Datalog逻辑程序调用语义及其应用研究

钟 勇[1,2] 郭伟刚[2] 钟昌乐[2] 刘凤玉[1,3] 李 宁[4]

(南京理工大学计算机科学与技术博士后流动站 南京 210094)[1]

(佛山科学技术学院信息与教育技术中心 佛山 528000)[2]

(南京理工大学计算机科学与技术学院 南京 210094)[3](佛山市泰达安全生产事务有限公司 佛山 528000)[4]

摘 要 提出Datalog逻辑程序调用语义和调用谓词,说明包含程序调用谓词的可更新U-Datalog程序的操作语义及其固定点语义。提出在有限分层调用情况下U-Datalog程序的通用评价(evaluation)算法。最后对Datalog程序调用语义在数字版权语言中的应用做了说明并给出示例。

关键词 Datalog语言,逻辑程序,程序调用,数字版权保护

中图法分类号 TP309 **文献标识码** A

Research on Call Semantic of Datalog Logic Program and its Application

ZHONG Yong[1,2] GUO Wei-gang[2] ZHONG Chang-le[2] LIU Feng-yu[1,3] LI Ning[4]

(Postdoctoral Mobile on Computer Application, Nanjing University of Science and Technology, Nanjing 210094, China)[1]

(Information and Educational Technology Center, Foshan University, Foshan 528000, China)[2]

(School of Computer Science and Technology, Nanjing University of Science and Technology, Nanjing 210094, China)[3]

(Taida Safety Produce Routine Co., LTD, Foshan 528000, China)[4]

Abstract The paper firstly gave the call semantic and call predicates of datalog logic programs, and discussed the operational and fixpoint semantics of U-datalog programs with call predicates. Then the algorithms that evaluate U-Datalog programs with call predicates under definite stratified calls were presented. Finally, applications of the call semantic of datalog programs in digital rights management were explained and demonstrated.

Keywords Datalog language, Logic program, Program call, Digital rights management(DRM)

1 引言

Datalog逻辑语言由于在表达力、灵活性和语义完整性上具有优势,因此在访问控制的研究中受到重视。如Jajodia等[1]提出了基于分层Datalog能表达多策略的授权框架FAF,Ninghui等提出了基于Datalog的可信管理代理逻辑[2]和基于限制Datalogc的可信管理语言[3],Bertino等在C-Datalog基础上提出了通用逻辑框架[4]等。使用控制(Usage Control)[5]和数字版权保护(digital rights management, DRM)技术由于需要表达主客体属性的可变性(mutability)和授权决策的持续性(continuity),因此要求授权语言具有可更新能力。U-Datalog[6]语言如我们在前期研究中提出的建立在Active-U-Datalog[7]语义基础上的LUC使用控制语言[8]和正在研制的数字版权保护脚本语言LucScript[9],作为具有延迟更新(deferred update)能力的Datalog语言直接扩展,在这些访问控制技术中具有较好的可应用性。

数字版权保护技术是数字网络环境下数字内容交易产权保护的重要技术。在DRM系统中,许可证(license)是重要的组成部分,许可证中包含数字内容的使用权限、密钥、元数据等关键信息。权利描述语言REL用来构造许可证,描述数字内容或服务的使用权利,是DRM系统中的重要研究课题。近年来,由于逻辑语言在表达力、灵活性和语义完整性上的优势,基于逻辑的REL语言的研究受到重视,典型的有License Script[10]语言、Gunter[11]等提出的抽象权利描述模型和语言以及Pucella等[12]在时态义务逻辑(deontic logic)的基础上对Gunter语言的扩展等。其中,LicenseScript语言通过使用Prolog程序作为许可证的表达语言,是迄今为止最具表达力和灵活性的REL语言。但LicenseScript语言使用Prolog语言作为许可证的权利表达语言,而许可证的管理规则使用多集重写方式(multiset rewriting)实现,使整体语言分为不同的两部分,不具有逻辑上的统一性,增加了语言上的复杂性。而且多集改写规则使管理规则必须固化在解释器中,从而影响了通用解释器的发展。

与LicenseScript语言相比,LucScript的许可证规则和许

到稿日期:2009-02-18 返修日期:2009-05-15 本文受中国博士后科学基金(20070421015),广东省自然科学基金(8452800001001086),江苏省博士后科研资助计划(0801045B),佛山市科技发展专项资金(200701002),国家自然科学基金(60673127),国家863计划(2007AA01Z404)资助。

钟 勇(1970—),男,博士后,副教授,主要研究方向为信息安全、数据库技术、数字版权保护技术等,E-mail:zhongyong1970@126.com;郭伟刚(1966—),男,副教授,主要研究方向为信息安全、数据库技术;钟昌乐(1974—),男,讲师,主要研究方向为信息安全、数字版权保护技术等;刘凤玉(1943—),女,教授,博士生导师,主要研究方向为网络性能保护和信息安全;李 宁(1963—),男,高工,主要研究方向为信息安全、生产安全。

可证管理规则均使用 Active-U-Datalog 语言，具有统一的逻辑基础。在该脚本语言中，许可证存放在许可池中，由许可池统一管理。许可证程序和许可池对许可证的管理程序均为独立逻辑程序，许可池与许可证之间以及许可证之间通过逻辑程序的调用语义来实现交互、更新和管理的需要。式(1)给出许可池管理规则：

$$delete\_lic() \leftarrow lic(d,\Delta), call(\Delta,(type('update')), \neg lic(d,\Delta)) \quad (1)$$

