arXiv:2611.32123v1 [cs.PL] 24 Nov 2026

非传递信息流安全的类型可靠性验证

Chenyi Zhang

新西兰基督城坎特伯雷大学计算机科学与软件工程系

摘要。 我们使用 Dafny，为一种命令式编程语言中的非传递非干扰（NTNI）信息流安全属性，给出了类型系统的机器检查可靠性证明。NTNI 最初是在一种加入内存访问运算符的函数式编程语言中提出的，其中安全策略被表达为针对基于组件的软件系统的粗粒度信息流约束。尽管后续研究从多个方向扩展了 NTNI 类型系统，但据我们所知，此前尚无研究为由类型系统强制实施的 NTNI 建立机器检查的可靠性证明。因此，现有 NTNI 强制机制的正确性尚未经过高置信度的形式化验证。

Dafny 是一种面向验证的编程语言和证明助手，支持对函数式、命令式和面向对象程序进行推理。在本研究中，我们在 Dafny 的项代数框架内形式化了该语言的操作语义、安全策略和类型规则。基于这些形式化结果，我们通过机械验证证明，每个被类型系统接受的程序都满足预期的 NTNI 安全保证。

1 引言

基于语言的安全利用程序语义和静态程序分析来强制实施安全属性。此类属性中研究最为广泛的一类是信息流安全，其旨在确保程序的机密性和完整性 [21]。自 20 世纪 90 年代以来，研究者已针对命令式 [27], 、函数式 [9], 和面向对象 [17] 编程语言，提出了许多基于安全类型系统和霍尔逻辑式证明系统的形式化方法，支持严格的非干扰策略以及受控的信息降级策略 [22]。

大多数基于语言的信息流控制（IFC）系统都建立在 Denning 的格模型 [3, 4], 之上，该模型假设信息流策略具有传递性。在这些系统中，安全级别构成一个偏序格，用于对输入、输出和中间变量等程序实体进行分类，从而能够跟踪整个程序中的信息依赖关系
执行。然而，在某些场景中，传递性假设过于严格或并不理想。现代软件系统越来越多地由

```
库          {  //  受保护代码：只能通过服务访问
   处理（源_数据）{              ...  }
   检索（键）{
      ...
      返回 某些_数据;
   }
}
服务 {//           特权代码
   日志文件;
   添加日志（x）{
      追加（日志文件,       x）;
   }
   打印（数据）{
      库. 处理（数据）;
   }
   查询_数据（键）{
      返回 库 . retrieve ( 键 );
   }
}
已下载_代码 {// 不可信代码
   v ,  d  =  . . . ;        //  不受信任的数据
   主函数 () {
      服务 . 添加日志 (d);
      服务 . 打印 (d. 数据 );
      v   =服务 . 查询_数据 (d. 键 );
   }
}
```

图 1. 混淆代理攻击。

从多样且可能不可信的来源获得的可复用组件构成。此类组件（包括从互联网下载的第三方代码）可能会与敏感系统组件在同一进程中并行执行。在这些环境中，安全策略不能仅依赖原始软件开发人员的意图。相反，应用程序开发人员、系统集成商或部署人员必须能够指定自己的安全需求，以保护可信系统组件免受不可信代码的侵害。由于用户通常不了解第三方组件的内部实现细节，因此他们通常只能根据每个组件的预期功能指定粗粒度安全策略。因此，安全标签关联的是软件组件，而不是单个程序变量。为满足这些需求，非传递非干扰（NTNI） [14] 被提出，作为安全软件组合更灵活的基础。

在图 1 所示的示例中，我们假设存在一个本地系统，其中库 受到保护，只能通过特权代码服务访问。来自互联网的下载代码不受信任，但可能被允许访问

服务，但不能访问库。例如，下载代码 可以调用addLog()，将一些（不受信任的）数据追加到不可执行日志文件中，供系统管理员手动检查。由于追加的数据不会进一步传播到系统的其他部分，因此该操作被视为安全。不受信任的下载代码 还会调用服务 中的print()，该函数通过process()使用受保护的库资源读取并打印数据。此调用是不安全的。尽管print()函数拥有从库调用process()的权限，并且下载代码 拥有使用服务 接口的权限，但下载代码 不应影响库 函数的执行。因此，这体现了混淆代理问题——虽然下载代码 无法直接访问库中的打印输出，但它仍可通过（被混淆的）代理服务影响打印输出。[1]

采用细粒度安全策略的传递性非干扰可以通过为服务中的各个程序变量分配安全级别来解决上述问题，例如，规定添加日志.x 可供所有代码访问，而打印.data 仅可供受信任的本地代码访问。然而，由于服务可能包含大量变量和函数，供各种第三方应用程序共享，因此，逐一指定程序变量可能既困难又容易出错。采用粗粒度安全策略的 NTNI 提供了一种更简单、更直观的解决方案：分别为代码库库、服务和下载代码分配安全级别 L、S 和 D，并指定$D \trianglerighteq S$、$S \trianglerighteq L$、$L \trianglerighteq S$和 $S \trianglerighteq D$（不允许 $D \trianglerighteq L$和 $L \trianglerighteq D$）。根据此策略，Service.query data() 将违反策略，因为它通过服务从库检索信息；由于 $L \trianglerighteq D$ 被禁止，该操作将被拒绝。

类型系统已被广泛用于在编程语言中强制实施信息流安全属性，例如 [21, 22]。然而，现有的大多数方法都没有提供可由机器检查的可靠性证明，少数例外采用了 Coq [10]和 Isabelle/HOL [16]。特别是，Murray 等人 [16] 通过机器检查证明，证明 seL4 微内核能够强制实施信息流安全，并针对 seL4 的实际 C 实现证明了非传递非干扰的一个变体。在另一个项目中，Kamm¨uller [10] 形式化定义了一种简单的字节码语言，并借助 Coq 的模块系统证明了非干扰。NTNI 的类型系统目前只有纸笔形式的可靠性证明 [14]。

