维普资讯 http://www.cqvip.com

一种过程定义模型及其验证性分析*)

费立蜀　顾　庆　陈道蓄
(南京大学软件新技术国家重点实验室　南京210093)

摘　要　软件过程是管理、开发、维护软件系统所需要的一系列活动的偏序集合。软件过程作为一种工作流，其建模和分析可以采用工作流的理论和技术作为支撑。因为任务控制流的图示表示办法直观方便，因此大多系统都采用这种定义方式为软件过程建模。而过程定义的合理性验证问题复杂度通常是非常高的，必须寻求合理的算法。图规约算法就是一种实际可行的算法。

关键词　软件过程，过程定义，建模，工作流，合理性，验证，图规约

A Process Definition Model and its Verification Analysis

FEI Li-Shu　GU Qing　CHEN Dao-Xu
(State Key Laboratory for Novel Software Technology, Nanjing University, Nanjing 210093)

Abstract　Software process is a partially ordered set of activities undertaken to manage, develop and maintain software systems. As a special workflow, the modeling and analysis of software process can be based on the theory and technique of workflow. Most systems use the task flow graph model to model software process because this kind of model is very intuitive. It's very important to find a sound algorithm to verify the workflow structure correctness because the soundness problem is normally very complex. Graph reduction algorithm is a practicable one.

Keywords　Software process, Definition, Modeling, Workflow, Soundness, Verification, Graph reduction

1　引言

软件过程是开发有质量保证的软件系统的关键因素，因为其目标是管理用户需求并把用户需求转换为符合满足需求的软件产品。因此，软件过程意味着生产一个软件系统的活动集合，这些活动由有组织的人员依赖一系列工具执行。

软件过程的出现很早，但是开始并没有明确定义。随着软件构造的过程的认识不断深化，软件过程的概念得到明确，可以定义为[2]：管理，开发，维护软件系统的所需要的一系列活动的偏序集合。因此，软件过程注重的是过程的构造而不是产品的输出。这个定义通常包含了相关执行人员的指定，人员组织成角色的方式，以及产生的中间和最终人工实体(artifacts)。而过程模型是软件过程的抽象表示。

工作流(Workflow)是从BPR(Business Process Reengineering)产生的概念。WfMc(Workflow Management coalition)提出了一个工作流参考模型[3]，描述了工作流系统的要素，包括了5个接口的标准。其中过程定义接口[4]是过程定义工具和工作流引擎之间的接口标准。

从概念上讲，软件过程是工作流在软件领域应用的一个特例，具备工作流的特征。因此，我们借用工作流的理论和技术来研究软件过程，如建模和分析。在本文的研究范围内，不区分软件过程和工作流。

研究软件过程或者工作流，通常有两种角度[1]：

·描述性的(Descriptive)：研究现存的过程中软件是如何被开发的。

·规定性的(Prescriptive)：定义需要的过程，回答"软件应该怎么开发"的问题。

不管从哪个角度，都可能会有下面几个方面的内容：

·表示，形式的或者非形式的表示过程模型。

·分析，通过形式的或者非形式的方法进行确认(validation)和验证(verification)。

另外，对于规定性的角度，还会有执行方面的内容，譬如调度执行策略。

主要的工作流模型定义方法有很多，实用化的原模型有任务流，状态转换，关系捕捉，基于Petri-net的，基于Logic的，基于通讯的[5]等。但是最直观方便的是基于任务流(控制流)的流程图示方法。

对于表示的问题，本文基于控制流模型，提出了一种过程定义模型。以此过程定义模型来支持软件过程的定义。文章第2部分叙述这个模型的图形化的非形式的定义和形式化的定义。对于分析的问题，本文基于定义的过程定义模型，进行验证性的工作。第3部分给出了工作流合理性的定义，并对其判断问题的复杂度作出了分析。第4部分，在这个基础上，寻求实际可行的验证方法，对图规约验证算法进行了研究。第5部分对相关的各种模型定义方法进行了一些比较。最后对本文进行了总结。

2　软件过程模型描述

描述一个软件过程，需要定义三个基本的维度：

1)控制流：定义任务和任务之间的转换和同步关系。应该包括的同步控制关系：顺序、选择分支、并发分支、循环。控制流的抽象表示视图称为控制流模型。

*)基金项目：国家863高技术项目"基于CMM的软件质量保障平台及应用"(编号：2001AA113090)。费立蜀　硕士研究生，研究方向：工作流系统，软件过程管理。顾　庆　博士，副教授，研究方向：分布式计算，工作流系统。陈道蓄　博士生导师，研究方向：分布式计算与并行处理。

维普资讯 http://www.cqvip.com

2)数据流:在工作流任务之间转换的过程当中,相关数据的流动,交换工作输入和输出。数据流的抽象表示视图称为数据流模型。

3)组织结构:角色是若干具有相同职责的组织内人员的集合。组织模型决定如何定义角色,以及角色和人员如何绑定,最终是把负责合适职责的人员分配到合适的任务上去。组织结构的抽象表示视图称为组织模型。

实际上,控制流离不开数据流,譬如选择分支条件和循环条件都需要对相关数据进行判断,因此条件转移的表达式的真值一般与环境数据有关。另外控制流的走向也取决与用户的交互。由用户交互和数据流所决定的工作流的一次执行,叫做一个工作流实例,简称实例。显然一个实例执行的任务是全部任务的一个子集。

