ПРОГРАММИРОВАНИЕ, *2024*, *№ 1*, *с. 88–99*

ИНФОРМАЦИОННАЯ БЕЗОПАСНОСТЬ

УДК: 004.43

ОПИСАНИЕ СЕМАНТИКИ ЯЗЫКА PARALOCKS В TLA+

© 2024 г. А. А. Тимаков[a], * (ORCID: 0000-0003-4306-789X)

[a]*МИРЭА – Российский технологический университет, 119454 Москва, пр. Вернадского, д. 78*

**E-mail: timakov@mirea.ru*

Поступила в редакцию 15.03.2023
После доработки: 07.04.2023
Принята к публикации: 03.07.2023

Одним из основных аспектов реализации контроля информационных потоков в программном обеспечении является язык описания политик безопасности или меток. Важно, чтобы такой язык позволял формировать политики безопасности элементов среды вычислений на основе правил управления доступом системного уровня. Соответственно язык должен быть достаточно гибким, поскольку на системном уровне могут использоваться разные механизмы: ролевое, мандатное управление доступом и др. Кроме того, в приложении могут действовать некоторые дополнительные ограничения. Наконец, желательно, чтобы язык позволял естественным образом учитывать возможность деклассификации данных (контролируемого раскрытия) в процессе вычислений. Одним из таких языков является *Paralocks*. Работа посвящена реализации семантики несколько упрощенной версии этого языка с использованием *TLA+*. *Paralocks* является языковой частью разработанной с участием автора платформы анализа информационных потоков в хранимых программных блоках баз данных *PLIF*. Приводятся доказательства свойств заданного отношения частичного порядка и решетки, построенной на основе множества всех возможных политик безопасности.

Ключевые слова: контроль потока информации, язык политики безопасности, логическая семантика, формальная верификация, проверка модели, программные модули

DOI: 10.31857/S0132347424010073 **EDN:** HJYWFT

1. ВВЕДЕНИЕ

Статья будет полезна читателям, хорошо знакомым с формальными моделями безопасности компьютерных систем, основанными на управлении доступом и контроле информационных потоков (КИП), обладающим достаточными знаниями в области формальной верификации и навыками работы с соответствующими инструментальными средствами. Необходимую предварительную информацию можно получить, обратившись к источникам: [1]—[5].

Подавляющее число реализованных механизмов контроля информационных потоков [4], [5] при описании множеств политик безопасности и абстрактной семантики информационных потоков опираются на понятие алгебраической решетки. При этом доказательства свойств предлагаемых решеток и корректности операторов вычисления минимальной верхней (максимальной нижней) грани часто не приводятся. Несмотря на кажущуюся очевидность предположений, отсутствие таких доказательств несколько подрывает доверие к реализованным механизмам.

В работе представлена семантика упрощенной версии языка описания политик безопасности *Paralocks* [4], лежащего в основе механизма КИП платформы *PLIF* [6], а также приводятся исчерпывающие пояснения к доказательствам свойств решетки, заданной на множестве всех возможных политик.

Для проверки обозначенных свойств применялся следующий подход. На предварительном этапе осуществлялся прогон модели на небольшом наборе исходных данных с использованием инструмента *TLC*. Отсутствие выявленных на этом этапе противоречий сделало оправданным переход ко второму этапу – доказательству свойств с использованием формальной логической системы *TLAPS*. Указанная система позволяет выстраивать доказательства в иерархическим стиле, который упрощает применение метода естественной дедукции. *TLAPS* опирается на инструменты автоматизированного доказательства теорем, такие как Isabelle [7], Zennon [8]. Описанный подход предоставляет дополнительные гарантии, что достаточно важно в области формальных моделей. По мнению некоторых авторов [9] до трети доказательств, предлагаемых в различных исследованиях, являются некорректными.

Отметим также, что определенной инновацией явилось использование подсистемы проверки типов *Apalache* [10] для доказательства инвариантов типов некоторых структур, используемых для описания

предложений политик безопасности и самих политик в спецификациях *TLA+*. Это, в свою очередь, позволило обоснованно использовать в доказательствах некоторые неочевидные предположения.

2. СЕМАНТИКА PARALOCKS В TLA+

Язык описания политик безопасности в программной среде *Paralocks* [4] является основой перспективной платформы контроля информационных потоков в программных блоках баз данных *PLIF*. Преимущества языка, предопределившие его выбор, описаны в нашей предыдущей статье, опубликованной в этом журнале.

Политика безопасности P_k определяется формулой логики предикатов, представимой в виде конъюнкции определенных дизъюнктов Хорна: $P_k \triangleq C_1 \wedge C_2 \wedge ...$, здесь C_n – предложение политики вида:

$$\forall x_1, ..., x_m \, l_1(\cdot) \wedge l_2(\cdot) \, l_3(\cdot) \ldots Flow(u),$$

где u – связанная переменная или константа, обозначающая пользователя, $Flow(u)$ – предикат, обозначающий легальность потока данных к u, $l_1 \ldots l_n$ — условия (блокировки), выполнение которых требуется для того, чтобы $Flow(u)$ принял значение *TRUE*. Предикаты $l_1 \ldots l_n$ могут быть параметрическими или непараметрическими.

Например, выражение политики: “Поток к произвольному пользователю *x* возможен, если а) открыта блокировка *guest* — для *x* задана роль *guest* – и открыта блокировка *t_expire* – истек заданный интервал времени **или** б) открыта блокировка *manager* — для *x* задана роль *manager*” – имеет формальный вид:

$$\forall x. \, (t\_expire \wedge guest(x) \Rightarrow Flow(x)) \wedge \\ \wedge \, (manager(x) \Rightarrow Flow(x)).$$

Для описания политик в спецификации *Paralocks.tla* [6] вводятся следующие константы: *UU* — множество допустимых имен связанных переменных, *U* — множество актеров (конкретных пользователей системы), E_0 — множество имен непараметрических блокировок, E_1 — множество имен параметрических (с одним параметром) блокировок. Известные варианты использования *Paralocks* для кодирования политик элементов программной среды (переменных, параметров функций, исключений и т.д.), преемственных к наиболее распространенным механизмам управления доступом (мандатному, ролевому), позволяют ограничиться лишь непараметрическими и однопараметрическими блокировками, а также внести иные допущения, упрощающие описание соответствующих сущностей в спецификациях *TLA+*. Положим, что множество имен связанных переменных *UU* включает один элемент, также будем считать, что, если связанная переменная появляется в одной из блокировок некоторого предложения политики, то эта же переменная является аргументом и в литерале Flow(·) того же предложения, и если связанная переменная является аргументом Flow(·) некоторого предложения политики, то все параметрические блокировки того же предложения также будут принимать эту переменную в качестве аргумента.

