ПРОГРАММИРОВАНИЕ, 2022, № 4, с. 27–49

ИНФОРМАЦИОННАЯ БЕЗОПАСНОСТЬ

УДК 004.43

КОНТРОЛЬ ИНФОРМАЦИОННЫХ ПОТОКОВ В ПРОГРАММНЫХ БЛОКАХ БАЗ ДАННЫХ НА ОСНОВЕ ФОРМАЛЬНОЙ ВЕРИФИКАЦИИ

© 2022 г. А. А. Тимаков[a,*]

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

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

Поступила в редакцию 08.06.2021 г.
После доработки 13.09.2021 г.
Принята к публикации 21.12.2021 г.

К настоящему времени проведено большое количество исследований в области формальных моделей безопасности компьютерных систем. В работе не рассматриваются криптографические методы защиты информации, которые, применительно к информационным системам, обеспечивают безопасность при реализации ими функций хранения и передачи данных. Основное внимание уделяется моделям, построенным на основе управления доступом и контроля информационных потоков и обеспечивающим безопасность при обработке данных. Модели, основанные на управлении доступом, нашли широкое применение при реализации механизмов защиты уровня операционных систем (ОС) и систем управления базами данных (СУБД). Контроль информационных потоков (КИП) сегодня активно внедряется в языковые платформы, предназначенные для создания как системного так и прикладного программного обеспечения. Однако, несмотря на все усилия, на практике при построении автоматизированных информационных систем промышленного уровня доминирует подход, ориентированный исключительно на реализацию общесистемных механизмов управления доступом. КИП на уровне приложений зачастую рассматривается без привязки к требованиям глобальной политики безопасности. В работе делается попытка обосновать необходимость дополнения глобальных систем управления доступом механизмами КИП в программном обеспечении. Приводится алгоритм внедрения КИП на уровне программных блоков баз данных (БД) на основе реализованного на системном уровне ролевого управления доступом.

DOI: 10.31857/S0132347422040057

1. ВВЕДЕНИЕ

Формальные методы обеспечения безопасности информационных систем можно разделить на две категории: криптографические и управления доступом/контроля информационных потоков. Криптографические механизмы защиты информации (КМЗИ) направлены на поддержание конфиденциальности и целостности информации в процессе ее хранения и передачи. Механизмы управления доступом/контроля информационных потоков защищают данные при их обработке.

В сложившейся практике область применения механизмов второй группы ограничена системным уровнем (ОС и СУБД). При проектировании безопасных систем предпочтение отдается комбинированным подходам, таким как мандатно-ролевое управление доступом. В настоящее время все больше специалистов признает необходимость расширения формальных моделей безопасности, в первую очередь, построенных на основе КИП, до вершины информационного стека – уровня приложений. Как будет показано далее, исследования активно продолжаются на протяжении последних 20 лет, в настоящее время появились первые языковые платформы, в которых реализованы полноценные механизмы контроля информационных потоков: Joana [1, 2], JIF [3], Paragon [4], Flow Caml [5]. Надо признать, что теоретические исследования в области формальных моделей безопасности приложений значительно опережают практику. При этом сложность современных систем и соответствующих угроз [6] не позволяет отказаться от идеи полноценного внедрения этих моделей на прикладном уровне и в целом на уровне специального программного обеспечения. Приведем ряд дополнительных аргументов в поддержку последнего утверждения.

В мире формальных моделей безопасности уровня ОС считается вполне обоснованным предположение о разделении всех процессов на доверенные и недоверенные. При этом доверенные процессы реализуют функции безопасности и не вступают в кооперацию с недоверенными при реализации запрещенных информационных

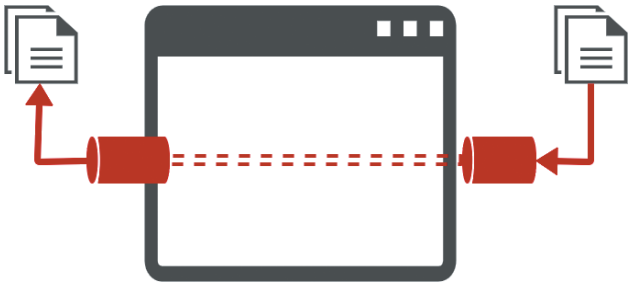


Рис. 1. Среда передачи данных.

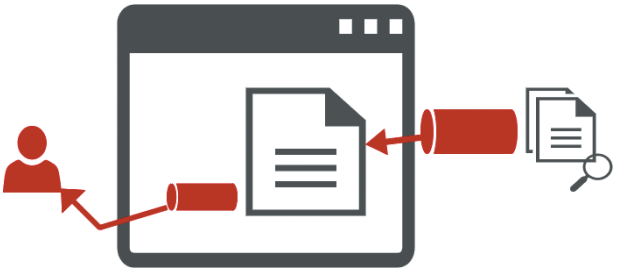


Рис. 2. Приложение – “витрина”.

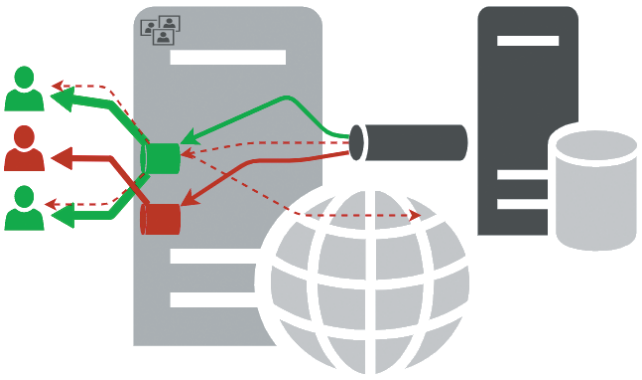


Рис. 3. Многопользовательские приложения.

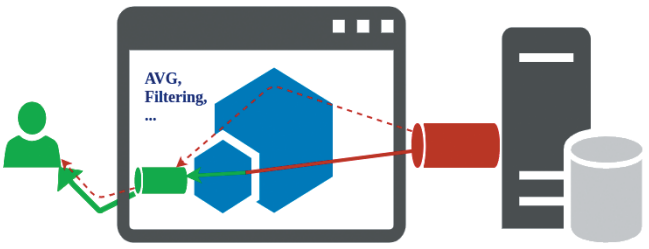


Рис. 4. Предобработка данных.

потоков. Возможность нарушения целостности программ, инициирующих доверенные процессы исключается. Мы утверждаем, что даже при этих существенных допущениях самодостаточными механизмы управления доступом системного уровня могут считаться только в случае, когда пользовательские приложения выступают в роли тривиальной среды передачи данных от источников к получателям или ”витрин”, то есть их функции ограничены представлением данных пользователям без искажения смысла (рис. 1, 2).

