约束规范仍是可满足。当 $t=3$时，如图12(c)所示，此时两个子图都已出现负环，根据定理3可得出结论：所检测的约束公式是不可满足的。

如果利用 Floyd-Warshall 求最短路径算法来搜索实例图中的负环，在带权有向图中找到负环，则对应的约束公式是不

可满足的。这个搜索算法的复杂度为$O(n^3)$，但大多数情况下，由于负环是出现在0结点，因此可以对Floyd-Warshall算法加以改进，使其只搜索以0结点开始的路径，那么算法复杂度就减小到$O(n^2)$。

这种方法能够实时地检测运行时时间约束的可满足性，在检测负环时，主要依赖求最短路径的搜索算法，提高最短路径搜索算法就可以提高检测的效率。文[9]扩展了这种方法，使其能够应用于分布式实时系统。

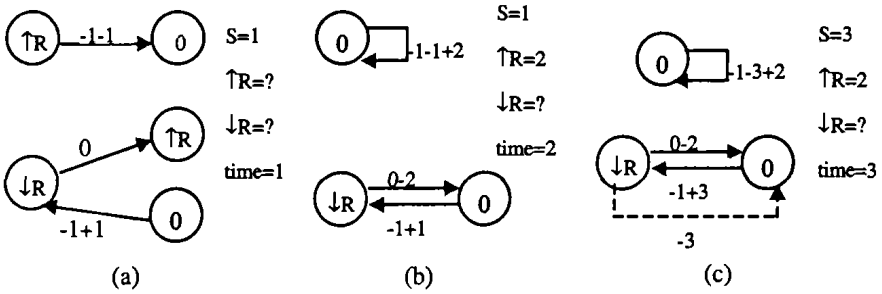

图12　实例化图

5　方法三

在给定的系统约束公式中，常常蕴含了一些额外的约束条件，这些条件容易被忽略，导致有时不能确保系统的实时性。例如约束条件为：$@(e_1,i)+1000 \geqslant @(e_2,i) \wedge @(e_2,i)-999 \geqslant @(e_3,i)$，蕴含了约束条件$@(e_1,i)+1 \geqslant @(e_3,i)$。假设事件$e_1$发生的第$i$次实例在时间1，到时间2还没有事件$e_2$和$e_3$任何的实例出现，如果没有考虑到蕴含约束条件，有时就不能立刻检测到系统的不满足性，达不到实时的要求。

实现动态时间约束规范有三种方法：选择适当的时间约束，调度构造时改变时间约束，调度定时实体。文[3,4,10]都阐述了一种选择适当的时间约束的方法，能够利用最小时间约束集来检测系统约束的可满足性，不但可以检测显式约束条件，而且还可以检测到隐式的约束条件，完全解决了上述问题，很好地实现实时检测约束条件。

定义6　一个简单约束表示为：$T_1+D \geqslant T_2$，其中D是一个整数常量；T_1、T_2是两个时间量，可以为@函数、相关@函数或0，且有如下限制：

·T_1、T_2至少有一个是@函数；

·T_1或T_2用@函数表示时，其参数必须是算术表达式$a*i+b$，变量$i \in N^+$，a、b是整数常量，且$a \geqslant 0$；

·如果T_1(或T_2)是相关@函数，那么T_2(或T_1)必须是它的参考时间，且其参数必须是整数常量。

简单约束在文[10]中也称为基本约束。当利用相关@函数表达时，可以使用简写形式。例如$@(e_1,i)+10 \geqslant @(e_2,@(e_1,i),2)$可以缩写为$@(e_1,i)+10 \geqslant @(e_2,\%2)$。

定义7　设简单约束集为S，$S_c=\{c_i \mid c_i \in S, c_i \neq c, i=1,\cdots,k, k \geqslant 1\}$。如果在计算过程中任一时间$t$，合取式$\wedge_{i=1}^{k} c_i$是可满足的，则约束$c$也是可满足的；反过来，$c$是不可满足的，合取式$\wedge_{i=1}^{k} c_i$也是不可满足的。那么就称约束$c$是不必要的。反之，就称约束$c$是必要的。