该管理规则删除所有类型是'*update*'的许可证，其中 $lic(d,\Delta)$ 是许可池中许可证的外延谓词，d 是数字媒体标识，Δ 是许可证程序标识，$call(\Delta,(type('update'))$ 是程序调用谓词，该谓词在许可证程序 Δ 中执行事务？$\neg(type('update'))$，以确定许可证类型是否是'*update*'，如果是则使用更新谓词 $\neg lic(d,\Delta)$ 删除该许可证。

由于管理规则与许可证规则相同，LucScript 许可池使用与数字媒体许可证结构相同的管理许可证来管理，不需要专门的解释器和固化在解释器中，使解释器具有更好的灵活性和通用性。

Datalog 程序之间的调用和交互问题是迄今 Datalog 逻辑程序中尚未涉及的问题。本文首先提出 Datalog 程序调用语义及其调用谓词形式，并说明包含程序调用谓词的 U-Datalog 程序的操作语义及其固定点(fixpoint)语义，然后给出在有限分层调用情况下 U-Datalog 程序的通用评价算法，最后给出 Datalog 程序调用语义在数字版权语言中的应用示例。

2 Datalog 程序调用语义和评价(evaluation)

既然基本 Datalog 程序(pure Datalog)可看作是 U-Datalog 的特例，那么本文就使用 U-Datalog 程序作为程序调用语义的示例。

2.1 U-Datalog 语言

U-Datalog 语言在语法上可看作是限制逻辑程序(CLP)的特殊实例，更新原子 $\pm p(t_1,t_2,\cdots,t_n)$ 作为 Datalog 规则中的限制，称除更新原子外的其他 Datalog 原子为查询原子(query atoms)。U-Datalog 程序的执行分为标记(marking)和更新(update)两阶段，标记阶段绑定事务变量和得到更新集，更新阶段再执行更新集。

定义 1(事务)[6] 事务是不含规则头且具如下形式的规则：

$$U_1,\cdots,U_k,L_1,\cdots,L_m \quad (2)$$

其中，$U_i(0\leqslant i\leqslant k)$ 是任意更新原子，$L_i(0\leqslant i\leqslant m)$ 是任意查询原子。对事务 T，称 T 的查询原子集为 T 的查询事务，在标记阶段得到的更新集为 T 的更新事务[1]。

2.2 程序调用语义

设 T 代表事务 $(U_1,\cdots,U_k,L_1,\cdots,L_m)$，$\Delta=IDB\cup EDB$ 表示 Datalog 程序，其中 IDB 表示内涵数据库，EDB 表示外延数据库。调用谓词包括表 1 中的谓词。

表 1 调用谓词

调用谓词符号表示	规则体包含谓词
?(T)	在调用程序中执行 T 的查询事务
?+(T)	在调用程序中执行 T 的查询和更新事务
Δ?(T)	在程序 Δ 中执行 T 的查询事务
Δ?+(T)	在 Δ 中执行 T 的查询和更新事务
$\Delta_1+\Delta_2$?(T)	在 Δ_1 中执行 T 的查询事务，在 Δ_2 中执行 T 的更新事务

表 1 中的调用谓词以符号形式表示，在实际程序中可以相应的谓词形式代替。如 $\Delta?(T)$ 可表示成 $call(\Delta,T)$ 等谓词形式。调用谓词在查询以及更新事务执行成功时返回 True，否则返回 False。称包含上述调用谓词的 U-Datalog 程序为嵌套 U-Datalog 程序，表示为 U-Datalog^r。为清晰起见，将事务中包含调用谓词的查询原子单独表示出来，因而 T 也可表示成 $(U_1,\cdots,U_k,C_1,\cdots,C_n,L_1,\cdots,L_m)$，其中 $U_i(0\leqslant i\leqslant k)$ 是任意更新原子，$C_i(0\leqslant i\leqslant n)$ 是任意包含调用谓词的查询原子，简称包含调用谓词的查询原子为调用原子，$L_i(0\leqslant i\leqslant m)$ 是除调用原子外的任意查询原子。

定义 2(操作语义) 设 DB 是 U-Datalog^r 程序，定义 DB 的操作语义 $O(DB)$ 为：

$$O(DB)=\{p(\tilde{X})\leftarrow\tilde{c},\tilde{u}\mid p(\tilde{X})\rightarrow^{\times}_{DB}\tilde{c},\tilde{u}\} \quad (3)$$

其中，$\tilde{c}$ 是调用原子合取式，$\tilde{u}$ 是更新原子合取式，$p(\tilde{X})$ 是除更新和调用原子外的任意原子，$p(\tilde{X})\rightarrow^{\times}_{DB}\tilde{c},\tilde{u}$ 表示 $p(\tilde{X})\leftarrow\tilde{c},\tilde{u}$ 能从 DB 中推导(derivations)出。

那么，定义 U-Datalog^r 的 Herbrand 基 β 由如下形式的文字组成：$L\leftarrow\tilde{c},\tilde{u}$，其中 L 是除更新和调用原子外的任意原子，$\tilde{c}$ 是调用原子的合取式，$\tilde{u}$ 是更新原子的合取式。在定义 U-Datalog^r 程序的固定点语义之前，类似文献[6]，先定义如下直接操作算子。

定义 3(直接操作算子) 设 DB 是 U-Datalog^r 程序，I 是一个解释(interpretation)，则定义 T_{DB} 算子 $(2^{\beta}\rightarrow 2^{\beta})$ 如下：

$$\begin{aligned}T_{DB}(I)=\{\ & p(\tilde{X})\leftarrow\tilde{c}',\tilde{u}' \mid \exists \text{重命名规则}\\ & p(\tilde{t})\leftarrow\tilde{c},\tilde{u},L_1(\tilde{Y}_1),\cdots,L_n(\tilde{Y}_n)\in DB\\ & \forall i=1,\cdots,n,\exists\theta(L_i(\tilde{X}_i)\leftarrow\tilde{c}_i,\tilde{u}_i)\in I,1\leqslant i\leqslant n\\ & \text{且在不共享变量的情况下有 } p(\tilde{X})=\theta p(\tilde{t})\\ & \tilde{c}'\equiv(\wedge_i\tilde{c}_i\wedge\tilde{c})\theta\\ & \tilde{u}'\equiv(\wedge_i\tilde{u}_i\wedge\tilde{u})\theta\\ \}\end{aligned} \quad (4)$$