В случае, когда приложения осуществляют существенные преобразования данных, приводящие, например, к деклассификации (рис. 4), или в случае, когда допускается многопользовательский режим работы, управление доступом на уровне ОС или СУБД в строгом смысле не может рассматриваться как самодостаточный механизм (рис. 3).

В соответствии с некоторыми рекомендациями в случаях, показанных на рис. 4, допускается (по аналогии с административными функциями) выносить выполнение критичных вычислений в отдельные доверенные процессы. Наличие подобных доверенных процессов в системе видится недопустимым послаблением. Очевидно, все модули программного обеспечения информационной системы создаются, как правило, одной командой разработчиков. Кроме того, подобное разделение на процессы существенно повышает сложность самой разработки.

Известный опыт внедрения мандатно-ролевого управления доступом на уровне ОС и СУБД одновременно свидетельствует о практической сложности достижения требуемого уровня детализации политики безопасности. В частности, говоря об объектах БД, разграничение доступа может быть эффективно реализовано на уровне пользовательских схем. Дальнейшая детализация осложняется большим числом различных объектов (представления, хранимые процедуры, синонимы, последовательности и др.) и скрытых информационных каналов.

Наконец, полезно провести аналогию с криптографическими методами защиты информации, которые нашли практическое применение на всем стеке информационных технологий: физическом, сетевом, системном и прикладном (Таблица 1).

Таким образом специальное программное обеспечение (пользовательские приложения, несистемные службы, программные блоки баз данных и др.) целесообразно воспринимать как нетривиальную среду передачи данных, анализ информационных каналов в которой представляет собой отдельную задачу. Широко распространенные формальные методы статического анализа применительно к этой задаче обладают рядом существенных ограничений. В первую очередь, описание политики безопасности информационных потоков (в отличие, например, от описания правил безопасной работы с памятью) является

Таблица 1. Реализация механизмов управления доступом и КМЗИ

	Канальный	Сетевой	Системный	Прикладной
Управление доступом КМЗИ	Port Security PPTP	фильтрация трафика IPSec (VPN)	управление доступом EFS (BitLocker)	**КИП – ?** Application Layer Encryption

Таблица 2. КИП в приложениях – исторический аспект

	Описание политики	Безопасность семантики	Механизм проверки	Реализация
Модель Белла–Лападулы [9]	+	–	±	–
Модель Деннинг (решетки) [10]	+	–	+	–
Система типов Волпано [11]	+	±	+	–
Flow Caml [5]	+	±	+	±
JIF [3]	+	–	+	+
Paragon [4]	+	+	+	+

специфичным для конкретной системы и требует значительного инструментирования кода: добавления комментариев, аннотаций, расширенных модификаторов доступа и т.д. Кроме того, с учетом сложности семантики информационных потоков, возникающих в программном обеспечении (явных, неявных, вероятностных, временных, случайных и т.д.), значительно возрастают затраты на выявление “ложных” срабатываний.

Далее в п. 2 кратко рассматривается исторический аспект развития методов КИП в программном обеспечении. Последовательно излагаются результаты некоторых наиболее значимых исследований в этой области, описываются направления развития и приводятся отдельные проекты, которые можно считать прототипами первых безопасных (в смысле информационного невлияния [7]) расширений языков программирования. В конце раздела формулируется идея разделения функций безопасной разработки и анализа с переносом механизма КИП из конкретной языковой среды в среду формальной верификации. Мотивирующий пример (см. п. 3) облегчит восприятие следующего раздела (см. п. 4), в котором более подробно описан подход, ориентированный на реализацию безопасных вычислений в среде, максимально приближенной к данным. Руководствуясь желанием сохранить в фокусе главную идею предложенного подхода, в данной работе ограничимся общими рассуждениями о корректности и применимости предложенного механизма контроля.

2. РЕТРОСПЕКТИВА

Читателям, знакомым с общей теорией и историей развития механизмов КИП, рекомендуется перейти к п. 3. Как уже отмечалось ранее, в последние годы проблеме КИП в программном обеспечении уделяется значительное внимание. Общее определение и классификация информационных потоков в программном обеспечении приводятся в [8]. Современные исследователи различают четыре аспекта проблемы: описание политики безопасности – языковая часть механизма КИП, условия безопасности вычислений (безопасность семантики), механизм проверки условий безопасности и доказательство его корректности, реализация. Полноценное внедрение КИП в информационную систему предполагает охват всех четырех аспектов.

В таблице 2 приведены наиболее значимые для развития теории КИП исследования, сделана попытка оценить их результаты применительно к перечисленным направлениям.

История механизмов КИП берет свое начало с первых формальных моделей безопасности информационных систем. Среди них особое место занимает модель Белла–Лападулы [9]. Авторы впервые предложили использовать для описания политики безопасности множество упорядоченных меток – уровней безопасности, соответствующих стандартной классификации данных по уровню конфиденциальности: “несекретно” ≤ “для служебного пользования” ≤ “секретно” ≤ “совершенно секретно”. Несмотря на то, что данная модель остается актуальной и в наше время, в работе не дано строгого определения условий безопасности.

В основу современных механизмов КИП положено понятие безопасности вычислений на основе информационного невлияния [7, 12]. В общем виде применительно к программным средам условие безопасности можно сформулировать

как отсутствие влияния элементов среды вычислений (переменных, входных-выходных потоков, исключений и т.д.) с заданным уровнем конфиденциальности (если используется решетка уровней конфиденциальности) на элементы с более низким уровнем конфиденциальности.

Следующей вехой развития теории КИП стали работы Д. Деннинг [10, 13]. Основными достижениями считаются: обобщение упорядоченного множества меток модели Белла–Лападулы до алгебраических решеток, предложения по реализации механизма проверки условий безопасности программ на основе статического анализа, ввод понятия неявного информационного потока (и контекста безопасности), который возникает в операторах ветвления и циклов при использовании конфиденциальных данных в условиях:

1 if high = 1 then low = 0; else low = 1; end if;

Дать определение безопасности вычислений на основе информационного невлияния и доказать корректность механизма проверки (в смысле информационного невлияния), предложенного Д. Деннинг, спустя почти двадцать лет удалось Волпано и др. [11]. Они создали первый прототип системы безопасных типов для простого императивного языка, объединив, таким образом, первые три аспекта КИП.