Множество возможных предложений политик безопасности и множество самих политик определяются как:

$$\begin{array}{l} ClausesSet \;\triangleq \\ \quad \{\langle u, \langle e0, e1\rangle\rangle : u \;\in (U \cup UU), \\ \quad\; e0 \in [E0 \to \text{SUBSET } \{NONE\}], \\ \quad\; e1 \in [E1 \to ((\text{SUBSET } (U) \cup \text{SUBSET } (UU)) \\ \qquad \setminus \{\{\}\}) \cup \{\{NONE\}\}]\} \\ \\ PoliciesSet \;\triangleq \\ \quad \{p \in \text{SUBSET } ClausesSet : \\ \quad\; \forall\, c1 \in p : \neg\exists\, c2 \in p : \\ \qquad \wedge\, c1 \neq c2 \\ \qquad \wedge\; \vee\, (compareClause(c1, c2) \\ \qquad\quad \vee\, compareClause(c2, c1))\} \\ \quad\; \cup \{\{\}\} \end{array}$$

Политика описывается множеством двумерных кортежей. Каждый кортеж – отдельное предложение политики, его первым элементом является связанная переменная или константа, обозначающие пользователя, к которому разрешен информационный поток; второй элемент – это двумерный кортеж, с его помощью описывается множество блокировок, которые должны быть открыты. Элементами этого кортежа являются записи, их поля соответствуют именам блокировок, определяемых константами модели, значения полей задают множества фактических параметров блокировок. В одной политике не может быть двух разных упорядоченных предложений.

Описанный ранее пример политики в спецификациях *PLIF* имеет вид[1]:

[1] Для интерпретации трасс, приводящих к ошибкам, PLIF используется упрощенная нотация, в которой пример выглядит как:

```
x: manager(x)
x: t_expire
```

$$\{\langle x, \langle [t\_expire \mapsto \{\}], [guest \mapsto \{x\},$$
$$reviewer \mapsto \{NONE\}, manager \mapsto \{NONE\},$$
$$organizer \mapsto \{NONE\}]\rangle\}\rangle],$$
$$\langle x, \langle [t\_expire \mapsto \{NONE\}], [guest \mapsto \{NONE\},$$
$$reviewer \mapsto \{NONE\}, manager \mapsto \{x\},$$
$$organizer \mapsto \{NONE\}]\rangle\rangle\}$$

Здесь: $x \in UU$, $t\_expire \in E_0$, {*guest, reviewer, manager, organizer*} $\subseteq E_1$, $NONE$ — специальное значение, которое обозначает, что открытие блокировки не требуется (в *TLA+* записи являются функциями, при этом частично определенные функции не поддерживаются).

Также с учетом заданных ограничений справедливы следующие предположения:

$$\text{ASSUME } U\_ASSM \triangleq U \neq \{\} \wedge NONE \notin U$$

$$\text{ASSUME } UU\_ASSM \triangleq UU = \{\text{“x”}\}$$

$$\text{ASSUME } Clause\_ASSM2 \triangleq$$
$$\forall c \in ClausesSet :$$
$$(\vee \wedge c[1] \in UU$$
$$\wedge \forall e1 \in E1 : c[2][2][e1] = UU$$
$$\vee \wedge c[1] \notin UU$$
$$\wedge \forall e1 \in E1 : c[2][2][e1] \cap UU = \{\})$$

Пояснения к некоторым иным предположениям, заданным в *Paralocks.tla* и используемым в формальных доказательствах, приводятся несколько позже.

На множестве предложений политик безопасности задан оператор сравнения:

$$compareClause(c1, c2) \triangleq$$
$$\wedge substMap3Equality(c1, c2)$$
$$\wedge \forall k \in E0 : \vee c1[2][1][k] = c2[2][1][k]$$
$$\vee c2[2][1][k] = \{\}$$
$$\wedge \forall e \in E1 : \vee c1[2][2][e] = \{NONE\}$$
$$\vee matchLocks(c1, c2, e)$$
$$\subseteq c2[2][2][e]$$

Оператор $compareClause(c1, c2)$ возвращает значение *TRUE* – предложение $c2$ является как минимум таким же строгим, как и предложение $c1$ — если выполняются два условия: а) существует подстановка, приводящая предикат *Flow*(·) предложения $c1$ к виду, идентичному предикату *Flow*(·) предложения $c2$; б) после применения соответствующей подстановки к блокировкам предложения $c1$ множество блокировок предложения $c1$ будет представлять подмножество блокировок предложения $c2$. Логическая интерпретация заданного таким образом отношения частичного порядка приводится в [4].

Параметром предиката *Flow*(·) может выступать связанная переменная (элемент множества *UU*) или конкретный пользователь (элемент множества *U*), поэтому истинность первого условия определяется результатом вычисления функции:

$$substMap3Equality(c1, c2) \triangleq$$
$$c1[1] \in UU \vee c1[1] = c2[1]$$

В принятой нотации вторым элементом предложения политики является двумерный кортеж записей, определяющих необходимые наборы непараметрических и параметрических блокировок. Если открытие некоторой непараметрической блокировки не требуется, соответствующее поле записи принимает значение {*NONE*}, в противном случае – {}. Поэтому вхождение множества непараметрических блокировок предложения $c1$ во множество непараметрических блокировок предложения $c2$ фактически означает:

$$\forall k \in E0 : \vee c1[2][1][k] = c2[2][1][k]$$
$$\vee c2[2][1][k] = \{\}$$

Для однопараметрических блокировок аналогично – значение {*NONE*} в поле соответствующей блокировки означает, что открытие блокировки не требуется. Если поле блокировки содержит не пустое множество элементов, отличных от {*NONE*}, значит, легитимность информационного потока к пользователю, заданному предикатом *Flow*(·), требует открытия некоторых экземпляров этой блокировки (определяются указанным множеством элементов). Функция *matchLocks*(·) применяет при необходимости соответствующую подстановку к выражениям однопараметрических блокировок предложения $c1$:

$$matchLocks(c1, c2, e) \triangleq$$
$$\text{LET } c \triangleq c1[2][2][e]$$
$$\text{IN}$$
$$\text{IF } c \cap UU \neq \{\}$$
$$\text{THEN } (c \setminus UU) \cup \{c2[1]\}$$
$$\text{ELSE } c$$

Для сравнения политик безопасности используется функция *comparePol*(·), ее определение тривиально:

$$comparePol(p1, p2) \triangleq \forall\, c2 \in p2 : (\exists\, c1 \in p1 : compareClause(c1, c2))$$

Далее в пояснениях к выражениям *TLAPS* в некоторых случаях вместо *compareClause*(·) и *comparePol*(·) будем использовать оператор.

