

基于 PAT 的使用控制模型的形式化规约与安全性分析

周从华, 陈伟鹤, 刘志锋

(江苏大学计算机科学与通信工程学院, 江苏 镇江 212013)

摘要: 使用控制模型 UCON 是高度分布式、网络化的异构开放式计算环境下实现数字资源保护的新型访问控制模型。首先, 利用态式时间进程代数 TCSP#建立了每个 UCON 核模型的形式化规约, 以及针对一般化 UCON 的组合规约机制。其次, 提出了各种基于单会话的进程组合机制, 实现了复杂并发会话、时间控制与非确定性的形式化规范, 从而使组合进程的可达空间即为所求空间。最后, 提出了基于可达空间的 UCON 安全性分析方法, 以及基于进程代数等价的控制规则冲突性分析方法。所有的工作都已在 PAT 上实现, 表明所提方法是切实可行的, 同时也为 UCON 的形式化规范与安全性验证寻找到了一个合适的工具。

关键词: UCON; 形式化规约; 安全性分析; 模型检测

中图分类号: TP309.7

文献标识码: A

doi: 10.11959/j.issn.2096-109x.2016.00038

Formal specification and security verification of usage control model based on PAT

ZHOU Cong-hua, CHEN Wei-he, LIU Zhi-feng

(School of Computer Science and Telecommunication Engineering, Jiangsu University, Zhenjiang 212013, China)

Abstract: Usage control (UCON) is an access control model to enforce digital resources protection in highly distributed, heterogeneous network computing environment. Firstly, each core model of UCON was specified formally with TCSP#, and a combination specification mechanism was proposed for general UCON. Secondly, as the basis of the security analysis, the concepts and calculation method of the reachable space were given. Various combination mechanisms of processes based on single-session was presented to achieve formal specifications of complex concurrent sessions, timings and nondeterminism. Then the reachable space of combined processes was the desired space. Finally, the security analysis method based on the reachable space and the conflict analysis of access control policies based on the equivalent checking in process algebras were proposed for UCON model. All the proposed work had been formal checked in PAT. The experiment result shows that the proposed approaches are feasible, and PAT is a really great tool for the systematic formal specification and security analysis of UCON.

Key words: UCON, formal specification, security analysis, model checking

1 引言

现今社会, 信息技术持续的创新促使各种移

动设备如智能手环、移动电话、PAD 等成为人们日常生活不可或缺的信息产品, 进而使对数字信息的访问变得无处不在, 访问环境也从封闭系统

收稿日期: 2016-02-12; 修回日期: 2016-03-05。通信作者: 周从华, chzhou@ujs.edu.cn

基金项目: 国家自然科学基金资助项目 (No.61300228)

Foundation Item: The National Natural Science Foundation of China (No.61300228)

转变为分布式、网络化的计算环境。在这种无处不在的计算环境下,传统的访问控制,如自主访问控制 DAC^[1]、强制访问控制 MAC^[1]与角色访问控制 RBAC^[2,3]已经不再适用。Park 和 Sandhu^[4]在 2002 年首次提出了 UCON 模型,目的在于实现无处不在的开放式计算环境下对信息的合法访问。

UCON 模型区别于传统访问控制模型的主要特征在于可变属性与连续的访问控制。UCON 定义了授权、义务和条件 3 个访问控制决策因素,必须当 3 个因素都被满足时访问才能得到许可。在 UCON 中,决策因素并不仅仅在访问执行之前被检测,在访问的过程中亦会被重复检测,只有在访问过程中主客体属性的变异和运行环境的变化没有导致某些决策因素不被满足时,访问才可以继续,否则撤销本次访问。UCON 模型不仅包含了 DAC、MAC 和 RBAC,还包含了数字版权管理与信任管理等,涵盖了现代电子商务和信息系统面临的安全和隐私 2 个重要的信息安全问题。因此,UCON 模型为无处不在的计算环境下信息的合法访问提供了一种新方法,被称作下一代访问控制模型。

一个正确的访问控制策略首先应该为用户提供充分的许可来执行他们的操作并获取合法信息,同时应该阻止用户恶意的行为。为了保证控制策略的正确性,控制策略的形式化规约与安全验证必不可少。规约可以帮助人们识别可能的系统行为,验证可以帮助人们验证安全性问题,如某个主体最终能否获得某个客体的访问权限。UCON 支持可变属性和连续决策,这为 UCON 的形式化规约与安全性验证带来了很大挑战。本文的研究动机主要在于应对此挑战,从规约到验证提供一个系统的解决方案,同时寻求现有成熟工具的支持,使方案变得切实可行。

目前,已有的 UCON 形式化规约方式可以分为基于逻辑的形式化规约^[5~7]与基于进程代数的形式化规约^[8,9]。文献[10]对所有的规约形式从表达力方面进行了系统的比较,认为目前已有的规约形式在表达力上有很大欠缺,特别是在并发使用会话之间的交互、时间控制与非确定性 3 个方面没有得到充分的表达。在 UCON 的安全性验证上,目前存在的工作比较少。文献[11]证明了在

属性值域是有限的前提下,仅仅考虑授权因素的 UCON 子模型 UCON_A的安全性是可判定的,但是并没有给出具体的方法来计算可达空间。文献[12]利用模型检测工具 SPIN^[13]对 UCON 在应用层的并发执行进行了建模与验证,其考虑的并发场景仅仅限于主体对客体的单次访问。

PAT^[14]是目前先进的模型检测工具,主要优点包括:1) PAT 的建模语言 TCSP#^[15]集成了高层规约和程序级规约,支持各种组合操作、实时性与自定义数据结构,具有表达力强、建模直观与便捷等特性;2) PAT 支持多种性质的验证,比如死锁、可达性、线性时态逻辑 LTL^[16]以及精化。鉴于 PAT 是一款出色的模型检测工具,本文的主要研究动机在于基于 PAT 平台,从访问控制策略的规约到安全性验证形成一套系统的方法,具体工作包括以下几个方面:首先,为每个 UCON 核模型建立 TCSP#规范,为一般化 UCON 建立规范组合机制,使对任意 UCON 按照这套机制可完整地建立其 TCSP#规范;其次,提出了各种基于单使用会话的进程组合机制,建立了并发使用会话、时间控制与非确定性的形式化规范,以此实现了任意时刻任意主体对任意客体发起任意访问请求的形式化规范以及可达空间的计算;最后,提出了基于可达空间的简单安全性、简单可用性等属性的分析方法,以及基于进程代数等价的控制策略冲突性分析方法。

2 使用控制 UCON

UCON 支持丰富、细粒度和持续的数字资源的访问控制。对于许多应用如 GRID、Web Service、云计算、无线自组网等 UCON 均提供了可行的、彻底的解决方案。UCON 共由 6 个部分组成:主体及其属性、客体及其属性、权限、授权、义务和条件,其中,授权、义务和条件是访问许可决策模块。

主体是资源访问的请求者,能够激活访问,同时会采取某些事件来完成访问。主体主要通过其属性来定义和表示,主体的属性是指那些能够进行决策的性质或者能力。客体是被主体执行访问权限的实体,客体的属性包括那些能够进行决策的性质或者能力。权限与传统访问控制中的访问权限一致,即表示那些能够被主客体利用的使用许可。

授权是定义在主客体属性上的谓词,是关于主客体属性的约束(如主体的角色必须为主治医生),谓词必须为真访问请求才可能得到许可。实际上从传统的访问控制系统开始,授权已经获得了广泛的应用。只不过在 UCON 中授权谓词可能在访问之前、访问的过程中进行评估,而传统访问控制只支持访问之前的评估。

义务是主体在访问客体之前、过程中必须完成的事件(如用户在注册前必须签署遵守某种约定的协议),增强了 UCON 模型的表达力。条件是对系统运行环境的约束,与主客体属性无关(如对客体的访问时间只能限于白天工作时间)。条件谓词的评估可以在访问实施之前或者在访问实施过程中执行。需要特别注意的是条件检测不能改变主体、客体和运行环境的属性。

在 UCON 中单次使用会话的基本过程可以总结如下。开始时主体 s 通过执行事件 `tryaccess.s.o.r` 发出对客体 o 执行权限为 r 的访问请求。系统评估授权谓词的真值,如果值为假,则立即通过执行事件 `denyaccess.s.o.r` 来拒绝此次访问请求,如果值为真,则执行事件 `permitaccess.s.o.r` 来同意此次访问请求,并继续执行事件 `preupdate.s.o.r` 来更新属性的值,执行事件 `doaccess.s.o.r` 完成访问请求,系统也进入在线使用控制的阶段。在访问过程中,主客体属性可能需要周期性更新,执行事件 `onupdate.s.o.r` 来完成该更新过程。在访问过程中,如果授权、条件和义务有一个失效,则系统立即通过执行事件 `revokeaccess.s.o.r` 来终止此次访问。在访问过程中,如果主体主动终止访问过程,可通过执行事件 `endaccess.s.o.r` 来实现。无论是主动终止还是失效终止,终止后属性的值都可能被更新,这种更新通过事件 `postupdate.s.o.r` 来执行。需要注意的是,主动终止与失效终止所引起的属性的变化可能会有所不同。

3 UCON 的 TCSP#规范

依据无处不在计算环境下访问控制的执行流程,可以定义如下 9 个原子事件。

- 1) `tryaccess.s.o.r`: 主体 s 对客体 o 产生一个权限为 r 的访问请求。
- 2) `permitaccess.s.o.r`: 访问请求 `tryaccess.s.o.r`

