一种基于Prolog的时间约束业务流程验证方法

陈鹤文 周 勇 燕雪峰
(南京航空航天大学计算机科学与技术学院 南京 210016)

摘 要 随着互联网技术的快速发展,对复杂系统业务流程建模的需求越来越大。针对带有时间约束的业务流程模型的正确性验证问题,提出了一种基于节点转换规则的图分解算法,将业务流程模型转换为运行时流程轨迹集合;设计了流程轨迹集合到Prolog的转换,将轨迹中的节点与时间约束转化为Prolog事实,提出了一种业务流程模型到Prolog语言的转换算法;将持续时间、周期循环与固定时刻3种时间模式转换为Prolog规则,以其支持业务流程模型3种时间模式的验证。最后对一个带有时间约束的医疗流程实例进行了验证。

关键词 时间约束,时间模式,业务流程模型,验证方法
中图法分类号 TP311 **文献标识码** A **DOI** 10.11896/j.issn.1002-137X.2016.6.035

Prolog Based Approach to Validate Time Constraints in Business Process

CHEN He-wen ZHOU Yong YAN Xue-feng
(College of Computer Science and Technology, Nanjing University of Aeronautics and Astronautics, Nanjing 210016, China)

Abstract With the rapid development of Internet technology, the demand for business process modeling of complex system is increasing. In order to verify the correctness of the business process model with time constraints, this paper put forward a kind of graph decomposition algorithm based on node switching rules, which transforms the business process model into execution trace set and transforms the execution trace set to Prolog. That is to say, the trace of nodes, gateway and time constraints are all converted into Prolog fact. This paper put forward an algorithm that transforms the business process model to Prolog language and transforms the duration time pattern, cycle time pattern and fixed time pattern into Prolog rules, supporting the validation of the business process model on three time patterns. Finally, a medical process instance with time constraints was verified.

Keywords Time constraints, Time pattern, Business process model, Verification method

1 引言

业务流对保持企业竞争力起着越来越重要的作用。良好的业务流程设计是保证企业灵活运行的关键,对企业的业务运营有着指导意义。传统的流程管理技术建立在微动态或静态环境下,已难以适应多变的、日益复杂的动态流程管理实践[1,2];与此同时,企业还希望在应对这些变化时能够有效控制成本,这就要求业务流程在开发和测试阶段能够尽可能地保证其正确性和合理性。由于开发人员与测试人员对该业务流需要描述的业务并不熟悉,难免会造成描述上的偏差,效率也会受到影响。因此,如何保证业务流程模型的正确性、合理性是当前业务流模型研究的重点。目前针对业务流程的验证方法已存在不少研究成果:文献[3,4]首先列举了流程中的所有节点,并将流转规则以子集的形式进行描述,然后基于路径搜索判断子集的可满足性,从而验证业务流程是否可以执行;此方法采用基于数学的形式描述进行业务流程验证,具有自动化的可行性,但子集的方法缺乏统一的标准,目前业界对此也无自动化的支撑工具。文献[5-7]在业务流程建模阶段采用Petri网,其具有图形化描述、精确的语义、强大的表达能力和基于状态而不是基于事件等优点,但Petri网具有很高的空间复杂度,若为了降低复杂度而限制Petri网有界,则会降低其表达能力。本文采用逻辑程序语言Prolog,通过将模型与时间约束转换为Prolog语言中的事实,自定义Prolog规则对其一致性进行验证,提出一种模型到Prolog的转换规则算法,其能够准确对模型中涉及到的时间约束进行验证,从而保证了该模型的一致性。

2 带有时间约束的业务流程模型

业务流程模型的主要目的是,以模型元素及规范的形式对复杂的流程结构与关系予以抽象表达,并通过模型使流程参与者对业务流程逻辑达成一致的理解。

传统的业务流程模型大多致力于执行顺序的研究,目前越来越多的业务流程模型需要涉及到对时间的约束。Andreas Lanz, Barbara Weber 和 Manfred Reichert 等通过对现有的医疗、教育等领域的研究总结出几种时间模式,用于刻画业务流程模型处理时序方面的能力[8]。按照不同的时间约束

到稿日期:2015-04-17 返修日期:2015-08-21 本文受国防科工局十二五重大基础科研项目(c0420110005,NS2013091)资助。
陈鹤文(1991—),男,硕士生,主要研究方向为软件工程,E-mail:313499522@qq.com;周 勇(1975—),男,副教授,主要研究方向为人工智能、专家系统等;燕雪峰(1975—),男,副教授,主要研究方向为计算机网络、分布交互仿真等。

种类，可将现阶段时间模式分为如下 3 种。

• 模式 1：持续时间模式(Dur)