Наиболее важными правилами семантиками языка *Paralocks* являются правила вычисления наименьшей верхней (НВГ) и наибольшей нижней граней (ННГ) двух заданных политик безопасности. ННГ двух произвольных политик безопасности $p1$ и $p2$ является некоторая максимально строгая политика p_{GLB}, такая, что $p_{\text{GLB}} \subseteq p1$ и $p_{\text{GLB}} \subseteq p2$. Поскольку каждое предложение политики представляет собой отдельную возможность возникновения разрешенного информационного потока, очевидно, справедливо:

$$GLB(p1, p2) \triangleq p1 \cup p2$$

НВГ двух произвольных политик безопасности $p1$ и $p2$ является некоторая минимально строгая политика p_{LUB}, такая, что $p1 \sqsubseteq p_{LUB}$ и $p2 \sqsubseteq p_{LUB}$. Для ее вычисления необходимо для всех возможных пар $\langle c1, c2\rangle$, таких, что $c1 \in p1$, $c2 \in p2$, и существует подстановка, приводящая предикат *Flow*(·) предложения $c1$ к виду, идентичному предикату *Flow*(·) предложения $c2$ или, наоборот, предикат *Flow*(·) предложения $c2$ — к виду, идентичному предикату *Flow*(·) предложения $c1$, сгенерировать новое предложение политики, применив соответствующую подстановку к блокировкам предложения $c1$ или $c2$ и объединив после этого полученные множества блокировок исходных предложений:

$$LUB(p1, p2) \triangleq \{x \in \{unionCl(c1, c2) : c1 \in p1, c2 \in p2\} : x \neq \langle\rangle\}$$

Здесь функция *unionCl*($c1, c2$) вычисляет новое предложение политики p_{LUB} на основе предложений $c1$ и $c2$, если существует соответствующая подстановка, или возвращает пустое предложение $\langle\rangle$:

```
unionCl(c1, c2) ≜
  LET capMap ≜ [e0 ∈ E0 ↦
      c1[2][1][e0] ∩ c2[2][1][e0]]
  IN
    IF substMap3Equality(c1, c2)
      THEN ⟨c2[1], ⟨capMap,
             [e1 ∈ E1 ↦
             IF  ∧ NONE ∈ c1[2][2][e1]
                 ∧ NONE ∈ c2[2][2][e1]

                 THEN {NONE}
                 ELSE (matchLocks(c1, c2, e1)
                   ∪ c2[2][2][e1]) \ {NONE}]⟩⟩
      ELSE
    IF substMap3Equality(c2, c1)
      THEN ⟨c1[1], ⟨capMap,
             [e1 ∈ E1 ↦
             IF  ∧ NONE ∈ c1[2][2][e1]
                 ∧ NONE ∈ c2[2][2][e1]
                 THEN {NONE}
                 ELSE (matchLocks(c2, c1, e1)
                   ∪ c1[2][2][e1]) \ {NONE}]⟩⟩
      ELSE ⟨⟩
```

Локальная функция *capMap* объединяет непараметрические блокировки предложений $c1$ и $c2$ (поле, соответствующее непараметрической блокировке, принимает значение {*NONE*}, если блокировка не требуется, в противном случае — {}).

Для доказательства некоторых свойств с использованием *TLAPS*, как уже отмечалось, удобно использовать допущения – предположения (assumptions). Доказательство их справедливости иногда является нетривиальной задачей, а порой невозможной, поскольку *TLAPS* имеет ряд ограничений. Например, в системе отсутствует полноценная поддержка множеств, заданных с использованием кортежей. Проверка таких свойств с использованием механизма проигрывания моделей на неполных наборах данных не дает формальных гарантий. Для инвариантов типов проблема, как показывает исследование, может быть частично решена с использованием инструмента вывода типов *Snowcat*, который является частью платформы *Apalache* [10]. Еще одним предположением, используемым в доказательствах, является *LUB_PS*. Его обоснованность подтверждается проверкой с использованием *Snowcat*. Стоит отметить, однако, что использование указанного инструмента как правило требует специальной разметки спецификации:

@*typeAlias*: $CLAUSE = \langle U, \langle E0 \to Set(U), E1 \to Set(U)\rangle\rangle$
@*typeAlias*: $POLICY = Set(CLAUSE)$;

@type: $Set(CLAUSE)$;
$ClausesSet \triangleq \{\ldots\}$

@type: $Set(POLICY)$;
$PoliciesSet \triangleq \{\ldots\}$

@type: $(POLICY, POLICY) \implies POLICY$;
$LUB(p1, p2) \triangleq \{x \in \{unionCl(c1, c2) : c1 \in p1, c2 \in p2\} : x \neq EC\}$

ASSUME $LUB\_PS \triangleq \forall p1, p2 \in PoliciesSet : LUB(p1, p2) \in PoliciesSet$

Ряд иных тривиальных предположений, описанных в *Paralocks.tla*, в данной работе не разбирается.

3. ПРОВЕРКА СВОЙСТВ РЕШЕТКИ, ЗАДАННОЙ НА МНОЖЕСТВЕ ПОЛИТИК БЕЗОПАСНОСТИ

Процесс вывода формальных доказательств необходимых свойств алгебраических структур, описанных в спецификации *Paralocks.tla*, с использованием логической системы *TLAPS* является весьма трудоемким, поэтому на начальном этапе указанные свойства проверялись для ограниченного набора данных с использованием инструмента оценки выражений над константами среды *TLA+Toolbox*, а в некоторых случаях и с использованием инструмента проигрывания моделей *TLC*. Результаты предварительного этапа в данной работе не приводятся, однако их можно обнаружить в файлах проекта [6]. Представим общий вид доказательства того, что функция *comparePol*(·) является отношением частичного порядка на множестве политик *PoliciesSet*.

Рефлексивность

Проверку удобно разбить на два случая: когда множество предложений произвольной политики является пустым и когда данное множество не является пустым.

THEOREM $Reflexitivity \triangleq \forall p \in PoliciesSet : comparePol(p, p)$
⟨1⟩ USE DEF *comparePol*, *compareClause*, *substMap3Equality*, *matchLocks*
⟨1⟩1 TAKE $pol \in PoliciesSet$
⟨1⟩2 CASE pol = { }
...
⟨1⟩3 CASE pol ≠ { }
...
⟨1⟩4. QED
BY ⟨1⟩2, ⟨1⟩3

В первом случае очевидно:

$\forall c2 \in pol : (\exists c1 \in pol : \text{compareClause}(c1,c2))$,

доказательство следует из определения *comparePol*(·).

Во втором случае требуется для произвольного $c1 \in pol$ доказать истинность *compareClause*(*c*1,*c*1) или с учетом определения *compareClause*(·):

...
⟨1⟩3.CASE $pol \neq \{\}$
⟨2⟩1 TAKE $c1 \in pol$
...
⟨2⟩3 $substMap3Equality(c1, c1)$
...
⟨2⟩4 $\forall k \in E0 : \lor c1[2][1][k] = c1[2][1][k] \lor c1[2][1][k] = \{\}$
...
⟨2⟩5 $\forall e \in E1 : \lor c1[2][2][e] = \{NONE\} \lor matchLocks(c1, c1, e) \subseteq c1[2][2][e]$
...
⟨2⟩6 QED
BY ⟨2⟩3, ⟨2⟩4, ⟨2⟩5
...

Доказательство шага ⟨2⟩ 3 очевидно, оно следует из определения *substMap3Equality*(·) и истинности $c1[1] = c1[1]$. Шаг ⟨2⟩ 4 также не требует доказательства. Для доказательства шага ⟨2⟩ 3 интересен случай, когда $c1[2][2][e] \neq NONE$.