定义8　设系统约束的合取式为C，SC是由C中的所有显式和隐式约束构成的集合。令$C'=\wedge_{a \in S} c_i$，其中$\forall S \subseteq SC$。如果在任一计算时间t，C'是可满足的，则C也是可满足的；反过来，C是不可满足的，C'也是不可满足的；那么就称S是C的有用约束集。令$S_c \subseteq S$，如果对于所有的有用约束集S，都有$|S_c| \leqslant |S|$，则称S_c是C的最小有用约束集。

定理4　设约束合取式为C，如果S_1是C的一个有用约束集，且$S_2=S_1-\{c\}$，c是S_1的一个不必要的约束，那么S_2是C的一个有用的约束集。

文[3]中给出了定理4的详细证明方法。

5.1　约束图

用约束图G来表示约束合取式C，其中约束函数用顶点表示，每个简单约束都用一条带权有向边表示，边上的权值为对应简单约束的截止时间或延迟。例如约束合取式C为：$@(e_1,i)-6 \geqslant @(e_2,i+1) \wedge @(e_1,i)+5 \geqslant @(e_3,@(e_1,i),1) \wedge @(e_2,i+1)+9 \geqslant @(e_3,@(e_2,i+1),1) \wedge @(e_1,i) \leqslant 10$，它对应的约束图如图13所示。

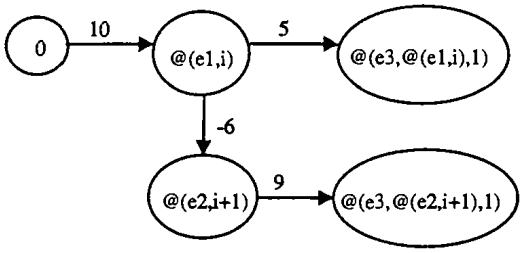

图13　约束图

约束图中，对于任意两个顶点u、$v \in G$，如果u和v存在一条权值为l的路径$p: u \square w \square v$，则加一条权值为l的边$u \square v$，且$u \square v \notin G$，那么所构成的图G'就称为完全约束图。如果在约束图中，一条路径的源顶点和目标顶点是同一顶点，则构成一个环。如果环的权值为负数，就称为负环。如果把约束合取式C中的变量用具体数字代替，就形成实例化的约束合取式C'，由C'构造出来的约束图为实例化约束图。

定理5　如约束图中出现一个负环，那么对应的约束合取式是不可满足的。

在第二种方法中，每检测一个运行实例就需要$O(n^3)$的时间复杂度，即使改进之后也需要$O(n^2)$。如果能够在运行之前处理大多数的最短路径，就能够很大程度上减少运行时的检测时间，因此必须进行约束图的优化，找出必要的路径，然后只需检测必要的约束，减少计算时间，从而更符合实时系统时间要求。

定理6　给定一个约束图$G=(V,E)$，$\forall u,v \in V$，l是从u到v最短路径的长度。如果权值为l的边$\langle u,v \rangle \in E$，那么从$u$到$v$任何长度大于$l$的路径都是不必要的。

从定理6可以引出一个推论。文[3]中给出了定理6及其推论的相关证明。

推论　正环是不必要的路径。

定理7　假设约束合取式为C，$G=(V,E)$为C对应的完全约束图，$G'=(V,E')$是C约束图，如果$\forall u,v \in V, u \neq v, \langle u,v \rangle \in E'$，且其权值为$l$，那么当$l$是$u$到$v$最短路径的长度时，由$G'$的边对应的约束所构成的约束集$S$是$C$的有用约束集。

定理7的证明见文[3]。定理7说明了最短路径所对应的约束构成了一个有用约束集，但并不是所有最短路径都表示必要的约束。因此还需要对最短路径进行筛选，判断它是否表示必要的约束。

定理8 给定约束图 $G=(V,E)$，$\forall u,v,w\in V,(u,v),(v,w)\in E$，$t_u,t_v,t_w$ 是顶点 u,v,w 在计算时的时间点，d_1,d_2 分别是边(u,v)和(v,w)的权值。为能够检测到路径 $p=u\square v\square w$ 对应隐式约束满足性的最早时间。如果总有 $t_v\leqslant t_p$，那么路径 p 是不必要的。