定义 4(固定点语义) 设 DB 是 U-Datalog^r 程序，定义 DB 的固定点语义 $Fix(DB)=T_{DB}\uparrow\omega$。

定理 1(操作语义和固定语义的同一性) 设 DB 是 U-Datalog^r 程序，则有 $O(DB)=Fix(DB)$。

文献[6]中说明了在不含调用原子的 U-Datalog 中以及在 CLP(Constraint Logic Programming)程序中该定理的正确性。如果将调用原子看作是一种特殊的限制类型，则类似的方法可证明定理 1 成立。

```
q(b);
t(a); //EDB
p(X)←q(X), ¬q(X);
r(X)←+t(X), p(X), ?(p(X));
k(a)←− t(a), ?(p(a));
s(X)←t(X), ?(r(X)) //IDB
```

图 1 U-datalogr 程序示例

例 1 设有 U-Datalog^r 程序，如图 1 所示，使用直接操作

[1] T 的更新事务包括 T 本身的更新原子和因执行事务以及程序调用引发的更新原子。

算子得到如下结果：

1) $T_0(DB)=\{q(b);t(a);k(a)\leftarrow -t(a),?(p(a))\}$；

2) $T_1(DB)=\{s(a)\leftarrow ?(r(a));p(b)\leftarrow -q(b)\}\cup T_0(DB)$；

3) $T_2(DB)=\{r(b)\leftarrow +t(b),-q(b),?(p(b))\}\cup T_1(DB)$；

4) $T_3(DB)=T_2(DB)=Fix(DB)$。

按照定理1，则有 $O(DB)=Fix(DB)=T_3(DB)$。

定义5(绑定算子 $b_{|T}$ 和替换算子 $eqn(\theta)$)[7] 给定绑定集 b 和事务 T，定义绑定算子 $b_{|T}$：

$$b_{|T}=\{(X=t)\in b|\ X\text{在}T\text{中出现}\} \tag{5}$$

给定替换 $\theta=\{V_1\leftarrow t_1,\cdots,V_n\leftarrow t_n\}$，定义替换算子 $eqn(\theta)$：

$$eqn(\theta)=\{V_1=t_1,\cdots,V_n=t_n\} \tag{6}$$

定义6(直接更新集和直接调用集) 给定 U-Datalog^r 程序 DB 及其固定点 $Fix(DB)$、事务 $T=\tilde{u}_0,\tilde{c}_0,L_1,\cdots,L_n$，定义在固定点 $Fix(DB)$ 语义下 T 在 DB 中的直接更新集 $\hat{Set\_u}(T,DB,Fix(DB))$ 如下：

$$\begin{aligned}&\hat{Set\_u}(T,DB,Fix(DB))=\{\langle b,\tilde{u}\rangle|^{2)}\\&\quad A_i\leftarrow\tilde{c}_i,\tilde{u}_i\in Fix(DB),1\leqslant i\leqslant n\\&\quad\theta=mgu((L_1,\cdots,L_n),(A_1,\cdots,A_n))\\&\quad b=eqn(\theta)_{|T}\\&\quad\tilde{u}=(\tilde{u}_0,\tilde{u}_1,\cdots,\tilde{u}_n)\theta\\&\}\end{aligned} \tag{7}$$

定义在固定点 $Fix(DB)$ 语义下 T 在 DB 中的直接调用集 $\hat{Set\_c}(T,DB,Fix(DB))$ 如下：

$$\begin{aligned}&\hat{Set\_c}(T,DB,Fix(DB))=\{\langle b,\tilde{c}\rangle|\\&\quad A_i\leftarrow\tilde{c}_i,\tilde{u}_i\in Fix\ (DB),1\leqslant i\leqslant n\\&\quad\theta=mgu((L_1,\cdots,L_n),(A_1,\cdots,A_n))\\&\quad b=eqn(\theta)_{|T}\\&\quad\tilde{c}=(\tilde{c}_0,\tilde{c}_1,\cdots,\tilde{c}_n)\theta\\&\}\end{aligned} \tag{8}$$

对任意事务 T，如果将三元组 (T,DB,DB) 作为根结点，$\hat{Set\_c}(T,DB,Fix(DB))$ 中所调用的事务、查询执行程序、更新执行程序的三元组作为子结点，对子结点做类似处理，可建立如图2所示的调用树。对任意结点 $(T_{i,j},DB_{i,j},EB_{i,j})$，称 $T_{i,j}$ 为调用事务，$DB_{i,j}$ 为调用程序，$EB_{i,j}$ 为更新程序，并称 T 为主调用事务，DB 为主调用或主更新程序。

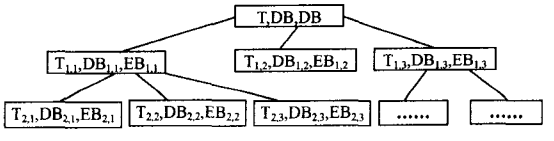

图2 事务 T 在程序 DB 中的调用树

定义7(分层调用树) 对任意事务 T 在 U-Datalog^r 程序 DB 的调用树 ct，如果 ct 中的任一结点 t_i 都不存在祖先结点 t_j，使得 $t_i\equiv t_j$，则称调用树 ct 是分层的，其中 $t_i\equiv t_j$ 表示两个结点内容完全相同。如果调用树的深度是有限的，则称 ct 是有限分层调用树。