模式 1 是时序限制中最常用的时间模式之一。该模式允许两个节点或活动之间存在时间限制，一般用来确保业务流程符合整体的规则和制度以及限制资源使用和保证满足外部的基准参照(例如服务等级约定)。

• 模式 2：周期循环模式(Cyc)

模式 2 允许定义两个节点活动之间存在周期循环，即两节点间可以重复执行，但必须满足循环周期中两点之间的时间限制。

• 模式 3：固定时刻模式(Fixed)

模式 3 描绘一个特定的活动或者业务流程实例要在一个特定的日期执行，常用的时间限制包含最早和最晚开始时间、最早和最晚结束时间。

业务流程模型一般由活动、事件和入口元素组成，定义如下。

定义 1 业务流程模型为一个多元组 $B=(N,E,\tau,\gamma,\Gamma,C)$。其中 N 表示按照模型中顺序保存的节点集合；$E\subseteq N\times N$ 是模型中节点之间边的集合；$\Gamma=\{\text{Activity},\text{Event},\text{EXCL},\text{INCL}\}$ 表示节点类型，包括活动、事件和两种入口元素(或元素和与元素)；$\tau: N\rightarrow\Gamma$ 是将节点标注成某一节点类型的函数；$\gamma: C\rightarrow\{\text{Dur},\text{Cyc},\text{Fixed}\}$ 是将时间约束标注成某一时间模式的函数；C 是定义在该模型上的时间约束集合。

3 基于 Prolog 的模型验证方法

采用 Prolog 对业务流程模型的性质进行验证。首先基于节点转换规则提出图分解算法，将业务流程模型转换为运行时流程轨迹集合；将轨迹中节点与时间的约束转化为 Prolog 事实，提出一种业务流程模型到 Prolog 语言的转换算法；将 3 种时间模式转换为 Prolog 规则，最后根据事实与规则进行一致性的验证。

执行轨迹是指在业务流程模型中一次运行的踪迹，由踪迹上的节点组成，定义如下。

定义 2 设 $B=(N,E,\tau,\gamma,\Gamma,C)$ 是业务流程模型，B 的一个执行轨迹为从起始活动到结束活动的一条有向路径，用 $T=\langle N_1,N_2,\cdots,N_n\rangle$ 表示，其中 N_i 表示轨迹 T 中包含的活动节点集合。用 C_T 表示为轨迹 T 中的时间约束集合。

3.1 业务流程模型的图分解算法

业务流程模型中的节点可以分为活动节点、与元素和或元素这 3 种。根据节点的不同，模型在执行时的流程轨迹有所区别。将前面 3 种节点按如下规则进行转换。

• 活动节点(Activity)

图 1 所示为活动节点。活动节点是业务流程模型中出现最多的一种节点，只需将活动节点直接加入到轨迹中。

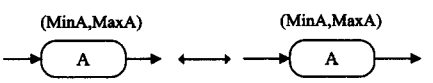

图 1 活动节点

• 或元素(EXCL)

图 2 所示为或元素。或元素使得模型同一时间内在节点 A 与节点 B 之间只能选择其一执行，故分裂为两个不同的执行轨迹，一条执行轨迹包含节点 A，另一条执行轨迹包含节点 B。

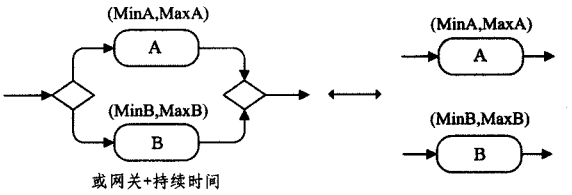

图 2 或元素

• 与元素(INCL)

图 3 所示为与元素(部分文献中也称平行元素)。与元素要求模型在某一事件内同时执行节点 A 与节点 B，故将两节点转换为一个新节点，同时将新节点的时间模式设定为两节点中时间模式的交集。

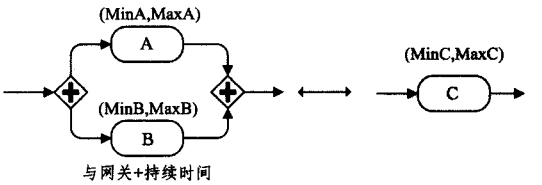

图 3 与元素

如图 3 所示，$MinC=\max(MinA,MinB)$，$MaxC=\max(MaxA,MaxB)$，汇聚后的节点为 C，则在这一元素中需满足限制($MinC$,$MaxC$)，最后将节点 C 加入到执行轨迹中。

根据以上节点转换规则，给出图分解算法。算法的输入为业务流程模型 B，输出为 B 的执行轨迹集合 $T=\{T_0,T_1,T_2,\cdots,T_n\}$，算法如下。