从定理8可得到一些推论，文[3]中有定理8及其推论的详细证明过程。

推论8.1 约束图 $G=(V,E)$，$\forall u\in V$，如果 v 是一个相关顶点，那么以 v 为源顶点或目标顶点的路径至少有一个中间顶点是不必要的。

推论8.2 约束图 $G=(V,E)$，$\forall$ 路径 $p\in G$，p 至少有一个中间顶点，如果路径 p 第一条边的权值是负数，那么 p 是不必要的。

推论8.3 约束图 $G=(V,E)$，$\forall$ 路径 $p\in G$，p 至少有一个中间顶点，如果路径 p 第一条边的权值是正数，且不大于路径 p 的长度，那么 p 是不必要的。

利用定理7和定理8及其推论来优化约束图。通过搜索任意两顶点之间的最短路径，使用删除不必要的路径，然后删除最短路径中不必要的约束。如果在优化过程中，检测到负环，根据定理3可知对应的约束是不可满足的。文[4]描述了约束图的优化算法，它利用 Floyd-Warshall 算法来搜索最短路径，算法复杂度为 $O(n^3)$。这个算法可以删除约束图大部分不必要的路径。

5.2 检测时间约束

定理9 约束图 $G=(V,E)$，$\forall u,w,(u,v)\in E$，设 t_u,t_v 表示顶点 u,v 在计算时的时间点，d 为边(u,v)的权值。如果 t 为最早检测到边(u,v)表示的约束不可满足性的时间，则有

$a.\ t\geqslant t_u+d$，当 $d\geqslant 0$

$b.\ t=t_u$，当 $d<0$

文[3]给出了定理7的详细证明方法。根据定理9可以检测一个简单约束的不可满足性。假设要检测的简单约束为：$T_1+d\geqslant T_2$。如果 T_1不是相对事件实例的发生时间，那么就需要在时间 T_1检测约束的可满足性，才能保证尽可能早的检测到约束的不可满足性。如果 T_1是相对事件实例的发生时间，则必须在时间 T_2检测不可满足性。同时，还需要设置定时器。对于约束 $T_1+d\geqslant T_2$，如果 $d\geqslant 0$检测时间在 T_1，且 T_2在这个时刻还没定义，那么截止时间定时器必须为事件实例 et_2设置截止时间，这个定时器的终止时间为 T_1+d。如果 et_2在设置的时间之前发生，则约束是可满足的，可以取消定时器的截止时间；否则当定时器终止时，约束就是不可满足的。如果 $d<0$，检测时间在 T_2，事件实例 et_1还没到发生时间 T_1，那么延迟定时器必须为检测 et_1发生而设置一个延迟，这个定时器的终止时间是 T_2-d。如果 et_1在设置的时间之前发生，则约束是不可满足的；否则当定时器终止时，约束就是可满足的。具体检测算法见文[4]，算法复杂度为 $O(n)$。

在一般情况下，系统给出的约束都不是单个的约束条件。对于给定的约束合取式 C，可以利用约束图简化约束条件。但在实例化图时，有些顶点可能会合并为一个顶点，即实例化之后它们都表示同一个事件的发生实例，这些顶点称为等价顶点，等价顶点会产生新的必要路径，因此需要进一步分析。

产生等价顶点有两种情况：

(1)当用特定值实例化发生函数的参数时，可能引起顶点的合并。如图14(a)所示，当 $i=1$时，v_1和 v_2顶点就合并为同一顶点，如图14(b)v 顶点，合并之后产生新的 u 到 w 的最短路径。

(2)运行时相关顶点可能会合并到其它的顶点。如图14(a)的顶点 v_1和 v_3，当 $i=2$，且 $\#(e_2,@(e_1,2))=1$时，它们就合并为图14(c)中的顶点 v'，合并之后产生新的 v 到 w 的最短路径。