不分层调用树中如果事务调用存在循环，将导致事务调用无法结束。调用树如果不是有限的，事务调用也无法结束，因而有限分层调用树才是可评价的。

定义8(成功调用树) 假定事务 T 在 U-Datalog^r 程序 DB 中的调用树是有限分层调用树 ct，对 ct 进行如下操作：

1) 从 ct 的叶结点开始，依后序遍历的顺序对 ct 结点进行事务成功与否的标记，标记方式如下：对结点 $(T_{i,j},DB_{i,j},EB_{i,j})$ 的任意子结点 (T_k,DB_k,EB_k)，如果该结点的事务执行结果($TRUE/FALSE$)使规则 $A_i\leftarrow\tilde{c}_i,\tilde{u}_i\in Fix(DB)$ 中的调用原子合取式 $\tilde{c}_i$ 的取值为 $FALSE$，则从 $Fix(DB_{i,j})$ 中删除该规则。对结点 $(T_{i,j},DB_{i,j},EB_{i,j})$ 及其子树进行规则删除后的固定点表示为 $Fix(DB_{i,j})/(T_{i,j},DB_{i,j},EB_{i,j})$，如果 $Fix(DB_{i,j})/(T_{i,j},DB_{i,j},EB_{i,j})|=T_{i,j}$ 则将结点 $(T_{i,j},DB_{i,j},EB_{i,j})$ 标记为 $TRUE$，否则标记为 $FALSE$。按后序遍历顺序标记 ct 直到根结点，并称操作结束后各个调用程序 $DB_{i,j}$ 经规则删除操作后得到的新固定点为 $DB_{i,j}$ 对事务 T 的成功固定点，表示为 $Fix(DB_{i,j},T)$。

2) 如果根结点的标记为 $True$，以 (T,DB,DB) 为根结点，$\hat{Set\_b}(T,DB,Fix(DB_{i,j},T))$ 中所调用的事务、查询执行程序、更新执行程序的三元组作为子结点的方式重新构造调用树。对子结点作类似处理，并且所有调用程序的固定点均使用成功固定点，称得到的调用树为 T 的成功调用树。

3) 如果根结点的标记为 $FALSE$，则定义 T 的成功调用树为 $NULL$。

定义9(更新集) 假定事务 T 在 U-Datalog^r 程序 DB 中存在不为 $NULL$ 的成功调用树 sct，对 sct 进行如下操作：从 sct 的叶结点开始，依后序遍历的顺序对 sct 结点的更新操作进行标记，标记方式如下：对结点 $(T_{i,j},DB_{i,j},EB_{i,j})$，首先让该结点的更新集 $Set\_U(T_{i,j},DB_{i,j},EB_{i,j})=\hat{Set\_u}(T_{i,j},DB_{i,j},Fix(DB_{i,j},T))$，对该结点的任意子结点 (T_k,DB_k,EB_k)，将 $Set\_U(T_k,DB_k,EB_k)$ 加入到 $Set\_U(T_{i,j},DB_{i,j},EB_{i,j})$。按后序遍历顺序标记 sct 直到根结点得到 $Set\_U(T,DB,DB)$，称 $Set\_U(T,DB,DB)$ 为事务 T 的更新集。

U-Datalog 更新语义包括强更新和弱更新[6]，强更新只允许插入(删除)现有 EDB 中不存在(存在)的原子。而弱更新的执行不需要知道现有 EDB 的前提状态。因而，在强更新环境中，对于插入原子 $+p(\tilde{t})$，当 $p(\tilde{t})\notin EDB$ 时更新才是一致的；对于删除原子 $-p(\tilde{t})$，$p(\tilde{t})\in EDB$ 时更新才是一致的。在弱更新环境中，不需要知道现有 EDB 的状态，只有当 $+p(\tilde{t})$ 和 $-p(\tilde{t})$ 同时存在更新事务中时，更新才是不一致的。对于 U-Datalog^r 程序的事务 T，其一致性是事务更新集 $Set\_U(T,DB,DB)$ 的一致性。

定义10(可观察元组)[7] 可观察元组是三元组 $\langle Ans,EDB,Res\rangle$，其中 Ans 是绑定集，EDB 是外延数据库，$Res\in\{Commit,Abort\}$，可观察元组集表示为 Oss。

定义11(更新合并函数) 给定更新集 U 及其相应更新程序的外延数据库 $EDB_0,EDB_1,\cdots,EDB_n(n\geqslant 0)$，定义更新合并函数如下：

$$incorp(U,EDB)=\bigcup_i in(U,EDB_i) \tag{9}$$

其中，$1\leqslant i\leqslant n$，$EDB=EDB_0\cup EDB_1\cup\cdots\cup EDB_n$，定义函数 in 如下：

2) 如果更新 $\tilde{u}$ 不在 DB 中执行而是在其他程序如 DB_i 中执行，那么绑定 b 也包括对 DB_i 中相应变量的绑定，下文同。

$in(U,EDB_i)=(EDB_i\backslash\{a|u\in\tilde{u}\wedge\langle b,\tilde{u}\rangle\in U\wedge\Delta_{EDB}(u)=EDB_i\wedge\Delta_{op}(u)=-a\})\cup\{a|u\in\tilde{u}\wedge\langle b,\tilde{u}\rangle\in U\wedge\Delta_{EDB}(u)=EDB_i\wedge\Delta_{op}(u)=+a\}\cup\{\emptyset u\in\tilde{u}\wedge\langle b,\tilde{u}\rangle\in U\wedge\Delta_{EDB}(u)=EDB_i\wedge u$ 中包含变量$\}$ (10)