本文介绍了在 Dafny [11] 中实现的一个针对非传递非干扰（NTNI）类型系统的机器检查的可靠性证明。与 Coq 和 Isabelle/HOL 等交互式定理证明器通常所需的大量证明工程工作相比，Dafny 为编程语言语义和类型系统的形式化与验证提供了一个轻量级框架。在 Dafny 中编码的规约、谓词和重写规则会自动转换为验证条件，并由后端 SMT 求解器完成证明，从而显著减少机械化

[1] 这个问题也可以通过动态 *stack inspection* 机制 [6, 28],识别，尽管栈检查与信息流这两种方法之间存在细微差异；本报告讨论了若干示例 [13]。

证明所需的人工工作。因此，Dafny 能够以相对简洁、结构化且易于理解的方式对类型系统进行形式化和验证。

此外，与 [14], 中最初提出的、基于函数式编程环境的 NTNI 形式化不同，我们将 NTNI 应用于一种 while 语言，其风格类似于 Volpano–Smith–Irvine (VSI) 类型系统 [27], 所使用的语言；VSI 类型系统是首个用于传递性非干扰的可靠类型系统。我们采用这种更简单的命令式环境，旨在让 NTNI 类型系统正确性的直觉比原始表述更加清晰、易于理解。

我们的 Dafny 形式化已公开发布[2]。语言语义、类型系统和可靠性证明均以精简的 Dafny 模块实现，共包含 635 行 Dafny 代码，其中包括注释、合理性测试和断言。（其中，语言模型为 195 行，类型系统为 180 行，可靠性证明为 260 行。）对于一种主要涉及少量量词算术和集合论推理的简单语言而言，这个规模易于管理，因此非常适合交由 SMT 后端验证证明，同时不会带来过多额外的信任要求。作为对比，Coq 中针对简单字节码语言的（传递性）非干扰模型约需 2，300 行代码 [10]。

我们的贡献总结如下。

1. 我们基于 [27,23] 的 while 语言，提出了一个针对强制执行非传递非干扰（NTNI）的类型系统的机器检查的可靠性证明。作为副产品，我们还提供了 Volpano–Smith–Irvine（VSI）类型系统的 Dafny 机械化可靠性证明；其原始论述完全依赖纸笔证明。请注意，我们不是 首个为 VSI 类型系统提供机器检查证明的研究者，因为已有已知工作，例如 [24] ，其中包含一个使用 Isabelle 完成的证明。
2. 我们对 NTNI 类型系统的证明阐明，对于每个安全级别ℓ，相应的安全推理都可以归约为在扩展的程序变量集合上的双级信息流安全问题。该归约直接基于程序的类型推导建立，因此仅适用于类型良好的程序。与 Ahmadpanah 等人 [1]，通过程序变换将 NTNI 归约为传递性非干扰、并可能增加程序的规模和复杂度的方法相比，我们的归约在概念上更简单、更直接。

2 系统模型

我们在图2所示的命令式编程语言中对安全条件进行编码，其中元变量 x 遍历变量， v 遍历整数值。我们用 0 表示 false，用非零值表示 true。命令也称为程序。给定一个程序， V 是程序中出现的变量集合。

[2] 代码位于 https://github.com/czhang-fm/TypeSoundnessNTNI/

$$
\begin{array}{lll}
e ::= & & \textit{Expressions} \\
\quad v & & \text{值} \\
\quad x & & \text{变量} \\
\quad e + e & & \text{二元运算} \\
c ::= & & \textit{Commands} \\
\quad \texttt{skip} & & \text{终止} \\
\quad x := e & & \text{赋值} \\
\quad \text{条件语句 } e, \text{ 那么 } c_1, \text{ 否则 } c_2 & & \text{条件语句} \\
\quad \text{while 循环 } e, \text{ 执行 } c & & \text{loop} \\
\quad c_1;\ c_2 & & \text{顺序组合}
\end{array}
$$

图2. 语言语法。

$$
\frac{M\ e_1 \Downarrow v_1 \qquad M\ e_2 \Downarrow v_2}{M\ e_1 + e_2 \Downarrow v_1 + v_2}\ [\text{EXP}] \qquad \frac{}{M\ v \Downarrow v}\ [\text{VAL}]
$$

$$
\frac{M(x) = v}{M\ x \Downarrow v}\ [\text{VAR}] \qquad \frac{M\ e \Downarrow v}{(M, x{:=}e) \to (M[x \mapsto v], \texttt{skip})}\ [\text{ASN}]
$$

$$
\frac{(M, c_1) \to (M', c_1')}{(M, c_1; c_2) \to (M', c_1'; c_2)}\ [\text{SEQ1}] \qquad \frac{(M, c_2) \to (M', c_2')}{(M, \texttt{skip}; c_2) \to (M', c_2')}\ [\text{SEQ2}]
$$

$$
\frac{M\ e \Downarrow v \quad v \neq 0}{(M, \texttt{while}\ e\ \texttt{do}\ c) \to (M, c; \texttt{while}\ e\ \texttt{do}\ c)}[\text{WHT}] \qquad \frac{M\ e \Downarrow 0}{(M, \texttt{while}\ e\ \texttt{do}\ c) \to (M, \texttt{skip})}[\text{WHF}]
$$

$$
\frac{M\ e \Downarrow v \quad v \neq 0}{(M, \texttt{if}\ e\ \texttt{then}\ c_1\ \texttt{else}\ c_2) \to (M, c_1)}[\text{TRU}] \qquad \frac{M\ e \Downarrow 0}{(M, \texttt{if}\ e\ \texttt{then}\ c_1\ \texttt{else}\ c_2) \to (M, c_2)}[\text{FLS}]
$$

图3. 结构操作语义。

程序 c 在内存状态 M： $V \to \mathbb{N}$ 下执行；该状态为每个变量赋一个值。其小步语义定义于图3，遵循 Gunter [8]，中的标准结构操作语义，其中 (M, c) →(M', c') 表示程序 c 在内存状态 M 下执行一步，随后变为 c'，内存状态则更新为 M'。