...
⟨2⟩5 $\forall e \in E1 : \lor c1[2][2][e] = \{NONE\} \lor matchLocks(c1, c1, e) \subseteq c1[2][2][e]$
⟨3⟩1 TAKE $e \in E1$
⟨3⟩2 CASE $c1[2][2][e] = \{NONE\}$
⟨4⟩1 $c1[2][2][e] = \{NONE\}$
PROOF BY ⟨3⟩2
⟨4⟩2 QED BY ⟨4⟩1
⟨3⟩3 CASE c1 [2] [2] [e] ≠ { NONE }
...
⟨3⟩4 QED BY ⟨3⟩2, ⟨3⟩1, ⟨3⟩3

```
...
```

В этом случае требуется доказательство истинности *matchLocks*(*c*1, *c*1, *e*) ⊆ *c*1[2][2][*e*]. Если *c*1[2][2][*e*] ∩ *UU* = {}, то доказательство следует из определения *matchLocks* (·) и истинности c1[2][2][*e*] ⊆ ⊆ *c*1[2][2][*e*]:

```
...
⟨4⟩1 matchLocks(c1, c1, e) ⊆ c1[2][2][e]
    ⟨5⟩1 CASE c1[2][2][e] ∩ UU = {}
        ⟨6⟩1 c1[2][2][e] ⊆ c1[2][2][e]
            PROOF OBVIOUS
        ⟨6⟩2 QED BY ⟨5⟩1, ⟨6⟩1 DEF matchLocks
...
```

Для случая c1[2][2][e] ∩ UU ≠ {} доказательство легко может быть получено следующим образом:

```
...
⟨4⟩1 matchLocks(c1, c1, e) ⊆ c1[2][2][e]
    ⟨5⟩1 CASE c1[2][2][e] ∩ UU = {}
        ...
    ⟨5⟩2 CASE c1[2][2][e] ∩ UU ≠ {}
        ⟨6⟩1 c1 ∈ ClausesSet
            PROOF BY ONLY DEF PoliciesSet
        ⟨6⟩2 {c1[1]} = UU
            PROOF BY ⟨5⟩2, ⟨6⟩1,
                     Clause_ASSM2, UU_ASSM
        ⟨6⟩3 QED BY ⟨5⟩2, ⟨6⟩2
...
```

Транзитивность

В виде теоремы в *TLAPS* свойство формулируется следующим образом:

```
THEOREM Transitivity ≜
          ∀ p1, p2, p3 ∈ PoliciesSet :
            ∧ comparePol(p1, p2)
            ∧ comparePol(p2, p3)
                    ⟹ comparePol(p1, p3)
...
```

Выбрав *p*1, *p*2, *p*3 из *PoliciesSet* и положив: *comparePol*(*p*1,*p*2) ∧ *comparePol* (*p*2,*p*3), необходимо доказать: ∀*c*3 ∈ *p*3: (∃*c*1 ∈ *p*1 : *compareClause*(*c*1,*c*3))[2]:

[2]Доказательство опирается на определения функций: *ComparePol*(·), *CompareClause*(·), *substMap*3*Equality*(·) и *MatchLocks*(·).

```
...
⟨1⟩ USE DEF comparePol, compareClause,
            substMap3Equality, matchLocks
⟨1⟩1 TAKE p1 ∈ PoliciesSet, p2 ∈ PoliciesSet,
          p3 ∈ PoliciesSet
⟨1⟩3 HAVE ∧ comparePol(p1, p2)
          ∧ comparePol(p2, p3)
⟨1⟩4 ∀ c3 ∈ p3 : (∃ c1 ∈ p1 :
                  compareClause(c1, c3))
    ...
⟨1⟩5 QED BY ⟨1⟩4
```

Очевидно, для любого *c*3 ∈ *p*3 можно найти такие *c*2 ∈ *p*2 и *c*1 ∈ *p*1, что справедливым будет: *c*2⊑ *c*3 ∧ *c*1⊑ c2. Тогда для доказательства теоремы достаточно выбрать соответствующие *c*1, *c*2, *c*3 и доказать справедливость *c*1⊑ *c*3 или с учетом определения *substMap*3*Equality*(·):

```
...
⟨2⟩1 TAKE c3 ∈ p3
    ⟨2⟩2 PICK c2 ∈ p2 : compareClause(c2, c3)
        PROOF BY ⟨1⟩3
    ⟨2⟩3 PICK c1 ∈ p1 : compareClause(c1, c2)
        PROOF BY ⟨1⟩3
    ⟨2⟩4 c1 [ 1 ] ∈ UU ∨ c1 [ 1 ] = c3 [ 1 ]
        PROOF BY ⟨2⟩3 , ⟨2⟩2
    ⟨2⟩5 ∀ k ∈ E0 : ∨ c1[2][1][k] = c3[2][1][k]
                    ∨ c3[2][1][k] = {}
        PROOF BY ⟨2⟩3, ⟨2⟩2
    ⟨2⟩6 ∀ e ∈ E1 : ∨ c1[2][2][e] = {NONE}
                    ∨ matchLocks(c1, c3, e)
                        ⊆ c3[2][2][e]
        ...
    ⟨2⟩7 QED BY ⟨2⟩4, ⟨2⟩5, ⟨2⟩6
...
```

Доказательства шагов ⟨2⟩ 4 и ⟨2⟩ 5 следуют из истинности ⟨2⟩ 2, ⟨2⟩ 3 и определений: *compareClause*(·), *substMap*3*Equality*(·). Для доказательства шага ⟨2⟩ 6 интересен случай, когда *c*1[2][2][*e*] ≠ NONE:

```
...
⟨2⟩6 ∀ e ∈ E1 : ∨ c1[2][2][e] = {NONE}
                ∨ matchLocks(c1, c3, e)
                    ⊆ c3[2][2][e]
```

⟨3⟩1 TAKE $e \in E1$
⟨3⟩2 CASE $c1[2][2][e] = \{NONE\}$
PROOF BY ⟨3⟩2
⟨3⟩3 CASE $c1[2][2][e] \neq \{NONE\}$
...
⟨3⟩4 QED BY ⟨3⟩1, ⟨3⟩2, ⟨3⟩3
...

В этом случае требуется доказательство истинности $matchLocks(c1,c3,e) \subseteq c3[2][2][e]$. Если $c1[2][2][e] \cap UU \neq \{\}$, доказательство может быть получено следующим образом:

...
⟨4⟩2 CASE $c1[2][2][e] \cap UU \neq \{\}$
⟨5⟩1 $((c1[2][2][e] \setminus UU) \cup \{c2[1]\}) \subseteq c2[2][2][e]$
PROOF BY ⟨4⟩2, ⟨3⟩3, ⟨2⟩3
⟨5⟩2 $((c1[2][2][e] \setminus UU) \cup \{c3[1]\})$
$\subseteq ((c2[2][2][e] \setminus UU) \cup \{c3[1]\})$
PROOF BY ⟨5⟩1
⟨5⟩3 $c2 \in ClausesSet$
PROOF BY ONLY DEF $PoliciesSet$
⟨5⟩4 $c2[1] \neq NONE$
PROOF BY ⟨5⟩3, U_ASSM, UU_ASSM, $Clause\_ASSM3$
⟨5⟩5 $c2[2][2][e] \neq \{NONE\}$
PROOF BY ⟨5⟩4, ⟨5⟩1
⟨5⟩6 QED BY ⟨5⟩5, ⟨5⟩2, ⟨2⟩2, ⟨2⟩3
⟨4⟩3 CASE $c1[2][2][e] \cap UU = \{\}$
...
⟨4⟩4 QED BY ⟨4⟩3, ⟨4⟩2
...