得到系统许可。

3) `denyaccess.s.o.r`: 访问请求 `tryaccess.s.o.r` 被系统拒绝。

4) `doaccess.s.o.r`: 主体 s 对客体 o 执行权限为 r 的访问。

5) `revokeaccess.s.o.r`: 系统撤销主体 s 对客体 o 正在执行的权限为 r 的访问。

6) `endaccess.s.o.r`: 主体 s 主动结束对客体 o 的权限为 r 的访问。

7) `preupdate.s.o.r`: 在访问被许可前对主客体的属性进行更新,或者在许可被拒绝后对主客体的属性进行更新。

8) `onupdate.s.o.r`: 在访问的过程中由系统来更新主客体的属性,这种更新操作在访问过程中可以重复进行。

9) `postupdate.s.o.r`: 在访问结束后由系统更新主客体的属性。

3.1 授权核模型的规约

依据属性更新的时机不同,对 UCON 中的每一个决策构件定义了多种核模型,本节主要讨论授权核模型的 TCSP#规范。在授权核模型中,使用控制的决策依赖于主客体的属性值。依据属性更新操作的时间点,即执行访问前、执行访问过程中和访问结束后 3 个阶段,共有 8 个核授权模型。

1) `preA0`: 由授权决定的使用控制决策,决策判定发生在访问之前,且在访问之前、访问过程中、访问之后没有属性更新操作。

2) `preA1`: 由授权决定的使用控制决策,决策判定发生在访问之前,且在执行访问操作前完成主客体相关属性的更新。

3) `preA2`: 由授权决定的使用控制决策,决策判定发生在访问之前,且在执行访问的过程中完成主客体相关属性的更新。

4) `preA3`: 由授权决定的使用控制决策,决策判定发生在访问之前,且在访问结束后完成主客体相关属性的更新。

5) `onA0`: 由授权决定的使用控制决策,决策判定发生在访问过程中,且在访问之前、访问过程中、访问之后没有属性更新操作。

6) `onA1`: 由授权决定的使用控制决策,决策

判定发生在访问过程中, 且在执行访问操作前完成主客体相关属性的更新。

7) onA2: 由授权决定的使用控制决策, 决策判定发生在访问过程中, 且在执行访问的过程中完成主客体相关属性的更新。

8) onA3: 由授权决定的使用控制决策, 决策判定发生在访问过程中, 且在访问结束后完成主客体相关属性的更新。

3.1.1 核模型 preA0

在 preA0 中授权决策由系统执行, 且发生在访问之前, 同时主体和客体的属性在访问全程中没有任何变化。利用 TCSP#描述的 preA0 如下。

```
preA0_S_O_R(s, o, r) = tryaccess.s.o.r->if(P1
&& ... &&Pn){permitaccess.s.o.r->doaccess.s.o.r->end.
s.o.r->Skip}else{denyaccess.s.o.r->Skip}
```

在 preA0_S_O_R(s,o,r)中, P1&&...&&Pn 是建立在主体和客体属性上的授权谓词。当该谓词为真时, 访问得到许可, 并开始执行, 否则访问被系统予以拒绝。为了区分不同的访问进程, 在进程的命名上, preA0 表示决策和属性更新的类型, S 表示具有某种共同特性的主体集, 如 STUDENT 表示由学生形成的主体的集合, O 表示具有某种共同特性的客体集, 如 FILE 表示由文件形成的集合, R 表示权限的类型, 如 READ 表示读权限。为了方便读者体会如何使用 TCSP#建模以及以往形式化规范工作的区别, 本文的实例尽量选自文献[5]。

实例 1 在强制访问控制模型中, 每个主体和客体均有一个安全级别, 主体对客体的访问主要通过安全级别的比较来实现。现在以经典的面向信息保密的 BLP (Bell-LaPadula)^[17]模型为例, 来说明如何利用 TCSP#具体描述 preA0 策略。BLP 对主体和客体都赋予一定的安全级别, 同时将安全级别进行排序。用户不能改变自身和客体的安全级别, 只有管理员才能够确定用户的访问权限。在实施访问控制时, 系统先对访问主体和受控客体的安全级别属性进行比较, 再决定访问主体能否访问受控客体。应用于保障信息保密性的访问主要有以下 2 种方式。

1) 向下读: 主体安全级别高于客体的安全级别时允许查阅的读操作。

2) 向上写: 主体安全级别低于客体的安全级别时允许写操作。

第一种方式的 TCSP#规范描述如下, 其中 s_level,o_level 分别表示主体和客体的安全等级。

```
preA0_S_O_READ(s,o,r)=tryaccess.s.o.r->if
(s_level>o_level){permitaccess.s.o.r->doaccess.s.o.r
->end.s.o.r->Skip}else{denyaccess.s.o.r->Skip}
```

第二种方式的 TCSP#规范描述如下。

```
preA0_S_O_WRITE(s,o,w)=tryaccess.s.o.w->
if (s_level<=o_level){permitaccess.s.o.w->doaccess.
s.o.w->end.s.o.w->Skip}else{denyaccess.s.o.w->Skip}
```

3.1.2 核模型 preA1

preA1 与 preA0 的主要不同在于在访问控制实施之前主客体的属性可能会被更新。一般情况下, 更新操作发生在主体发出访问请求且请求得到许可之后。利用 TCSP#描述的 preA1 如下。

```
preA1_S_O_R(s,o,r)=tryaccess.s.o.r->if(P1&&
... &&Pn){permitaccess.s.o.r->preupdate.s.o.r{ ... }->
doaccess.s.o.r->end.s.o.r->Skip}else{denyaccess.s.o.r
-> Skip}
```

在属性更新操作 preupdate{...}中可以完成多个属性的更新, 同时还有时序性约束, 如 preupdate{x=x+1;y=y+1}表示先完成对 x 的更新, 然后完成对 y 的更新。

实例 2 在一个按次计费的数字版权保护系统中, 主体拥有属性 credit, 表示拥有的存款, 客体拥有属性 value, 表示每次访问的代价。当主体的 credit 超过客体的 value 时, 主体可以对客体执行读的访问。在访问开始前, 主体的存款 credit 将减少 value。具体的访问过程如下。

```
preA1_S_O_READ(s,o,r)=tryaccess.s.o.r->if
(s_credit>=o_value){permitaccess.s.o.r->preupdate.
s.o.r{s_credit=s_credit-o_value;}->doaccess.s.o.r->
end.s.o.r->Skip}else{denyaccess.s.o.r->Skip}
```

3.1.3 核模型 preA2

在模型 preA2 中, 授权决策由系统在访问之前检测, 且在访问过程中会有多个属性值被更新。利用 TCSP#描述的 preA2 如下。

```
preA2_S_O_R(s,o,r)=tryaccess.s.o.r->if(P1&&...
&&Pn){permitaccess.s.o.r->doaccess.s.o.r->onup
date.s.o.r{...}->end.s.o.r->Skip}else{denyaccess.s.o.r
```

->Skip}

3.1.4 核模型 preA3

在模型 preA3 中, 授权决策由系统在访问之前检测, 且在访问结束后会有多个属性值被更新。利用 TCSP#描述的 preA3 如下。

```
preA3_S_O_R(s,o,r)=tryaccesss.o.r->if(P1&&
...&&Pn){permitaccess.s.o.r->doaccess.s.o.r->end.
s.o.r->postupdate.s.o.r{...}->Skip}else{denyacc
ess.s.o.r->Skip}
```

实例 3 在一个基于会员的数字版权管理系统中, 读者拥有属性 expense 和 readingGroup, 书拥有属性 readingGroup 和 readingCost。如果读者和书属于同一个阅读组, 则读者可以阅读此书, 同时阅读结束后读者的阅读费用在原 expense 的基础上增加 readingCost。具体访问控制过程如下。

```
preA3_S_O_READ(s,o,r) = tryaccesss.o.r -> if
(s_readingGroup==o_readingGroup){permitacc
ess.s.o.r->doaccess.s.o.r->end.s.o.r->postupdate.s.o.
r{s_expense=s_expense+o_readingCost}->Skip}els
e{denyaccess.s.o.r->Skip}
```

3.1.5 核模型 onA0

在授权决策由系统在访问之前检测的模式下, 访问请求得到许可后将不再进行安全检查。在模型 onA0 中, 授权决策将被重复检测, 一旦检测到授权谓词真值为假, 系统立即撤销此次访问。在访问的过程中, 主体随时可能结束访问, 引入事件 continue.s.o.r 表示主体继续访问, 引入事件 nocontinue.s.o.r 表示主体主动结束访问。利用 TCSP#描述的 onA0 如下。

```
onA0_S_O_R_1(s,o,r)=tryaccess.s.o.r->onA0_
S_O_R_2(s,o,r)
onA0_S_O_R_2(s,o,r)=doaccess.s.o.r->if(P1&
& ... &&Pn){continue.s.o.r->onA0_S_O_R_2(s,o,r)[
nocontinue.s.o.r->end.s.o.r->Skip}else{revokeaccess.
s.o.r->Skip}
```

实例 4 在一个组织机构中, 用户 Bob (角色为 employee) 可以参与该组织的一个短期项目。临时证书 temp_cert 用以证明 Bob 的身份。当 Bob 访问项目中的敏感信息时, 他的数字证书 temp_cert 会被重复检测。如果检测到证书在证书撤销列表中, Bob 的临时身份将会被撤销, 进而