VSI 类型系统 [27] 强制实施两级安全策略 $L \trianglerighteq H$，假设程序变量可划分为 L 和 H，并且存储在标记为安全级别 L 的变量中的信息可以流向具有安全级别 H 的变量，但反向则不允许。定义这一安全属性的一种更精确的方法是采用 非干扰 的概念，该概念最初由 Goguen 和 Meseguer [7] 提出。在编程语言中，我们需要定义内存状态之间的等价关系来表达这一属性。定义 $M_1 =_L M_2$：如果两个内存状态 M_1 和 M_2 对所有标记为 L 的变量的值都一致（即所谓的 低等价状态），并且“$\to^*$”是图3中定义的小步转移关系“$\to$”的传递闭包，前提是执行能够终止。

定义 1（终止不敏感非干扰，TI-NI）。程序c 满足关于策略 $L \trianglerighteq H$的 *TI-NI*，当且仅当对于所有内存状态 M_1 和 M_2，

$$\frac{\Sigma \vdash e_1 : \tau \qquad \Sigma \vdash e_2 : \tau}{\Sigma \vdash e_1 + e_2 : \tau}\ [\text{TEXP}] \qquad \frac{}{\Sigma \vdash v : L}\ [\text{TVAL}]$$

$$\frac{\Sigma(x) = \tau\ \texttt{var}}{\Sigma \vdash x : \tau}\ [\text{TVAR}] \qquad \frac{\Sigma \vdash x : \tau \quad \Sigma \vdash e : \tau}{\Sigma \vdash x{:=}e : \tau}\ [\text{TASN}]$$

$$\frac{\Sigma \vdash c_1\ \tau\ \texttt{cmd} \quad \Sigma \vdash c_2\ \tau\ \texttt{cmd}}{\Sigma \vdash c_1; c_2 : \tau\ \texttt{cmd}}\ [\text{TSEQ}] \qquad \frac{\Sigma \vdash e : \tau \quad \Sigma \vdash c : \tau\ \texttt{cmd}}{\Sigma \vdash \texttt{while}\ e\ \texttt{do}\ c : \tau\ \texttt{cmd}}\ [\text{TWH}]$$

$$\frac{\Sigma \vdash e : \tau \quad \Sigma \vdash c_1 : \tau\ \texttt{cmd} \quad \Sigma \vdash c_2 : \tau\ \texttt{cmd}}{\Sigma \vdash \texttt{if}\ x\ \texttt{then}\ c_1\ \texttt{else}\ c_2 : \tau\ \texttt{cmd}}\ [\text{TIF}]$$

$$\frac{}{L \trianglerighteq H}\ [\text{BASE}] \qquad \frac{\tau_1 \trianglerighteq \tau_2}{\tau_2\ \texttt{cmd} \trianglerighteq \tau_1\ \texttt{cmd}}\ [\text{CMD}] \qquad \frac{}{\rho \trianglerighteq \rho}\ [\text{REFL}]$$

$$\frac{\rho_1 \trianglerighteq \rho_2 \quad \rho_2 \trianglerighteq \rho_3}{\rho_1 \trianglerighteq \rho_3}\ [\text{TRANS}] \qquad \frac{\Sigma \vdash c : \rho_1 \quad \rho_1 \trianglerighteq \rho_2}{\Sigma \vdash c : \rho_2}\ [\text{SUB}]$$

图 4。 Volpano-Smith-Irvine 类型系统。

我们有

$$\left.\begin{array}{c} M_1 =_L M_2 \\ (M_1, c) \to^* (M_1', \texttt{skip}) \\ (M_2, c) \to^* (M_2', \texttt{skip}) \end{array}\right\} \implies M_1' =_L M_2'$$

我们注意到，定义 1 是终止不敏感的，与此同时， [27], 本文不考虑由程序终止导致的信息泄漏。因此，如果上下文含义明确，我们通常会在定义中省略“TI”。

显式流。 当具有高安全级别的变量直接出现在赋值给较低安全级别变量的表达式中时，就会发生显式信息流，例如赋值 $l := h + 2$，其中 l 标记为 L，而 h 标记为 H。显式流模式也会出现在输入/输出操作以及返回值函数或方法调用中。

隐式流。 当高安全级别变量影响程序的控制流，从而间接影响低安全级别变量的值时，就会发生隐式信息流。一个典型示例是语句 if h then l:=1 else $l := 0$，其中 $h \in \{0, 1\}$。在这种情况下， h 的值实际上通过分支结构被复制到 l 。

我们简要回顾图 4 中定义的 VSI 类型系统。VSI 系统在安全策略 $L \trianglerighteq H$ 下定义了两种基本安全类型： L 和 H。与变量 x 关联的变量类型 τ var 表示， τ 是根据程序员或用户的指定分配给 x 的安全类型。类型 τ cmd 仅用于命令（或程序）的类型检查。VSI 类型系统的关键洞见是：表达式始终被分配其所依赖信息中的最高安全级别；而命令 c 被分配类型$\tau\ cmd$，其中 τ 是执行 c 期间可能被更新的变量中的最低安全级别。

定理 1（VSI 正确性 [27]）。条件语句 c 在 *VSI* 中类型正确，则 c 满足相对于策略 $L \trianglerighteq H$的非干扰性质。

有时有人认为，给定一个多级安全策略（$\mathbb{S}$，$\trianglerighteq$），其中$\mathbb{S}$ 是安全级别的集合，$\trianglerighteq$是 $\mathbb{S}$上的偏序，安全推理可以归约为一组双级安全问题，每个安全级别 $\ell \in \mathbb{S}$对应一个问题。具体而言，对于固定级别 ℓ，令 $L_\ell = \{\ell' \mid \ell' \trianglerighteq \ell\}$ 和 $H_\ell = \mathbb{S} \setminus L_\ell$。随后，只需针对每个安全级别 $\ell \in \mathbb{S}$，检查来自 H_ℓ 的信息是否会干扰在 L_ℓ 中的级别上可观察到的信息，即可验证安全性。