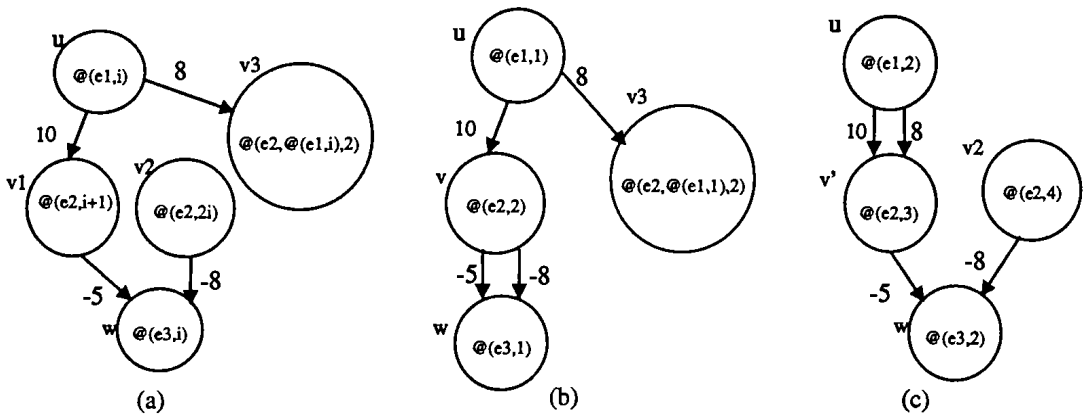

图14 等价顶点

对于第一种情况，可以用静态实例化或者动态实例化两种方法来解决。静态实例化就是在编辑约束图时，用具体值代替参数变量，把约束图实例化，然后找出所有的不同的实例约束图，如果约束图有 n 个顶点，则最坏情况下共有 $n(n-1)/2$ 种实例图。这种方法在编辑约束图时就能找出所有的等价顶点及其产生的最短路径，因此在运行检测时不会受到影响。但它的代价就是空间复杂度从 $O(n^2)$增加到 $O(n^4)$。动态实例化就是把等价顶点检测推迟到运行时才执行，这种方法将增加额外的 $O(n^3)$时间复杂性。

至于第二种情况，由于涉及到相关顶点，因此必须等到运行期间，当参考事件的实例发生时，才能确定等价顶点，这就要求在运行时搜索更多的必要路径。

定理10 给定一个已优化的约束图 $G=(V,E)$，$v,v_r,v_e\in V$，其中 v_r 是相关顶点，v 是 v_r 的参考顶点。假设在一个计算中，在 t_v 时刻顶点 v_r 和 v_e 实例化为等价顶点，并产生新的最短路径 p。如果 v_r 是一个将来相关顶点并且 v 是 p 的源顶点或目标顶点，那么 p 才是必要的。

文[3]给出了定理10的详细证明过程。根据定理10，可以在运行时标识出由相关顶点实例化产生的等价顶点，并找出新的最短路径，然后更新约束图。更新算法见文[4]，它的时间

复杂性为O(n)。

从上面的分析，可以找出实例约束图的所有必要路径，然后检测每个必要约束条件，从而实现整个约束合取式C满足性的检测。如果有一个必要约束不可满足，则C是不可满足的。只有在所有的必要约束都能满足时，C才是可满足的。具体可满足性检测算法见文[4]，如果采用静态实例化方法来找等价顶点引起的最短路径，则这个检测算法的时间复杂性为O(n)。如果采用动态实例化方法，则这个检测算法的时间复杂性为$O(n^3)$。

这种方法也能够实现实时检测运行时的时间约束可满足性，而且与第二种方法相比，它不仅可以检测显式的时间约束，而且还能够检测隐式时间约束的可满足性。在运行时检测的时间复杂度为O(n)，明显小于第二种方法，但它必须事先进行约束规则的优化，这需要增加额外的时间和空间复杂度。

6 分析比较

上述三种检测实时系统中时间约束满足性的方法，都是先把时间约束条件用实时逻辑表达出来，然后转化为图形的方法来解决。但它们都有各自的特点，适合于检测不同情形的实时系统。

第一种方法利用系统规范和安全性断言的否定构造一个带权有向图，然后进行有向图的正环检测，最后根据检测到的正环判断系统规范与安全性断言的一致性，从而达到检测的目的。这种方法包括三个步骤：构造有向图、检测正环和一致性检测。构造有向图需要$O(n^2)$的时间复杂度。正环检测的时间复杂度为$O(n^2)$。假设检测到的正环个数为k，S_2中P_i包含2个约束条件，则算法时间复杂度为$O(2^k)$。这种方法适合于设计系统对系统规范的验证，但约束图是不可判定的。文[5]对此进行了改进，增加了可决定性因素。与另外两种方法相比，这种方法并不能很好地实现运行时的约束检测。这种方法已应用于德克萨斯大学开发的SARTOR项目中。