Для случая $c1[2][2][e] \cap UU = \{\}$ доказательство выводится аналогично.

Антисимметричность

Свойство антисимметричности отношения частичного порядка на множестве политик безопасности *PoliciesSet* можно выразить следующим образом:

THEOREM $Antisymmetry \triangleq$
$\forall p1, p2 \in PoliciesSet :$
$\wedge\ comparePol(p1, p2)$
$\wedge\ comparePol(p2, p1) \implies p1 = p2$

Общая схема доказательства не требует пояснений:

...
⟨1⟩ USE DEF $comparePol$, $compareClause$, $substMap3Equality$, $matchLocks$
⟨1⟩1 TAKE $p1 \in PoliciesSet, p2 \in PoliciesSet$
⟨1⟩2 HAVE $\wedge\ comparePol(p1, p2) \wedge comparePol(p2, p1)$
⟨1⟩4 $p1 = p2$
⟨2⟩1 $\forall c2 \in p2 : \exists c1 \in p1 : c2 = c1$
⟨2⟩2 $\forall c1 \in p1 : \exists c2 \in p2 : c1 = c2$
...
⟨2⟩5 QED BY ⟨2⟩1, ⟨2⟩2
⟨1⟩5 QED BY ⟨1⟩4

Доказательства шагов ⟨2⟩ 1 и ⟨2⟩ 2, очевидно, идентичны.

Рассмотрим далее структуру доказательства шага ⟨2⟩ 1. Предложение политики безопасности c, как уже отмечалось, состоит из трех компонент: предиката $Flow(\cdot) - c[1]$ (представлен параметром предиката), множества непараметрических блокировок и множества параметрических блокировок — $c[2][1]$ и $c[2][2]$ (имеют форму записей, в которых поля соответствуют наименованиям блокировок, значения — параметрам). Поэтому доказательство сводится к проверке эквивалентности соответствующих элементов:

...
⟨2⟩1 $\forall c2 \in p2 : \exists c1 \in p1 : c2 = c1$
...
⟨3⟩7 $c1[1] = c2[1]$
...
⟨3⟩8 $c1[2][1] = c2[2][1]$
...
⟨3⟩9 $c1[2][2] = c2[2][2]$
...
⟨3⟩10 QED BY ⟨3⟩7, ⟨3⟩8, ⟨3⟩9
DEF $PoliciesSet$, $ClausesSet$
...

На начальном этапе возьмем произвольное $c2$ из $p2$ и выберем такие $c1$ из $p1$ и $c3$ из $p2$, что $c1 \sqsubseteq c2$ и $c3 \sqsubseteq c1$. Докажем, что $c3 = c2$:

...
⟨3⟩1 TAKE $c2 \in p2$
⟨3⟩3 PICK $c1 \in p1 : compareClause(c1, c2)$
PROOF BY ⟨1⟩2
⟨3⟩4 PICK $c3 \in p2 : \wedge\ compareClause(c3, c1)$
PROOF BY ⟨1⟩2

```
    ...
⟨3⟩6 c3 = c2
...
```

Поскольку выражение политики безопасности не может содержать двух различных предложений $c1$ и $c2$, таких, что $c1 \sqsubseteq c2$ или $c2 \sqsubseteq c1$, для доказательства шага $\langle 3 \rangle 6$ достаточно доказать, что $c3 \sqsubseteq c2$:

```
...
  ⟨3⟩5 compareClause(c3, c2)
      ...
  ⟨3⟩6 c3 = c2
      PROOF BY ⟨3⟩5  DEF PoliciesSet
...
```

Истинность первых двух конъюнктов выражения *compareClause*($c3$,$c2$) непосредственно следует из истинности шагов: $\langle 3 \rangle$ 3, $\langle 3 \rangle$ 4 ($c1 \sqsubseteq c2 \wedge c3 \sqsubseteq c1$), определения *compareClause*($\cdot$) и предположения о том, что для любого c, принадлежащего *ClausesSet*, справедливо: $c[2][1] \in [E0 \to \{\{NONE\},\{\}\}]$:

```
...
⟨3⟩5 compareClause(c3, c2)
    ⟨4⟩1 c3[1] ∈ UU ∨ c3[1] = c2[1]
        PROOF BY ⟨3⟩3, ⟨3⟩4
    ⟨4⟩2 ∀ e ∈ E0 :
          ∧ ∨ c2[2][1][e] = c3[2][1][e]
            ∨ c2[2][1][e] = {}
          PROOF BY ⟨3⟩3, ⟨3⟩4, Clause_ASSM4
    ⟨4⟩3 ∀ e ∈ E1 : ∨ c3[2][2][e] = {NONE}
                    ∨ matchLocks(c3, c2, e)
                        ⊆ c2[2][2][e]
    ...
    ⟨4⟩4 QED BY ⟨4⟩1, ⟨4⟩2, ⟨4⟩3
...
```

Для доказательства истинности последнего условия выражения *compareClause*($c3$,$c2$) необходимо рассмотреть три случая:

```
...
⟨4⟩3 ∀ e ∈ E1 : ∨ c3[2][2][e] = {NONE}
                ∨ matchLocks(c3, c2, e)
                    ⊆ c2[2][2][e]
    ⟨5⟩1 TAKE e ∈ E1
    ⟨5⟩2 CASE c3[2][2][e] = {NONE}
        PROOF BY ⟨5⟩2
    ⟨5⟩3 CASE c3[2][2][e] ∩ UU = {}
              ∧ c3[2][2][e] ≠ {NONE}
    ...
    ⟨5⟩4 CASE c3[2][2][e] ∩ UU ≠ {}
              ∧ c3[2][2][e] ≠ {NONE}
    ...
    ⟨5⟩5 QED BY ⟨5⟩2, ⟨5⟩3, ⟨5⟩4
                DEF matchLocks
...
```

Доказательство шага $\langle 5 \rangle$ 2 тривиально. Доказательства шагов: $\langle 5 \rangle$ 3 и $\langle 5 \rangle$ 4 в полном объеме для краткости не приводятся. Отметим лишь, они основаны на том факте, что $c1 \sqsubseteq c2 \wedge c3 \sqsubseteq c1$, а также на предположениях: *U_ASSM* и *UU_ASSM*.