Первой реализацией КИП на уровне реального языка программирования стал проект *Flow Caml* [5] – безопасное расширение функционального языка *ML*. Авторами был реализован механизм контроля информационных потоков на основе строгой схемы информационного невлияния. Данный факт стал бесспорным преимуществом проекта и одновременно причиной его ограниченного распространения – проект получил популярность лишь в академической среде и стал отправной точкой для дальнейших исследований.

Можно сказать, что проект *FlowCaml* завершил начальный этап становления теории КИП. В дальнейшем исследования продолжались по всем четырем направлениям. Несколько более подробная информация по ним приводится далее. Среди наиболее значимых проектов, доведенных до практической реализации в промышленных языках программирования, выделяются *JIF* [3] и *Paragon* [4].

Описание политики. По сути, речь идет о языковой части механизма КИП. Грамматика языка описания политики может быть основана на таких понятиях, как: субъект, объект, право доступа, роль, уровень доступа, метка безопасности и т.д. Политика безопасности информационных потоков, реализуемая на уровне специального программного обеспечения, должна строго соответствовать более общей политике управления доступом, реализуемой на системном уровне. С учетом общепринятого подхода к управлению информационными потоками на основе алгебраических решеток наиболее гладко трансформация требований общей политики в метки элементов среды вычислений может осуществляться в системах с мандатным управлением доступом.

В значимой части известных механизмов КИП за основу принимается решетка уровней конфиденциальности данных [10] – $(SC, \sqsubseteq)$, где SC – конечное множество уровней конфиденциальности, $\sqsubseteq$ – отношение частичного порядка. В научных трудах для упрощения системы доказательств чаще всего принимается двухуровневая решетка: $\{H, L\}$, где $L \sqsubseteq H$.

Еще одним известным способом описания политики безопасности информационных потоков является децентрализованная модель на основе меток (ДММ) [3]. В основу положены двухкомпонентные метки, определяющие владельцев и "читателей" информации. Например, метка $\{A : B, C\}$ будет означать, что A является владельцем данных, а B и C предоставлено право чтения. В общем случае, метка переменной (или элемента среды вычислений иного типа) может включать несколько владельцев, каждый из которых может иметь свой список "читателей". В [3] задано отношение частичного порядка на множестве меток, минимальный и максимальный элементы, таким образом, данное множество также образует решетку, и для систем, использующих ДММ, применим подход [10]. Можно сказать, ДММ соответствует дискреционному принципу управления доступом.

В последнее время распространение получил принципиально новый перспективный язык *Flow Locks* [4, 14, 15]. Математической основой выражений языка являются конъюнкции определенных дизъюнктов Хорна вида: $\{L_1 \Rightarrow A_1, \ldots L_n \Rightarrow A_n\}$, где L_n – множество блокировок, которые должны быть открыты для того, чтобы данные могли быть прочитаны соответствующим актером (пользователем). В отличие от статических политик безопасности, политики, описываемые *Flow Locks*, могут быть динамическими, то есть зависеть от состояния некоторых элементов (блокировок) среды вычислений выражений. Например, политика $P \triangleq \{High, TExpired \Rightarrow Low\}$ означает, что соответствующее значение может быть всегда прочитано актером $High$, и актером Low – при условии открытой блокировки $TExpired$.

Расширение этого языка *Paralocks* позволяет описывать параметрические блокировки, обладает высокой выразительной способностью и позволяет, например, создавать политики управления информационными потоками на основе логических ролей приложений, в том числе, в предположении о том, что пользователи могут переключать роли в текущих сеансах.

Безопасность семантики. Отвечает на вопрос, какая система считается безопасной или описывает условия безопасности. Важно отметить, что понятие безопасности вычислений не должно определять алгоритм или какой-либо порядок проверки свойств системы. Например, для дискреционных моделей управления доступом часто принимается, что система является безопасной, если в ней отсутствует возможность получения субъектом нового права доступа к какому-либо объекту без участия владельца. Безусловно, это требование является недостаточно строгим, поскольку отсутствие доступа не исключает наличие запрещенных информационных потоков между соответствующими сущностями. Обязательное требование мандатного разграничения доступа в критичных системах во многом продиктовано стремлением исключить нисходящие информационные потоки между сущностями с разными уровнями доступа (уровнями конфиденциальности).

Как отмечалось ранее, в основу формального определения безопасности систем с контролем информационных потоков положено понятие информационного невлияния. Введем обозначения и определения по аналогии с [16]. Пусть C – множество выражений некоторого языка, $\Lambda \triangleq \Upsilon \times \Gamma \times \Theta$ – множество состояний среды вычислений, где Υ – множество состояний среды переменных, Γ – множество состояний среды входных потоков, Θ – множество состояний среды выходных потоков, $\langle c, S\rangle$ – контекст вычисления, $\langle c, S\rangle \downarrow o$ – отношение вычисления, которое обозначает, что при вычислении выражения c в состоянии $S \in \Lambda$ наблюдается поведение o. Под наблюдаемым поведением шага вычислений следует понимать значение, которое становится доступно внешнему обозревателю после его завершения (значение, сохраненное в переменную или выходной поток, временная задержка, информация об исключении и т.д.). Для простоты воспользуемся двухуровневой решеткой уровней конфиденциальности и будем полагать, что состояния S_1 и S_2 среды вычислений находятся в отношении низкой эквивалентности: $S_1 \approx S_2$, если значения их низкоуровневых элементов совпадают. Тогда:

Определение 2.1. *Программа P* удовлетворяет свойству информационного невлияния ИН(P), если для любых двух состояний S_1 и S_2 среды вычислений, состоящих в отношении низкой эквивалентности, успешное выполнение программы в первом (начальном) состоянии с наблюдаемым поведением o_1 гарантирует успешное выполнение программы во втором (начальном) состоянии с наблюдаемым поведением o_2, при этом o_1 и o_2 являются неразличимыми, то есть:

$$\text{ИН}(P) \triangleq \forall S_1, S_2, o_1, o_2 : S_1 \approx S_2 \wedge$$

$$\wedge \langle P, S_1\rangle \downarrow o_1 \wedge \langle P, S_2\rangle \downarrow o_2 \Rightarrow o_1 \approx o_2, \qquad (2.1)$$

где: $\approx$ – *отношение неразличимости поведений*.

В строгом смысле свойство информационного невлияния требует низкой эквивалентности итоговых состояний, то есть поведения считаются неразличимыми, если в итоговых состояниях S_1', S_2' полностью совпадают значения всех низкоуровневых элементов, включая значения соответствующих переменных. На практике такие ограничения считаются слишком строгими, кроме того, иногда допускается некоторое влияние высокоуровневых элементов на низкоуровневые. Например, приложение может использовать статистическую функцию *AVG* для расчета среднего значения стоимости контракта. Данное значение считается открытым, в то время как информация об отдельном контракте – конфиденциальной.