因此要检验过程的合理性,有两种情形。

定义1(结构合理性) 又称静态合理性,指过程定义的静态结构没有结构冲突。

定义2(系统合理性) 又称动态合理性,指过程执行时具备完备的实例集合,且所有的实例都是结构上合理的。

显然判断结构合理性,只需对过程定义的控制流进行判断即可,不考虑数据和组织结构。而判断系统合理性,必须从过程动态执行的历史和可能执行的路径结合数据进行判断。显然,系统合理性包含结构合理性,但是系统合理性的判断要涉及到动态的执行情况,因此难度要大得多;另一方面,结构合理性已经包含了主要的系统合理性。本文对合理性的判断,主要立足于结构合理性。

把每个任务与数据流和人员指派相结合,就可以执行此过程了。我们把某个人员集合需要在一定的数据资源上完成的任务,叫做一个活动。

结合了控制流模型,数据模型和组织模型三个子模型的整体称为软件过程模型。控制流模型是过程模型的关键部分,本文主要考虑控制流模型,下面所指过程模型也指控制流模型,这时,我们把活动和任务当作一个词,不区分它们之间的区别。

2.1 过程模型图形表示

软件过程模型定义所需各元素如图1所示。

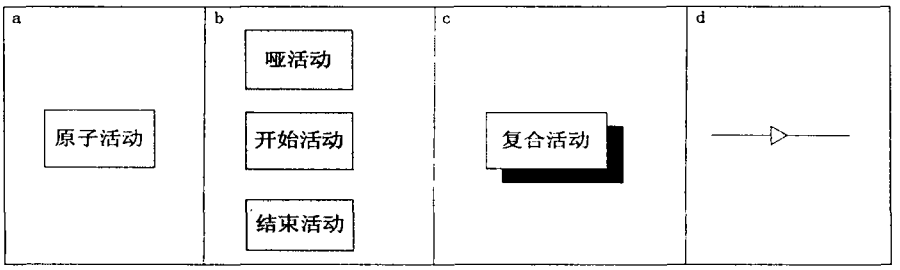

图1

活动:用矩形框表示活动。有三类活动:原子活动,哑活动,复合活动。原子活动表示负载一定任务的活动,表示一个具体的任务,如1(a)所示。哑活动不是表示具体的任务,目的是为了工作流的同步控制关系的需要,加入的辅助节点,如1(b)所示。其中开始活动和结束活动是两个特殊的哑活动,分别表示一个过程的开始和结束。复合活动表示一个子过程,从而形成过程的分层结构。并以两个重叠的矩形框表示,如1(c)所示。

转换:用带箭头的直线段表示,箭头表示转换的方向。用以连接两个相连续的活动。

控制关系:没有单独的显式的控制图元表示,控制关系隐含在活动的同步属性中,如图2所示。

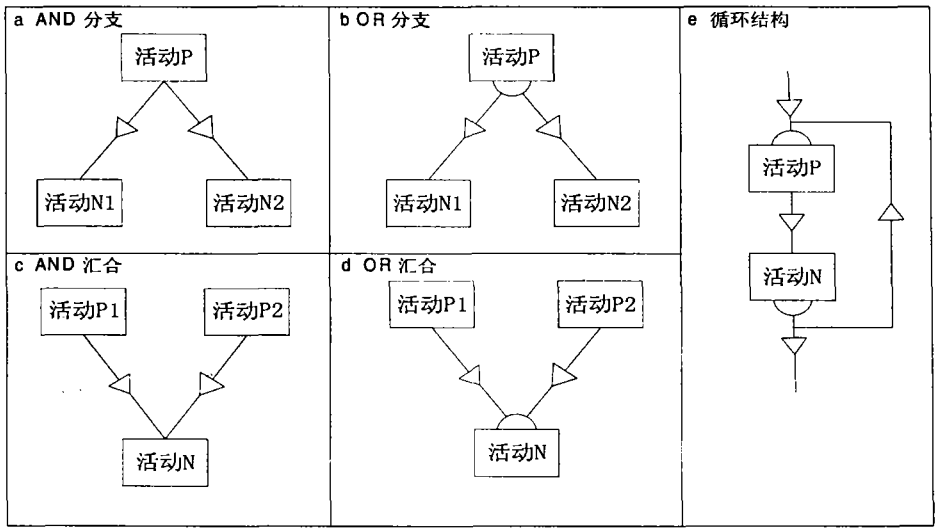

图2

图2(a)表示 AND 并发分支。语义上表示,活动 P 结束后,触发活动 N1和 N2的执行,并且 N1和 N2都要执行。但是在执行时间上,并不一定是同时执行。

图2(b)表示 OR 选择分支。语义上表示,活动 P 结束后,触发活动 N1和 N2中的一个执行。至于选择哪一个,需要看转换条件决定。

维普资讯 http://www.cqvip.com

图2(c)表示 AND 汇合。语义上表示,活动 N 执行的前提,必须是活动 P1和 P2都要执行完毕以后,才能执行 N。因此表示并发同步。

图2(d)表示 OR 汇合。语义上表示,活动 N 执行的前提,只要活动 N1和 N2中的一个执行完毕就可以执行 N 了。因此表示选择同步。

需要说明的是,表示 AND 分支和 AND 汇合的关系,只需要默认的活动图元表示就可以了。但是要表示 OR 分支和 OR 汇合的关系,我们采取了在默认图元的基础上,增加一个半圆表示 OR 的关系,如图2(b)和2(d)中所示。并不采用单独的表示 AND 和 OR 关系的图元,这是一种隐式的语义表示方法。这样做的好处是可以凸显活动之间的执行顺序和关系,把同步关系看作是活动的附属属性;因此活动、转换和同步关系不在同一个层次上。同时,这样做可以减少图示图元的数量。可以简化工作流的表示。