Опираясь на определения *compareClause*($\cdot$) и *substMap3Equality*($\cdot$), предположение о том, что множество допустимых связанных переменных состоит из одного элемента (*UU_ASSM*), а также на истинность шагов: $\langle 3 \rangle$ 3, $\langle 3 \rangle$ 4 и $\langle 3 \rangle$ 6 имеем: ($c3[1] = x \vee$ $\vee c3[1] = c1[1]) \wedge (c1[1] = x \vee c1[1] = c2[1]) \wedge c2[1] =$ $= c3[1]$. Отсюда следует истинность шага $\langle 3 \rangle$ 7 — $c1[1] = c2[1]$. Доказательство шага $\langle 3 \rangle$ 8 — $c1[2][1] =$ $= c2[2][1]$ — является также тривиальным.

```
...
⟨3⟩7 c1[1] = c2[1]
    PROOF BY ⟨3⟩3, ⟨3⟩4, ⟨3⟩6, UU_ASSM
⟨3⟩8 c1[2][1] = c2[2][1]
    ⟨4⟩1 ∀ e0 ∈ E0 : c1[2][1][e0] = c2[2][1][e0]
        PROOF BY ⟨3⟩3, ⟨3⟩4, ⟨3⟩6
    ⟨4⟩2 QED BY ⟨4⟩1, Clause_ASSM4
            DEF PoliciesSet
...
```

Остается доказать истинность шага $\langle 3 \rangle$ 9 — $c1[2][2] = c2[2][2]$. Для этого достаточно выбрать произвольное значение $e1$ из $E1$ и доказать: $c1[2][2]$ $[e1] = c2[2][2][e1]$. Общая структура доказательства этого шага может быть представлена как:

```
...
⟨3⟩9 c1[2][2] = c2[2][2]
    ⟨4⟩1 SUFFICES ASSUME NEW e1 ∈ E1
        PROVE c1[2][2][e1] = c2[2][2][e1]
        PROOF BY Clause_ASSM1
            DEF PoliciesSet
    ⟨4⟩2 CASE c2[2][2][e1] = {NONE}
        ...
```

```
⟨4⟩3 CASE  ∧ c2[2][2][e1] ∩ UU = {}
           ∧ c2[2][2][e1] ≠ {NONE}
           ...
⟨4⟩4 CASE  ∧ c2[2][2][e1] ∩ UU ≠ {}
           ∧ c2[2][2][e1] ≠ {NONE}
           ...
⟨4⟩5 QED BY ⟨4⟩2, ⟨4⟩3, ⟨4⟩4
...
```

Если $c2[2][2][e1] = NONE$, то из $c1 \sqsubseteq c2$, определений *compareClause*(·) и *matchLocks*(·), предположений о представлении параметрических блокировок в предложениях политик безопасности, допустимых множествах связанных переменных и актеров следует:

```
...
⟨4⟩2 CASE c2[2][2][e1] = {NONE}
    ⟨5⟩1 ∨ c1[2][2][e1] = {NONE}
         ∨ (c1[2][2][e1] \ UU)
           ∪ {c2[1]}  ⊆ {NONE}
         ∨ c1[2][2][e1] ⊆ {NONE}
         PROOF BY ⟨3⟩3, ⟨4⟩2, Clause_ASSM1,
                  U_ASSM, UU_ASSM
                  DEF matchLocks
    ⟨5⟩2 c1 ∈ ClausesSet ∧ c2 ∈ ClausesSet
         PROOF BY  DEF PoliciesSet
    ⟨5⟩3 QED BY ⟨4⟩2, ⟨5⟩1, ⟨5⟩2,
                Clause_ASSM3, Clause_ASSM1,
                U_ASSM, UU_ASSM
...
```

Вывод доказательства для случая ⟨4⟩ 2 далее тривиален. Для шагов ⟨4⟩ 3 и ⟨4⟩ 4 с целью сохранить краткость изложения далее приводится лишь общая структура вывода, для которого ключевыми являются факты, определенные шагами: ⟨3⟩ 3, ⟨3⟩ 4 и ⟨3⟩ 6.

```
...
⟨4⟩3 CASE  ∧ c2[2][2][e1] ∩ UU = {}
           ∧ c2[2][2][e1] ≠ {NONE}
    ⟨5⟩1 ∨ (c1[2][2][e1] \ UU) ∪ {c2[1]}
           ⊆ c2[2][2][e1]
         ∨ c1[2][2][e1] ⊆ c2[2][2][e1]
         ...
    ⟨5⟩2 CASE c1[2][2][e1] ⊆ c2[2][2][e1]
         ...
    ⟨5⟩3 CASE (c1[2][2][e1] \ UU)
              ∪ {c2[1]} ⊆ c2[2][2][e1]
         ...
    ⟨5⟩4 QED BY ⟨5⟩1, ⟨5⟩2, ⟨5⟩3
⟨4⟩4 CASE  ∧ c2[2][2][e1] ∩ UU ≠ {}
           ∧ c2[2][2][e1] ≠ {NONE}
         ...
    ⟨5⟩9  c2[2][2][e1] ⊆ c1[2][2][e1]
         ...
    ⟨5⟩10 c1[2][2][e1] ⊆ c2[2][2][e1]
         ...
    ⟨5⟩11 QED BY ⟨5⟩9, ⟨5⟩10, UU_ASSM
...
```

Полное доказательство транзитивности, заданного на множестве *PoliciesSet* отношения *comparePol*(·), можно найти в [6].

Решетка

Предположим, что заданная ранее функция $LUB(\cdot)$ действительно является функцией нахождения наименьшей верхней грани двух политик безопасности, и сформулируем в виде теоремы утверждение о том, что множество *PoliciesSet* с заданным на нем отношением частичного порядка *comparePol*(·) представляет собой алгебраическую решетку:

```
THEOREM ParalocksLattice ≜
        ∀ p1, p2 ∈ PoliciesSet :
          ∧ comparePol(p1, LUB(p1, p2))
          ∧ comparePol(p2, LUB(p1, p2))
          ∧ ∀ y ∈ PoliciesSet :
            ∧ comparePol(p1, y)
            ∧ comparePol(p2, y)
              ⟹ comparePol(LUB(p1, p2), y)
```

Отметим, даже на малых наборах данных (две параметрических блокировки, одна непараметрическая блокировка, один конкретный пользователь во множестве *U*, одна связанная переменная во множестве *UU*) проверить справедливость теоремы с использованием инструмента оценки выражений над константами среды *TLA + Toolbox* не удалось. На предварительном этапе пришлось воспользоваться инструментом проигрывания моделей *TLC*. В качестве переменных были выбраны: *Policies2Set* — множество всех возможных пар политик из множества *PoliciesSet*, *curSet* — текущая пара политик. *LUB*(*p*1, *p*2) – возвращает для пары политик *p*1 и *p*2 минимальную верхнюю грань, если она существует.