3 非传递非干扰

在由传递信息流策略支配的系统中，与特定安全级别相关联的信息可以流向通过策略关系的传递闭包可达的任何其他安全级别。这一特性可能使传递性策略不适合表达基于组件的系统中的粗粒度安全需求。

例如，假设 Alice 拥有的一个组件只愿意与另一个组件 Bob 共享信息。在传递性策略下，如果 Bob 获准与其他组件共享信息，那么 Alice 的信息也可能通过策略关系的传递闭包间接流向这些组件。这类间接信息流可能违反 Alice 的预期安全需求，因此并不可取。

上述需求促使我们采用一种限制性更强的方案：移除策略的传递性要求 $\trianglerighteq$，该策略是安全级别集合上的二元关系。对于图 1 中示例所需的策略，其中包含代码库 库 (L)（受保护代码）、服务 (S)（特权代码）和 下载代码 (D)，安全级别属于 $\mathbb{S} = \{L, S, D\}$，允许的信息流关系为 $D \trianglerighteq S$、$S \trianglerighteq L$、$L \trianglerighteq S$ 和 $S \trianglerighteq D$（即不允许 $D \trianglerighteq L$ 和 $L \trianglerighteq D$）。以 下载代码 D 为例，安全策略要求，类型为 D 的变量的最终值只能受到输入值为类型 D 或 S 的变量的影响（由于 $S \trianglerighteq D$），而不能受到 L 的影响。这引出了以下定义。

对于每个 $\ell \in \mathbb{S}$，定义 $\trianglerighteq(\ell) = \{\ell' \mid \ell' \trianglerighteq \ell\}$，它是允许影响 ℓ 的安全级别集合。此外，我们假设 $\trianglerighteq$是自反的，因此对于任意 $\ell \in \mathbb{S}$，都有 $\ell \in \trianglerighteq(\ell)$。若对于所有被赋予安全级别 $\ell \in T$ 的变量 x，均有 $M_1(x) = M_2(x)$，则记为 $M_1 \overset{T}{\sim} M_2$。

定义 2（终止不敏感非传递非干扰，TI-NTNI）。[14] 程序 c 满足关于非传递策略 $\trianglerighteq$ 的 TI-NTNI，当且仅当对于所有内存状态 M_1 和 M_2 以及安全级别 $\ell \in \mathbb{S}$，均有

$$\left.\begin{array}{c} M_1 \overset{\trianglerighteq(\ell)}{\sim} M_2 \\ (M_1, c) \to^* (M_1', \texttt{skip}) \\ (M_2, c) \to^* (M_2', \texttt{skip}) \end{array}\right\} \Longrightarrow M_1' \overset{\{\ell\}}{\sim} M_2'$$

图 5 中描述的类型系统在安全级别集合 $\mathbb{S}$ 上强制实施非传递非干扰策略 $\trianglerighteq$。类型上下文 Σ 为每个变量分配一个安全级别

$$\frac{\Sigma, \Gamma \vdash_{pc} e_1 : \tau \qquad \Sigma, \Gamma \vdash_{pc} e_2 : \tau}{\Sigma, \Gamma \vdash_{pc} e_1 + e_2 : \tau}\ [\text{NEXP}] \qquad \frac{}{\Sigma, \Gamma \vdash_{pc} v : \emptyset}\ [\text{NVAL}]$$

$$\frac{\Sigma(x) = \ell}{\Sigma, \Gamma \vdash_{pc} x : \{\ell\}}\ [\text{NVAR}] \qquad \frac{\Sigma, \Gamma \vdash_{pc} e : \tau \quad \tau \cup pc \subseteq \Gamma(x) \subseteq \trianglerighteq(\Sigma(x))}{\Sigma, \Gamma \vdash_{pc} x{:=}e : \{x\}}\ [\text{NASN}]$$

$$\frac{\Sigma, \Gamma \vdash_{pc} c_1 : \rho \quad \Sigma, \Gamma \vdash_{pc} c_2 : \rho}{\Sigma, \Gamma \vdash_{pc} c_1; c_2 : \rho}\ [\text{NSEQ}] \qquad \frac{\Sigma, \Gamma \vdash_{pc} e : \tau \quad \Sigma, \Gamma \vdash_{pc \cup \tau} c : \rho}{\Sigma, \Gamma \vdash_{pc} \texttt{while}\ e\ \texttt{do}\ c : \rho}\ [\text{NWH}]$$

$$\frac{\Sigma, \Gamma \vdash_{pc} e : \tau \quad \Sigma, \Gamma \vdash_{pc \cup \tau} c_1 : \rho \quad \Sigma, \Gamma \vdash_{pc \cup \tau} c_2 : \rho}{\Sigma, \Gamma \vdash_{pc} \texttt{if}\ x\ \texttt{then}\ c_1\ \texttt{else}\ c_2 : \rho}\ [\text{NIF}]$$

$$\frac{\Sigma, \Gamma \vdash_{pc} e : \tau_1 \quad \tau_1 \subseteq \tau_2}{\Sigma, \Gamma \vdash_{pc} e : \tau_2}\ [\text{E-SUB}] \qquad \frac{\Sigma, \Gamma \vdash_{pc} c : \rho_1 \quad \rho_1 \subseteq \rho_2}{\Sigma, \Gamma \vdash_{pc} c : \rho_2}\ [\text{C-SUB}]$$