循环结构可以用基本的 OR 选择分支结构表示,即一个 OR 选择分支和一个 OR 汇合结构的组合,如图2(e)所示。

定义3(循环) 存在活动 A 到活动 B 的路径和活动 B 到 A 的路径(路径只考虑有向图,不考虑这条路径上活动的入口和出口同步属性),且 A 的入口同步属性和 B 的出口同步属性都是"Or",那么从 A 到 B 之间构成一个循环。显然,从 A 到 B 的路径可以有多条,叫做一个路径集;从 B 到 A 的路径也可以有多条。

定义4(正规循环) 如果路径集的所有活动除了端点 A 入口转换和 B 的出口转换之外不存在与本路径集外的活动之间的转换,那么这个循环叫做正规循环。或者叫封闭循环,反之叫做开放循环。

串行结构很简单,就是前后相续的若干活动的连接,中间没有分支。

对于转换,都有一个条件表达式与之相联系。对于 AND 并发关系这个表达式无意义。在 OR 选择关系中,每个转换的条件是这个相联系的表达式的计算结果。上面我们已经论述了条件表达式属于动态结构的部分。因此,不在图中表示出来,而隐含在转换的属性中。

2.2 过程模型语法结构

定义5(活动) 表示一个任务。活动集合用 A 表示。

定义6(原子活动) 表示一个具体的可以执行的任务,被分配了角色成员并且指定了输入输出数据。原子活动集合用 AA 表示。且 $AA \subseteq A$。

定义7(哑活动) 表示一个占位活动,并不承担实际的任务,用于协助完成活动之间同步关系的建模。哑活动集合用 DA 表示。且 $DA \subseteq A$。

定义8(复合活动) 表示一个嵌套的子过程。复合活动集合用 CA 表示。且 $CA \subseteq A$。

定义9(过程) 形式的定义为 $P=\langle A,T,C,s,e\rangle$

其中:

1. A 为活动集合,包括原子活动 AA,哑活动 DA,复合活动 CA,即;且 $A=AA \cup DA \cup CA$;且 $|A|=|AA|+|DA|+|DA|$,即活动子集之间交集为空。

2. $T \subseteq A \times A$ 为转换集合。若 $t=(a,b) \in T$,则 a 称为 b 的前驱活动,简称前驱;b 成为 a 的后继活动,简称后继。任何一个活动都可能有若干个前驱和后继。

3. $C:T' \rightarrow$ Prd 为 T 到条件谓词上的映射。

4. s 为起始活动,为一哑活动,即 $s \in DA$,s 入度为0,即 s[ind]=0。

5. e 为终止活动,为一哑活动,即 $e \in DA$,e 出度为0,即 e[outd]=0。

定义10(属性) 若有活动 $a \in A$,a 的属性 attr 表示为 A[attr]。

特别的。

定义11(入度) 活动 a 的入度为 a 的前驱的个数,表示为 a[ind],$a[ind]=\{a \mid \exists\ b \in A,(b,a) \in T\}$ 的元素个数。

定义12(出度) 活动 a 的出度为 a 的后继的个数,表示为 a[outd],$a[outd]=\{a \mid \exists\ b \in A,(a,b) \in T\}$ 的元素个数。

定义13(入口约束) 活动 a 的入口约束定义为进入 a 的入口同步关系,表示为 $a[inres] \in \{And,Or\}$。

定义14(出口约束) 活动 a 的出口约束定义为进入 a 的出口同步关系,表示为 $a[outres] \in \{And,Or\}$。

工作流过程的定义没有定义活动和活动之间的同步关系,这种同步关系体现在入口约束和出口约束之中。这是一种隐式的同步定义方式。工作流的另外一种定义方式,是把同步关系作为一种单独的节点显式定义出来[6,8,9]。

3 验证性问题及其复杂性

与程序设计语言编写的程序类似,因为存在着入口,出口,选择分支,并发分支,循环等同步关系,所以定义的过程同样可能会出现结构上的缺陷,譬如死循环,不可到达的节点等等。检验结构合理性是非常重要的。

首先必须进一步定义什么样的过程定义是结构上正确的,或者说是合理的。

定义15(合理性) 过程定义是合理的,当且仅当满足以下条件:

1)满足定义9中定义的过程的定义,其中的约束性条件是必须存在且只存在一个开始活动和一个结束活动。

2)任何一个活动都是可达的,一个活动可达的含义是指从开始活动开始执行,一定存在一个活动执行序列,最后能够执行到这个活动。

3)任何一个活动都是可结束的,一个活动可结束的含义是指从这个活动开始执行,存在一个活动执行序列,最后一定能够执行到结束活动。

4)任何一个活动的入口约束是"Or",那么其前驱活动中同时只能有一个被触发执行。

其中,第一个条件是按照过程的定义必须满足的条件,存在一个可以识别的开始和结束活动,这个条件的判断非常容易,只要判断各个活动的入度和出度就可以了。

第二个条件是为了避免存在死路径的情况,如果存在不能够由开始活动可以到达的活动,那么这个活动就是死的,永远不会执行,因此也就没有存在的必要。选择分支的一个分支的条件谓词是重言式的情况就是一种存在死路径的情况。另外还可能因为由于死锁导致不能执行后续的活动,图3(C)中就是一个死锁的情况,因为活动 N1的条件不可能得到满足,因此活动 N2是不能执行到的。