算法 1 Module_TransTo_Trace

```
1. 输入：业务流程模型 B=(N,E,τ,γ,Γ,C)
2. 输出：B 的执行轨迹集合 T={T1,T2,…,Tn}
3. T={T0}
4. For each Ni in N
5. If (!Visited(Ni))//屏蔽分支中的节点
6. If τ(Ni)=Event//对于 event 节点不做处理
7. Goto next(Ni)
8.    Else If τ(Ni)=Activity
9.       For each Tj in T
10. AddNodeToTrace(Tj,Ni)
11.      End for
12. Goto next(Ni)
13. Else if τ(Ni)=EXCL then//EXCL 表示或入口元素
14.      Ntemp<-GetBranchActivityNodes(Ni)
15. If(Ntemp)>1 //处理两个及两个以上的分支
16. For each Tj in T
17. If (NextNode(Tj,Ni)) //选取后继节点是 Ni 的轨迹
18. For each Nm in Ntemp
19. Visited(Nm)//标记访问过的节点
20.              Tm<- Tj
21. AddNodeToTrace(Tm,Nm)
22. AddTraceToSet(T,Tm)
23. End for
24. DeletTrace(T,Tj)//删除被覆盖的轨迹
25.          End if
26. End for
27.      End if
28. Goto next(Ni)
29. Else if τ(Ni)=INCL then
```

```
30.  N_temp<-GetBranchActivityNodes(N_i)
31. Min<-GetMinDuration(N_temp)
32.      Max<-GetMaxDuration(N_temp)
33. N_new<-New(Min,Max)
34.      Visited(N_temp)//标记访问过的节点
35. For each T_j in T
36. If NextNode(T_j,N_i)//选取后继节点是 N_i 的轨迹
37. AddNodeToTrace(T_j,N_new)
38.        End if
39.      End for
40. Goto next(N_i)
41. End if
42. End if
43. End for
```

其中，*Visited*(N_i)将 N_i 节点进行标记；*AddNodeToTrace*(T_i,N_i)是将 N_i 节点加入到 T_i 轨迹中；*AddTraceToSet*(T,T_m)是将 T_m 轨迹添加至 T 集合中；*DeletTrace*(T,T_i)删除集合 T 中的 T_i 执行轨迹；*NextNode*(T_j,N_i)判断执行轨迹 T_j 的后继节点是否为 N_i；Goto *next*(N_i)是指继续遍历下一个节点；*GetBranchActivityNodes*(N_i)是根据该 N_i 入口元素得到其后的所有分支；*GetMinDuration*(N_{temp})与 *GetMaxDuration*(N_{temp})则是得到所有分支中最小和最大持续时间，*New*(*Min*,*Max*)则是新建一个节点，并将相应的最大最小时间赋予到该节点中。

算法中依次遍历业务流程模型中的节点 N_i，当 $\tau(N_i)$为事件时，不做处理，直接跳到下一节点(第 6 行，第 7 行)；当 $\tau(N_i)$为活动时，则直接将该节点加入执行轨迹集合 T 的各个元素中(第 8 行－第 12 行)；当 $\tau(N_i)$为或入口元素时，先得到该或入口所有分支节点集合 N_{temp}，循环现有执行轨迹集合中各元素 T_j，之后嵌套循环分支节点集合 N_{temp}，T_j 赋值给 T_m 临时变量后，若 N_i 是 T_j 的后继节点，则将分支节加入到 T_m，同时将 T_m 添加至集合 T 中，N_{temp} 循环结束后，将 T_j 从集合 T 中删除，这样就把该或入口分成若干条顺序路径(第 13 行－第 28 行)；当 $\tau(N_i)$为与入口元素时，先将所有分支中节点的最小持续时间取最小，最大持续时间取最大，之后将所有分支节点合并为一个节点 N_{new}，新节点的前置节点就是原分支中节点的前置节点，后置节点就是原分支中节点的后置节点，并且用新合并的最小最大持续时间进行限制，对于循环集合 T 中每一个元素 T_j，若 N_i 是 T_j 的后继节点，将新节点 N_{new} 加入其中(第 29 行－第 40 行)，至此业务流程模型中所有执行轨迹均被添加至执行轨迹集合中。