其中，$\Delta_{EDB}(u)$操作映射 u 的更新程序的外延数据库，$\Delta_{op}(u)$操作映射 u 的更新原子，谓词 a 是基原子，不包含未绑定变量。当更新操作 u 包含未绑定的变量（ungroundness）时，无法决定更新操作的执行，只执行空操作。

定义 12（U-Datalog^r 程序语义） 给定 U-Datalog^r 程序 $DB_0,DB_1,\cdots,EDB_n(n\geqslant 0)$（外延数据库 $EDB_0,EDB_1,\cdots,EDB_n$）、事务 T，通过 Sem 函数定义事务 T 在主调用程序 DB_0 的语义如下：

$$Sem_{DB0}(T)=S(T)(\langle\emptyset,EDB,Commit\rangle) \quad (11)$$

其中，$EDB=EDB_0\cup EDB_1\cup\cdots\cup EDB_n$，定义函数 $S(Oss\rightarrow Oss)$如下：

1）如果 $\rho=Abort$，则

$S_{IDB}(T)(\langle\alpha,\varepsilon,\rho\rangle)=\langle\emptyset,\varepsilon,Abort\rangle$

2）如果 $Set_U(T,DB_0,DB_0)$是不一致的，则

$S_{IDB}(T)(\langle\alpha,\varepsilon,\rho\rangle)=\langle\emptyset,\varepsilon,Abort\rangle$

3）否则，

$S_{IDB}(T)(\langle\alpha,\varepsilon,\rho\rangle)=\langle Ans,incorp(Set\_U(T,DB_0,DB_0),EDB),Commit\rangle$，其中 $Ans=\{b|\langle b,\tilde{u}\rangle\in Set\_U(T,DB_0,DB_0)\}$。

定义 12 说明事务 T 的语义包括两个阶段：第一阶段得到 T 的更新集 $Set\_U(T,DB_0,DB_0)$；第二阶段将更新集 $Set_U(T,DB_0,DB_0)$合并到 EDB 中。

2.3 通用评价算法

U-Datalog^r 程序的评价算法包括标记阶段和更新阶段，标记阶段算法的主要目的是计算事务的更新集。标记算法既可以直接使用 T_{DB} 算子或 semi-navie 等算法及其变体[13]的优化措施，自底向上（bottom-up）得到固定点 $Fix(DB)$，再得到 T 的更新集；也可使用类似于 magic set[14] 或 WF（constraint memoing）[15]等自顶向下（top-down）算法的优化措施来得到 T 的更新集。通用算法 1 采用自底向上的方法并基于宽度优先算法来搜集所有更新。算法在发现事务 T 的调用树是非分层的或调用层级超过 Max_level 时返回错误，否则将各个执行成功调用事务的更新收集到 T 的更新集 $Set_U(T,DB_0,DB_0)$中。

算法 1 基于宽度优先的 U-Datalog^r 程序评价的标记阶段算法

输入：主调用事务 T，主调用程序 DB_0 和其他调用程序 $DB_1\cdots DB_n$，最大调用层级 Max_level

输出：执行成功与否标志（TRUE/FALSE），T 的更新集 $Set\_U(T,DB_0,DB_0)$

```
(1) level←0; //层级计数初始化
(2) return transaction(T,DB0,DB0) //子程序并返回执行是否成功标记
(3) BOOL function transaction (T,DB,EB)
(4) {
(5)   if (++level>Max_level) exit(1);//如果超过最大调用层级数则终止程
(6)   if (! stratified(T)) exit(1); //T 不是分层的，即存在祖先结点等于 T，则终止程序
(7)   if (DB 的固定点 Fix(DB)未计算过) {
(8)     执行直接操作算子 T_DB(I) until T_i(DB)= T_{i+1}(DB);//计算固定点
(9)     Fix(DB)←T_i(DB);}
(10)  if (Fix(DB)|≠T) { level← level−1;return FALSE;} //如果 Fix(DB)不能满足 T 则返回错误
(11)  利用 Fix(DB)绑定 T 的变量值后，计算 T 的直接调用集 Set_c^(T,DB);
(12)  for each tr in Set_c^(T,DB) {//T 的每个调用事务 tr
(13)    result=transaction(tr,tr.DB,tr.EB);//执行 tr,tr.DB 和 tr.EB 分别是 tr 的调用和更新程序
(14)    if (result==FALSE) //如果事务 tr 执行不成功
(15)      RemoveRuleFrom(Fix(DB),tr);//从固定点解释中将包含 tr 调用的规则删除
(16)    else
(17)      Success_set(T)←tr;//否则将 tr 加入到 T 的成功调用集 Success_set(T)中 }
(18)  if (Fix(DB)|≠T) { level← level−1;return FALSE;} //如果删除不成功事务后的 Fix(DB)不能满足 T 则返回错误
(19)  利用 Fix(DB)计算 T 的更新并保存到更新集 Set_U(T,DB,EB)
(20)  for each tr in Success_set(T) //对执行成功的调用事务 tr
(21)  Set_U(T,DB,EB)←Set_U(tr,tr.DB,tr.EB) //将事务 tr 的更新集加入到 T 的更新集中
(22)  level← level−1;
(23)  return TRUE;
(24) }
```

更新阶段算法目的是执行 T 的更新集 $Set\_U(T,DB,DB)$，由算法 1 可知，由于 T 的子事务中可能存在执行不成功的情况，T 的直接更新集 $Set_\hat{u}(T,DB)$并不一定是 $Set_U(T,DB,DB)$的子集。