第三个条件是为了避免存在死锁的情况,如果从某个活动开始执行,永远都不能到达结束活动,那么必定存在一个死锁。图3(A)所示是顺序的同步关系搭配错误导致的死锁,活动 N1和 N2只能执行其中的一个,所以活动 Q 的触发条件不可能得到满足。图3(D)是一种死循环的情况,活动 N2执行完毕以后,因为其出口约束是"And",因此 N1和 N2会不断重复

维普资讯 http://www.cqvip.com

执行，构成死循环。

第四个条件是为了避免失同步错误，既然入口约束是“Or”，语义上是表示若没有循环结构，入口只有一个选择分支到达这个活动，如果有多个活动能够同时到达，就出现了选择同步错误。图3(B)是另外一种同步关系搭配错误，显然活动Q会得到两次执行。

可开始条件和可结束条件的判断都不是可以在多项式时间内可以完成的。A. H. M. ter Hofstede 等[7,9]给出的关于可开始条件和可结束条件的两个定理，证明了这一点。参照其证明方法，针对本文给出的过程定义模型，可以证明可开始条件的判断问题是NP-complete问题。

定理1 可开始条件的判断问题是NP-complete问题。

证明：只需证明可满足性问题，即SAT问题，可以通过一个多项式变换为一个可开始条件的判断问题，因为SAT问题是一个已知的NP-complete问题。

设命题 $P=\wedge_{i=0}^{n}C_i$，$C_i=\vee_{j=1}^{m}x_{ij}$，C_i 是子句。与P相对应，按自底向上的层次顺序构造过程结构如下：目标活动为P，每个子句 C_i 对应一个活动，作为P的前驱，且P的入口约束为“Or”；对于每个 C_i，构造其前驱活动为 $\langle\neg x_{ij},C_i\rangle$，其中¬为逻辑否定的符号，且活动 C_i 的入口约束为“And”；对于所有的活动 $\langle\neg x_{ij},C_i\rangle$，构造活动 $\neg x_{ij}$ 为其前驱，入口约束和出口约束都是“And”，如果不同的活动标记 $\langle\neg x_{ij},C_i\rangle$ 前件是同一个元素 $x=\neg x_{ij}$，那么这几个不同的活动是同一个活动 $\neg x_{ij}$ 的后继；最后，每对活动 $\neg x_{ij}$ 和 x_{ij} 都是一个哑活动 Dx_{ij} 的后继，哑活动 Dx_{ij} 的出口约束为“Or”，表示两者择其一执行。

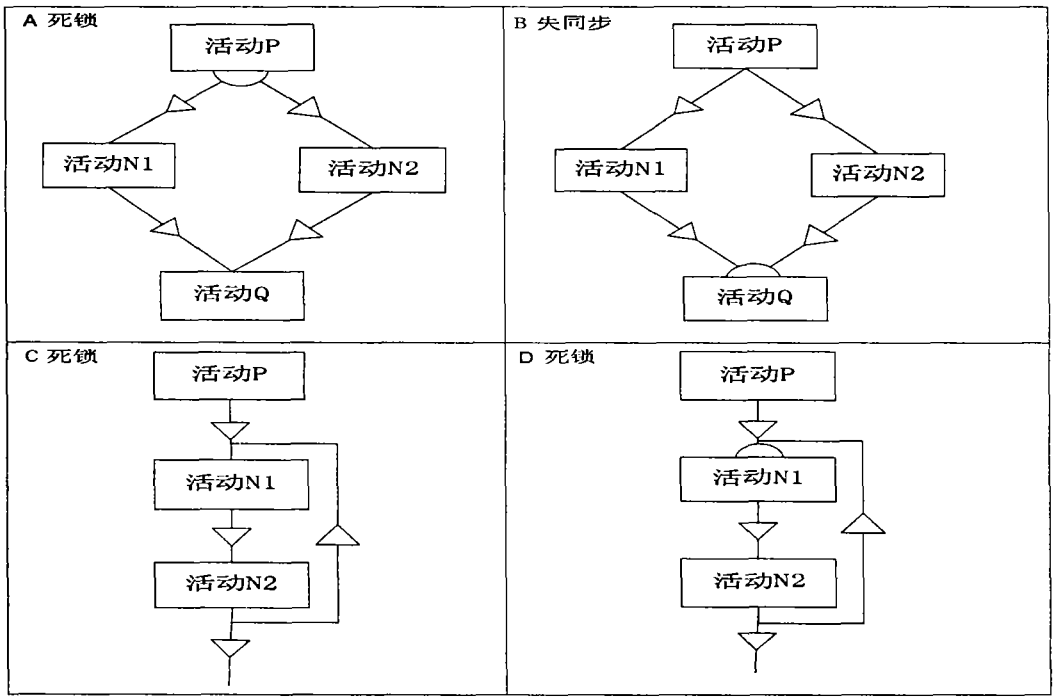

图3

这种构造可以保证命题P不可满足当且仅当活动P是可以开始的。若 $P=\wedge_{i=0}^{n}C_i$ 不可满足，则不论任何真值指派T，必有一个子句 $C_i=\vee_{j=1}^{m}x_{ij}$ 为假，则在指派T下，其所有的元素 x_{ij} 都必定为假。相应地，活动 C_i 开始依赖于其前驱 $\langle\neg x_{i1},C_i\rangle,\cdots,\langle\neg x_{im},C_i\rangle$ 能够开始，活动 $\langle\neg x_{ij},C_i\rangle$ 的开始又取决于活动 $\neg x_{ij}$ 的开始。因此哑活动 Dx_{ij} 选择 $\neg x_{ij}$ 执行，最终就能使活动P开始。与此类似，可证如果P可满足，那么任何活动 C_i 都不能开始。