很多情况下，实时系统有许多实时要求，如果不能满足系统时间约束将会导致系统错误的反应。在设计一个安全性关键的实时系统时，好的工程实践意味着具有清晰、准确的系统行为规范。当系统实现之后应该能确保它的行为满足需求，就需要检测运行时的行为[16]。第二和第三种方法就实现了在运行时的时间约束检测。

第二种方法说明了同步与异步的检测模型能够实时地对系统约束条件的检测。运行时利用Floyd-Warshall算法来搜索有向图的正环，时间复杂度为$O(n^3)$。考虑到大多数情况下，负环出现在0结点，对Floyd-Warshall算法加以改进，使其算法复杂度减小到$O(n^2)$。通过对搜索算法的改进还可以进一步减小算法的时间复杂度。与第三种方法相比在运行时，参加检测的约束规范显得有点冗余。目前已在IBM RS/6000工作站AIXv.3上应用。

第三种方法相对于前两种在运行时检测时间约束具有更好的实时性，不过它必须在运行前先优化处理约束图，减少运行时系统检测的约束条件。如果采用静态实例化方法解决等价顶点问题，则检测算法的时间复杂性为O(n)，但相应的空间复杂度从$O(n^2)$增加到$O(n^4)$。如果采用动态实例化方法，却增加额外的$O(n^3)$时间复杂性，检测算法的时间复杂性为$O(n^3)$。另外，这种方法不但能够检测系统给定的约束条件，还能够检测到蕴含的时间约束条件，能够尽可能早地检测到系统约束的可满足性。但它必须事先进行约束规则的优化，在运行前需要增加额外的时间和空间复杂度。

结论 实时系统是对时间具有严格的要求，运行时的时间约束条件检测是非常重要的。利用实时逻辑非常适合于表达实时系统的时间约束，并通过利用图形化检测方法实现系统时间规范的验证，确保了系统的实时性。

本文针对具有严格时间要求的系统，阐述并分析了三种利用实时逻辑实现时间约束检测的方法。第一种方法通过检测系统规范和安全性断言的一致性来验证约束的满足性，非常适合于系统规范的设计与可满足性检测，算法的时间复杂度是$O(n^2)+O(n^2)+O(2^k)$。第二种方法利用实时逻辑与约束图的方法实现运行时的时间约束检测，但检测时系统约束条件不够第三种方法简约，算法时间复杂度为$O(n^3)$，改进之后为$O(n^2)$。第三种方法通过对约束图的处理，减少运行时系统检测的约束条件，从而减少运行时的时间约束条件的搜索时间，算法的时间复杂度为O(n)，在实时性和检测效率明显优于前两种方法。但需要运行前优化约束规则，将会增加额外的时间和空间复杂度。

通过对三种检测方法的分析，进一步说明实时逻辑在实时系统中约束规范检测的重要作用，显示出实时逻辑与图论相结合在解决实时系统中时间约束问题的优越性。

参考文献

1 Jahanian F, Mok A K. A Graph-Theoretic Approach for Timing Analysis in Real Time Logic. In: Proc. Real-Time Systems Symp. 1986

2 Chodrow S E, Jahanian F, Donner M. Run-time monitoring of real-time systems. In: Robert Werner, ed. Proc. of the Real-Time Systems Symposium, San Antonio, Texas, USA, December 1991. IEEE Computer Society Press 1991. 74～83

3 Mok AK, Liu G. Early Detection of Timing Constraint Violations: [Technical Report], Real-Time System Lab. Department of Computer Sciences, The University of Texas at Austin, 1997

4 Mok A K, Liu G. Efficient Run-Time Monitoring of Timing Constraints: [Technical Report]. Real-Time system Lab. Department of Computer Sciences, The University of Texas at Austin, 1997, an abridged version to appear in Proc. Third IEEE Real-Time Technology and Applications Symposium, June1997