他不能继续对敏感信息进行访问。访问控制过程如下。

```
onA0_S_O_R_1(s,o,r)=tryaccess.s.o.r->onA0_
S_O_R_2(s,o,r)
onA0_S_O_R_2(s,o,r)=doaccess.s.o.r->if(s_rol
e==employee&&s_temp_cert==1){continue.s.o.r->
onA0_S_O_R_2(s,o,r)[nocontinue.s.o.r->end.s.o.r->
Skip}else{revokeaccess.s.o.r->Skip}
```

上述控制过程中, s_temp_cert==1 表示证书不在撤销列表中。

3.1.6 核模型 onA1

与模型 onA0 不同之处主要在于, 模型 onA1 中主客体的属性在访问开始前可以被更新。利用 TCSP#描述的 onA1 如下。

```
onA1_S_O_R_1(s,o,r)=tryaccess.s.o.r->preupda
te.s.o.r{...}->onA1_S_O_R_2(s,o,r)
onA1_S_O_R_2(s,o,r)=doaccess.s.o.r->if(P1&
& ... &&Pn){continue.s.o.r->onA1_S_O_R_2(s,o,r)[
nocontinue.s.o.r->end.s.o.r->Skip}else{revokeaccess.
s.o.r->Skip}
```

3.1.7 核模型 onA2

与模型 onA1 的不同之处主要在于, 模型 onA2 中主客体的属性在访问过程中重复被更新。利用 TCSP#描述的 onA2 如下。

```
onA2_S_O_R_1(s,o,r)=tryaccess.s.o.r->onA2_
S_O_R_2(s,o,r)
onA2_S_O_R_2(s,o,r)=doaccess.s.o.r->onup
date.s.o.r{...}-> if(P1&&...&&Pn){continue.s.o.r->
onA2_S_O_R_2(s,o,r)[nocontinue.s.o.r->end.s.o.r->
Skip}else{revokeaccess.s.o.r->Skip}
```

3.1.8 核模型 onA3

与模型 onA2 的不同之处主要在于模型 onA3 中主客体的属性在访问结束后进行更新。利用 TCSP#描述的 onA3 如下。

```
onA3_S_O_R_1(s,o,r)=tryaccess.s.o.r->onA3_
S_O_R_2(s,o,r)
onA3_S_O_R_2(s,o,r)=doaccess.s.o.r->if(P1&
& ... &&Pn){continue.s.o.r->onA3_S_O_R_2(s,o,r)
[nocontinue.s.o.r->end.s.o.r->postupdate.s.o.r{ ... }->
Skip}else{revokeaccess.s.o.r->postupdate.s.o.r{ ... }
->Skip}
```

实例 5 考察一个数字版权管理系统，该系统允许对客体的并发访问数量不超过 10 个。任何一个用户在任何时候只能发送一个新的访问请求。当系统并发访问的数量达到 10 时，新的访问请求的到达必将导致一个在线进行的访问控制被撤销。撤销的策略有多种，可以依据访问的开始时间、访问过程中的空闲时间或者访问持续的时间。对于不同的策略需要定义不同的系统属性。设 o 表示客体， s 表示主体，定义如下几个属性。

1) $s_o_accessing$ 表示主体 s 对客体 o 的访问，值为 1 表示主体 s 发出了对客体 o 的访问请求，值为 0 表示主体 s 没有发出对客体 o 的访问请求。

2) $o_accessing_number$ 表示访问客体 o 的主体的数量。

3) sys_clock 表示系统时间，为系统属性。

4) $s_o_starttime$ 是主体属性，表示主体 s 开始对客体 o 开始访问的时间。

5) $minstarttime$ 表示最先开始访问的时间。

现在以最早开始时间为撤销策略为例。因为属性的更新发生在访问开始前和访问结束后，所以，整体的使用控制策略定义为 $onA1$ 与 $onA3$ 的组合。

```
onA13_S_O_R_1(s,o,r)=tryaccess.s.o.r->preupdate.s.o.r {s_o_accessing=1; o_accessing_number =
o_accessing_number+1;s_o_starttime=sys_clock}->
onA13_S_O_R_2(s,o,r)
```

```
onA13_S_O_R_2(s,o,r)=doaccess.s.o.r->if!(o_
accessing_number==11&& o_s_starttime== minstart-
time)){continue.s.o.r->onA13_S_O_R_2(s,o,r)[
nocontinue.s.o.r->end.s.o.r->postupdate.s.o.r{o_
accessing_number=o_accessing_number-1; s_o_
starttime=0}->Skip} else {revokeaccess.s.o.r->
postupdate.s.o.r{s_o_accessing_s=0;o_accessing_s_
number=o_accessing_s_number-1;s_o_starttime=
0}->Skip}
```

3.2 义务核模型的规约

义务是 UCON 中除了授权之外另外一个重要的构件。本节将探讨义务的特 CSP#规范。由于访问决策的连续性，在 UCON 中有 2 种类型的义务：预义务，即在主体开始访问客体前必须完成的事件；在线义务，即在访问的过程中必须完成

的事件。类似于授权核模型，基于更新操作发生的时间段，可以进一步将义务核模型分解成 8 个子模型。

1) $preB0$ ：由义务决定的使用控制决策，决策判定发生在访问之前，且在访问之前、访问过程中、访问之后没有属性更新操作。

2) $preB1$ ：由义务决定的使用控制决策，决策判定发生在访问之前，且在执行访问操作前完成主客体相关属性的更新。

3) $preB2$ ：由义务决定的使用控制决策，决策判定发生在访问之前，且在执行访问的过程中完成主客体相关属性的更新。

4) $preB3$ ：由义务决定的使用控制决策，决策判定发生在访问之前，且在访问结束后完成主客体相关属性的更新。

5) $onB0$ ：由义务决定的使用控制决策，决策判定发生在访问过程之中，且在访问之前、访问过程中、访问之后没有属性更新操作。

6) $onB1$ ：由义务决定的使用控制决策，决策判定发生在访问过程之中，且在执行访问操作前完成主客体相关属性的更新。

7) $onB2$ ：由义务决定的使用控制决策，决策判定发生在访问过程之中，且在执行访问的过程中完成主客体相关属性的更新。

8) $onB3$ ：由义务决定的使用控制决策，决策判定发生在访问过程之中，且在访问之后主客体的属性将会被更新。

3.2.1 核模型 $preB0$

$preB0$ 规定只有当所有的义务事件完成后，访问才能被许可。以一个义务事件 ob 为例来说明 $preB0$ 对应的访问控制流程，主体可以执行义务，也可以不执行，引入事件 $noob$ 表示主体没有执行义务 ob 。利用 TCSP#描述的 $preB0$ 如下。

```
preB0_S_O_R(s,o,r)=tryaccess.s.o.r->(ob.s.o.r->
doaccess.s.o.r->end.s.o.r->Skip[]noob.s.o.r->
denyaccess.s.o.r->Skip)
```

实例 6 在一个在线电子市场中，客户为了完成订购，需要点击接受订购协议的同意按钮。定义事件 $click_agreement$ 表示客户点击了接受订购协议的同意按钮。该策略的 TCSP#规范如下。

```
preB0_S_O_R(s,o,r)=tryaccess.s.o.r->(click
```

_ agreement.s.o.r->doaccess.s.o.r->end.s.o.r->Skip[]
nonclick_agreement.s.o.r->denyaccess.s.o.r->Skip)

3.2.2 核模型 preB1

PreB1 规定只有当所有的义务事件完成后,访问才能被许可,且在访问开始之前需要更新主客体的属性。利用 TCSP#描述的 preB1 如下。

preB1_S_O_R(s, o, r)=tryaccess.s.o.r->(ob.s.o.r->preupdate.s.o.r{...}->doaccess.s.o.r->end.s.o.r[]
noob.s.o.r->denyaccess.s.o.r->Skip)

3.2.3 核模型 preB2

PreB2 规定只有当所有的义务事件完成后,访问才能被许可,且在访问过程中更新主客体的属性。利用 TCSP#描述的 preB2 如下。

preB2_S_O_R(s, o, r)=tryaccess.s.o.r->(ob.s.o.r->doaccess.s.o.r->onupdate.s.o.r{...}->end.s.o.r->Skip[]nonob.s.o.r->denyaccess.s.o.r->Skip)

3.2.4 核模型 PreB3

PreB3 规定只有当所有的义务事件完成后,访问才能被许可,且在访问结束后更新主客体的属性。利用 TCSP#描述的 preB3 如下。

preB3_S_O_R(s, o, r)=tryaccess.s.o.r->(ob.s.o.r->doaccess.s.o.r->end.s.o.r->postupdate.s.o.r{...}->Skip[]nonob.s.o.r->denyaccess.s.o.r->Skip)

实例 7 在实例 7 中,当顾客完成订购,在订单列表中添加一项订单时,订单列表需要更新。引入主体属性 s_o_order 表示主体 s 是否对客体 o 下了订单:值为 1 表示下了订单,值为 0 表示没有下订单。此策略的 TCSP#规范表示如下。

preB3_S_O_R(s, o, r)=tryaccess.s.o.r->(ob.s.o.r->doaccess.s.o.r->end.s.o.r->postupdate.s.o.r{s_o_order=1}->Skip[]nonob.s.o.r->denyaccess.s.o.r->Skip)