显然这个构造过程是多项式转换。最后，如果给出一个过程定义的实例，那么某一个活动是否能够开始，这个验证过程是可以在多项式时间内完成的。因此，可开始判断问题是一个NP-complete问题。

看一个例子，$P=A\wedge B\wedge C=(x\vee\neg y\vee z)\wedge(x\vee\neg z)(y\vee\neg z)$，按上面的构造方法得到的过程定义为图4所示，图中“¬”用“－”表示。

同样可以得出结束判断问题相似的结论。可结束问题是DSPACE(exp)-hard问题。更进一步，检测死锁的问题也是DSPACE(exp)-hard问题。

除了基于控制流图的分析以外，还有比较适合理论分析的基于Petri-net的工作流模型，W. M. P van der Aalst[10]提出的基于Petri-net的任意一个工作流WF-Net，其合理性的验证都是EXPSPACE-hard的。Hasan Davulcu 等[13]提出了一种基于CTR(concurrent Transaction Logic)的工作流表示模型，用CTR公式concurrent-Horn goal表示工作流图，用一种简单的代数约束表示事件之间和时序上的正确性的约束关系。基于这种模型，同样也得出验证工作流模型的正确性是NP-complete问题。

对任意的工作流的验证，各种模型都得出相似的结论。如果对工作流进行语法上的约束，可能会降低验证的复杂度。Aalst提出了三种对任意WF-net进行约束的结构：Free-choice，well-structured，以及S-Coverable，前两种作为S-Coverable的特例，是有其特定的针对性。虽然有些不满足S-Coverable的结构也可能是合理的，但是作为实际上比较可行的约束，这个约束是合适的。这几种结构的WF-net的合理性的验证都是可以在多项式事件内完成的。上面基于CTR的方法也指出，如果只把约束关系限定为顺序约束的话，验证性问题也是可以在多项式时间之内解决的。