图 5. 非传递非干扰类型系统。

$\Sigma(x) \in \mathbb{S}$，并且类型上下文 Γ 为每个变量 x 分配一个可能（传递地）流向 x 的安全级别集合。具体而言，对于每个表达式 e，类型系统为其分配一个类型 τ，其中包含所有可能影响 e 值的信息；对于每个命令 c，命令类型 ρ 是在执行 c 时可能被更新的变量集合。特别地，[C-SUB] 表明，如果 ρ_1 是可能被更新的变量集合（Σ、$\Gamma \vdash_{pc} c :\ \rho_1$），那么可以安全地将 c 用更大的集合进行类型检查（即，Σ、$\Gamma \vdash_{pc} c :\ \rho_2$，且满足 $\rho_1 \subseteq \rho_2$）。同样，对于 [E-SUB], ，如果表达式 e 可以用标签集合 τ_1 进行类型检查，那么在给定 $\tau_1 \subseteq \tau_2$ 的情况下，也可以用 τ_2 进行类型检查。现在，子类型关系就是子集关系，因此，VSI 类型系统中出现的自反性和传递性类型规则，在 NTNI 的类型系统中已不再需要。[3] 在这些规则中，控制依赖由 pc 携带，并通过诸如 [NIF] 和 [NWH], 的规则进行传播；当赋值位于分支结构或循环结构内部时，则由 [NASN] 强制执行。在 Dafny 模型中，命令类型 ρ 用于指示所提供的类型上下文对 Σ 和 Γ 是否对某个程序有效，因此，无效的类型上下文应将 ρ 设为无效。

最重要的约束或许定义在规则 [NASN] 中，该规则为语句 $x{:=}e$赋予类型。在这种情况下，我们需要确保右侧（即 $e :\ \tau$）所包含的所有信息，以及控制依赖的信息（即 pc），都被允许影响 x（即 $pc \subseteq \trianglerighteq(\Sigma(x))$）。同一规则还在全局层面强制执行 $\Gamma(x) \subseteq \trianglerighteq(\Sigma(x))$。关于控制依赖的信息编码在 pc 中，并通过规则 [NIF] 和 [NWH] 从上层语法结构传播下来。

图 5 中引入的类型系统强制实施 NTNI。该定理将在下一节介绍的 Dafny 模型中得到证明。[4]

[3] 在为整个程序建立类型的过程中，任意扩大某个表达式或语句的类型都会受到限制。对于程序 c，我们只要求存在类型上下文 Σ 和 Γ，以便为 c 建立 NTNI。此类类型系统通常是不完备的，也就是说，可能存在一个程序

定理 2（NTNI可靠性）。条件语句 c 相对于安全策略 $\triangleright$在 *NTNI* 类型系统中具有良好类型，则 c相对于 $\triangleright$满足非传递非干扰。

4 使用 Dafny 编码 NTNI

Dafny 是一种证明助手，同时支持函数式编程风格的规约和命令式编程风格的规约 [11]。Dafny 中的引理或定理本质上是一种方法，它确保只要其前置条件得到满足，所有后置条件也都得到满足。这个基本的 while 循环程序语言通过如下用户定义的（变体）类型进行建模。

```
// 所有表达式的类型
数据类型 表达式 = | 数字 (n: 整数 ) | 变量(x: 变量 )
                  | 加法 ( e1 : 表达式 , e2 : 表达式 )
// 所有命令（语句）的类型
数据类型 命令 = | 跳过 | 赋值(x: 变量 , e: 表达式 )
                | 条件语句（条件: 表达式, then 分支: 命令, else 分支: 命令)
                | while 循环（条件: 表达式, c : 命令)
                | 序列（c1: 命令, c2 : 命令）
```

列表 1.1. 程序语法

```
函数 小步语义(s: 机器状态, c: 命令) : (机器状态, 命令)
// ...
{
    匹配 c {
        case Skip => (s, c)
        case 赋值 (x, e) => (s[x := 求值 (s, e)], 跳过 )
        case 条件语句 ( e , c1, c2) =>// 条件语句 - then - else 分支
            var res := 求值 (s , e);
            if (res != 0) then (s, c1) else (s, c2)
        case while 循环 (e, c1) => // while - 循环
            变量 结果: = 求值 (s , e);
            if (结果 != 0) then (s, 序列 (c1, c))
                else (s, 跳过)
        分支 序列 (c1, c2) => // 顺序执行 组合
            匹配 c1 {
                case Skip => (s, c2) // 移除 1 个跳过
                分支 c1 =>
                    var (s', c') := 小步语义 (s, c1);
                    (s', 序列 (c', c2)) }
    }
}
```

列表 1.2. 结构化操作语义

它满足 NTNI，但无法在图 5 所定义的系统中通过类型检查。根据程序和策略计算类型上下文也不在本文的讨论范围内。

结构操作语义（SOS）定义了此类程序的执行方式，其本质上被建模为一个函数：该函数读取内存状态 s 和命令 c 作为输入，然后执行单步操作语义，并可能将内存状态 s 更新为一个带有（可能）更短命令的新状态。在执行模型中，Evaluate(s, e) 根据内存状态 s 对表达式 e 求值；如果 c 是顺序组合，则该函数可以按照图3中的 SOS 规则 [SEQ1] 递归调用自身。

类型检查过程由两个函数建模。第一个函数 HasExprType(vctx, e) 收集流入表达式 e 的信息（安全标签集合）。具体而言，变量 x 所表示的信息量是 x 变量本身以及流入 x(vctx[x])的信息。流入 e1 +e2 的信息递归定义为流入 e1 和流入 e2 的信息的连接，其中 Join 运算符本质上是集合并。

```
函数 HasExprType ( vctx : VContext,                e: Expr ): 短语类型
{
    匹配 e {
        分支 Num(n)   => ExprType ({})
        分支 变量（x）   => ExprType ( vctx [x ] +{ x})
        case 加 (e1    ,  e2 )  =>
            变量 t1: = HasExprType (vctx,        e1 );
            变量 t2: = HasExprType (vctx,        e2 );
            ExprType ( Join (t1 ,    t2 ))
    }
}
```