算法 2 以弱更新为例说明更新阶段算法。弱更新环境不需要知道现有 EDB 的状态，而只需要更新集 $Set_U(T,DB,DB)$的一致性。用投影函数 $proj(Set_U(T,DB,DB),DB_i)$表示 T 的更新集中对程序 DB_i 的所有更新。则定义插入操作 $+p(\tilde{t})\in proj(Set_U(T,DB,DB),DB_i)$如下：

$$+p(\tilde{t})\rightarrow DB_i\Rightarrow\begin{cases}Abort & \exists -p(\tilde{t})\in proj(Set\_U(T,DB,DB),DB_i)\\ \emptyset & \text{if } \tilde{t} \text{ 中包含变量}(ungroundness)\\ DB_i\cup p(\tilde{t}) & \text{其它}\end{cases} \quad (12)$$

定义删除操作 $-p(\tilde{t})\in proj(Set_U(T,DB),DB_i)$如下：

$$-p(\tilde{t})\rightarrow DB_i\Rightarrow\begin{cases}Abort & \exists +p(\tilde{t})\in proj(Set\_U(T,DB,DB),DB_i)\\ \emptyset & \text{if } \tilde{t} \text{ 中包含变量}(ungroundness)\\ DB_i\backslash p(\tilde{t}) & \text{其它}\end{cases} \quad (13)$$

当更新操作 $\pm p(\tilde{t})$包含未绑定的变量时，无法决定更新操作的执行，只执行空操作；当发生不一致时更新操作失败，事务不进行提交。更新阶段算法步骤如算法 2 所示。

算法 2 U-Datalog^r 程序评价的更新阶段算法

输入：主调用事务 T，主调用程序 DB_0 和其他调用程序 $DB_1,\cdots,DB_n$，T 的更新集 $Set\_U(T,DB_0,DB_0)$

输出：执行成功与否标志（*TRUE/FALSE*），执行更新后的调用程序 $DB_0, DB_1, \cdots, DB_n$

(1) for each $(\gamma p(\tilde{t}), DB)$ in $Set\_U(T, DB_0, DB_0)$ { //$\gamma \in (+,-)$

(2) if $(\exists \delta p(\tilde{t}), (\delta p(\tilde{t}) \in proj(Set\_U(T, DB_0, DB_0), DB)) \wedge (\delta \neq \gamma)$

{ //更新不一致

(3) abort(T);//废除所有更新；

(4) return *FALSE*;}

(5) 按照式(12)或式(13)做相应的更新；}

(6) commit(T);//提交所有更新

(7) return *TRUE*; //更新成功

例 2　设有 U-Datalogr 程序 DB_0 和 DB_1，如图 3 所示，假设主调用程序为 DB_0，主调用事务 $T=(r(X), s(a))$。

$q(b)$;	
$t(a)$; //*EDB*	
$p(X) \leftarrow q(X), -q(X)$;	程序 DB_1
$r(X) \leftarrow +t(X), p(X), DB_1 + DB_0?(p(X))$;	
$k(X) \leftarrow -t(X), ?(p(X))$;	
$s(X) \leftarrow t(X), DB_1? +(r(X))$ //IDB	
$q(a)$;	
$q(b)$;	
$t(a)$; //*EDB*	程序 DB_0
$p(X) \leftarrow q(X), -q(X)$;	
$r(X) \leftarrow +t(X), p(X), ? +(p(X))$;	
$k(X) \leftarrow -t(X), DB_0? +(p(X))$; //IDB	

图 3　U-datalog^r 程序 DB_0 和 DB_1

(1) 首先使用直接操作算子，如例 1 所示的方法计算 DB_0 的固定点，得到如下结果：

$Fix(DB_0)=\{ q(b); t(a);$

$k(a) \leftarrow -t(a), ?(p(a));$

$s(a) \leftarrow DB_1? +(r(a));$

$p(b) \leftarrow -q(b);$

$r(b) \leftarrow +t(b), -p(b), DB_1 + DB_0?(p(b))\}$

由于 $Fix(DB_0)|=(r(b), s(a))$，绑定事务 T 的变量 $\{X=b\}$，并且有：$\hat{Set\_c}(T, DB_0)=\{DB_1? +(r(a)), DB_1 + DB_0?(p(b))\}$。

(2) 再类似计算 DB_1 的固定点：

$Fix(DB_1)=\{ q(a); q(b); t(a);$

$k(a) \leftarrow -t(a), DB_0? +(p(a));$

$p(a) \leftarrow -q(a);$

$p(b) \leftarrow -q(b);$

$r(a) \leftarrow +t(a), -p(a), ? +(p(a));$

$r(b) \leftarrow +t(b), -p(b), ? +(p(b))\}$

(2.1) 由 $Fix(DB_1)|=(r(a))$，可得到事务 $(DB_1? +(r(a)))$ 的调用集 $\hat{Set\_c}((DB_1? +(r(a))), DB_1)=\{? +(p(a))\}$；再由 $Fix(DB_1)|=p(a)$，得到调用事务 $? +(p(a))$ 的调用集 $\hat{Set\_c}(? +(p(a)), DB_1)=\{\}$，因而可得到事务 $? +(p(a))$ 的更新集为 $Set\_U(? +(p(a)), DB_1, DB_1)=\{\langle DB_1, -q(a)\rangle\}$；该更新加入上级事务 $(DB_1? +(r(a)))$ 更新集得到 $Set\_U((DB_1? +(r(a))), DB_1, DB_1)=\{\langle DB_1, -q(a)\rangle, \langle DB_1, +t(a)\rangle, \langle DB_1, -p(a)\rangle\}$。