以图 4 为例，从 e_0 开始，其对于 $A1$ 节点来说是活动，直接将该节点加入到轨迹 T_0 中。判断 e_2 为或元素，则通过 *GetBranchNodes*(e_0)函数获得 $A3$ 和 $A4$ 节点，首先循环 $A3$ 节点，由于此时轨迹集合 T 中只有 T_0 轨迹，因此将节点加入后增加新的轨迹 T_m，其所含节点为($A1$,$A3$)，并且将原来节点删除，如此再循环 $A4$ 节点，则在分解 e_2 或元素后轨迹集合 T 中轨迹为{($A1$,$A3$),($A1$,$A4$)}。判断 $A6$ 为活动节点，则将其直接加入 T 集合各轨迹中后，新的 T 集合为{($A1$,$A3$,$A6$),($A1$,$A3$,$A6$)}。在到达 e_7 或元素节点时，同理可得新的 T 集合。当达到终点 e_{12} 时，可得最后的执行轨迹 T 集合为{($A1$,$A3$,$A6$,$A8$,$A11$),($A1$,$A3$,$A6$,$A9$,$A11$),($A1$,$A4A6$,$A8$,$A11$),($A1$,$A4$,$A6$,$A8$,$A11$)}。

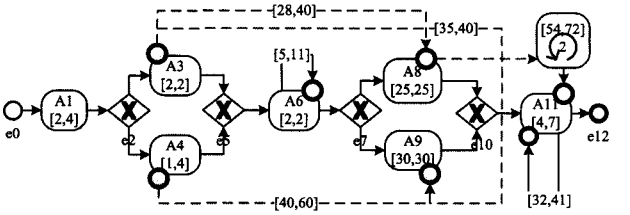

图 4　医疗流程图

3.2　业务流程模型到 Prolog 的转换

Prolog 是一种逻辑编程语言[9]，它建立在逻辑学的理论基础之上，最初被运用于自然语言等研究领域。由业务流程模型到 Prolog 转换的思路是将模型与时间模式转换为 Prolog 语言中的事实，实现用 Prolog 语言来对该业务流程模型的描述。

在第一步的模型分解中，通过算法可以将包含 3 种节点的模型分解为执行轨迹集合，第二步为了验证，需要将执行轨迹转换为 Prolog。用 *Facts*(T)表示执行轨迹集合 T 转换为 Prolog 事实的集合。具体算法如下。

算法 2　Trace_To_Pro

```
1. 输入：执行轨迹集合 T={T_1,T_2,…,T_n}
2. 输出：Facts(T)
3. For each T_i in T
4.   For each N_j in T_i
5.       Miny<-GetMinDuration(N_j)
6.       Maxy<-GetMaxDuration(N_j)
7. N_next<-FindNextActivity(N_j)
8. If Find(activity(N_j,N_next,Miny,Maxy),Facts)
9. Goto next(N_i)
10.      else
11. Add(Facts,activity(N_j,N_next,Miny,Maxy))
12.      End if
13.   End for
14. For each C_j in T_i
15.       If γ(C_j)=Dur
16. N_fore<-FindHeadActivity(C_j)
17. N_next<-FindTailActivity(C_j)
18. Miny<-GetMinDuration(C_j)
19. Maxy<-GetMaxDuration(C_j)
20. If FindConstra(constrainttp1(N_fore,N_next,Miny,Maxy),Facts)
21.         Goto next(C_i)
22. Else
23.          Add(Facts,constrainttp1(N_fore,N_next,Miny,Maxy))
24.         End if
25. Else if γ(C_j)=Cyc
26. N_fore<-FindForActivity(C_j)
27. N_next<-FindNextActivity(C_j)
28. Miny<-GetMinDuration(C_j)
29. Maxy<-GetMaxDuration(C_j)
30. Time<-GetLoopTimes(C_j)
31. If FindConstra(constrainttp2(N_fore,N_next,Time,Miny,Maxy),
    Facts)
32.         Goto next(C_i)
33. Else
34. Add(Facts,constrainttp2(N_fore,N_next,T,Miny,Maxy))
35.         End if
```

```
36. Else if γ(Cj)=Fixed
37.        Nnext<-FindNextActivity(Cj)
38.        Miny<-GetMinDuration(Cj)
39.        Maxy<-GetMaxDuration(Cj)
40. If FindConstra(constrainttp3(1,Nnext,Miny,Maxy),Facts)
41.         Goto next(Ci)
42. Else
43. Add(Facts,constrainttp3(1,Nnext,Miny,Maxy))
44.        End if
45. End if
46.     End for
47. End for
```

算法中，*activity*(*fore*,*next*,*min*,*max*)用来对活动节点进行描述，其中 *fore* 表示该节点的序号，*next* 表示该节点的后续节点，*min* 和 *max* 表示该活动节点的最小和最大可持续时间。对于属于时间模式 1 来说，用 *constrainttp*1(*head*,*tail*,*min*,*max*)描述其时间限制，其中 *head* 表示头节点，*tail* 表示尾节点，*min* 和 *max* 表示最小和最大的可持续时间；对于属于时间模式 2 来说，用 *constrainttp*2(*head*,*tail*,*times*,*min*,*max*)描述其时间限制，其中 *head* 表示头节点，*tail* 表示尾节点，*times* 表示需要循环的次数，*min* 和 *max* 表示最小和最大的可持续时间。