5 Andrei S, Chin W-N, Incremental Satisfiability Counting for Real-Time Systems. 10th IEEE Real-Time and Embedded Technology and Applications Symposium, 2004

6 Sodhi S, Cheng A M K. Optimizing Timing Analysis and Verification of Embedded Systems using Rule-Based-Analytic Techniques. In: Proc. WIP Session of IEEE-CS Real-Time Systems Symposium, Cancun, Mexico, Dec. 2003

7 Dasdan A. A Strongly Polynomial-Time Algorithm for Over-Constraint Resolution. 10th International Workshop on Hardware/Software Co-Design, 2002

8 Jahanian F. Run-Time Monitoring of Real-Time Systems. In Advances in Real-Time systems, S. Son (ed), 1995

9 Raju S C V, Rajkumar R, Jahanian F. Monitoring timing constraints in distributed real-time systems. In: Proc. of IEEE 13th Real-Time Systems Symposium, Phoenix, AZ, Dec. 1992. 57～67

10 Liu G, Mok A K, Konana P. A Unified Approach for Specifying Timing Constraints and Composite Events in Active Real-Time Database Systems. In: Proc. of real-Time Technology and Applications Symposium, June 1998

11 Ekelin C, Jonsson J. Real-Time System Constraints: Where do They Come From and Where do They Go. In: Proc. of the Intl. Workshop on Real-Time Constraints, Alexandria, Virginia, Oct. 1999

12 Gupta R K. A Framework for Interactive Analysis of Timing Constraints in Embedded Systems. In: Proc. Int. Workshop on Hardware/Software Codesign, 1996

13 Anderson S, Küster Filipe J. Guaranteeing temporal validity with a real-time logic of knowledge. In: Proc. of the 23rd IEEE International Conference on Distributed Computing Systems Workshops - First Intl. Workshop on Data Distribution for Real-Time Systems 2003. Providence, Rhode Island, USA, IEEE Computer Society Press, 2003. 178～183

14 Stuart D A, et al. Formal Methods for Real-Time Systems. Dissertation Presented to the Faculty of the Graduate School of The University of Texas at Austin in Partial Fulfillment of the Require-

ments for the Degree of Doctor of Philosophy, May 1996
15 Fohler G, Huber Ch. Integration of RTL and Precedence Graphs with a Static Scheduler as Verifier. In: Proc. Euromicro Workshop on Real-Time Systems, Va¨ steraos, Sweden, June 1994
16 Peters D K, Parnas D L. Requirements-Based Monitors for Real-Time Systems. IEEE Transactions on Software Engineering, 2002
17 Shaw A, Rupp A. Real-time programming with time-stamped event histories. TR#UW-CSE-96-05-02, Dept. of Computer Science and Eng., U. of Washington, 1996

(上接第186页)

向服务器发送资金帐号和指纹特征值，接收服务器对登记的指纹存盘状态等。当银行系统在执行储户开户或增加指纹提款方式时，需要调用该函数。

该函数的主要参数：储户资金帐号、指纹总数、指纹验证方式、网点编号、操作员姓名等。返回登记是否成功的标志。

(2)银行储蓄指纹比对 YhcxMatchFp()

该函数主要完成：接收指纹验证命令，通过储户资金帐号向服务器索取对应的指纹数据，向指纹设备控制器发送在服务器中索取的指纹数据，接收与新输入的指纹的比对结果，根据比对结果向银行系统发送是否可以继续取款等相关业务。当银行系统执行储户取款等有关业务时，如果需要确认储户身份，则调用该函数。

该函数的主要参数：储户资金帐号、网点编号、操作员姓名等。返回指纹验证是否成功的标志。

(3)银行指纹数据的增加 YhcxInsertFp()

当储户需要再增加其它手指的指纹数据时，需要调用该函数。该函数主要完成：指纹数据的采集、服务器中指纹数据的新增等。

该函数的主要参数：储户资金帐号、网点编号、操作员姓名等。返回增加指纹是否成功的标志。

(4)银行指纹数据的删除 YhcxDeleteFp()