Для ослабления требования информационного невлияния введено понятие деклассификации данных (в случае контроля целостности – очистки данных). Под деклассификацией понимается контролируемое раскрытие информации с ограничениями по месту (*where*), времени (*when*), инициатору (*who*) и, конечно, содержимому (*what*). Требование информационного невлияния может быть еще больше ослаблено в зависимости от возможностей нарушителя, например, способности эксплуатировать скрытые информационные каналы, а также характера вычислений: пакетное выполнение команд, интерактивные вычисления. В последнем случае наблюдаемые значения могут генерироваться на каждом шаге вычислений. Двигаясь в сторону адаптации строгих требований безопасности к требованиям, реализуемым на практике – "допустимой некорректности", исследователи разработали дополнительные схемы информационного невлияния. Далее указаны некоторые из них в нисходящем порядке, от более строгих к менее строгим: информационное невлияние ИН, количественное информационное невлияние КИН, информационное невлияние с учетом деклассификации ИНД, терминальное-зависимое информационное невлияние ТЗИН, терминально-независимое информационное невлияние ТНЗИН, прогресс-зависимое информационное невлияние ПЗИН, прогресс-независимое информационное невлияние ПНЗИН и т.д. Подробное описание приведенных схем представлено в [16]. Отдельную ветвь составляют исследования, посвященные безопасности многопоточных программ и параллельных вычислений в целом. В работах [17–19] представлены схемы информационного невлияния, зависимые от вероятностных каналов и каналов по времени. Наибольший интерес представляет схе-

ма на основе принципа вероятностной бисимуляции [17].

В заключение хотелось бы отметить принципиально новый подход к определению безопасности семантики [4, 20]. Предпосылками к его появлению стали попытки адаптировать стандартное требование информационного невлияния к условиям, в которых допускается деклассификация данных. Предложенные модифицированные схемы ИН (для деклассификации) не отличались интуитивной простотой. Новый подход построен на основе эволюции знаний нарушителя, он хорошо подходит для анализа безопасности информационных потоков на основе динамических политик, то есть политик, требования которых могут изменяться на этапе выполнения (например, в случае успешной проверки условий деклассификации). Под знаниями нарушителя на шаге n выполнения программы P с соответствующей трассой наблюдаемых поведений $\vec{o}$, будем понимать множество начальных состояний среды вычислений, при которых выполнение P приводит к генерации $\vec{o}$. Дадим формальное определение безопасности программы на основе понятия статичности знаний нарушителя аналогично [21].

Определение 2.2. *Программа P удовлетворяет свойству статичности знаний нарушителя* СЗН(P), *если во время выполнения на нормальных шагах (не относящихся к деклассификации) знания нарушителя не меняются, то есть:*

$$\text{СЗН}(P) \triangleq \forall S, A, \vec{C}c' : K_A(S, P, \vec{C}c') = \\ = K_A(S, P, \vec{C}), \tag{2.2}$$

где: $K_A(S, P, \vec{C})$ *— знания нарушителя A после прохождения трассы $\vec{C}$ программы P из начального состояния S, c' — выражение, вычисляемое вне контекста деклассификации.*

Механизм проверки. Существует большое количество способов организации скрытых информационных каналов в программной среде. К таковым, например, относится измерение потребляемых ресурсов и времени выполнения. Эксплуатация таких каналов на практике сложна и маловероятна, часто они просто игнорируются. Проверка безопасности стандартных информационных потоков (явных и неявных), а также потоков, возникающих вследствие наблюдения за прогрессом вычислений, традиционно реализуется средствами статического и динамического анализа программ. В настоящее время превалируют статические методы анализа на основе безопасных систем типов. Как уже отмечалось ранее, катализатором этого направления послужили исследования Д. Деннинг [10, 13] и Д. Волпано [11]. Основным преимуществом статического анализа, очевидно, является полнота покрытия кода (при динамическом анализе проверке подвергаются все же отдельные траектории), а также отсутствие необходимости реализовывать дополнительные проверки на этапе выполнения программы. К преимуществам динамического анализа относят, главным образом, точность, что особенно важно для платформ, поддерживающих динамическую загрузку (генерацию) кода, таких как *JavaScript*. При этом, как доказано в [22], статический и динамический анализ позволяют в равной степени гарантировать безопасность программы в смысле ПНЗИН. Попытки объединить преимущества обоих подходов привели к появлению гибридных механизмов КИП [23]. Полный сравнительный анализ обозначенных подходов приведен в [16].

Значимая часть исследований посвящена внедрению механизмов контроля на уровне расширенных безопасных систем типов. Анализ на основе более полных программных моделей, например межпроцедурных графов потоков управления, является, безусловно, более точным, но и более сложным с точки зрения практической реализации (требуется построение самих моделей, отсутствует возможность композиционного анализа). Тем не менее, отдельные исследования направлены на сращивание и этих подходов [24].

Не угасает интерес к анализу информационных потоков при параллельных вычислениях. В проекте *Veronica* [25] предложена, по утверждению авторов, новая композиционная логика проверки, поддерживающая широкий спектр политик безопасности.

Методы формальной верификации несмотря на общий рост популярности для решения проблемы КИП в программном обеспечении применяются редко. Объясняется это достаточно просто. Проверка условий безопасности программного обеспечения должна быть комплексной, то есть применяться ко всем модулям системы. Значительный объем кода современных приложений не позволяет даже с учетом абстрагирования добиться приемлемого числа состояний для проверки инвариантных свойств программы, описанной в виде спецификации, с использованием инструментов проигрывания моделей типа *TLC*. Получение формальных доказательств является трудоемким процессом, и на практике может применяться лишь для выделенных компонент. Среди отдельных работ, где формальная верификация применяется для проверки условий безопасности информационных потоков, следует отметить: [26, 27]. Данные исследования, однако, посвящены анализу бизнес-процессов. В некоторых случаях формальная верификация используется для доказательства корректности современных моделей управления доступом в ОС и СУБД [28].

```
public void play{P1<-* meet P2<-*}(PrintStream[{}]
    {P1<-* meet P2<-*} output) throws
        (SecurityException,
         IllegalArgumentException{P1<-* meet P2<-*})
         where authority (P1, P2)
```

Рис. 5. Пример описания политики безопасности в JIF.

```
public !bottom void play
        (?bottom PrintStream<bottom> output)
         throws !bottom IllegalArgumentException
```

Рис. 6. Пример описания политики безопасности в Paragon.