3.2.5 核模型 onB0

onB0 规定主体必须在访问控制的过程中重复执行义务事件,否则访问将被取消,且在整个访问过程中无需更新主客体的属性。利用 TCSP#描述的 onB0 如下。

onB0_S_O_R_1(s, o, r)=tryaccess.s.o.r->onB0_S_O_R_2(s, o, r)

onB0_S_O_R_2(s, o, r)=doaccess.s.o.r->(ob.s.o.r->(continue.s.o.r->onB0_S_O_R_2(s, o, r))[]nocontinue.s.o.r->end.s.o.r->Skip[])noob.s.o.r->revo

keaccess.s.o.r->Skip)

实例 8 为了使用在线服务,广告标语必须在客户端打开,否则服务就失效。该控制策略的 TCSP#规范表示如下。

onB0_S_O_R_1(s, o, r)=tryaccess.s.o.r->onB0_S_O_R_1(s, o, r)

onB0_S_O_R_2(s, o, r)=doaccess.s.o.r->(openbanner.s.o.r->(continue.s.o.r->onB0_S_O_R_2(s, o, r))[]nocontinue.s.o.r->end.s.o.r->Skip[])noopenbanner.s.o.r->revokeaccess.s.o.r->Skip)

3.2.6 核模型 onB1

onB1 规定主体必须在访问控制的过程中重复执行义务事件,否则访问将被取消,且在访问开始前更新主客体的属性。利用 TCSP#描述的 onB1 如下。

onB1_S_O_R_1(s, o, r)=tryaccess.s.o.r->preupdate.s.o.r{...}->onB2_S_O_R_1(s, o, r)

onB1_S_O_R_2(s, o, r)=doaccess.s.o.r->(ob.s.o.r->(continue.s.o.r->onB1_S_O_R_2(s, o, r))[]nocontinue.s.o.r->end.s.o.r->Skip[])noob.s.o.r->revokeaccess.s.o.r->Skip)

3.2.7 核模型 onB2

onB2 规定主体必须在访问控制的过程中重复执行义务事件,否则访问将被取消,且在访问过程中重复更新主客体的属性。利用 TCSP#描述的 onB2 如下。

onB2_S_O_R_1(s, o, r)=tryaccess.s.o.r->onB2_S_O_R_2(s, o, r)

onB2_S_O_R_2(s, o, r)=doaccess.s.o.r->oneupdate.s.o.r{...}->(ob.s.o.r->(continue.s.o.r->onB2_S_O_R_2(s, o, r))[]nocontinue.s.o.r->end.s.o.r->Skip[])noob.s.o.r->revokeaccess.s.o.r->Skip)

3.2.8 核模型 onB3

onB3 规定主体必须在访问控制的过程中重复执行义务事件,否则访问将被取消,且在访问结束或者撤销后更新主客体的属性。利用 TCSP#描述的 onB3 如下。

onB2_S_O_R_1(s, o, r)=tryaccess.s.o.r->onB2_S_O_R_2(s, o, r)

onB2_S_O_R_2(s, o, r)=doaccess.s.o.r->(ob.s.o.r->(continue.s.o.r->onB2_S_O_R_2(s, o, r))[]noconti

```
nue.s.o.r->end.s.o.r->postupdate.s.o.r{ ... }->Skip)[]
noob.s.o.r->revokeaccess.s.o.r->postupdate.s.o.r
{...}-> Skip)
```

实例 9 在一个在线服务应用中, 用户需要每隔 30 min 点击广告窗口, 否则服务自动撤销。主体属性 `s_usagetime` 表示服务应用的在线使用时间, 访问开始时 `s_usagetime` 的初始值设置为 0, 在访问过程中 `s_usagetime` 不断增加, 在访问结束后 `s_usagetime` 又被设置为 0, 因此该访问控制策略由 `onB1`、`onB2`、`onB3` 三者组合而成。

此实例中义务的执行是有条件的, 即只有当在线时间达到 30 min 时才必须执行点击广告的事件, 因此在重复检测的过程中需要加入条件进行判断。以下是该控制流程的 TCSP#规范。

```
onB123_S_O_R_1(s,o,r)=tryaccess.s.o.r->preu
pdate{s_usagetime=0}->onB123_S_O_R_2(s,o,r)
onB123_S_O_R_2(s,o,r)=doaccess.s.o.r->onup
date1.s.o.r{s_usagetime=s_usagetime+1}->if(s_usa
getime==30){(clickad.s.o.r->onupdate2.s.o.r {s_
usagetime=0}->(continute.s.o.r->onB123_S_O_R_2
(s,o,r)[nocontinue.s.o.r->end.s.o.r->postupdate.s.o.r
{s_usagetime=0}->Skip))[(noclickad.s.o.r->
revokeaccess.s.o.r->postupdate.s.o.r{s_usagetime=0}
->Skip)}else{(continute.s.o.r->onB123_S_O_R_2(s,
o,r))[(nocontinue.s.o.r->end.s.o.r->postupdate.s.o.
r{s_usagetime=0}-> Skip)}
```

3.3 条件核模型的规约

条件构件是面向环境和系统的决策因素, 条件谓词用于估算当前的环境和系统状态。与授权、义务不同, 条件不能改变主客体的属性。基于条件谓词估算的时间点的不同, 条件核模型可以分为预条件和在线条件 2 种类型。

3.3.1 核模型 preC0

在模型 `preC0` 中条件谓词必须在访问开始前检测, 且没有任何的主客体属性被更新。利用 TCSP#描述的 `preC0` 如下。

```
preC0_S_O_R(s,o,r)=tryaccess.s.o.r->if(PC1
&& ... &&PCn){permitaccess.s.o.r->doaccess.s.o.r->
end.s.o.r->Skip}else {denyaccess.s.o.r->Skip}
```

3.3.2 核模型 onC0

在模型 `onC0` 中条件谓词在访问的过程中不

断被检测, 如果谓词真值为真, 则继续访问, 否则立即撤销当前的访问。此外在访问整个过程中没有任何的主客体属性被更新。利用 TCSP#描述的 `onC0` 如下。

```
onC0_S_O_R_1(s,o,r)=tryaccess.s.o.r->onC0_
S_O_R_2(s,o,r)
onC0_S_O_R_2(s,o,r)=doaccess.s.o.r->if(PC
1&&...&&PCn){continue.s.o.r->onC0_S_O_R_2(s,o, r)
[nocontinue.s.o.r->end.s.o.r->Skip}else{revokeaccess.
s.o.r->Skip}
```

实例 10 访问控制策略定义一个值日班的职员(角色为 `dayshifter`) 仅仅在白天能够访问对象、定义 `currenttime` 为系统属性, 表示当前时间。该策略是 `preA0`、`preC0`、`onC0` 的组合, TCSP#描述如下

```
preA0_preC0_OnC0_S_O_R_1(s,o,r)=tryacce
s.s.o.r->if(s_role==dayshifter&&8am<=currenttime
<=5pm){permitdoaccess.s.o.r->doaccess.s.o.r->
preA0_preC0_OnC0_S_O_R_2(s,o,r)}else{denya
ccess.s.o.r->Skip}
preA0_preC0_OnC0_S_O_R_2(s,o,r)=if(8am<
=currenttime<=5pm){doaccess.s.o.r->(continue.s.o.r
->preA0_preC0_OnC0_S_O_R_2(s,o,r)[nocontinute.
s.o.r->end.s.o.r->Skip)}else{revokeaccess.s.o.r->
Skip}
```

4 并发、时间约束、非确定性

UCON 应用于计算无处不在的环境, 因此访问的并发性、访问时间的控制与访问的非确定性是 UCON 中最重要的特性。目前还没有任何工作对此 3 个问题进行详细探讨, 文献[10]也将此问题列为 UCON 的主要挑战。如何建立正确描绘此 3 个问题的模型至关重要, 本节将从建模的角度对此 3 个问题展开详细分析。

4.1 并发

UCON 的服务对象是分布式、网络化系统, 在某一段时间同一个主体访问不同客体, 不同主体访问同一个主体将是常态, 因此并发访问是 UCON 应用中一个非常基础的特征。在访问过程中属性的更新操作将会影响另外一个正在发生的访问控制的授权决策。由于主体的行为不受约束,

它可以在任意时刻发起对任意客体的访问，依据并发的复杂性，将并发分成多种情况，以建立不同并发类型的 TCSP#规范。以实例 12 为例，进行详细阐述。

实例 11 在一个基于用户预消费的数字版权管理系统中，读者 s 拥有属性 *expense*，表示账户里的预存款，书 o 拥有属性 *readingcost* 表示用户阅读本书一次的价格。访问控制策略是用户发出请求后，如果账户的存款余额不小于阅读的代价，则此次访问请求得到许可，并从预存款中扣除相应的价格。此外假设管理系统已经规定同一本书被多个用户阅读的次数累计不超过 K 次。

现在假设有两个读者 $s0$ 和 $s1$ ，其初始拥有的 *expense* 分别为 11 和 9。假设有两本书 $b0$ 与 $b1$ 。 $b0$ 的属性 *readingcost* 为 5， $b1$ 的属性 *readingcost* 为 4。定义数组 $s[2]$ ， $s[i](i=0,1)$ 表示第 i 个主体的属性 *expense*。定义数组 $b[2]$ ， $b[i](i=0,1)$ 表示第 i 个客体的属性 *readingcost*。定义数组 $number[2]$ ， $number[i](i=0,1)$ 用以记录客体 i 被阅读的次数，初始值为 0。