列表 1.3。 表达式类型检查（为便于阅读而简化）

HasCmdType(vctx, vpc, c) 函数用于检查在上下文 vctx 下，考虑由 vpc 表示的控制依赖时， c 是否可进行类型检查。若类型检查成功，则返回 c 中可能被更新的变量集合；否则返回 Invalid。任何包含 Invalid 组件的命令也无法通过类型检查。我们只关注赋值和 if 条件这两种情况，因为这两种结构足以说明 HasCmdType() 的工作方式。对于 Assn(x, e), x 的安全级别存储在 t_1 中，而 e 中的信息存储在 t_2 中，其标签来自 e labels。随后，模型会强制要求存储在 t_2 中的安全标签（即 e labels）是 CanFlow(t1) 的子集；根据安全策略，CanFlow(t1) 是允许流入 x 的信息的上界。控制依赖变量（pc 标签）也会根据 CanFlow(t1) 进行检查。对于 If(e, c1, c2)，要求使用一个新的（可能更大的）控制依赖集合（vpc+te）检查两个分支 c1 和 c2，其中通过 te 纳入了来自 e 的新信息。

```
函数 HasCmdType ( vctx : VContext ,  vpc:  集合 <变量 >,  c: 命令 ): 短语类型//
...{
```

```
    4 匹配 c { 5 分支 Skip  =>  CmdType ({}) 6 分支 Assn (x , e)  =>7 // t1 是
x 的策略标签 8 // t2 表示来自等式右侧 =的信息 9 var t1 , t2 := label [x] ,
HasExprType ( vctx , e); 10 // 获取允许影响 x 的标签集合 11 var flowsFrom :=
CanFlow (t1 ); 12 var pc_labels :=( set x  | x in vpc :: label [x]) ; 13 var e_
labels :=( set x  | x in t2 :: label [x]) ; 14 if ( t2 + vpc +{ x})  <= vctx [x]15 &&
( e_labels + pc_labels )  <= flowsFrom 16 then CmdType ({ x }) 17 else 无效
18 分支 If (e , c1 , c2 )  =>19 var te := HasExprType (vctx , e); 20 var t1 :=
HasCmdType ( vctx , vpc +te , c1 ); 21 var t2 := HasCmdType ( vctx , vpc +te ,
c2 ); 22 if t1  ==  Invalid  ||  t2  ==  Invalid then Invalid 23 else (
CmdType ( 连接 (t1 , t2 ))) 24 分支 While (e , c1 )  =>// ... 25 分支 Seq (c1 , c2 )
 =>// /... 26 } 27 }
```

清单 1.4。 命令类型检查（为便于阅读而简化）

```
1 非干扰引理（ vctx：VContext，... s1、s2、s1’、s2’：状态）// 其他前置条件 ... 2 要
求 HasCmdType（vctx，{}，c）!=无效// 类型检查已通过 4 要求 等价（vctx，（变
量集合 x  | x 属于变量集合 && 粗粒度_标签 [x] 属于 CanFlow（l）:: x），s1，s2）
5 要求 终止（ c，s1，s1’） && 终止（ c，s2，s2’） 6 确保 等价（ vctx，（变量集
合 x  | x 属于变量集合 && 粗粒度_标签[x] ==l :: x），s1’，s2’）
```

清单 1.5。 类型系统实现的 NTNI 安全性（定理 2），为便于阅读而简化

清单 1.5 是定理 2 的简化 Dafny 表示。这里，（变量集合 x| x 属于变量集合 && 粗粒度标签 [x] 属于 CanFlow(l) :: x）表示标签属于 $\triangleright(\ell)$ 的变量集合；而记号（变量集合 x | x 属于变量集合 && 粗粒度标签 [x] == l::x）表示带有 x 和 $\underline{\Sigma}(\bar{x}) = \ell$ 的变量集合（即，粗粒度标签 [x] 会获取为 x 分配的安全标签）。下面，我们将讨论如何为定理 2 建立证明。

迈向 NTNI 的可靠性证明 通过证明在一对初始状态 $=_L$ 上可以保持等价关系，确立了双层安全策略 $L \trianglerighteq H$ 下 VSI 类型系统的可靠性（定理 1）

s_1 和 s_2 在良类型程序 c中沿着所有执行步骤前进时也是如此。因此，我们需要通过结构归纳法检查程序c的所有可能构造。

然而，这种证明技术无法直接应用于非传递性非干扰的分支（定理 2）。定义 2 的假设要求两个内存状态在安全级别 $\trianglerighteq(\ell)$ 中存储的所有值上保持一致，而这通常比定义 2 结论中出现的具有安全级别 ℓ 的变量集合更大。给定 $\trianglerighteq(\ell) \subset \sim^{\{\ell\}}$，一种可能的解决方案是，寻找一个位于 $\sim^{\trianglerighteq(\ell)}$ 与 $\sim^{\{\ell\}}$ 之间的新状态等价关系，并使其以与定理 1 的证明中低等价关系（$=_L$）相同的方式工作。如果类型检查成功，则可以根据以下变量集合构造这种关系：$\{y \in V \mid \exists x.\Sigma(x) = \ell \wedge y \in \Gamma(x)\}$，其中 $\Gamma(x)$ 是一个（特定的）函数，它包含在类型检查期间可能（传递性地）影响每个变量 x的所有变量；类型检查由清单 1.4 中的类型上下文 vctx 表示。 [5]

对于每个 $\ell \in S$，定义 $Max(\ell) = \{x \in V \mid \Sigma(x) = \ell\} \cup \{y \in V \mid \exists x.\Sigma(x) = \ell \wedge y \in \hat{}\Gamma(x)\}$。给定内存状态 M_1、 M_2 和 $V' \subseteq V$，对于所有变量 $x \in V'$，若 $M_1(x) = M_2(x)$，则写作 $M_1 \overset{V'}{\sim} M_2$。我们将建立以下对证明至关重要的性质。

1. For all x, y, z, $x \in \hat{\Gamma}(y)$ and $y \in \hat{\Gamma}(z)$ implies $x \in \hat{\Gamma}(z)$;
2. $\{x \in V \mid \Sigma(x) = \ell\} \subseteq Max(\ell) \subseteq \{x \in V \mid \Sigma(x) \trianglerighteq \ell\}$.