Реализация. Наиболее значимые проекты были обозначены ранее. Отметим также, что в целом число инструментов и платформ, предназначенных для решения задачи КИП, достаточно велико, известны решения для проверки web и мобильных приложений [29–32] платформы для анализа безопасности информационных потоков в системах, построенных на основе баз данных [33–35]. К сожалению, ни один проект не получил широкого распространения. Отказ от строгих схем информационного невлияния, внедрение КИП в самые популярные языковые платформы не оправдали ожиданий. Примечательным является исследование [36], в котором автор делает попытку оценить возможность использования существующих решений для разработки полноценных безопасных Java-приложений. В работе были рассмотрены проекты *JIF* и *Paragon*. Сложность описания политик безопасности и необходимость значительного инструментирования кода привели к тому, что работа, по сути, была сведена к анализу известных демонстрационных примеров[1].

Причиной сложившейся ситуации видится следующее. Все реализации, несмотря на возможность описания гибких политик безопасности и проверки их требований, предполагают существенное изменение языковой грамматики в отсутствии расширения функциональных возможностей (см. рис. 5 и 6).

Безопасность всегда оставалась вторичной по отношению к функционалу, и ситуация не будет меняться. Вряд ли стоит ожидать широкого распространения безопасных языков программирования. Стоит также отметить, что за последние 25 лет (с момента публикации [11]) мир существенно изменился: расширились сферы применения языков программирования, значительно увеличились объемы кода в типовых проектах, наблюдается тенденция к упрощению языковых грамматик, делается акцент в сторону повышения производительности написания кода (rapid application development) и сокращению времени внедрения новых функций в существующее программное обеспечение. Активно развивается архитектура микросервисов и модульная разработка, происходит постепенный отказ от монолитного принципа написания программ. Все перечисленное плохо сочетается с трудоемкими процедурами КИП, лежащими в основе “языковой” безопасности (language based security). Решение видится в разделении функций разработки и анализа безопасности информационных потоков.

Предположение 2.1. *Смещение механизма проверки условий безопасности в область формальной верификации позволит решить проблему при выполнении следующих требований к реализации:*

– автоматическая генерация спецификаций с приемлемым уровнем абстракции (наличие утилиты типа C2TLA [37]);

– удобный интерфейс аналитика для описания политик безопасности элементов среды вычислений;

– приемлемые временные затраты на проверку инвариантных свойств безопасности (надежности);

– наглядное представление результатов анализа.

Вынося механизм проверки за пределы языковой среды, мы, в определенном смысле, теряем гарантии безопасности, поскольку, соответствие программы заданным свойствам, в конечном счете, определяется соответствием программы проверенной модели (спецификации). Тем не менее, с учетом описанных выше тенденций развития индустрии программного обеспечения, частичный отказ от таких гарантий в пользу декомпози-

[1] В основе обоих проектов лежат безопасные системы типов, по этой причине разметке подвергаются не только входные и выходные значения, но и заголовки методов. Для них могут задаваться метки формальных параметров, эффектов записи (writing effects), выходных значений, генерируемых исключений. Описание политики для систем, основанных на использовании межпроцедурного ГПУ, например Joana [2], выглядит несколько проще.

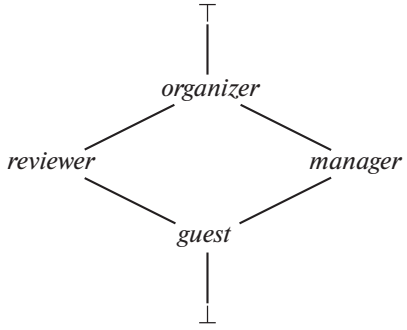


Рис. 7. Решетка ролей.

ции функций вовлеченных в процесс разработки специалистов и упрощения работы программистов (от них требуется лишь внесение конкретных проверок, сформированных по результатам тестирования спецификации) видится обоснованным. Основные положения предлагаемого подхода представлены в п. 4.

3. МОТИВИРУЮЩИЙ ПРИМЕР

Для удобства воспользуемся известным примером описания системы управления конференциями. Она позволяет участникам заявлять доклады, организаторам – осуществлять их рецензирование и распределение по секциям.

Глобальная политика безопасности, заданная в терминах ролевой модели управления доступом (рис. 7), диктует ряд ограничений, например: рецензент имеет доступ по чтению к содержимому статей, но не может ассоциировать работы с конкретными авторами. Статус доклада (одобрен или нет) доступен по чтению менеджеру, который занимается составлением программы конференции и распределением докладов по секциям, и другим пользователям по истечении определенного временного интервала. Программа конференции доступна по чтению всем пользователям. Схема данных состоит из следующих отношений: *Papers* – доклады, *Submissions* – заявки на участие с докладом, *Conferences* – конференции, *Sections* – секции, *Allocations* – распределение докладов по секциям, *Logs* – журнал ошибок.

Функции системы, связанные с обработкой данных, реализуются посредством хранимых процедур и функций БД, полный состав которых можно найти в [38]. Ключевыми для нас хранимыми программными блоками являются: *p_submit_paper* – добавляет строку в таблицу *Submissions* (регистрирует заявку на участие с докладом), *p_chahge_status* – обновляет значение поля *Status* для заданной строки в таблице *Submissions* после рецензирования соответствующего доклада, *p_allocate* – добавляет сведения о докладе в таблицу *Allocations* после осуществления ряда проверок, *f_getsection_program* – возвращает программу заданной секции, для формирования отчета обращается к таблицам: *Papers* и *Allocations*.

Назначение привилегий в соответствии с заданными ограничениями политики безопасности может быть представлено как:

```
1 grant execute on p_submit_paper to guest;
2 grant execute on p_chahge_status to reviewer;
3 grant execute on p_allocate to manager;
4 grant execute on f_getsection_program to guest;
```

Требование политики по предоставлению доступа к атрибуту *Status* доклада (менеджеры или **иные пользователи по истечению тайм-аута**) является примером использования процедуры деклассификации данных. Корректность реализации этой процедуры определяется, в том числе, и “доверенным” сервисом, который формирует данные о программах конференций. На рис. 8 показан пример нарушения заданного требования (правый фрагмент кода) через неявный информационный поток, и пример правильной проверки условий вставки данных в таблицу *Allocations* (статус 2 указывает на доклады приглашенных специалистов, их рецензирование не требуется).

Отсутствие утечек информации о принадлежности докладов конкретным авторам на этапе рецензирования также зависит от свойств соответствующих программных модулей, и может не гарантироваться системой управления доступом уровня БД.