1) Type-1 并发：某个时刻主体可以发出对客体的单次访问。

在访问控制流程中主客体的属性将在访问开始前予以更新，因此属于核模型 $preA1$ ，具体的 TCSP#规范如下。

```
preA1_S_O_R_Type_1(i,j,r)=tryaccess.i.j.r->if
(s[i]>=b[j]&&number[j]<K)
{permitaccess.s.o.r->preupdate.i.j.r{s[i]=s[i]-b[j];
number[j]=number[j]+1;}->doaccess.i.j.r->end.i.j.r
->Skip}else{denyaccess.i.j.r->Skip}
```

2) Type-2 并发：上次访问结束后主体可以继续对同一客体发出多次相同的访问请求。

在访问过程中主客体的属性将在访问开始前予以更新，因此属于核模型 $preA1$ 。同时允许访问结束后主体可以对客体继续访问，这种许可在 TCSP#中可以使用进程 $P()=a->P()$ 来定义，具体的 TCSP#规范如下。

```
preA1_S_O_R_Type_2(i,j,r)=tryaccess.i.j.r->if
(s[i]>=b[j]&&number[j]<K){permitaccess.s.o.r->
preupdate.i.j.r{s[i]=s[i]-b[j];number[j]=number[j]
+1;}->doaccess.i.j.r->end.i.j.r->preA1_S_O_R_Type
```

```
e_2(i,j,r)}else{denyaccess.i.j.r->preA1_S_O_R_Type
e_2(i,j,r)}
```

Type-2 并发与 Type-1 并发的主要区别在于，将 $preA1_S_O_R_Type_1(i,j,r)$ 中所有 Skip 事件替换为 $preA1_S_O_R_Type_2(i,j,r)$ 。依据 TCSP#的语义解释，当事件 end 或者 denyaccess 执行结束后，按照 $preA1_S_O_R_Type_2(i,j,r)$ 的规范重新开始运行，从而实现了新一轮次的访问。

3) Type-3 并发：在一段时间内主体可以发出对同一客体的多次相同的访问请求，且并不需要等待上次访问的结束，即所设计的进程必须能够描述在访问过程中主体可以随时发出对客体的又一次访问请求的规范。

在 TCSP#中算子 $\diamond$ 实现了多个进程之间的随机选择，因此在原理上可以对进程 $preA1_S_O_R_Type_2(i,j,r)$ 中的每一个事件添加一个可以发出访问请求的选项来达到一段时间内发出多次访问请求的目的。但是， $\diamond$ 实现的是进程之间而不是事件之间的随机选择，因此首先要将 $preA1_S_O_R_Type_2(i,j,r)$ 中的每一个事件重新定义为一个进程，然后按照 $preA1_S_O_R_Type_2(i,j,r)$ 组织进程之间的序，具体的 TCSP#规范如下。

```
Proc_tryaccess(i,j,r)=tryaccess.i.j.r->if(s[i]>=b
[j]&&number[j]<K){Proc_permitaccess(i,j,r)}else{
Proc_denyaccess(i,j,r)}
```

```
Proc_permitaccess(i,j,r)=permitaccess.i.j.r->Pr
oc_preupdate(i,j,r)<> preA1_S_O_R_Type_3(i,j,r)。
```

```
Proc_preupdate(i,j,r)=preupdate.i.j.r{s[i]=s[i]-
b[j]; number[j]=number[j]+1;}->Proc_doaccess(i,j,r)
<> preA1_S_O_R_Type_3(i,j,r)
```

```
Proc_doaccess(i,j,r)=doaccess.i.j.r->Proc_end
(i,j,r)<> preA1_S_O_R_Type_3(i,j,r)
```

```
Proc_denyaccess(i,j,r)=denyaccess.i.j.r->preA1_
S_O_R_Type_3(i,j,r)
```

```
Proc_end(i,j,r)=end.i.j.r->preA1_S_O_R_Type
_3(i,j,r)
```

```
preA1_S_O_R_Type_3(i,j,r)=Proc_tryaccess(i,
j,r)<> preA1_S_O_R_Type_3(i,j,r)
```

4) Type-4 并发：同一时间主体可以对客体发出多次相同的访问。

定义进程 $preA1_S_O_R_Type_4(i,j,r,k)$ 来描

绘某个时刻主体可以发出对客体的单次访问。与 $\text{preA1_S_O_R_Type_1}(i,j,r)$ 的不同之处在于引入 k 用来标记发出的第几次访问请求。

```
preA1_S_O_R_Type_4(i,j,r,k)=tryaccess.i.j.r->
if(s[i]>=b[j]&&number[j]<K){preupdate.i.j.r.k{s[i]
=s[i]-b[j]; number[j]=number[j]+1;}->doaccess.i.j.r.
k->end.i.j.r->Skip} else {denyaccess.i.j.r.k-> Skip}
```

定义进程 $\text{preA1_S_O_R_Type_4_parallel}(i, j, r, m)=\text{preA1_S_O_R_Type_4}(i,j,r,0) \parallel \cdots \parallel \text{preA1_S_O_R_Type_4}(i,j,r,m-1)$ 。

进程 $\text{preA1_S_O_R_Type_4_parallel}$ 中, 符号 $\parallel$ 表示多个进程之间的平行组合。为了实现同时发出多次访问请求的描述, 引入了事件 tryaccess.i.j.r 进行同步。平行组合意味着 $\text{preA1_S_O_R_Type_4}(i,j,r,0), \cdots, \text{preA1_S_O_R_Type_4}(i,j,r,m-1)$ 必须同时执行 tryaccess.i.j.r 事件, 从而实现了 m 个进程同时发出访问请求的目的。

5) Type-5 并发: 在一段时间内主体可以对不同的客体产生访问请求。

在 TCSP# 中多进程之间的交互意味着在任意时刻可从多进程中任意选择一个进程执行, 利用交互算子 $\parallel$ 定义进程 $\text{preA1_S_O_R_Type_5}(i,r)=\text{preA1_S_O_R_Type_1}(i,0,r) \parallel \cdots \parallel \text{preA1_S_O_R_Type_1}(i,n-1,r)$ 。算子 $\parallel$ 确保了在任意时刻可从 n 个子进程中选择一个进程执行, 从而达到主体对不同客体的访问。

6) Type-6 并发: 在一段时间内不同主体可以对相同的客体产生访问请求。

与 Type-5 并发类似, 定义进程 $\text{preA1_S_O_R_Type_6}(j,r)=\text{preA1_S_O_R_Type_1}(0,j,r) \parallel \cdots \parallel \text{preA1_S_O_R_Type_1}(m-1,j,r)$ 。

7) Type-7 并发: 同一时间主体可以对不同的客体产生访问请求。

首先定义进程 $\text{preA1_S_O_R_Type_7}(i,j,r)$ 描绘主体对客体的单次访问, 与 $\text{preA1_S_O_R_Type_1}(i,j,r)$ 不同的是, 这里引入事件 tryaccess 用于进程之间的同步

```
preA1_S_O_R_Type_7(i,j,r)=tryaccess->if(s[i]
>=o[j]&&number[j]<K){preupdate.i.j.r{s[i]=s[i]-o[j];
number[j]=number[j]+1;}->doaccess.i.j.r->end.i.j.r
->Skip} else {denyaccess.i.j.r->Skip}
```

定义进程 $\text{preA1_S_O_R_Type_7_syn}=\text{preA1_S_O_R_Type_7}(i,0,r) \parallel \cdots \parallel \text{preA1_S_O_R_Type_7}(i,n-1,r)$ 。在 $\text{preA1_S_O_R_Type_7_syn}$ 中, $\text{preA1_S_O_R_Type_7}(i,j,r)(j=0, \cdots, n-1)$ 必须同时执行事件 tryaccess , 然后独立运行, 从而实现了同一时间点主体对不同客体产生访问的请求。

8) Type-8 并发: 同一时间不同主体可以对相同的客体产生访问请求。

与 Type-7 并发类似, 定义进程 $\text{preA1_S_O_R_Type_8_syn}=\text{preA1_S_O_R_Type_7}(0,0,r) \parallel \cdots \parallel \text{preA1_S_O_R_Type_7}(m-1,0,r)$ 。

9) Type-9 并发: 在一段时间内不同主体可以对不同的客体产生访问请求。

$\text{preA1_S_O_R_Type_9}()=\text{preA1_r_Type_1}(0,0,r) \parallel \cdots \parallel \text{preA1_r_Type_1}(0,n-1,r) \parallel \cdots \parallel \text{preA1_r_Type_1}(m-1,j,r)$ 。 $\parallel$ 表示进程之间的交互执行, 因此上述进程表示 m 个主体在一段时间内对 n 个客体产生了访问请求。

10) Type-10 并发: 同一时间不同主体可以对不同的客体产生访问请求。

$\text{preA1_S_O_R_Type_10}()=\text{preA1_S_O_R_Type_7}(0,0,r) \parallel \cdots \parallel \text{preA1_S_O_R_Type_7}(0,m-1,r) \parallel \cdots \parallel \text{preA1_S_O_R_Type_7}(n-1,m-1,r)$ 。在上述进程中所有进程必须同时执行事件 tryaccess , 从而实现同一时间不同主体可以对不同客体产生访问请求。