Ввиду того, что иерархическое доказательство является достаточно объемным, ограничимся лишь

его общей структурой. Этапы доказательства очевидны:

```
⟨1⟩1  TAKE p1, p2 ∈ PoliciesSet
⟨1⟩2  LUB(p1, p2) ∈ PoliciesSet
      PROOF BY LUB_PS
⟨1⟩3  PICK l ∈ PoliciesSet : l = LUB(p1, p2)
      PROOF BY ⟨1⟩2
⟨1⟩4  ∀ l1 ∈ l :
        ∧ ∃ c1 ∈ p1 : compareClause(c1, l1)
        ∧ ∃ c2 ∈ p2 : compareClause(c2, l1)
⟨1⟩5  ∀ y ∈ PoliciesSet :
        comparePol(p1, y) ∧ comparePol(p2, y)
                          ⟹ comparePol(l, y)
⟨1⟩6  QED BY ⟨1⟩3, ⟨1⟩4, ⟨1⟩5  DEF comparePol
```

Доказательство шага ⟨1⟩ 4 строится на том факте, что любое предложение политики *l* является результатом применения некоторой подстановки к блокировкам предложения *c*1 или *c*2 с последующим объединением блокировок этих предложений — *unionCl*(*c*1,*c*2), где *c*1 ∈ *p*1 и *c*2 ∈ *p*2. Для доказательства шага ⟨1⟩ 5 в общем виде необходимо выбрать произвольную политику *y* ∈ *PoliciesSet*, произвольное предложение *y*1 ∈ *y* и доказать: ∃*c*1, *c*2, *l*1 : *c*1 ∈ *p*1 ∧ *c*2 ∈ *p*2 ∧ *l*1 ∈ *l* (*c*1 ⊑ *y*1 ∧ *c*2 ⊑ *y*1 => *l*1 ⊑ *y*1):

```
...
⟨1⟩5 ∀ y ∈ PoliciesSet :
     comparePol(p1, y) ∧ comparePol(p2, y)
                       ⟹ comparePol(l, y)
   ⟨2⟩1 TAKE y ∈ PoliciesSet
   ⟨2⟩2 SUFFICES ∀ y1 ∈ y :
          ( ∧ ∃ c1 ∈ p1 : compareClause(c1, y1)
            ∧ ∃ c2 ∈ p2 : compareClause(c2, y1)) ⟹
                ∃ l1 ∈ l : compareClause(l1, y1)
          PROOF BY ⟨1⟩3  DEF comparePol
   ⟨2⟩3 TAKE y1 ∈ y
   ⟨2⟩4 HAVE ∧ ∃ c1 ∈ p1 : compareClause(c1, y1)
             ∧ ∃ c2 ∈ p2 : compareClause(c2, y1)
   ⟨2⟩5 PICK c1 ∈ p1 : compareClause(c1, y1)
        PROOF BY ⟨2⟩4
   ⟨2⟩6 PICK c2 ∈ p2 : compareClause(c2, y1)
        PROOF BY ⟨2⟩4
...
```

Далее необходимо сгенерировать некоторое предложение *u* = *unionCl*(*c*1, *c*2) и доказать: а) *u* принадлежит политике *l* = *LUB*(*c*1, *c*2) и б) *u* ⊑ *y*1. Условие б) требует декомпозиции на три подусловия в соответствии с определением *compareClause*(·):

```
...
⟨2⟩9 PICK u : u = unionCl ( c1 , c2 )
     PROOF OBVIOUS
...
⟨2⟩13 u ∈ l
      PROOF BY ...
...
⟨2⟩16 u [ 1 ] ∈ UU ∨ u [ 1 ] = y1 [ 1 ]
      PROOF BY ...
...
⟨2⟩24 ∀ k ∈ E0 : ∨ u[2][1][k] = y1[2][1][k]
                 ∨ y1 [ 2 ] [ 1 ] [ k ] = { }
      PROOF BY ...
⟨2⟩25 ∀ e ∈ E1 :
      ∨ u[2][2][e] = {NONE}
      ∨ matchLocks(u, y1, e) ⊆ y1[2][2][e]
      ⟨3⟩1 TAKE e ∈ E1
      ⟨3⟩2 CASE u[2][2][e] = {NONE}
           PROOF BY ⟨3⟩2
      ⟨3⟩3 CASE ∧ u[2][2][e] ≠ {NONE}
                ∧ u[2][2][e] ∩ UU = {}
           ...
      ⟨3⟩4 CASE ∧ u[2][2][e] ≠ {NONE}
                ∧ u[2][2][e] ∩ UU ≠ {}
           ...
      ⟨3⟩5 QED BY ⟨3⟩2, ⟨3⟩3, ⟨3⟩4
⟨2⟩26 QED BY ⟨2⟩16, ⟨2⟩13, ⟨2⟩24, ⟨2⟩25
        DEF compareClause, substMap3Equality
...
```

Здесь доказательство шага ⟨2⟩ 25 является наиболее трудоемким и требует рассмотрения трех случаев. В целом логический вывод доказательств для каждого из них осуществляется в таком же стиле, что и выводы доказательств для аналогичных этапов ранее рассмотренных утверждений – последовательная проверка выражений функции *compareClause*(·), отвечающих за сравнение параметрических блокировок предложений политики безопасности, с опорой на определенный набор обоснованных предположений.

4. ЗАКЛЮЧЕНИЕ

Контроль информационных потоков стоит в одном ряду с некоторыми иными [12], динамично развивающимися направлениями теории языков программирования. Работа является продолжением серии публикаций, посвященных разработанной с участием автора технологии контроля информационных потоков в программном обеспечении автоматизированных информационных систем уровня

предприятия. В отличие от известных подобных технологий, основанных на методах статического и динамического анализа программного обеспечения и требующих от программистов решения дополнительных нетривиальных задач, связанных с разметкой исходного кода, а также интерпретацией результатов анализа, разрабатываемая технология позволяет строго разделить функции написания кода и контроля корректности его логики с учетом специфики предметной области и с привязкой к принятой в системе политике безопасности.

Представленное исследование обладает самостоятельной ценностью, поскольку дает вариант описания политик *Paralocks*, имеющих внедрение в *Java* (проект *Paragon* [13], [11]), в формальном языке *TLA+*, который часто применяется для моделирования программ и проверки их свойств. Таким образом, в работе делается шаг в сторону создания применимой на практике платформы КИП, основанной на методах формальной верификации и проигрывания моделей.

Сама идея проверки безопасности информационных потоков на основе методов формальной верификации ранее выдвигалась и получила определенное признание [14]. Однако в указанной работе проводились только теоретические исследования, не привязанные к конкретному языку программирования, алфавит политик безопасности в ней представлен простой двухуровневой решеткой $\langle\{\text{Hi,Low}\}, \subseteq\rangle$.

Полученные в рамках исследования доказательства формальных свойств решетки выражений, составляющих алфавит политик безопасности, являются важным аспектом разработки механизма контроля информационных потоков.

Практическое значение может иметь предложенный вариант использования формальной логической системы *TLAPS*. Как уже отмечалось, многие объемные доказательства, публикуемые в современных статьях, содержат некорректные утверждения. Человеческий фактор является причиной возникновения простейших ошибок. Например, часто на основе установленного (заданного) факта вида: “$\forall x \in X$ выполняется $P(x)$” делается шаг: “возьмем $x \in X$, для которого выполняется $P(x)$”. Шаг является некорректным, так как требует проверки дополнительного условия “X не является пустым”. Другим ошибочным примером является использование для доказательства истинности некоторого утверждения фактов вида: $x1 =$ if $e1$ then $x2$ else $x3$ и $x1 = x2$. Недостающим условием в данном случае является: $x2 \neq x3$, его проверка может оказаться нетривиальной.