4. КОНТРОЛЬ ИНФОРМАЦИОННЫХ ПОТОКОВ В ПРОГРАММНЫХ БЛОКАХ БД

Выбор **программной среды** для внедрения механизма КИП – хранимые программные блоки БД, объясняется следующими особенностями: близость к данным, малый объем кода – в среднем размер хранимой процедуры (функции) не превышает 60 строк, отсутствие вспомогательного кода – PL/SQL код фактически не реализует вспомогательные функции (сфокусирован на основном алгоритме), наличие развитых средств анализа зависимостей – позволяет эффективно выделять релевантное относительно чувствительной информации подмножество хранимых программных блоков, сериализация транзакций с использованием различных стратегий управления блокировками – позволяет сократить количество состояний модели, описывающей параллельные вычисления.

Для **описания политик безопасности** в исследовании используется язык *Paralocks*. Он обладает простой математической интерпретацией – основан на выражениях логики предикатов первого порядка, отличается высокой гибкостью и выразительной способностью – позволяет описывать правила мандатной, ролевой моделей управления

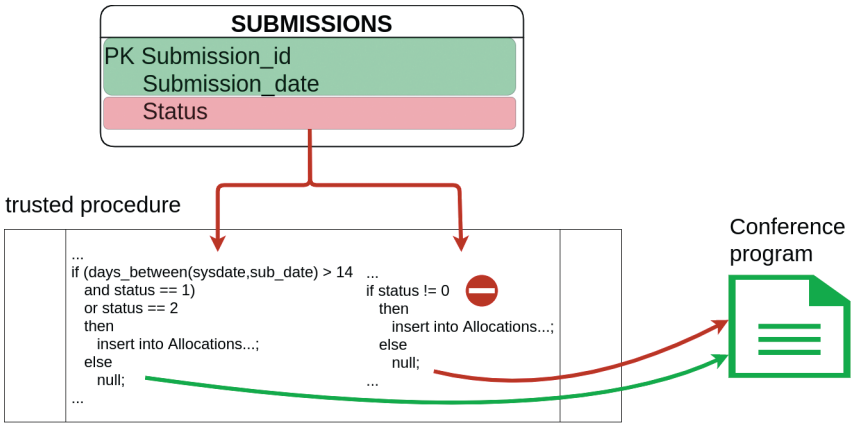


Рис. 8. Пример нарушения политики безопасности.

доступом, децентрализованной модели на основе меток, позволяет задавать правила деклассификации (очистки) данных – ослабляющие условия являются частью самих выражений политики безопасности. Спецификация *Paralocks* [38] содержит *TLA+* представление примитивов, необходимых для определения политики. В данной работе, однако, воспользуемся упрощенной решеткой меток конфиденциальности (рис. 9) (для описания дуальной политики целостности может использоваться аналогичная решетка). Введем следующие обозначения ролей, которые могут быть заданы в сеансе пользователя: g – *guest*, r – *reviewer*, m – *manager*, o – *organizer*. Метка конфиденциальности определяется как множество вида: $L_k = \{R_1, \ldots R_n\}$, здесь $R_1, \ldots R_n$ – условия предоставления доступа. Для чтения данных необходимо выполнение одного из условий. С учетом требования преемственности политики базовой ролевой модели положим:

$$\forall R_n, R_m, L_k : R_n \le R_m \wedge \\ \wedge\ R_n \in L_k \Rightarrow R_m \in L_k \qquad (4.1)$$

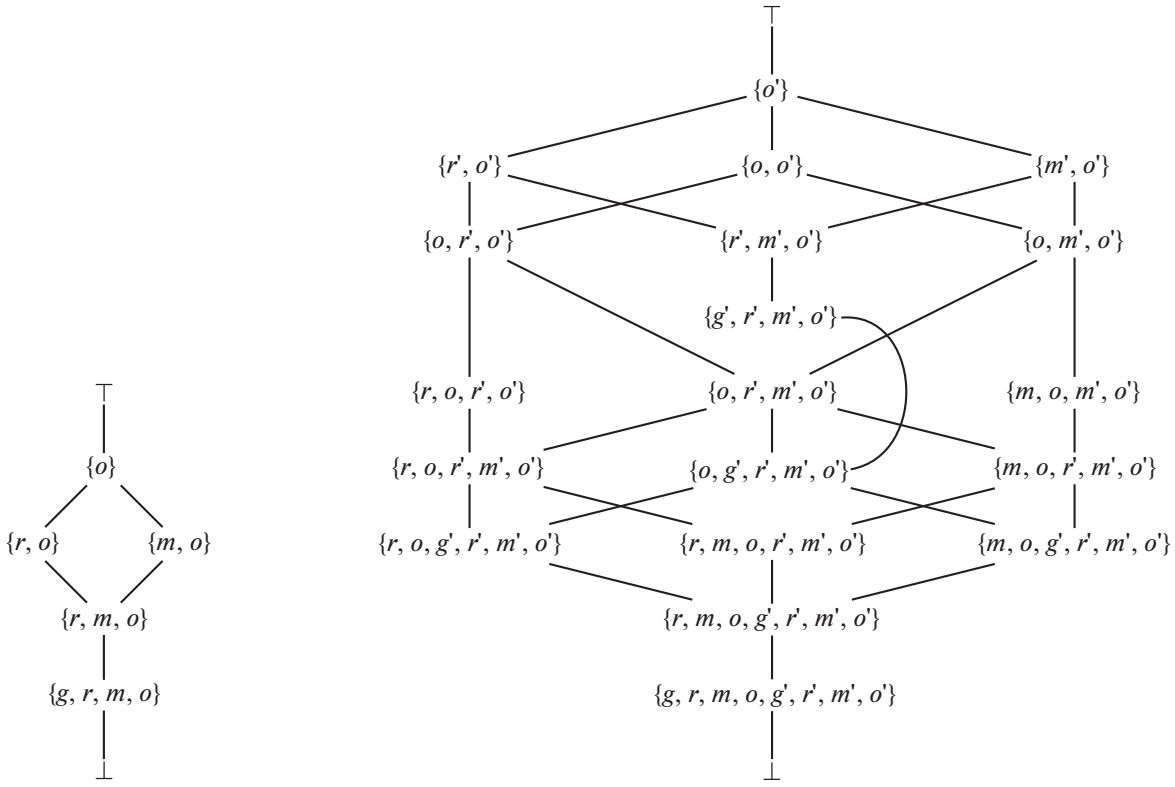
Тогда метка данных, доступ на чтение к которым предоставлен g, будет иметь вид: $\{g, r, m, o\}$. На рис. 9а представлена решетка меток конфиденциальности на основе иерархии ролей (рис. 7) без учета дополнительных ограничений. Отношение частичного порядка $\sqsubseteq$ может быть задано как: $L_1 \sqsubseteq L_2$ тогда и только тогда, когда для любого $R_n \in L_2$ имеем: $R_n \in L_1$. Оператор пересечения (нахождения наибольшей нижней грани) определяется как объединение условий (ролей) соответствующих меток конфиденциальности: $L_1 \sqcap L_2 = L_1 \cup L_2$, оператор соединения (нахождения наименьшей верхней грани) – как пересечение условий: $L_1 \sqcup L_2 = L_1 \cap L_2$.