11) Type-11 并发: 在一段时间内不同主体可以对不同的客体产生多次访问请求。

定义进程 $\text{preA1_S_O_R_Type_11}()=\text{preA1_S_O_R_Type_2}(0,0,r) \parallel \cdots \parallel \text{preA1_S_O_R_Type_2}(0,n-1) \parallel \cdots \parallel \text{preA1_S_O_R_Type_2}(m-1,n-1)$, 其中 $\text{preA1_S_O_R_Type_2}(i,j,r)$ 实现了对同一客体的多次访问请求, $\parallel$ 实现了访问之间的交互。

12) Type-12 并发: 与系统环境的交互。

在实例 11 中, 系统可能随时往主体的账户上面充值, 从而导致主体的 expense 随时发生变化。对于此种情况, 引入进程 $\text{Proc_update}(i)=\text{update.i}\{s[i]=s[i]+\text{Fee}\}->\text{Skip}$ 描述充值过程, 然后可通过算子 $\parallel$ 与上述不同并发进程实现交互。

TCSP# 作为 CSP 的扩展, 在并发的描述上比 CSP 具有更强的表达能力, 上述 12 种并发类型的

成功描绘充分说明了 TCSP#能够完美刻画 UCON 中的并发情况。

4.2 时间控制

在 UCON 中当主体发出访问请求后,系统必须在一段时间内进行响应,客体也必须在一定的时间内完成义务事件的执行。例如,在一个在线电子商务中,顾客必须点击同意购买协议的按钮才能继续购物。这种点击必须有时间限制,不可能为消费者等待无穷时间,如可以设置如果在 5 min 内不点击就撤销此次访问请求。为此考察以下几种时间约束的 TCSP#规范。

1) Type-1 时间控制:请求发出后,必须在某段时间内得到响应,否则继续发送访问请求。

以核模型 preA0 为例来说明时间控制的处理。定义如下进程。

```
preA0_S_O_R_Type_1_Timing(s,o,r)=tryaccess.s.o.r->((if (P1&&...&&Pn) {permitaccess.s.o.r->doaccess.s.o.r->end.s.o.r->Skip} else {denyaccess.s.o.r->Skip}))timeout[d](denyaccess.s.o.r->preA0_S_O_R_Type_1_Timing(s,o,r)))
```

在 preA0_S_O_R_Type_1_Timing (s,o,r)中, timeout[d]表示主体执行完事件 tryaccess.s.o.r 后,如果进程 if (P1&&...&&Pn){permitaccess.s.o.r->doaccess.s.o.r->end.s.o.r->Skip} else {denyaccess.s.o.r->Skip} 在 d 个时间单元内执行了某个事件,则继续按照此进程的规范执行,否则执行进程 denyaccess.s.o.r->preA0_S_O_R_Type-1_Timing (s,o,r)。

2) Type-2 时间控制:访问得到许可后,必须在某段时间内执行完毕,否则撤销此次访问。

以核模型 preA0 为例来说明时间控制的处理。定义如下进程。

```
preA0_S_O_R_Type_2_Timing(s,o,r)=tryaccess.s.o.r->if (P1&&...&&Pn) {permitaccess.s.o.r->((doaccess.s.o.r->end.s.o.r->Skip)timeout[d] (denyaccess.s.o.r->preA0_S_O_R_Type_2_Timing(s,o,r)))} else {denyaccess.s.o.r->Skip }
```

在 preA0_S_O_R_Type_2_Timing(s, o, r)中 timeout[d]表示系统执行完事件 permitaccess.s.o.r 后,如果进程 doaccess.s.o.r->end.s.o.r->Skip 在 d 个时间单元内执行了事件 doaccess.s.o.r,则继续

按照此进程的规范执行,否则执行进程 denyaccess.s.o.r->preA0_S_O_R_Type_2_Timing (s,o,r)。

3) Type-3 时间控制:义务的执行必须在某个时间段内完成,否则撤销此次访问。

以核模型 preB0 为例来说明时间控制的处理。定义如下进程。

```
preB0_S_O_R_Type_2_Timing(s,o,r)=tryaccess.s.o.r->((ob.s.o.r->doaccess.s.o.r->end.s.o.r)timeout[d] (denyaccess.s.o.r->Skip))
```

在 preB0_S_O_R_Type_2_Timing 中,进程 ob.s.o.r->doaccess.s.o.r->end.s.o.r 的第一个事件 ob.s.o.r 必须在时间 d 内执行,否则执行进程 denyaccess.s.o.r->Skip。

实例 12 考察网格系统 (grid system) 中安全套接字 (securing socket) 的使用控制策略。主要有以下几个访问控制策略。

策略 1 套接字必须在发送数据前打开。

策略 2 只有注册的用户才可以打开编号在 1 000 到 2 000 之间的安全套接字进行连接。

策略 3 每个用户发送数据包的时间不能超过 10 min。

策略 1 到策略 3 都没有改变主客体的属性,而且访问控制决策是在访问开始前进行的,因此对应的核模型为 preA0,同时访问得到许可后,必须在 10 min 内执行完毕,对应的时间控制类型为 Type-2 Timing,具体进程定义如下。

```
preA0_S_O_R_Type_2_Timing(s,o,r)=tryaccess.s.o.r->if(1000<=o<=2000){register.s.o.r->((doaccess.s.o.r->end.s.o.r->Skip)timeout[10](revokeaccess.s.o.r))} else {denyaccess.s.o.r->Skip }
```

上述 3 种时间控制类型是 UCON 中的典型情况,应用 TCSP#成功刻画了这些场景,充分说明了 TCSP#刻画时间控制的能力。

4.3 非确定性

当主体对客体的访问请求得到许可后,主体通过执行一系列的原语操作实现对客体的访问。这些原语操作的执行顺序是随机的,如用户登录到邮箱帐户后可能采取的操作,但是不同的操作序列对主体和客体属性的影响是不同的,从而导致后面的访问控制受到影响。

在第 5.1 节的并发建模中利用一个抽象的事件 `doaccess` 来表示访问执行过程。假设 `doaccess` 可以分解成 `act_1, act_2, ..., act_n` 等 n 个原子事件, 这些事件的执行顺序是随机的。通过交互算子 $\parallel$ 实现访问操作的随机性, 具体进程定义为: $(act_1 \rightarrow Skip \parallel \dots \parallel act_n \rightarrow Skip)$ 。

非确定性的另外一种呈现方式是访问结束后属性的更新。依据是正常结束还是撤销结束的形式不同, 属性的更新操作也可能不同。如主体的信誉, 正常结束可以提高主体的信誉, 撤销结束将降低主体的信誉。对于这种由于结束方式不同所引起的属性更新的不确定性, 分别用进程 `end.i.j.r->postupdate.i.j.r` 和进程 `revokeaccess.i.j.r->postupdate.i.j.r` 实现两者之间的区别。

5 安全性分析

安全性分析是访问控制系统中最基础最重要的问题, 其主要目标是判断主体能否最终获得对客体的访问权限。对于传统访问控制模型, 主要通过遍历控制系统的可达状态空间进行安全性分析。这种方法也适用于 UCON。

5.1 全局可达空间的构造

文献[11]已经证明如果主客体属性的值域是有限的, 且没有新的主客体的产生, 则 $UCON_A$ (仅仅包含授权因素的 UCON 模型) 安全性问题是可判定的。这个问题比较容易理解, 因为这 2 种约束保证了状态空间的有穷性, 可以通过穷举状态空间的方法来实现安全性分析。关于状态空间的构造, 文献[11]只给出了一个抽象的概念, 即从初始状态出发, 通过执行各种访问控制请求计算下一步状态, 循环执行从而计算出所有的可达空间。现在的问题是如何构造一个进程, 使在 PAT 中该进程的可达空间就是整个 UCON 的可达空间。与文献[11]一样, 主要考虑 $UCON_A$ 模型上的可达性分析。

构造进程代数的关键在于必须能够表达任意时刻任意主体对任意客体发送任意的访问请求。在给出构造方法之前, 先做一些假设: 主体数目为 m , 客体数目为 n , 访问权限只有 r 一种。具体的进程构造过程如下。

Procedure 1 面向可达空间计算的进程代

数的构造。

Step1 构造初始化进程定义进程 $P() = initialization\{\dots\} \rightarrow Skip$ 用于对变量进行初始化。

Step2 依据 Type-1 并发构造某个时刻主体对客体的单次访问流程, 记为 $Proc_1(i, j, r)$ 。

Step3 依据 Type-2 并发, 在 $Proc_1(i, j, r)$ 的基础上构造上次访问结束后主体可以继续对同一客体发出访问请求的流程, 记为 $Proc_2(i, j, r)$ 。

Step4 依据 Type-3 并发, 在 $Proc_2(i, j, r)$ 的基础上构造在访问过程中主体可以随时发出对客体的再一次相同的访问请求, 记为 $Proc_3(i, j, r)$ 。

Step5 构造同一主体对 n 个客体的交互访问。定义进程 $Proc_4(i) = Proc_3(i, 0, r) \parallel \dots \parallel Proc_3(i, n-1, r)$ 。

Step6 构造任一主体对任一客体的交互访问。定义进程 $Proc_5() = Proc_4(0) \parallel \dots \parallel Proc_4(m-1)$ 。