Использование таких средств, как *TLAPS*, безусловно влечет дополнительные трудозатраты, вызванные необходимостью трансляции доказательств в формулы *TLA+*, но в то же время предоставляет дополнительные гарантии корректности получаемых результатов.

5. БЛАГОДАРНОСТИ

Автор выражает признательность Джуру Куковецу (Институт разработки информационных систем, Вена, https://informatics.tuwien.ac.at/orgs/e194), одному из разработчиков инструмента *Apalache* [10], предназначенного для проверки спецификаций *TLA+* на основе символьного выполнения, за важные рекомендации по оптимизации отдельных функций и операторов, образующих *TLA+* семантику используемой версии языка *Paralocks*. Несмотря на то, что на данном этапе полностью адаптировать спецификации *PLIF* для применения *Apalache* не удалось, дальнейшая работа в этом направлении, по мнению автора, имеет перспективу и будет продолжена.

СПИСОК ЛИТЕРАТУРЫ

1. *Девянин П.Н. и др.* Интеграция мандатного и ролевого управления доступом и мандатного контроля целостности в верифицированной иерархической модели безопасности операционной системы // Тр. Института системного программирования РАН. 2020. Т. 32. № 1. С. 7–26.
2. *Lamport L.* Specifying systems: the TLA+ language and tools for hardware and software engineers. 2002.
3. *Denning, Dorothy E.* A lattice model of secure information flow // Communications of the ACM. 1976, № 5. P. 236–243.
4. *Broberg, Niklas, Sands, David.* Paralocks: Role-based information flow control and beyond // Conference Record of the Annual ACM Symposium on Principles of Programming Languages, 2010. P. 431–444.
5. *Myers, C. Andrew, Liskov, Barbara.* A decentralized model for information flow control // ACM SIGOPS Operating Systems Review. 1997. № 5. P. 129–142.
6. *Тимаков А.А.* PLIF. 2021. GitHub. https://github.com/timimin/plif.
7. *Blanchette J. C., Bulwahn L., Nipkow T.* Automatic proof and disproof in Isabelle/HOL //Frontiers of Combining Systems: 8th International Symposium, FroCoS 2011, Saarbrucken, Germany, October 5–7, 2011. Proceedings 8. Springer Berlin Heidelberg, 2011. P. 12–27.
8. *Bonichon R., Delahaye D., Doligez D. Zenon.* An extensible automated theorem prover producing checkable proofs //LPAR. 2007. Т. 4790. P. 151–165.
9. *Lamport L.* How to write a proof //The American mathematical monthly. 1995. Т. 102. № 7. P. 600–608.
10. *Kukovec J., Konnov I.* Type Inference for TLA in Apalache.

11. *Broberg N.* Thesis for the Degree of Doctor of Engineering Practical, Flexible Programming with Information Flow Control. 2011.
12. *Кораблин Ю.П.* Эквивалентность схем программ на основе алгебраического подхода к заданию семантики языков программирования // Russian Technological Journal. 2022. № 10(1). С. 18–27.
13. *Broberg N., van Delft B., Sands D.* Paragon for practical programming with information-flow control //Programming Languages and Systems: 11th Asian Symposium, APLAS 2013, Melbourne, VIC, Australia, December 9–11, 2013. Proceedings 11. Springer International Publishing, 2013. С. 217–232.
14. *Clarkson M. R. et al.* Temporal logics for hyperproperties // Principles of Security and Trust: Third International Conference, POST 2014, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2014, Grenoble, France, April 5–13, 2014, Proceedings 3. – Springer Berlin Heidelberg, 2014. С. 265–284.

DESCRIPTION OF PARALOCKS LANGUAGE SEMANTICS IN TLA+

A. A.Timakov[a]

[a] *MIREA – Russian Technological University, Pr. Vernadskogo 78, Moscow, 119454 Russia*

One of the basic aspects of information flow control in applications is security policy language. Such language should allow to define security policies for evaluation environment elements in coherence with higher level access control rules. So the language is expected to be flexible because there may be different access control paradigms implemented on the system level: mandatory, role-based etc. An application may also have its own specific restrictions. Finally it is also desirable that the language support declassification (controlled release of information) during the computation. One of such languages is Paralocks. The research is devoted to the logical semantics of modified version of Paralocks realized in TLA+. Paralocks represents a language basis for the perspective information flow control platform PLIF which is being developed for the analysis of PL/SQL program blocks with author's participation. It includes the proofs of the partial order and lattice defined on the set of security policy expressions.

Keywords: information flow control, security policy language, logical semantics, formal verification, model checking, program units

REFERENCES

1. *Devyanin P.N. and others.* Integration of mandatory and role-based access control and mandatory control in a verified hierarchical security model of LED systems // Proceedings of the Institute of System Programming of the Russian Academy of Sciences. – 2020. – T. 32. – No. 1. – P. 7–26.
2. *Lamport L.* Specifying systems: the TLA+ language and tools for hardware and software engineers. – 2002.
3. *Denning E.D.* A lattice model of secure information flow // Communications of the ACM, 1976, No 5. P. 236–243.
4. *Broberg N., Sands D.* Paralocks: Role-based information flow control and beyond // Conference Record of the Annual ACM Symposium on Principles of Programming Languages, 2010. P. 431–444.
5. *Myers C.A., Liskov B.* A decentralized model for information flow control // ACM SIGOPS Operating Systems Review, 1997, No 5. p. 129–142.
6. *Timakov A.A.* PLIF. 2021. GitHub. https://github.com/timimin/plif.
7. *Blanchette J.C., Bulwahn L., Nipkow T.* Automatic proof and disproof in Isabelle/HOL //Frontiers of Combining Systems: 8th International Symposium, FroCoS 2011, Saarbrucken, Germany, October 5-7, 2011. Proceedings 8. – Springer Berlin Heidelberg, 2011. – C. 12–27.
8. *Bonichon R., Delahaye D., Doligez D.Z.* An extensible automated theorem prover producing checkable proofs // LPAR. – 2007. – T. 4790. – C. 151–165.
9. *Lamport L.* How to write a proof //The American mathematical monthly. – 1995. – T. 102. – №. 7. – C. 600–608.
10. *Kukovec J., Konnov I.* Type Inference for TLA in Apalache.
11. *Broberg N.* Thesis for the Degree of Doctor of Engineering Practical, Flexible Programming with Information Flow Control. 2011.
12. *Korablin Yu.P.* Equivalence of program schemes based on the algebraic approach to specifying the semantics of programming languages // Russian Technological Journal, 2022, 10(1), P. 18–27.
13. *Broberg N., van Delft B., Sands D.* Paragon for practical programming with information-flow control // Programming Languages and Systems: 11th Asian Symposium, APLAS 2013, Melbourne, VIC, Australia, December 9–11, 2013. Proceedings 11. – Springer International Publishing, 2013. – C. 217–232.
14. *Clarkson M.R. et al.* Temporal logics for hyperproperties // Principles of Security and Trust: Third International Conference, POST 2014, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2014, Grenoble, France, April 5–13, 2014, Proceedings 3. – Springer Berlin Heidelberg, 2014. – C. 265–284.