В случае, если в политике используется дополнительное ограничение, например “time expired”, решетка меток конфиденциальности задается несколько сложнее. Пусть R' означает, что чтение возможно, если в текущем сеансе установлена роль R и выполняется условие “time expired”. Будем также полагать, что на множестве $\{R'_1, \ldots R'_n\}$ сохраняется отношение частичного порядка, то есть: $R_1 \le R_2 \Rightarrow R'_1 \le R'_2$, и условие R' всегда является более строгим, чем R, то есть, если для чтения данных достаточно R, то достаточно и R' или:

$$\forall R_n, L_k : R_n \in L_k \Rightarrow R'_n \in L_k \qquad (4.2)$$

С учетом выражений 4.1 и 4.2 метка данных, доступ на чтение к которым предоставлен менеджерам – m или обычным пользователям при истечении заданного интервала времени – g', будет иметь вид: $\{g', r', m', o', m, o\}$. Отношение частичного порядка, операторы пересечения и соединения задаются также, как и для решетки на рис. 9а. Графически решетка меток конфиденциальности на основе иерархии ролей (рис. 7) и дополнительного ограничения “time expired” показана на рис. 9б.

Безопасность семантики в данной работе рассматривается в контексте прогресс-зависимого информационного невлияния. В отличие от стандартной схемы (Определение 2.1), в данном случае проверка условий безопасности информационных потоков выполняется после каждого шага вычислений, при этом сами условия ослабляются (контролируется только вывод данных в выходные потоки). Описанная далее инструментированная абстрактная семантика безопасных информационных потоков (далее просто – абстрактная семантика) положена в основу механизма генерации спецификаций программных блоков и предполагает монотонное изменение меток безопасности внутренних глобальных (далее – глобальных) переменных среды (таблиц) на этапе выполнения. Под внутренними будем понимать объекты, недоступные для прямого просмотра. Пусть $M_n : X \to \{L_1, \ldots L_k\}$ – отображе-



(а) на основе ролей (б) на основе ролей и ограничения – “time expired”

Рис. 9. Решетка меток безопасности.

ние множества переменных на множество меток безопасности на шаге вычислений n.

Определение 4.1. *Программа P удовлетворяет свойству прогресс-зависимого информационного невлияния для начального и конечного отображений множества переменных на множество меток безопасности M_1 и M_2 – ПЗИН$(P)_{M_1,M_2}$, если для любых двух состояний S_1 и S_2 среды вычислений, состоящих в отношении низкой эквивалентности относительно некоторого уровня конфиденциальности L при отображении M_1*: а) *каждый шаг вычислений сопровождается генерацией одинаковых наблюдаемых значений относительно уровня L или приводит к “расхождению” для обоих состояний*”; б) *соответствующие финальные состояния – S_1' и S_2' находятся в отношении низкой эквивалентности относительно уровня L при отображении M_2*:

$$\text{ИЗИН}(P)_{M_1,M_2} \stackrel{\Delta}{=} \forall L, S_1, S_2, o_1, o_2 : S_1 \approx_{M_1,L} S_2 \wedge$$
$$\wedge P(S_1) \downarrow \langle S_1', o_1\rangle \wedge P(S_2) \downarrow \langle S_2', o_2\rangle \Rightarrow \qquad (4.3)$$
$$\Rightarrow o_1 \approx_L o_2 \wedge S_1' \approx_{M_2,L} S_2'$$

Условие б) в данном определении позволяет применять композиционный подход при проверке заданных свойств программного обеспечения [8]. Это важно, поскольку в общем случае сеанс взаимодействия пользовательского приложения с БД неограничен по количеству вызываемых процедур (функций), то есть модель параллельных вычислений, соответствующая такому взаимодействию, имеет бесконечное число состояний. В текущей версии проекта с помощью model checker проверяется основной инвариант безопасности (соответствует условию а) для системы, имитирующей однократное выполнение программного блока в каждом сеансе. С помощью отдельного предиката на последнем шаге каждой возможной траектории сеанса проверяется соответствие начальных и конечных меток безопасности всех глобальных переменных. В случае расхождения, требуется повторный прогон модели, при котором в начальном состоянии для глобальных переменных задаются метки безопасности последнего шага предыдущего прогона. Алгоритм является конечным, поскольку решетка меток безопасности имеет фиксированный размер, и все правила абстрактной инструментированной семантики являются монотонными в отношении глобальных переменных.

Учет скрытых вероятностных информационных каналов и каналов по времени потребует дополнительных ограничений, например, запрета

синхронизации для программных блоков, в которых возникают явные или неявные информационные потоки, связанные с чувствительными данными. Учитывая сложность эксплуатации таких каналов на практике, особенно в условиях многопользовательского, удаленного режима работы, на данном этапе они игнорируются.

Как уже отмечалось в п. 2, **проверку условий безопасности информационных потоков** в специальном программном обеспечении предложено вынести в область формальной верификации. В качестве инструмента создания спецификаций предлагается использовать язык *TLA+* [39]. Основными преимуществами, предопределившими его выбор среди множества альтернатив, являются: открытость, хорошая документированность и сопровождение, удобство интегрированной среды разработки *TLA Toolbox*, улучшенная поддержка свойств “живучести” (liveness), позволяющих более точно описывать поведение системы. Для непосредственной проверки свойств используются инструменты типа model checker: *TLC*, *Apalache* [40].

Пользовательские сеансы взаимодействия с БД посредством хранимых программных блоков описываются как параллельные процессы. Абстрактную семантику последовательных вычислений в рамках выделенного сеанса можно описать системой переходов состояний вида: $\langle PC, c, M \rangle$, где: PC – метка безопасности счетчика команд (program counter) в текущем сеансе; c – исполняемая команда текущего сеанса; M – абстрактное состояние среды, которое задает отображение локальных и глобальных переменных, входных и выходных потоков на соответствующие им метки безопасности. Пусть $D = \{d_1, \ldots d_n\}$ – множество активных сеансов, $O : D \times C$ – функция отображения множества сеансов на множество команд. Тогда глобальная семантика, описывающая переходы между параллельными процессами (сеансами), задается правилами GLOBAL RULES (Приложение 2).