(2.2) 由 $Fix(DB_1)|=(p(b))$，可得到事务 $(DB_1 + DB_0?(p(b)))$ 的调用集 $\hat{Set\_c}((DB_1 + DB_0?(p(b))), DB_1)=\{\}$，因而可得到事务 $DB_1 + DB_0?(p(b))$ 的更新集为：$Set\_U((DB_1 + DB_0?(p(b))), DB_1, DB_0)=\{\langle DB_0, -q(b)\rangle\}$。

(3) 由于 $\hat{Set\_c}(T, DB_0)$ 中的调用均成立，$Fix(DB_0)$ 保持不变，有 $Fix(DB_0)|=(r(b), s(a))$，并有 $\hat{Set\_c}(T, DB_0)=\{DB_1? +(r(a)), DB_1 + DB_0?(p(b))\}$，将 $\hat{Set\_c}(T, DB_0)$ 事务的更新集加入，得到 $Set\_U(T, DB_0, DB_0)=\{\langle DB_0, -q(b)\rangle, \langle DB_0, +t(b)\rangle, \langle DB_0, -p(b)\rangle, \langle DB_1, -q(a)\rangle, \langle DB_1, +t(a)\rangle, \langle DB_1, -p(a)\rangle\}$。

(4) 执行更新 $Set\_U(T, DB_0, DB_0)$，得到更新后的调用程序 DB_0，DB_1，如图 4 所示。

$t(a)$;//EDB	
$p(X) \leftarrow q(X), -q(X)$;	
$r(X) \leftarrow +t(X), p(X), DB_1 + DB_0?(p(X))$;	程序 DB_1
$k(X) \leftarrow -t(X), ?(p(X))$;	
$s(X) \leftarrow t(X), DB_1? +(r(X))$//IDB	
$q(b)$;	
$t(a)$;//EDB	
$p(X) \leftarrow q(X), -q(X)$;	程序 DB_0
$r(X) \leftarrow +t(X), p(X), ? +(p(X))$;	
$k(X) \leftarrow -t(X), DB_0? +(p(X))$;//IDB	

图 4　更新后的程序 DB_0 和 DB_1

3　应用示例

在数字版权保护技术的研究中，目前主要存在两类权利描述语言（REL）：一类是基于逻辑的，典型的语言包括 LicenseScript 等；另一类是基于 XML 的语言，如 XrML（http://www.xrml.org）和 ODRL（http://www.odrl.net）等。基于 XML 的 REL 语言发展较为完善，已处于实用阶段。

但 REL 对开放性、灵活性、可扩展性以及支持各类使用权利描述的要求使基于 XML 的语言难于满足：(1) 当使用条件变得复杂时，基于 XML 的 REL 语言的语法也变得复杂和难以理解；(2) 缺乏正式的语义使基于 XML 的 REL 语言的确切含义严重依赖特定理解；(3)无法表达许多有用的版权法相关的权限要求。逻辑语言由于在表达力、灵活性和语义完整性上的优势，基于逻辑的 REL 语言的研究受到重视，但现存的逻辑语言存在的一些问题，如 LicenseScript 语言规则的不统一性、语言的复杂性、许可证更新困难等造成基于逻辑的 REL 语言无法进入实用阶段。LucScript 语言是我们正在研制的数字版权保护脚本语言[9,17]，该语言的许可证规则和更新管理规则均使用 Active-U-Datalog^r 语言，其核心是包含调用谓词子的 U-Datalog^r。LucScript 实现框架如图 5 所示。

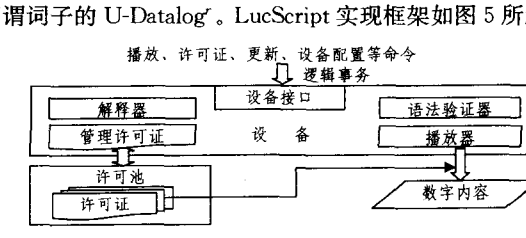

图 5　LucScript 实现框架

在 LucScript 实现平台中，所有数字媒体播放、许可证、更新、设备配置等命令均以逻辑事务的形式提交到设备接口，再

由逻辑解释器解释后执行。许可证由 U-Datalog^r 程序组成，存放在许可池中。许可池由管理许可证管理，管理许可证由与许可证相同的 Active-U-Datalog^r 程序组成。

LucScript 语言中许可证 lic 是四元组 $\{D, IDB, AR, BV\}$，其中 D 是许可证所保护的数字内容的唯一识别符，IDB 是内涵规则集，AR 是触发规则集，BV 是表示为 $name \equiv value$ 形式的属性绑定形式。称 $P = IDB \cup AR \cup BV$ 为许可证程序部分。许可证表示成 $lic(D, \Delta)$，其中 Δ 是许可证程序标识符。图 6 是许可证示例。

```
(1) License //许可证示例
(2) (e_film_star_war,//许可证数字媒体标识
(3)     { play(D)← counter(n),n≥1,update_counter();
(4)       update_counter()←counter(n),n = n1+1,−counter(n),
          +counter (n1);}, //IDB
(5)     { …… },//AR,
(6)     { counter≡50;version≡'11. 6. 1';expire≡'07/12/31';
          type≡'use' },//BV
(7) )
```

图 6　使用许可证示例

图 6 语句(2)说明该许可证管理的数字媒体标识。语句(3)－(4)是 IDB 部分，语句(3)说明只有当计数器不小于 1 才能播放该媒体并使用语句(4)进行计数器更新。语句(4)将计数器的值减 1，其中 $-counter(n)$ 删除计数器的当前值，$+counter(n_1)$ 插入新的计数值。语句(5)是主动规则部分(略)。语句(6)是许可证外延绑定部分，如 $counter$ 谓词说明该许可证允许播放媒体 50 次。