第一个性质源于这样一个事实：类型检查会收集变量上的二元流关系，而该关系始终具有传递性；这一点通过为 t2（列表 1.4 的第 9 行）和 te（列表 1.4 的第 19 行）构造 ExprType（列表 1.3 的第 8 行）得以确立，并在列表 1.4 的第 14 行得到强制执行。因此，直观地说，未包含在 $Max(\ell)$ 中的所有其他变量都不能影响 $Max(\ell)$ 中的变量，正如下述引理所示。列表 1.6 给出了一个简化的 Dafny 表示，其中将关于 $Max(\ell)$ 的上述性质编码为前置条件。

引理 1（由类型检查得出的传递性非干扰）。条件语句 c在安全策略 $\trianglerighteq$下符合 *NTNI* 类型系统的类型要求，则对于所有 $\ell \in S$，

$$\left.\begin{array}{c} M_1 \overset{Max(\ell)}{\sim} M_2 \\ (M_1, c) \to^* (M_1', \texttt{skip}) \\ (M_2, c) \to^* (M_2', \texttt{skip}) \end{array}\right\} \implies M_1' \overset{Max(\ell)}{\sim} M_2'$$

引理 1 以及上述关于 $Max(\ell)$ 的第二个性质，可以直接推导出定理 2。

[5]与可由程序和安全策略唯一确定的 Σ 不同， Γ 可能有多个候选项，因此可能存在多组有效的类型上下文对，满足图 5 中列出的约束。

备注。 Ahmadpanah 等人 [1] 提出的程序变换通过引入多个程序变量副本，将 NTNI 安全策略转换为原始程序规范形式上的传递性非干扰策略。上述构造表明，这种重复可能并非必要：对于类型良好的程序，类型系统已经为每个安全级别 $\ell \in S$ 诱导出一个划分，该划分可以模拟双层安全策略（$V \setminus Max(\ell)$）$\not\trianglerighteq Max(\ell)$。

```
引理{:induction false}MaxSetConsistency(vctx:VContext,
      变量集合: 集合 <变量 >     , ..., s1,s2,s1',s2':状态)
// 其他   前置条件          ...
要求 HasCmdType（vctx,          vpc,   c) !=  无效
 要求 对于所有 x、y :: x 属于 变量集合 &&          y  在 vctx [x]  ==>
      vctx [y]  <=vctx [x]  <=变量集合
 要求 对于所有 x, y :: x 属于 变量集合 &&      y  在 vctx [x]  ==>
      粗粒度_标签 [y] 属于 CanFlow（粗粒度_标签 [x]）
 要求       等价（vctx,    变量集合, s1,  s2）
 确保       等价（vctx,    变量集合, s1',  s2'）
```

列表 1.6. 为便于阅读而简化的引理 1

5 相关工作

最相关的工作是 Lu 和 Zhang [14], 提出 NTNI 概念的研究，以及 Ahmadpanah 等人 [1], 通过引入流敏感性和输入输出构造扩展 NTNI 的研究。这些论文讨论的关键概念已在前文各节中介绍。

非传递非干扰（INI）的概念并不意味着底层流策略是非传递的；相反，它描述了对信息流的受控中介。给定策略 $A \trianglerighteq B \trianglerighteq C$，来自 A 的信息可以流向 C，前提是 B 充当可信中介，例如降级器或净化器。Rushby 在基于状态的环境中形式化了非传递非干扰 [20], ，并引入展开条件作为验证的证明技术。这一概念随后在多种环境中得到进一步发展，包括基于轨迹的系统模型 [15] 以及进程代数 CSP [19]。van der Meyden 提出了一种与 INI 相关的变体 [26, 5], ，要求降级器（例如 B）对发送者（例如 A）所传递的信息至少拥有与接收者（例如 C）同等程度的知识。这些模型通常将策略表示为安全域上的图；信息流可以沿某些路径传递，但不能沿其他路径传递。本文采用的 NTNI 表述借鉴了此前关于粗粒度非传递信息流控制及其类型理论强制实施的研究 [14, 1]。在 NTNI 框架内，或许可以模拟 INI 中的策略，使设计者能够通过复杂的策略结构指定哪些边存在以及哪些路径是合法的。

诸如 Coq 和 Isabelle/HOL 之类的证明助手已被用于形式化证明类型系统和信息流安全性质的健全性。

例如，Murray 等人 [16] 提出了一项机器检查证明，证明 seL4 微内核能够实施信息流安全。该证明为 seL4 的实际 C 实现建立了非传递非干扰的一种变体；该实现约包含 8，830 行代码。另一项研究则涉及 Kamm¨uller [10]，他形式化了一种简单的字节码语言，并使用 Coq 模块系统证明了非干扰。

关于降级和信息解密降级的研究构成了另一条独立的研究路线。基于语言的 IFC 系统（如 Jif [17] 和 Fabric [12]）强制执行传递性安全策略，同时通过显式的降级原语满足现实世界的需求，其中包括用于机密性的降密和用于完整性的背书。DC 标签 [25, 29] 通过引入去中心化策略，并支持稳健解密降级和背书 [18, 2]，扩展了这一方法。在这些系统中，降级被视为一种显式的操作机制，程序员必须谨慎地引入并论证其合理性。

6 结论

非传递非干扰（NTNI）是一类安全属性，适用于以粗粒度方式指定信息流策略的场景。本文使用 Dafny 证明助手，为一种简单的命令式编程语言给出了可由机器检查的 NTNI 证明。我们将语言的简洁性视为有助于教学和清晰机械化的优势，并展示如何使复杂的安全证明简洁易懂。相关证明开发成果已公开发布。目前，我们正在继续扩展该语言和证明，以支持完整的面向对象编程特性，例如类和方法调用。我们希望这项工作能为现有的理论发展以及 NTNI 未来可能的应用提供更坚实的基础。