算法依次遍历轨迹集合 T 中的轨迹 T_i，单个轨迹 T_i 的组成部分为 $T_i=\langle N_1,N_2,\cdots,N_n\rangle$，通过循环其中 N_i 节点将其以后续节点、最大最小持续时间的形式记录为 *activity* 事实存入 *Facts* 集合，若已经存在相同事实则不存入(第 4 行—第 13 行)；通过循环其中时间限制 C_i，按照该时间限制属于不同的时间模式将其以头节点、尾节点、最大最小持续时间的形式记录为 *constrainttpn* 事实存入 *Facts* 集合，若已经存在相同事实则不存入(第 14 行—第 46 行)。至此，轨迹集合 T 中所有的节点和时间限制将加入到 *Facts* 集合中，以便下一步的验证。

以 3.1 节转换后的执行轨迹(*A*1,*A*3,*A*6,*A*8,*A*11)为例，算法循环该轨迹中第一个节点转换为 Prolog 事实 *activity*(1,3,2,4)；循环该轨迹中第一个时间限制，判断其属于 Dur 时间模式后，通过获取头尾节点和最大最小时间约束，转换为 Prolog 事实 *constrainttp*1(3,8,30,40)。依此类推，可将执行轨迹集合转换为 Prolog 事实。

3.3 Prolog 验证规则

通过前述算法转换将业务流程模型中的描述信息转换为 Prolog 中事实。对于规则部分，则需要通过分析待验证的时间约束，进而将其转换为 Prolog 规则。自定义 Prolog 规则对其一致性进行验证，提出使用 Prolog 来描述模型中时序限制的转换规则。

· 规则 1

通过 *constrainttp*1 和 *constrainttp*2 两种事实来描述模型中的时序限制，在这两种事实中通过指定首、尾节点后对时序限制进行描述，所以使用 *link*(*A*,*B*,*L*)函数来获得首节点为 *A*、尾节点为 *B* 的执行序列，并将该序列赋值给列表 *L*。在 Prolog 中[*A*,*B*]表示首节点为 *A*、尾节点为 *B* 的列表，[*H*|*T*]使用此列表可与任意的列表匹配，匹配成功后，*H* 绑定为列表的第一个项目的值，称为表头，而 *T* 则绑定为剩下的列表，称为表尾，*member*(*A*,*M*)则表示 *A* 节点为列表 *M* 中的元素。

```
link(A,B,L):-link(A,B,L,[]).
link(A,B,[A,B],M):-not(A=B),not(member(A,M)),not(member(B,M)),activity(A,B,_,_).
link(A,B,[A,H|T],M):-activity(A,H,_,_),not(member(A,M)),link(H,B,[H|T],[A|M]).
```

· 规则 2

使用 *getmin*([*A*,*B*],*S*)来计算在给定一个列表[*A*,*B*]下得到的节点中最小持续时间的总和并将其赋值给 *S*，提供后续验证时序限制的需要，Prolog 程序如下：

```
getmin([A,B],S):-activity(A,B,S,_). getmin([M,K|N],Min):-activity(M,K,M1,_),getmin([K|N],Min2),Min is Min2+M1.
```

· 规则 3

使用 *getmax*([*A*,*B*],*S*)来计算在给定一个列表[*A*,*B*]下，得到的节点中最大持续时间的总和并将其赋值给 *S*，提供后续验证时序限制的需要，Prolog 程序如下：

```
getmax([Q,W],E):-activity(Q,W,_,E).
getmax([R,T|Y],Max):-activity(R,T,_,M2),getmax([T|Y],Max2),Max is Max2+M2.
```

在此需要说明一点，当序列中所有节点的最小持续时间总和小于该时间限制的最大持续时间，并且序列中所有节点的最大持续时间总和大于该时间限制的最小持续时间时，称该序列满足时间限制的可能性(possible/notpossible)；当序列中所有节点的最小持续时间总和大于该时间限制的最小持续时间，并且序列中所有节点的最大持续时间总和小于该时间限制的最大持续时间时，称该序列满足时间限制的必然性(necessary/notnecessary)。