LucScript 许可证使用本文的逻辑程序调用谓词进行交互和更新。增加调用语义后，许可证具有自拆分能力。如假定在图 1 的许可证中增加下列 IDB 规则：

$$clone(lic(D, \Delta)) \leftarrow create(lic(D, \Delta)), call(\Delta, (+IDB(r))), IDB(r) \quad (14)$$

其中，$create(lic(D, \Delta))$ 创建许可证 $lic(D, \Delta)$，IDB(r) 是许可证结构谓词，代表许可证中的 IDB 规则，$call(\Delta, (+IDB(r)))$ 是本文的 $\Delta?+(T)$ 型调用谓词，在程序 Δ 中执行 $+IDB(r)$ 插入。整个规则创建临时许可证 $lic(D, \Delta)$ 并将图 6 许可证中的 IDB 规则全部插入到该临时许可证中，通过这种方法，使许可证本身的逻辑程序具有拆分和创建新许可证等管理功能，从而使管理许可证可由与许可证相同的 Active-U-Datalog^r 程序组成。

LucScript 版权保护框架建立在具有统一逻辑基础的非过程逻辑语言上，使用 Datalog 程序调用语义来完成许可证的管理、更新和交互等操作，克服了 LicenseScript 等现存逻辑版权语言中缺乏统一逻辑基础、使用过程性逻辑语言、更新困难等问题。另外，LucScript 许可池使用与许可证结构相同的管理许可证来管理，不需要专门的解释器和固化在解释器中，使解释器具有更好的灵活性和通用性。

结束语　Datalog 程序的调用和交互问题是迄今 Datalog 逻辑程序中尚未涉及到的问题。本文提出在可更新 U-Datalog 程序中的调用语义及其调用谓词形式，并说明包含程序调用谓词的 U_Datalog^r 程序操作语义及其固定点语义，给出在有限分层调用情况下的 U-Datalog^r 程序的通用评价算法。

目前，U_Datalog^r 应用到我们正在开发的数字版权保护脚本语言 LucScript 中。本文对该应用做了说明并给出示例。既然现存的权利描述语言缺乏统一的逻辑基础，U_Datalog^r 的应用将给解决该问题提供良好的基础。下一步将继续扩展调用谓词在包含限制域和负原子的 U_Datalog 语言中的语义及其应用。最后，本文的研究也说明了 Datalog 程序调用语义的提出自有其理论和应用价值。

参 考 文 献

[1] Jajodia S, Samarati P, Sapino M L, et al. Flexible support for multiple access control policies[J]. ACM Transaction on Database System, 2001, 26(2): 214-260

[2] Li Ninghui, Grosof B, Feigenbaum J. A practically implementable and tractable delegation logic[C]// Proceedings of the 2000 IEEE Symposium on Security and Privacy. 2000: 27-42

[3] Li Ninghui, Mitchell J C. Datalog with constraints: a foundation for trust-management languages[C]// Proceedings of the Fifth International Symposium on Practical Aspects of Declarative Languages. New Orleans, USA, 2003: 58-73

[4] Bertino E, Catania B, Ferrari E, et al. A logical framework for reasoning about access control models[C]// Proceedings of the sixth ACM symposium on Access control models and technologies table of contents. Chantilly, Virginia, USA, 2001: 41-52

[5] Park J, Sandhu R. Towards usage control models: beyond traditional access control[C]// Proceedings of the seventh ACM symposium on Access control models and technologies. Monterey, California, USA, 2002: 57-64

[6] Montesi D, Bertino E, Martelli M. Transactions and updates in deductive databases[J]. IEEE Trans. Knowl. Data Eng., 1997, 9(5): 784-797

[7] Bertino E, Catania B, Gori R. Active-U-Datalog: integrating active rules in a logical update language[C]// Proceedings of International Seminar on Logic Databases and the Meaning of Change. Schloss Dagstuhl, Germany, 1998: 107-133

[8] 钟勇，秦小麟，郑吉平，等. 一种灵活的使用控制授权语言框架[J]. 计算机学报，2006，29(8)：1408-1418

[9] Zhong Y, Zhu Z, Lin D M, et al. A Method of Fair Use in Digital Rights Management[C]// Proceeding of the 10th International Conference on Asian Digital Libraries. LNCS 4822. Hanoi, Vietnam, 2007: 160-164

[10] Chong C N, Corin R, et al. LicenseScript: a logical language for digital rights management[J]. Annales des Telecommunications, 2006, 61(3/4): 284-331

[11] Gunter C A, Weeks S T, Wright A K. Models and languages for digital rights[C]// Proc. of the 34th Annual Hawaii International Conference on Systems Sciences. Maui, Hawaii, 2001: 4034-4038

[12] Pucella R, Weissman V. A logic for reasoning about digital rights[C]// IEEE Proc. of the Computer Security Foundations Workshop. Cape Breton, Nova Scotia, CA, 2002: 282-294

[13] Ullman J. Principle of data and knowledge-based systems vol Ⅰ and Ⅱ[M]. New York: Computer Science Press, 1989

[14] Catriel B, Raghu R. On the power of magic[J]. Journal of Logic Programming, 1991, 10(3): 255-299

[15] Toman D. Memoing Evaluation for Constraint Extensions of Datalog[J]. Constraints, 1997, 2(3/4): 337-359

[16] Revesz P Z. Safe stratified Datalog with integer order programs[C]// Proceedings of Principles and practice of constraint programming. Cassis, France, 1995: 154-169

[17] 钟勇，秦小麟，刘凤玉. ODRL 权利描述语言逻辑实施机制研究[J]. 计算机科学，2009，36(4)：133-139