维普资讯 http://www.cqvip.com

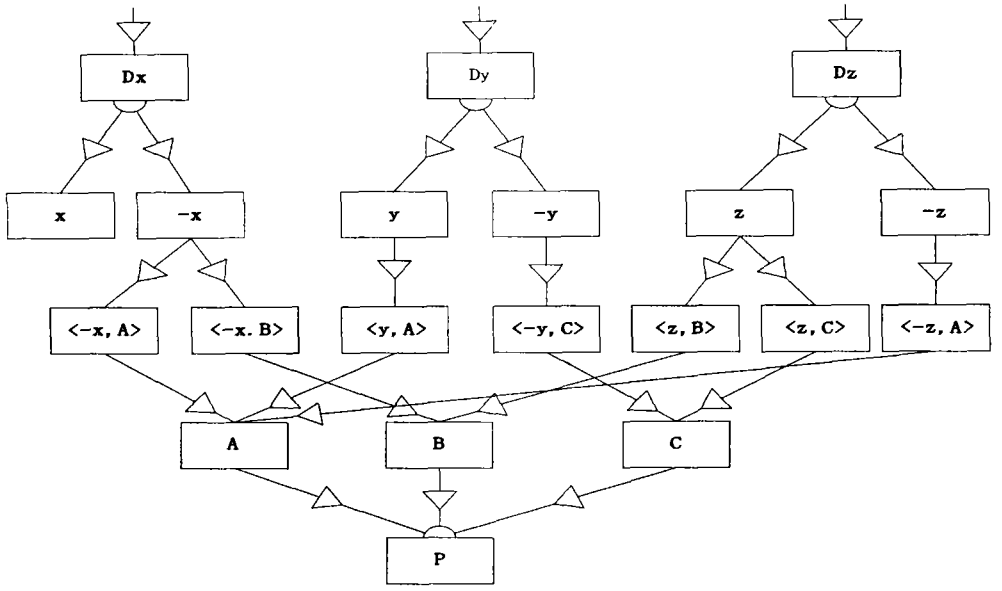

图4

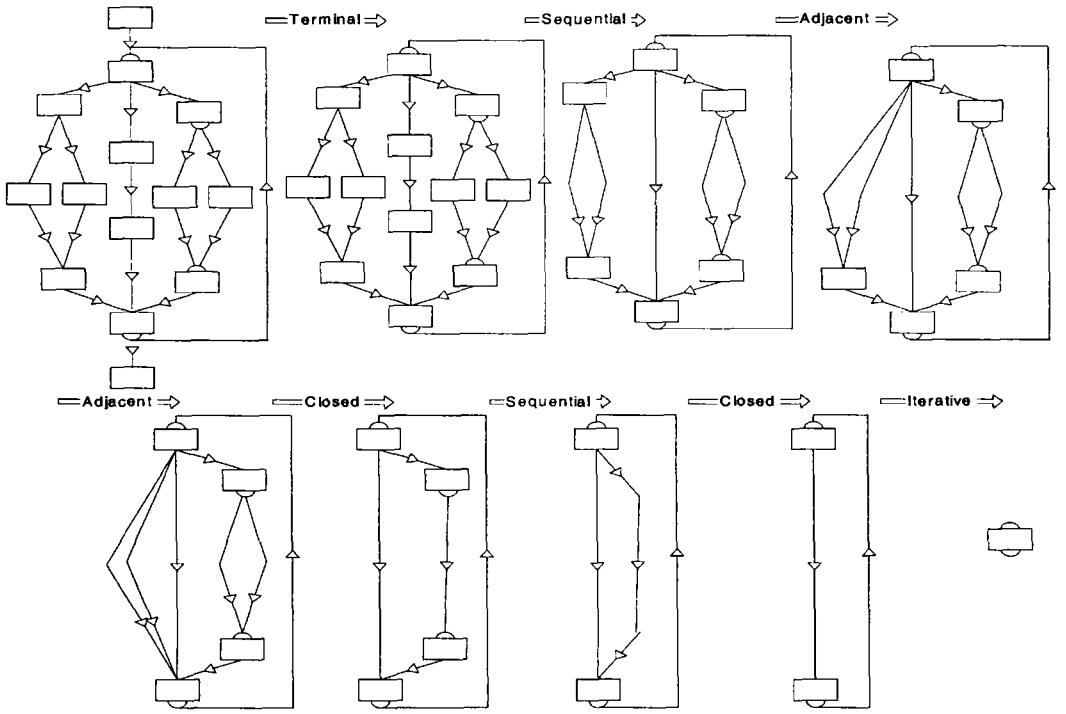

图5

4 验证策略

直接判断过程定义合理性的方法实际上是不可行的。因此，或者对过程定义模型进行语法上的限制，降低验证的难度；或者寻求其他的可行性的算法。

对语法结构进行限制，虽然可以降低验证的难度，但是会限制其表达能力；显然，限制语法结构和增加表达能力之间是一个平衡(trade-off)。

另外一种是从另外的角度，寻求新的验证算法。Sadiq[6]提出了一种基于DAG规约的办法，提出了五种规约规则，如果能把工作流图按照这五种规约最终规约到空图，那么就说明是合理的。实际上，Sadiq提出的这五种规约是不完全的，存在一些正确的工作流图并不能规约为空。文[8]在这个基础上，对这五个规则进行修改和扩充，提出了一个最小完备规约集，并且其算法的复杂度为 $o((size[G])^2)\cdot((size[N])^2)$，其中 size[G]为DAG图的大小，size[N]为DAG中所有节点的大小，size[E]为DAG中所有边的大小，size[G]=size[N]+size[E]。并且证明了，应用这些规则不断对工作流图进行规约，如果能最终规约为空图，说明是正确的，否则必存在结构上的冲突。

这个规约集仍然只是对DAG图进行规约，不包括循环结构，我们对这个规约集扩充加入循环结构的规约，并按照上面定义的过程模型进行相应的结构修改，用于本文所述过程定义的正确性验证。但是任意的循环结构的规约非常复杂，我们采取只允许使用正规循环的结构的语法限制办法，降低复杂度，只是需要牺牲一定程度的表达的灵活性。因为正规循环

维普资讯 http://www.cqvip.com

结构实际上是由两个独立的子图构成的，如果活动A到活动B构成一个正规循环，那么从A到B的路径集和从B到A的路径集可以分别单独规约，如果这两个子图是结构合理的，那么最后应用下面的规则5就能把正规循环结构规约为空。图5中的循环结构是一个正规循环。

规则1(Terminal 规则) 如果一个活动的入度和出度的总和为1，删除这个活动。

规则2(Sequential 规则) 如果一个活动的入度和出度都为1，删除这个活动。

规则3(Adjacent 规则) 两种活动，一种是与其前驱的类型一致，且其前驱有一个前驱多个后继；另一种是与其后继类型一致，且其后继有一个后继多个前驱。这个活动连同相联系的转换删去，并连接其前驱和后继。

规则4(Closed 规则) 两个活动之间存在多条转换(边)，删除其余，只留一条。

规则5(Iterative 规则) 一个或者两个活动之间构成正规循环，把这个循环缩为一个入口约束和出口约束都为"Or"的活动，取消构成环的两条转换(边)。

图5所示，为一个应用这些规则进行规约的例子。

规则6(Choice-Convergence 规则) 一个活动的出口约束为"Or"，表示选择分支，其所有的后继活动的出口约束为"And"(即都是并发分支活动)且都有一个转换到达同一个入口约束为"Or"的后继活动。见图6(a)到(b)的转换，增加一个出口约束为"And"的活动Fx，直接连接活动M1，且与M1连接的F1,F2,…,Fn与M1断开连接，并且Fx与原入口相连，Fx的入口约束与C1的入口约束保持一致。

规则7(Synchronizer-Convergence 规则) 一个活动的出口约束为"And"(并发分支)，其所有的后继活动的入口约束为"Or"(选择同步活动)，且这些活动都只有一个共同的入口约束为"And"的后继(并发汇聚)。见图6(c)到(d)的转换。与规则6类似，需要增加一个入口约束为"Or"的活动Mx，入口活动F1，出口活动S1直接连接到Mx，取消从F1到中间层活动M1,M2,…,Mn之间的转换。Mx的出口约束与S1的出口约束保持一致。

规则8(Merge-Fork 规则) 见图6(e)到(f)的转换。条件是入口活动M的入口约束为"Or"(选择汇聚)，出口活动F的出口约束为"And"(并发分支)。