在第 1 节中将时间模式划分为持续时间、周期循环和固定时刻 3 种类型，下面分别讨论用 Prolog 语言描述的时序限制规则。

· 持续时间模式

由图 4 可得在该时间模式上的时间限制共 3 个，在 Prolog 的事实部分通过 *constrainttp*1(*Start*,*End*,*Min*,*Max*)对其进行描述，其中以序列所满足该时间限制的可能性和必然性作为输出结果。验证该时间限制的代码如下。

```
1. verify :-constrainttp1(Start,End,Min,Max),
2. link(Start,End,List),
3.    getmin(List,Min1),
4.    activity(End,_,Min,_),
5.    Min2 is Min1+Min,
6.    getmax(L,Max1),
7.    activity(End,_,_,Max),
8.    Max2 is Max1+Max,
9. write(),
10.       nl,fail.
```

· 周期循环模式

由图 4 可得在该时间模式上的时间限制共 1 个，在 Prolog 的事实部分通过 *constrainttp*2(*Start*,*End*,*Times*,*Min*,*Max*)对其进行描述，其中以序列所满足该时间限制的可能性和必然性作为输出结果。验证该时间限制的代码如下。

```
1. verify :-constrainttp2(Start,End,Times,Min,Max),
2. link(Start,End,List),
3.    getmin(List,Min1),
4.    activity(End,_,Min,_),
5.    Min2 is Min1+Min,
6.    getmax(List,Max1),
7.    activity(End,_,_,Max),
```

```
8.  Max2 is Max1+Max,
9. Min3 is Min2 * Times,
10.  Max3 is Max2 * Times,
11. write(),
12.  nl,fail.
```

· 固定时刻模式

由图 4 可得该时间模式上的时间限制共 2 个，在 Prolog 的事实部分通过 *constrainttp3*(1, *Tail*, *Earily*, *Late*)对其进行描述，其中以序列所满足该时间限制的可能性和必然性作为输出结果。验证该时间限制的代码如下。

```
1. verify:-constrainttp3(1,Tail,Earily,Late),
2.  link(1,Tail,List),
3.  getmin(List,Min1),
4.  activity(Tail,_,I,_),
5.  Min2 is Min1+I,
6.  getmax(List,Max1),
7.  activity(Tail,_,_,A),
8.  Max2 is Max1+A,
9.  write(),
10.  nl,fail.
```

4 实例验证与分析

4.1 实例验证

本文对业务流程模型进行验证，分析其时间模式的一致性，探讨基于 Prolog 与常规模型一致性验证方法的不同点与先进性。这里以文献[10]中的医疗模型为例。

如前文图 4 所示，以病人的医疗流程为例进行一致性验证。图中节点代表不同时间段的行为，如登记、化验、手术等，节点中的属性表示在该活动的最小、最大持续时间，在不同节点之间对其持续时间进行了限制，周期循环的两节点之间同样存在限制。

图 4 中各节点含义如表 1 所列。

表 1 节点含义

A1	A3	A4	A6	A8	A9	A11
(2,4)	(2,2)	(1,4)	(2,2)	(25,25)	(30,30)	(4,7)
进入监护中心	低血压	无低血压	采取纤溶治疗	治疗有效促进 PCI	治疗无效执行 PCI	评价病人状态

从图 4 中可知，该医疗模型中含有 11 个活动节点，根据 2.1 小节提出的业务流程转换规则，转换后该模型的执行轨迹 T 为{(*A*1,*A*3,*A*6,*A*8,*A*11),(*A*1,*A*3,*A*6,*A*9,*A*11),(*A*1,*A*4,*A*6,*A*8,*A*11),(*A*1,*A*4,*A*6,*A*9,*A*11)}。执行轨迹 T 集合作为输入应用到 2.2 小节中的模型转换算法中，可以得表 2 中的 Prolog 事实。

表 2 图 4 对应的 Prolog 事实

活动节点	Prolog
A1	activity(1,3,2,4)
A1	activity(1,4,2,4)
A3	activity(3,6,2,2)
A4	activity(4,6,1,4)
A6	activity(6,8,2,2)
A8	activity(8,11,25,25)
A9	activity(9,11,30,30)
A11	activity(11,12,4,7)

将该事实与 2.3 小节中的转换规则合并作为 Prolog 程序，在 SWI-Prolog 环境中运行可得表 3 所列结果。

表 3 运行结果

时间约束	序列	结果
持续时间	[3,6,8]	possible&necessary
	[3,6,9]	notpossible&notnecessary
	[4,6,9]	notpossible&notnecessary
周期循环	[8,11]	possible&necessary
	[1,3,6]	possible&notnecessary
	[1,4,6]	possible&notnecessary
固定时刻	[1,3,6,8,11]	possible&notnecessary
	[1,3,6,9,11]	possible&necessary
	[1,4,6,8,11]	possible&notnecessary
	[1,4,6,9,11]	possible&necessary