Step7 定义进程 $Proc() = P(); Proc_5()$ 。

定理 1 Procedure 1 构造出的进程 $Proc()$ 的可达空间为整个访问控制系统的可达空间。

5.2 分析

安全性分析主要讨论访问许可泄露问题, 其主要目标在于对给定的主体、客体和权限, 该主体能否最终获得对该客体的访问权限。在 UCON 中, 主体对客体的访问权限的许可完全取决于当前状态下主体与客体属性值, 如在实例 11 中如果读者 1 拥有的 *expense* 为 3, 则其不能阅读书 *b0*, 如果增加到 6, 则可以阅读书 *b0*。UCON 的主要优点在于访问许可的执行会引起主客体属性值的变化, 而正是这种变化给 UCON 模型的安全性分析带来了很大的困难。文献[19]将安全性分成简单安全性、简单可用性、约束安全性、活性与包含性 5 类, 本节将给出 PAT 框架下 5 种属性的分析方法。

简单安全性是指是否存在一个可达状态, 使该状态下指定的主体能够获取指定客体的访问权限。假设谓词 $P1 \&\& \dots \&\& Pn$ 用来决策指定主体能否获取指定客体的访问权限, 可通过可达性检测完成安全性分析: 断言 `#assert reaches P1&&...&&Pn` 为真, 则说明权限被许可, 否则说明权限未被许可。

简单可用性是指在每个可达状态下,指定的主体是否能够获取指定的客体的访问权限。

假设谓词 $P1 \&\& \dots \&\& Pn$ 用来决策指定主体能否获取指定客体的访问权限,可通过验证 LTL 属性 $\Box(P1 \&\& \dots \&\& P2)$ 来实现简单可用性的验证:此属性成立说明在每个可达状态下谓词 $P1 \&\& \dots \&\& Pn$ 为真,即在每个可达状态下访问得到了许可,反之说明简单可用性不成立。

限界安全性是指在每个可达状态下,对于给定的客体,可获取该客体访问权限的主体是否在指定的主体范围内。对于主体 i , 客体 j , 访问访问权限 r , 定义谓词 $\text{Predicate}.i.j.r$ 表示访问权限的决策项。不失一般性,假设有 m 个主体,编号分别为 $0, \dots, m-1$, 1 个客体,编号为 0 , 一种访问权限,编号为 0 , 指定的主体范围是 $\{0, \dots, k\}$, 指定的客体是 0 , 限界安全性可以用过验证 LTL 属性 $\Box!(\text{Predicate}.(k+1).0.0 \parallel \dots \parallel \text{Predicate}.(m-1).0.0)$ 来实现。该属性为真说明在任一可达状态下谓词 $\text{Predicate}.(k+1).0.0$ 到 $\text{Predicate}.(m-1).0.0$ 均为假,即主体 $k+1$ 到主体 $m-1$ 都没有获取相应的权限,反之限界安全性不成立。

活性是指在任一可达状态下至少有一个主体能够获得对某个客体的访问权限。活性的验证等于验证是否存在一个可达状态,使得没有一个用于访问决策判断的谓词为真,因此可以通过可达性检测来验证活性。假设有 m 个主体, n 个客体, k 个权限。对于主体 i , 客体 j , 访问访问权限 r , 定义谓词 $\text{Predicate}.i.j.r$ 表示访问权限的决策项,断言 $\#assert \text{ reaches } !(\bigvee_{i \in \{0, \dots, m-1\}, j \in \{0, \dots, n-1\}, r \in \{0, \dots, k-1\}} \text{Predicate}.i.j.r)$ 为真,则说明存在一个可达状态,使没有一个决策谓词为真,即活性不成立,反之成立。

包含性是指在任一可达状态下对某个客体拥有某种访问权限的主体是否也拥有对另外一个客体的另外一种权限。不失一般性,假设有 2 个客体,编号分别为 0 和 1 , 有 2 种权限,编号分别为 0 和 1 。设计断言 $\#assert \text{ reaches } !(\bigwedge_{i \in \{0, \dots, m-1\}} \text{Predicate}.i.0.0 \rightarrow \text{Predicate}.i.1.1)$ 。该断言为真,则说明存在一个可达状态,该状态下 $\text{Predicate}.i.0.0 \rightarrow \text{Predicate}.i.0.1$ 为假,即对指定的客体拥有某种访问权限的主体没有拥有另外一种

权限。断言为假,说明包含性成立。

5.3 完全性与相容性

完全性与相容性是访问控制系统中另外 2 个重要的基本问题。关于 UCON 的完全性和冲突性目前还未见任何相关的工作,文献[10]也将这 2 个问题列为 UCON 未来需要解决的问题,本节将对如何利用 PAT 完成 2 个属性的验证展开系统的讨论。

完全性是指对于主体发出的任意请求,系统必须给出准确的回应,要么同意,要么许可。UCON 采取授权、义务和条件 3 种因素共同决策访问的许可,当 3 个因素都成立时,访问得到许可,否则拒绝。对 UCON 而言,只有一种情况会造成控制策略的不完全,即遗漏了某个主体对某个客体发出某种权限的访问请求的 TCSP#规范。因此 UCON 的不完全性可以归结为:对于任意的主体 i , 任意的客体 j , 任意的权限 r , 是否进行了相应的 TCSP#规范描述。可以通过验证 LTL 性质 $\Box! \text{tryaccess}.i.j.r$ 达到验证完全性的目的:如果存在主体 i , 客体 j , 权限 r 使 $\Box! \text{tryaccess}.i.j.r$ 为真,则说明 UCON 是不完全的,否则是完全的。

不相容性是指对于某个访问,存在不少于 2 个访问控制决策,依据其中的一个规则访问得到许可,依据另外一条规则,则访问被拒绝。2 个访问控制决策是相容的,意味着对于任意一个访问请求,决策的结果应该是一致的,换句话讲从决策的结果判断,2 个进程应该是等价的,为此建立如下相容性检测过程。

Procedure 2 相容性检测。

Step1 依据 Type-1 并发对第一个访问控制规则建立 TCSP#规范,记为 $\text{Procedure}_1(i,j,r)$ 。

Step2 依据 Type-1 并发对第二个访问控制规则建立 TCSP#规范,记为 $\text{Procedure}_2(i,j,r)$ 。

Step3 建立进程 $P()$ 用以对变量进行初始化。

Step4 建立进程 $\text{Proc}_1 = P(); \text{Procedure}_1(i,j,r)$ 。

Step5 建立进程 $\text{Proc}_2 = P(); \text{Procedure}_2(i,j,r)$ 。

Step6 建立断言 $\#assert \text{ Proc}_1 \text{ refines } \text{Proc}_2$ 。

Step7 建立断言 $\#assert \text{ Proc}_2 \text{ refines } \text{Proc}_1$ 。

Step8 验证 2 个断言是否成立:都成立说明 2 个规则相容,否则说明不相容。

实例 13 在一个医院的信息管系统中有 3 个

用户组: 医生、护士、主治人员, 其中主治人员包括主治医师和主治护士 2 个子角色。访问控制策略主要有 3 个^[20]。

Policy1 医生不能对病人进行查房。

Policy2 主治医生可以对病人进行查房。

Policy3 医生可以查阅任何病人的医疗记录。

现在验证 Policy 1 与 Policy 2 的相容性。初始假设: 2 个医生, 编号分别为 0 和 1; 2 个护士, 编号分别为 0 和 1; 2 个病人, 编号分别为 0 和 1; 0 号病人的主治医生是 0, 主治护士是 0, 1 号病人的主治医生是 1, 主治护士是 1。

定义数组 $S[2]$ 表示主体属性: $s[i]=1$, 表示 i 是医生。定义数组 $o[2]$ 表示客体属性: $o[i]=j$, 表示 j 是 i 号病人的主治医生。

Step1 依据 Type-1 并发对 Rule1 建立 TCSP#规范。

```
Procedure_1( $i, j, r$ )=tryaccess.i.j.r->if( $s[i]==1$ )
{denyaccess.i.j.r->Skip} else {permitaccess.i.j.r->
doaccess.i.j.r->end.i.j.r->Skip}
```

Step2 依据 Type-1 并发对 Rule2 建立 TCSP#规范。

```
Procedure_2( $i, j, r$ )=tryaccess.i.j.r->if( $o[j]==i$ )
{permitaccess.i.j.r->doaccess.i.j.r->end.i.j.r->Skip}
else {denyaccess.i.j.r->Skip}
```

Step3 建立进程 $P()$ 用以对变量进行初始化。

```
P()=initialization{s[0]=1;s[1]=1;o[0]=0;o[1]=1}
->Skip
```

Step4 建立进程 $Proc_1 = P()$; Procedure_1(i, j, r)。

Step5 建立进程 $Proc_2 = P()$; Procedure_2(i, j, r)。

Step6 建立断言 #assert Proc_1 refines Proc_2。

Step7 建立断言 #assert Proc_2 refines Proc_1。

Step8 验证 2 个断言是否成立, 都成立说明 2 个规则相容, 否则说明不相容。

利用 PAT 工具, 验证发现这 2 个断言都不成立, 因此 Policy 1 和 Policy 2 是不相容的。

6 会议论文评审系统

考察一个会议论文评审系统^[21]。该系统包含的对象是评审专家与准备评审的论文。评审系统

遵循如下访问控制策略。

Policy1 程序委员会主席为每个评审专家分配评审的论文, 每篇论文的评审专家为 2 人。

Policy2 对于任意一篇论文 p , 如果在当前系统中 p 没有指派给评审专家 a 评审, 则 a 可以查看评审专家 b 对论文的评审意见。

Policy3 如果论文 p 被同时指派给了评审专家 a 与 b 评审, 只有当 a 完成审稿后才能查看 b 对论文的评审意见。

Policy4 如果论文 p 已经被指派给评审专家 a 评审, 在没有完成审稿之前 a 随时可以终止本次审稿。

Policy5 如果论文 p 已经被指派给评审专家 a 评审, 在 a 没有完成审稿之前, 程序委员会主席可随时撤销 a 对 p 的审稿权。

制定这些控制策略的主要目的是阻止评审专家的意见受其他评审专家的影响。Policy 1 可以设计一个初始化进程来完成。Policy 2 和 Policy 3 没有改变任何主体与客体属性, 且必须满足某个条件才能查看其他评审专家提交的评审意见, 因此对应的策略控制核心模型为 preA0。Policy 4 改变了主体的属性, 且发生在访问请求得到许可之后、执行之前, 因此对应的策略控制执行模型为 preA1。Policy 5 改变了客体的属性, 且发生在访问请求得到许可之后、执行之前, 因此对应的策略控制执行模型为 preA1。具体分析的安全性如下。