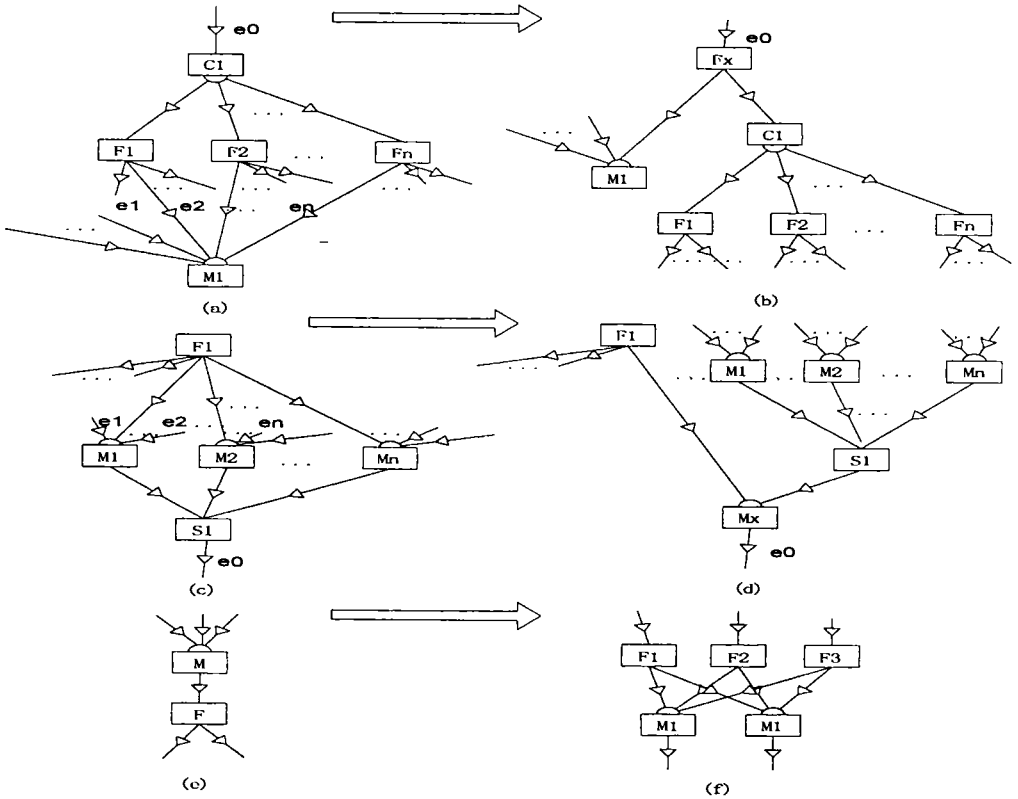

图6

对循环结构进行语法限制之后，需要额外的结构判断。判断是否有循环存在，有成熟的深度优先和广度优先算法，并可以方便地求其路径集。进而容易判断是否正规循环。

加入循环规约对这个规约集的影响，我们有下面的定理。

定理2 只有规则8会破坏正规循环结构。

证明：即要证明其他规约规则不会把正规循环的结构转换为开放循环。

规则1～5显然不会破坏正规循环结构。

规则6,7(a)中的结构是正规循环的一个路径集的一部分，如果不包含循环的端点，即C1和M1都不是循环的端点，那么经过规则6的变换，所有的活动仍然在原来的路径集中。如果C1是循环的入口，经过变换后，Fx成为循环的入口，仍然保持正规循环的结构。如果M1是循环的出口，那么如果F1,F2,…,Fn除了到M1的转换外，还有到其他活动的转换，那么这些活动必然是路径集的一部分，即这些活动必然都汇聚到M1上，因此变换后，仍然是正规循环。规则7的变换与此类似，也仍然保持正规循环的结构。

规则8,7(e)中，如果M是循环的入口，那么变换后，显然会分裂成两个循环，入口分别是M1和M2，M1和M2分属两个循环，因此破坏了正规循环的结构。证毕。

因此，要应用上面的规约规则，不能在正规循环结构中使用规则8。

算法

输入：工作流图。

维普资讯 http://www.cqvip.com

输出:规约后的工作流图。

步骤1:不断应用规则1～7,直到不能继续规约。

步骤2:如果规约为空图,原始图是合理的,停止规约。

步骤3:查找是否存在未处理过的正规循环。

步骤4:如果不存在,转步骤5;如果存在,对其左右两个路径集作为两个子图调用此算法进行规约。然后应用规则5,对规约路径集后的循环进行规约。转步骤3。

步骤5:不断应用规则1～4,6～8,直到工作流图不能继续规约。

步骤6:如果规约为空图,原始图是合理的,停止规约。完毕。

步骤3,4中可能包含递归调用,使图到达步骤5时,已经不存在正规循环结构了,因此,这时应用规则8没有问题。

这个扩充的规则集的完备性,有下面的定理。

定理3,规约规则1～8是最小完备规约集。

证明:如果存在正规循环结构,显然增加的规则5是必需的。

分两个方面来说明增加规则5后的完备性:

1)如果工作流图是合理的,那么一定可以规约为空图。

若不存在循环结构,应用除规则5之外的规则,一定可以规约为空图;若存在正规循环结构,因为可以递归的对正规循环的路径集作为独立的子图进行分别规约,最后应用规则5,也可以规约为空。

2)如果工作流图不合理,那么一定不能规约为空图。

若不存在循环结构,显然不能规约为空图;若存在开放循环结构,无法应用规则5,循环结构不可能消除,因此不能规约为空图;若循环结构都是正规循环,那么如果循环内的结构不合理,必然是其路径集不合理,因为算法对循环规约是递归的,因此必有一个不含循环的路径集应用除规则5以外的规则不能规约为空,最终不能应用规则5消除循环。