致谢。作者感谢 SOFSEM 2027 程序委员会的匿名评审人，他们帮助发现了早期草稿中的若干事实和表述问题。

参考文献

1. Mohammad M. Ahmadpanah、Aslan Askarov 和 Andrei Sabelfeld。非传递策略的转译。载于 *2021 IEEE European Symposium on Security and Privacy (EuroS&P)* 543 561 2021 ，第 – 页。IEEE，。2. Ethan Cecchetti、Andrew C. Myers 和 Owen Arden。不可篡改的信息流控制。载于 *the ACM Conference on Computer and Communications Security*，第 1875–1891 页。ACM，2017。3. Dorothy E. Denning。安全信息流的格模型。 *Commun. ACM*，19 5 236( ): –243，1976。4. Dorothy E. Denning 和 Peter J. Denning。安全信息流程序认证。 *Commun. ACM*，20(7):504–513，1977。

5. Kai Engelhardt、Ron van der Meyden 和 Chenyi Zhang。非确定性系统中的非传递非干扰。载于 *Communications Security* 869 880, 2012 ，第 – 页。6. Cédric Fournet 和 Andrew D. Gordon。栈检查：理论与变体。载于 *Proceedings of the 29th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages* 02, 307 318, 2002 ，POPL ' 第 – 页。7. Joseph A. Goguen 和 Jose Meseguer。《安全策略与安全模型》。载于 *1982 IEEE Symposium on Security and Privacy*，第 11–11 页，1982 年 4 月。8. Carl A. Gunter。 *Semantics of Programming Languages: Structures and Techniques*。麻省理工学院出版社，马萨诸塞州剑桥，1992。9. Nevin Heintze 和 Jon G. Riecke。slam演算：以保密性和完整性进行编程。载于 *Proceedings of the 25th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages (POPL)* 365 377 1998 ，第 – 页。ACM，。10. Florian Kamm¨uller。用 Coq 为简单字节码语言形式化非干扰。 *Formal Aspects of Computing*，20(3):259–275，2008。11. K. Rustan M. Leino。Dafny：用于函数正确性的自动程序验证器。载于 *Proceedings of the 16th International Conference on Logic for Programming Artificial Intelligence and Reasoning (LPAR 2010)* 6355 *Lecture* , , , 第 卷, *Notes in Computer Science* 348 370 2010 页码 –。Springer，。12. Jed Liu、Owen Arden、Michael D. George 和 Andrew C. Myers。Fabric：以安全为构造原则构建开放式分布式系统。 *Journal of Computer Security*，25 4 5 367 426，2017 (-): –。13. Yi Lu、Raghavendra Kagalavadi Ramesh、Chenyi Zhang 和 Padmanabhan Krishnan。统一访问控制与信息流：适用于由可信代码和不可信代码构成的程序的安全模型。技术报告，Oracle Labs，2016 年 5 月。14. Yi Lu 和 Chenyi Zhang。用于粗粒度信息流控制的非传递安全类型。载于 *Proceedings of the 33rd IEEE Computer Security Foundations Symposium (CSF 2020)* 199 213 2020 ，第 – 页。IEEE，。15. Heiko Mantel 和 David Sands。基于非传递非干扰的受控降密。载于 *Programming Languages and Systems*，计算机科学讲义，第 129–145 页，2004。16. Toby C. Murray、Daniel Matichuk、Matthew Brassil、Peter Gammie、Timothy Bourke、Sean Seefried、Corey Lewis、Xin Gao 和 Gerwin Klein。seL4：从通用系统到信息流强制执行的证明。载于 *IEEE Symposium on Security and Privacy (SP)* 415 429 2013 ，第 – 页，IEEE，。17. Andrew C. Myers。Jflow：实用的近似静态信息流控制。载于 *Proceedings of the 26th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages (POPL)* 228 241 1999 ，第 – 页。ACM，。18. Andrew C. Myers、Andrei Sabelfeld 和 Steve Zdancewic。强制执行稳健降密和限定稳健性。 *Journal of Computer Security* 14 2 157 196, , 2006 ( ): –。19. Andrew W. Roscoe 和 Michael H. Goldsmith。什么是非传递非干扰？载于 *Proceedings of the 12th IEEE Computer Security Foundations Workshop*，第 228–238 页，1999。20. John Rushby。非干扰、传递性与信道控制安全策略。技术报告：计算机科学实验室报告 CSL-92-02，SRI International，1992。21. Andrei Sabelfeld 和 Andrew C. Myers。基于语言的信息流安全。*IEEE Journal on Selected Areas in Communications*，21(1):5–19，2003。22. Andrei Sabelfeld 和 David Sands。降密：维度与原则。*Journal of Computer Security*，17(5):517–548，2009。

23. Geoffrey Smith。《安全信息流分析原理》。载于 Mihai Christodorescu、Somesh Jha、Douglas Maughan、Dawn Song 和 Cliff Wang 编，*Malware Detection* 27 *Advances in Information Security* 291 307 ，第 卷，页码 –。Springer，2007。24. Gregor Snelting 和 Daniel Wasserrab。Volpano/Smith 安全类型系统的正确性证明。 *Archive of Formal Proofs*，2008年9月。https:// isa-afp.org/entries/VolpanoSmith.html，形式化证明开发。25. Deian Stefan、Alejandro Russo、David Mazi`eres 和 John C. Mitchell。析取类别标签。载于 *Nordic Conference on Secure IT Systems (NordSec)*，第 223–239 页，2011。26. Ron van der Meyden。非传递非干扰究竟是什么？载于 *European Symposium On Research In Computer Security* 235 250, 2007 ，页码 –。27. Dennis Volpano、Geoffrey Smith 和 Cynthia Irvine。用于安全流分析的可靠类型系统。 *Journal of Computer Security*，4(2):167–187，1996。28. Dan S. Wallach 和 Edward W. Felten。理解 Java 栈检查。载于 *Proceedings of the 1998 IEEE Symposium on Security and Privacy*，第 52–63 页。IEEE计算机学会，1998。29. Lucas Waye、Pablo Buiras、Dan King、Stephen Chong 和 Alejandro Russo。这是我的特权：控制 DC标签中的降级。载于 *Security and Trust Management (STM)* 9331 *LNCS* 203 219 2015 ，第 卷，页码 –。Springer，。