1) 简单安全性: 某个评审专家最终能否获取某篇论文的评审权。

2) 简单可用性: 任意时刻某个评审专家能否获取某篇论文的评审权。

3) 限界安全性: 任意时刻获取某篇论文评审权的主体是否在指定的主体范围内。

4) 活性: 在任意可达状态下至少有一个评审专家能够获得对某篇论文的访问权限。

5) 包含性: 任意时刻某个评审专家拥有对某篇论文的评审权是否也同时拥有拒绝某篇论文评审的权利。

6) 完全性: 任意时刻某个评审专家对某篇论文发出任何访问请求都会得到响应。

7) 相容性: 任意时刻某个评审专家对某篇论

表 1 不同评审专家和论文数量情况下的实验结果

规模	性能参数	简单安全性	简单可用性	限界安全性	活性	包含性	完全性	相容性	公平性
2,3	Visited States	51	7	7	61	11	667	2	1 855
	Total Transitions	80	70	70	630	112	8243	1	16 711
	Time/s	0.126 849 7	0.080 961 3	0.009 358	0.050 039 1	0.009 759 5	0.276 498	0.005 870 8	0.442 401 9
	Memory/KB	9 320.76	9 112.904	9 135.736	12 821.24	9 496.912	21 611.824	32 660.28	14 762.848
4,4	Visited States	232	473	473	12 297	711	90 130	2	189 955
	Total Transitions	523	10 961	10 961	285 347	16 345	176 152	1	1 079 567
	Time/s	0.088 303 4	0.499 98	0.493 146 2	12.169 622 8	0.624 068 8	3.289 180 3	0.004 463 5	32.791 591 8
	Memory/KB	9 330.84	25 630.4	27 264.616	444 924.864	37 957.12	103 927.336	32 653.784	266 899.36
8,6	Visited States	838	235	235	N/A	3 412	2 027 340	2	N/A
	Total Transitions	2 461	328	328	N/A	5 084	4 019 980	1	N/A
	Time/s	0.139 493 4	0.083 753 6	0.031 495 6	N/A	0.269 614 8	104.038 858 3	0.052 275 4	N/A
	Memory/KB	11 348.336	12 065.008	12 517.896	N/A	20 573.32	2 178 426.368	32 793.048	N/A

文发出任何访问请求，不会出现既得到许可又得到拒绝的响应。

除了上述简单的安全属性外，还需考察公平性：评审专家对论文的评审不受其他评审意见的影响，即任意评审专家在提交某篇论文的评审意见前没有看到其他评审专家对该论文的评审的意见。

现在通过实验结果来评估算法的性能。所有的实验均基于 PAT3.5.1 完成，实验环境为：Windows 8 操作系统，2.83 GHz Intel Q9550 CPU，4 GB 内存。运行时间的上限设置为 60 min。表 1 收集了不同评审专家和论文数量情况下的实验结果。在表 1 中，用一个二元组 (i, j) 表示规模，其中 i 表示评审专家的人数， j 表示需要评审的论文数。

从表 1 的数据可以看出，除了活性和公平性，其他属性的验证时间是完全可以接受的，内存空间的使用是非常有限的。因此从时空效率的角度而言，使用 PAT 来完成 UCON 的安全性分析是完全可行的。

7 结束语

本文依托于先进的模型检测工具 PAT，利用 PAT 的建模语言 TCSP#建立了所有 UCON 核模型的规范，以及一般化 UCON 模型的规范机制，对复杂的使用会话场景，如并发、时间控制与非确

定性也建立了一套规范机制。可达空间的计算是分析 UCON 安全性的基础，本文提出了一种进程代数构造机制，使该进程代数可以刻画任意时刻任意主体对任意客体放出任意访问请求，因此该进程代数的状态空间就是访问控制系统的可达空间。基于可达空间，给出了安全性分析技术，基于进程代数的等价关系，提出了控制策略冲突性检测方法。总而言之，在 PAT 下从 UCON 的规约到安全性分析，提出了一套系统的解决方案。

在第 6 节的实例研究中，发现随着主客体数量的增长，可达空间的增长非常快，这为大规模 UCON 的验证带来了挑战，未来的主要工作是如何应对状态空间增长过快的问題。

参考文献：

- [1] PFLEEGER C P, PFLEEGER S L. Security in computing[M]. Prentice Hall Professional Technical Reference, 2002.
- [2] SANDHU R S, COYNE E J, FEINSTEIN H L, et al. Role-based access control models[J]. Computer, 1996, 29(2): 38-47.
- [3] FERRAILOLO D F, SANDHU R, GAVRILA S, et al. Proposed NIST standard for role-based access control[J]. ACM Transactions on Information and System Security(TISSEC), 2001,4(3): 224-274.
- [4] PARK J, SANDHU R. The UCON ABC usage control model[J]. ACM Transactions on Information and System Security (TISSEC), 2004, 7(1): 128-174.
- [5] ZHANG X, PARISI-PRESICCE F, SANDHU R, et al. Formal model and policy specification of usage control[J]. ACM Transactions on Information and System Security (TISSEC), 2005, 8(4): 351-387.

- [6] ZHANG X. Formal model and analysis of usage control[D]. Fairfax County, Virginia: George Mason University, 2006.
- [7] JANICKE H, CAU A, ZEDAN H. A note on the formalisation of UCON[C]//The 12th ACM Symposium on Access Control Models and Technologies. c2007: 163-168.
- [8] MARTINELLI F. A model for usage control in GRID systems[C]//The Third International Conference on Security and Privacy in Communications Networks and the Workshops. c2007: 520-520
- [9] JAGADEESAN R, MARRERO W, PITCHER C, et al. Timed constraint programming: a declarative approach to usage control[C]//The 7th ACM SIGPLAN International Conference on Principles and Practice of Declarative Programming. c2005: 164-175.
- [10] LAZOUSKI A, MARTINELLI F, MORI P. Usage control in computer security: a survey[J]. Computer Science Review, 2010, 4(2): 81-99.
- [11] ZHANG X, SANDHU R, PARISI-PRESICCE F. Safety analysis of usage control authorization models[C]//The 2006 ACM Symposium on Information, Computer and Communications Security. c2006, 6: 243-254.
- [12] RAJKUMAR P V, GHOSH S K, DASGUPTA P. Concurrent usage control implementation verification using the SPIN model checker[C]//The Third International, CNSA 2010, Chennai, India. Berlin Heidelberg Springer. c2010: 214-223.
- [13] HOLZMANN G J. The model checker SPIN[J]. IEEE Transactions on Software Engineering, 1997, 23(5): 279-295.
- [14] SUN J, LIU Y, DONG J S, et al. PAT: towards flexible verification under fairness[C]//The 21st International Conference on Computer Aided Verification. c2009: 709-714.
- [15] SUN J, LIU Y, DONG J S, et al. Modeling and verifying hierarchical real-time systems using stateful timed csp[J]. ACM Transactions on Software Engineering and Methodology(TOSEM), 2013, 22(1):139-176.
- [16] VARDI M Y. An automata-theoretic approach to linear temporal logic[M]//Logics for concurrency. Berlin Heidelberg: Springer, 1996.238-266.
- [17] MCLEAN J. A comment on the 'basic security theorem' of bell and LaPadula[J]. Information Processing Letters, 1985, 20(2): 67-70.
- [18] KATT B, HAFNER M, ZHANG X. A usage control policy specification with petri nets[C]//The 5th International Conference on Collaborative Computing: Networking, Applications and Worksharing. c2009: 1-8.
- [19] LI N, TRIPUNITARA M V. Security analysis in role-based access control[J]. ACM Transactions on Information and System Security (TISSEC), 2006, 9(4): 391-420.
- [20] ADI K, BOUZIDA Y, HATTAK I, et al. Typing for conflict detection in access control policies[M]//E-technologies: Innovation in An Open World. Berlin Heidelberg: Springer, 2009: 212-226.
- [21] ZHANG N, RYAN M, GUELEV D P. Synthesising verified access control systems through model checking[J]. Journal of Computer Security, 2008, 16(1): 1-61.
- [22] LAMPORT L. The temporal logic of actions[J]. ACM Transactions on Programming Languages and Systems (TOPLAS), 1994, 16(3): 872-923.

作者简介:



周从华(1978-), 男, 江苏盐城人, 博士, 江苏大学副教授, 主要研究方向为访问控制、形式化方法。



陈伟鹤(1974-), 男, 江苏丹阳人, 博士, 江苏大学副教授, 主要研究方向为访问控制、数据挖掘。



刘志锋(1981-), 男, 江苏无锡人, 博士, 江苏大学副教授, 主要研究方向为访问控制、形式化方法。