因此,这个规约集是最小完备集。证毕。

5 相关比较

有很多定义过程模型的元模型,其中比较常见的是控制流程图的模型[5,6,8],其优点是便于定义全局的控制流,并且非常直观。缺点是缺乏全局性的约束关系,譬如任意两个任务之间的执行次序,只有相邻任务之间的局部约束。大部分控制流程图的模型都是把任务和同步关系的分别描述,我们认为,这两者不是同一个层次上,因此采取隐式的表示方式,既能保证同样的表达能力,又使表示简洁清楚。有些控制流图没有循环结构,利用复合活动对子过程嵌套实现循环结构,这样虽然能保证同一个层次上的过程模型是DAG,但是把为了实现循环,把活动内部的执行复杂化了。

基于状态转换的模型[5],是针对每个角色的状态转换的描述,从每个角色的角度来看,非常清楚,但是不易获得整体描述。

关系捕捉的模型[5],是捕捉角色之间的交互关系,能够显示角色之间的任务同步关系,并且能够"翻译"成控制流程图,也能很容易"翻译"成 Pertri-net 模型。但是仍然缺乏整体描述。

基于 Perti-net 的模型[10～12],研究得比较多,优点是理论性强,便于理论分析,表达能力强,基于 Petri-net 的模型,已经得到了许多有用的理论结果。但是其缺点是不直观,实现的系统复杂,不宜理解。

基于 CTR 的模型[13],利用逻辑代数,能很方便地表达全局约束,也非常利于理论分析,且与控制流程图和 Petri-net 能够相互转换。但其缺点是局部约束表达能力弱,并且没有直观的图表示,非常难以理解。

对于过程的合理性验证,基于这些不同的模型,对于任意模型的验证,都能证明其复杂度是至少 NP-Complete 的[6,7,9,10]。针对语法约束的方法,基于 Petri-net 的模型提出了 S-Coverable 结构限制,基于 CTR 的模型对全局约束做出了限制。基于控制流程图的模型[8]和基于 Petri-net 的模型[11]都可以采用图规约的算法判断其合理性,不必限制语法结构,但是只能对 DAG 图进行处理,并包含循环结构的处理。本文提出的算法,在对结构进行适当的限制的基础上对图进行规约,既保证了基本的表达灵活性,又降低了验证的复杂度。

总结 软件过程作为一种工作流在软件领域的应用特例。其表示和分析等问题可以以工作流的理论和技术为基础。

过程定义,或工作流定义,存在很多模型定义方式,其中最直观的是控制流图示方式。可以基于图的理论,对其进行非形式和形式的定义。

过程定义的正确性或者是合理性的验证是很复杂的,直接按照合理性的条件进行验证都需要非多项式的时间和空间复杂度,因此实际操作中是不可行的。解决的两条途径一是牺牲表达能力对其进行语法上的限定,二是寻找其他可行的验证途径。应用图规约方法可以在多项式时间内完成对工作流图的验证,因此寻求最小完备的规约集是一个好的选择。针对有循环结构的过程定义,我们把这两种办法结合起来,既保证基本灵活的表达能力,又能降低检验合理性的复杂度。

参考文献

1 Lonchamp J. A Structured Conceptual and Terminological Framework for Software Process Engineering. In: Proc. of the 2nd Intl. Conf. on the Software Process-Continuous Software Process Improvement, 1993

2 Acua S A, et al. Software Process Modelling. url = "citeseer. nj. nec. com/479098. html"

3 Workflow Management Coalition. The Workflow Reference Model. Document Number TC00-1003 Document Status - Issue 1.1 19-Jan-95

4 Workflow Management Coalition. Interface 1: Process Definition Interchange Process Model. Document Number WfMC TC-1016-PDocument Status - Version 1.1 (Official release)Issued on Oct. 1999

5 Yu Lei, Singh M P. A Comparison of Workflow Metamodels. In: Proc. of the ER-97 Workshop on Behavioral Modeling and Design Transformations: Issues and Opportunities in Conceptual Modeling, Los Angeles, Nov. 1997

6 Sadiq W, Orlowska M E. Analyzing Process Models Using Graph Reduction Techniques. Information System, 2002, 25(2): 117～134

7 ter Hofstede A H M, Orlowska M E. On the Complexity of Some Verification Problems in Process Control Specifications. The Computer Journal, 1999, 42(5): 349～359

8 Lin H, Zhao Z, Li H, Chen Z. A Novel Graph Reduction Algorithm to Identify Structural Conflicts. In: Proc. of the Thirty-Fourth Annual Hawaii Intl. Conf. on System Science (HICSS-35). IEEE Computer Society Press, 2002

9 ter Hofstede A H M, Orlowska M E, Rajapakse J. Verification Problems in Conceptual Workflow Specifications. In: Intl. Conf. on Conceptual Modeling / the Entity Relationship Approach, 1996. 73～88

10 van der Aalst W M P. Workflow Verification: Finding Control-Flow Errors Using Petri-Net-Based Techniques. Business Process Management, 2000. 161～183

11 van der Aalst W M P, Hirnschall A, Verbeek H M W. An Alternative Way to Analyze workflow Graphs. url = "citeseer. nj. nec. com/556825. html"

12 Adam N, Atluri V, Huang W-K. Modeling and Analysis of Workflows Using Petri Nets. Journal of Intelligent Information Systems, 1998, 10: 131～158

13 Davulcu H, Kifer M, Ramakrishnan C R, Ramakrishnan I V. Logic Based Modeling and Analysis of workflows. In: Proc. of the ACM Symposium on Principles of Database Systems (PODS'98), Seattle WA USA, 1998