当储户不再采用指纹取款时，需要将原来登记的指纹取消。该函数主要完成：将指定的资金帐号的全部指纹数据删除。

该函数的主要参数：储户资金帐号、网点编号、操作员姓名等。返回删除指纹是否成功的标志。

(5)银行储蓄指纹重新登记 YhcxReInputFp()

当储户需要重新替换指纹数据时，必须调用该函数。该函数主要完成：接收指纹重新登记命令，向指纹设备控制器发送指纹登记命令，接收指纹设备控制器发来的指纹特征值，向服务器发送资金帐号和指纹特征值，接收服务器对重新登记的指纹存盘状态等。

该函数的主要参数：储户资金帐号、指纹总数、指纹验证方式、网点编号、操作员姓名等。返回登记是否成功的标志。

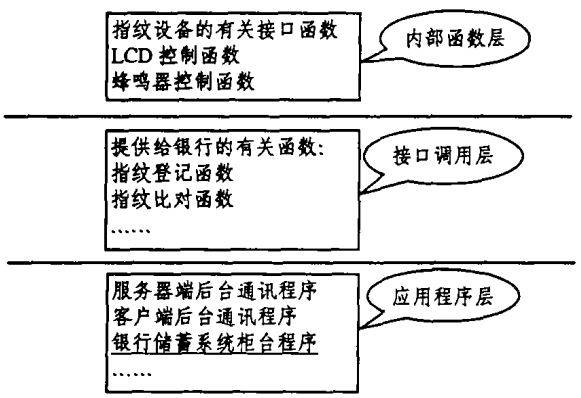

图4 程序或函数的层次结构示意图

(6)银行键盘字符输入 YhcxKeybStr()

如果指纹设备有键盘连接时，可以通过该函数调用来获取键盘的输入值，作为密码输入。

该函数的主要参数：需要输入的字符长度等。返回输入密码是否成功的标志，以及输入的密码串。

4.4 程序或函数的层次结构

本系统分内部接口、外部接口；底层调用、接口性调用；服务器端后台程序、客户端后台程序、银行储蓄柜台系统等，其层次结构如图4所示。

5 数据库表结构设计

数据库存储储户在银行预留的指纹数据，在储户取款等有关业务时使用。本系统提供给银行系统一个数据库(当然也可以使用原银行系统现有的数据库)，存放在银行 Unix 服务器上。可用 Informix、Sybase、Oracle 等数据库系统。系统提供了三个表。

(1)资金帐号表：主要是本系统与原银行系统之间的一个联系表。主要有：{资金帐号、指纹总数、指纹验证方式}等字段。

(2)指纹数据表：该表存储储户的指纹数据。主要有：{资金帐号、指纹数据}等字段。

(3)指纹日志表：该表存储对指纹数据的操作日志。主要有：{日期、时间、网点编号、终端机号、操作摘要、操作员}等字段。

结束语 本文设计的系统包括两部分：一部分是银行指纹设备，它与 Unix 系统的终端相连，主要完成指纹的登记、指纹验证、密码输入等功能。另一部分是本系统提供的通讯管理程序及相函数，它主要与指纹设备进行通讯获取数据或传递数据，还要与原来的银行系统进数据交换，另外还要管理服务器上的指纹数据等。通过这两部分的有机结合，从而实现了银行储蓄系统柜台程序对指纹设备的操作，与储户指纹数据的交换，达到从采集指纹到验证指纹身份的目的。并且将原来的数字密码系统转换为本文设计的指纹密码系统，使储户的指纹密码永不丢失，保证了储户资金的安全。实现指纹密码后，不仅对银行是一种福音，更是储户的福音。不论从社会效益，还是从经济效益来看，都是不可估量的，对传统的数字密码更是一场革命。

参 考 文 献

1 杨光友，朱宏辉. 单片微型计算机原理及接口技术. 北京：中国水利水电出版社，2002
2 余永权，等. 单片机在控制系统中的应用. 北京：电子工业出版社，2003
3 张成海，张铎. 现代自动识别技术与应用. 北京：清华大学出版社，2003
4 张健沛. 数据库原理及应用系统开发. 北京：中国水利水电出版社，1999
5 杨孝如，等. Sybase 数据库系统管理指南. 北京：中国水利水电出版社，1997
6 付继彬，等. Oracle8入门与提高. 北京：清华大学出版社，2000