由表 3 结果可得，持续时间模式中序列 1 满足可能且必须，序列 2 满足不可能且不必须，序列 3 满足不可能且不必须；周期循环模式中序列 1 满足可能且必须，序列 2 满足可能且不必须，序列 3 满足可能且不必须；固定时刻模式中序列 1 满足可能且不必须，序列 2 满足可能且必须，序列 3 满足可能且不必须，序列 4 满足可能且必须。

4.2 实例分析

在现有研究业务流程上时间约束的成果中，针对上述 3 种时间模式也都做了相应的分析。表 4 列出了现有成果针对业务流程模型上时间约束的研究情况，其中 Yes 表示方法支持该时间模式的验证，No 表示不支持。

表 4 与现有方法的对比

现有研究	持续时间	周期循环	固定时刻
Chen 等(2011)	Yes	No	Yes
Du 等(2013)	No	No	Yes
Combi 等(2012)	Yes	No	Yes
Huai 等(2010)	Yes	No	Yes
Eder 等(2008)	Yes	No	Yes
Bettini 等(2002)	Yes	No	No
This paper	Yes	Yes	Yes

从表 4 可以看出，由于使用的验证工具或者验证方法等的一系列限制，很少有方法能够将上述 3 种时间模式统一在同种方法中进行验证，并且在工具和方法上的可扩展性也有所欠缺。这里提到的可扩展性是指在模型不变的情况下，任意添加或者修改时间模式后方法所能表现出来的适应性。

Prolog 语言通过事实对模型和时间约束进行描述，利用规则刻画某一时间模式的一致性检测。随着业务流程的发展，限制在模型上的时间模式也会变化，通过 Prolog 语言则不需要改变事实部分，只需要针对变化后的时间模式对规则进行修改，在有新的时间模式情况下可以通过增加相应的规则来实现，考虑按照需求封装规则内部的实现以便接口化，在需要验证任一时间模式时直接调用该接口，从而实现了与具体实例分离，达到了通用性验证与可扩展性验证的效果。

另一方面，传统方法需要手动将模型转换为其需要的描述信息，在 2.1 小节中通过转换算法将模型自动转换为执行轨迹集合，之后利用 2.2 小节中模型到 Prolog 的转换算法实现了模型中时序约束一致性的自动检验。本文输出结果旨在说明特定执行轨迹是否满足在它之上的时间约束，可以通过调整输出函数的内容来详细列出该序列是否满足时间约束，若不满足，则列出原因，并且通过输出最少持续时间总和来重新制定时间约束。例如表 3 中序列[3,6,9]，结果显示该序列对于持续时间的约束既不满足可能性也不满足必然性，定位到图 4 中包含该序列的部分，根据需要重新调整活动节点中时间参数达到满足时间约束的条件，实现了时间约束与具体执行轨迹的绑定，

增强了实用效果。

结束语 本文提出了业务流程模型转换规则算法、执行轨迹到 Prolog 的自动转换算法以及 Prolog 对于时间约束的验证规则来验证分析带有时间约束的 BPMN。将模型转换为执行轨迹的集合后利用 Prolog 语言直接对其时间约束进行检验，实现了业务流程模型的自动化验证。

参 考 文 献

[1] Fan Yu-shun, Wu Cheng. Research on workflow modeling to improve system flexibility[J]. Journal of Software, 2002, 13(4): 833-839(in Cinese)
范玉顺，吴澄. 一种提高系统柔性的工作流建模方法研究[J]. 软件学报，2002，13(4)：833-839

[2] Van Der Aalst W, Van Hee K M. Workflow management: models, methods, and systems[M]. MIT press, 2004: 30-150

[3] Aalst V D, Wil M P. Business Process Management: A Comprehensive Survey[J]. Isrn Software Engineering, 2012, 2013(2): 125-143

[4] Börger E. Approaches to modeling business processes: a critical analysis of BPMN, workflow patterns and YAWL [J]. Software & Systems Modeling, 2012, 11(3): 305-318

[5] Wu N Q, Zhou M C. Modeling, analysis and control of dual-arm cluster tools with residency time constraint and activity time variation based on Petri nets[J]. IEEE Transactions on Automation Science and Engineering, 2012, 9(2): 446-454

[6] L Ye-bai, M Fu-qi. Research of the verification in workflow process modeling on the application of Petri nets[C]// International Conference on e-Education, e-Business, e-Management, and e-Learning, 2010(IC4E'10). IEEE, 2010: 21-24

[7] Szpyrka M, Nalepa G J, Ligęza A, et al. Proposal of formal verification of selected BPMN models with Alvis modeling language [M] // Intelligent Distributed Computing V. Springer Berlin Heidelberg, 2012: 249-255

[8] Lanz A, Weber B, Reichert M. Time patterns for process-aware information systems [J]. Requirements Engineering, 2014, 19 (2): 113-141

[9] Zhou N F. The language features and architecture of B-Prolog [J]. Theory and Practice of Logic Programming, 2012, 12(1/2): 189-218

[10] Combi C, Gozzi M, Posenato R, et al. Conceptual modeling of flexible temporal workflows[J]. ACM Transactions on Autonomous and Adaptive Systems (TAAS), 2012, 7(2): 451-457

(上接第 159 页)

降低数据维度的思想，通过对非参数估计的训练样本集进行主成分分析来改进样本的输入因子数；再利用主成分分析中的方差贡献率作为由经验算法求得的带宽矩阵的权重，改进了传统的非参数估计法，建立了一种新的软件失效预测模型。模型从复杂系统建模的角度研究了变量间的关系，其优点在于对于单输出(结果)多维系统，不需诸多限制条件，可以检验变量之间在非线性意义上的因果关系。实验结果表明：该方法在预测时，一定程度上消除了各输入因子对结果的作用程度的不同所造成的影响，在预测的精度和稳定性上得到了进一步提高。

本模型由于使用改进后的非参数方法进行训练，因此数据的训练集应尽可能大，以使模型得到充分训练，这样才能达到更好的预测效果。另外，本文算法是进行单步预测，每预测一步后，再把其加入训练集来预测下一个点，随着预测点数的增加，导致在后面点数的预测中误差会越来越大，所以此模型更适合于进行短期预测。

参 考 文 献

[1] Wang Q, Wu S J, Li M S. Software defect prediction[J]. Journal of Software, 2008, 19(7): 1565-1580(in Chinese)
王青，伍书剑，李明树. 软件缺陷预测技术[J]. 软件学报，2008，19(7)：1565-1580

[2] Nagappan N, Ball T, Zeller A. Mining metrics to predict component failures[C]// 28th International Conference on Software Engineering (ICSE). 2006: 452-461

[3] Liu Ya-nan, Wei Zhi-nong, Zhong Lin-juan, et al. Study on the forecasting model of power supply reliability based on PCA and RVM[J]. Power System Protection and Control, 2012, 40(20): 101-105(in Chinese)
刘亚南，卫志农，钟淋涓，等. 基于 PCA 和 RVM 的电网供电可靠性预测模型研究[J]. 电力系统保护与控制，2012，40(20)：101-105

[4] Catal C. Software fault prediction: A literature review and current trends[J]. Expert Systems with Applications, 2011, 38(4): 4626-4636

[5] Sandamali Dharmasena L, Zeephongsekul P. Fitting software reliability growth curves using nonparametric regression methods [J]. Statistical Methodology, 2010, 7(2): 109-120

[6] Couto C, Montandon J E, Silva C, et al. Static correspondence and correlation between field defects and warnings reported by a bug finding tool[J]. Software Quality Journal, 2013, 21(2): 241-257

[7] Catal C. Software fault prediction: A literature review and current trends[J]. Expert Systems with Applications, 2011, 38(4): 4626-4636

[8] D'Ambros M, Lanza M, Robbes R. An extensive comparison of bug prediction approaches[C]// 7th IEEE Working Conference on Mining Software Repositories (MSR). 2010: 31-41

[9] Hwang J N, Lay S R, Lippman A. Nonparametric multivariate density estimation: a comparative study[J]. IEEE Transactions on Signal Processing, 1994, 42(10): 2795-2810

[10] Couto C, Montandon J E, Silva C. Static correspondence and correlation between field defects and warnings reported by a bug finding tool[J]. Software Quality Journal, 2013, 21(2): 241-257

[11] Nagappn N, Ball T. Using software dependencies and churn metrics to predict field failures: an empirical case study[C]// First International Symposisum on Empirical Software Engineering and Measurement. 2007: 364-373

[12] Lee H J, Naish L, Ramamohanarao K. Study of the relationship of bug consistency with respect to performance of spectra metrics [C]// 2nd IEEE International Conference on Computer Science and Information Technology. 2009: 501-508

[13] Okutan A, Yildiz O T. Software defect prediction using Bayesian networks[J]. Empir Software Eng, 2014, 19(1): 154-181

[14] Couto C, Piresa P. Predicting software defects with causality tests[J]. Journal of Systems and Software, 2014, 93: 154-181

[15] 叶阿忠. 非参数和半参数计量经济模型理论[M]. 北京：科学出版社，2008：30-150