С учетом выбранной позиции относительно скрытых вероятностных информационных каналов и каналов по времени, использование равномерного планировщика выглядит обоснованным.

Состояние системы, представленной как набор спецификаций [38], описывается более сложной структурой, включающей, в том числе, состояние памяти (значения локальных и глобальных переменных, параметров и т.д.), множество эксклюзивных табличных блокировок, множество открытых блокировок политики *Paralocks*, трассу вычислений и другие сущности. Кроме того, в проекте за основу принята расширенная схема ПЗИН, допускающая деклассификацию данных.

Опишем подробнее ключевые правила абстрактной семантики (малого шага) последовательных вычислений. Для простоты воспользуемся подмножеством языка *PL/SQL*, BNF-грамматика которого представлена в табл. 3.

Представленное подмножество PL/SQL не включает динамических запросов и команд, параметрических циклов и циклов с постусловием, а также некоторых иных конструкций. Тем не менее, полагаем, большая часть возможностей стандартного PL/SQL остается доступной. Исчерпывающие правила абстрактной семантики информационных потоков приведены в Приложении 2. Условия *inv* соответствуют инварианту безопасности.

Метка арифметического выражения (E-OPER) вычисляется как минимальная верхняя грань меток составляющих его операндов, это же утверждение справедливо для условных выражений (E-C-OPER) и выражений в списках SELECT курсоров (E-L-OPER). Метка записи (E-VAR-REC) определяется метками соответствующих полей. Метку таблицы было бы логично представить как минимальную верхнюю грань меток ее строк и значения, определяющего ее текущий размер, по аналогии с коллекциями (E-VAR-ARR), однако, с целью упростить *TLA+* спецификации, описывающие информационные потоки в программной среде БД, поднимем уровень абстракции при вычислении меток до столбцов (E-TAB-COL). Правила (C-INS), (C-UPD), (C-DEL) гарантируют корректность изменения этих меток. Значения меток столбцов монотонно возрастают в соответствии с метками изменяемых данных, условиями и контекстом вызова DML-операторов. При вычислении меток курсоров (E-CUR) учитываются метки условных выражений во фразе WHERE. Атрибуты курсоров (E-CUR-FND) наследуют метки курсоров. Заданная языковая грамматика допускает вызов сторонних процедур и функций (далее – просто сторонних функций), распространение информации в которых не контролируется. Будем различать доверенные и чистые функции. Полагаем, что при вызове доверенных функций $x_f(x_1 \to e_1, \ldots x_m \to e_m)$ запрещенных сторонних эффектов (side effects) не возникает, метка возвращаемого значения задается явно аналитиком (E-TR-LIB). Условие *inv* ограничивает метки фактических параметров и проверяет соответствие метки счетчика команд метке эффекта записи p_w (writing effect) вызываемой функции. Если “разметка” вызываемой сторонней функции не произведена, полагаем, что формальным параметрам и эффекту записи присваивается минимальная метка $\perp$, возвращаемому значению – максимальная метка $\top$. Под чистыми понимаются функции, в которых отсутствует эффект записи, то есть данные не сохраняются в

Таблица 3. Грамматика подмножества *PL/SQL*

(*values*)	$\nu ::=$	$n \mid b$
(*declarations*)	$d ::=$	x **number** $\mid x_1\ x_2 \mid d_1; d_n \mid$
		type x_r **is record** $(x_{f_1}\ x_1, \ldots x_{f_n}\ x_n) \mid$
		type x_t **is table of** $x \mid$ **exception** $x \mid$
		cursor x **is select** $e_1, \ldots e_n$
		from $x_{t_1}, \ldots x_{t_n}$ **where** $cnd \mid$
		procedure $x_p(x_1, \ldots x_n)$ **as** d
		begin c_1 **[exception]**c_2 **end**;$\mid$
		function $x_{fn}(x_1, \ldots x_n)$ **return** x_{rt} **as** d
		begin c_1**[exception** c_2**] end**;
(*expressions*)	$e ::=$	$\nu \mid x \mid x[e] \mid x_1.x_2 \mid e_1 \odot e_2 \mid x(e_1, \ldots e_2) \mid$
		$x\%$ **found** $\mid$
(*conditions*)	$cnd ::=$	$e_1 * e_2 \mid cnd_1 \star cnd_2$
(*statements*)	$c ::=$	$x := e \mid c_1; c_2 \mid x_f(x_1 \to e_1, \ldots x_n \to e_n) \mid$
		if e **then** c_1 **else** $c_2 \mid$ **while** e **do** $c \mid$
		endif $\mid$ **endwhile** $\mid c_1 \vee c_2 \mid$
		select $e_1, \ldots e_n$ **into** $x_1, \ldots x_n$ **from** $x_{t_1}, \ldots x_{t_n}$
		where $cnd \mid$
		insert into $x_t(x_1, \ldots x_n)$ $(e_1, \ldots e_n) \mid$
		update x_t **set** $x_1 = e_1, \ldots x_n = e_n$ **where** $cnd \mid$
		delete from x_t **where** $cnd \mid$
		open $x_{cur} \mid$ **fetch** x_{cur} **into** $x \mid$ **close** $x_{cur} \mid$
		throw $x_{exc} \mid$ **when** x_{exc} **then** $c \mid$ **endexc** $\mid$
		commit $\mid$ **rollback** $\mid$ **null** $\mid$ **return**$(x_{out} \to e) \mid$
(*program*)	$p ::=$	**declare** d **begin** c_1 **[exception** c_2**] end**;

глобальные переменные и не выводятся в выходные потоки, возвращаемое значение полностью зависит от фактических параметров (E-PUR-LIB). Как уже отмечалось выше, метки внутренних переменных могут изменяться на этапе выполнения. Правила, в названиях которых присутствует “EXT”, применяются в тех случаях, когда метка изменяемого объекта является статической, например, если к таблице кому-либо предоставлен прямой доступ (не через программные блоки PL/SQL). Еще одним источником неточности процедуры формальной верификации могут выступать правила (C-IF), (C-WHILE), как показано, они не предполагают проверку истинности условий. Применение этих правил добавляет в метку счетчика команд, представленную кортежем вида $\langle pc_1, \ldots pc_n \rangle$, новый элемент, соответствующий метке условия. В [41] показан вариант реализации запрещенного вероятностного информационного потока в многопоточном приложении через глобальные переменные в условиях *while*. Ограничения вида $e - local$ легко проверяются в ходе ручного анализа кода и позволяют гарантировать отсутствие описанных информационных потоков. Через $\to_p$ в (C-OR) обозначается вероятностный переход. Появление недетерминизма в данном случае не вызывает опасений, поскольку верификация модели с использованием model checker предполагает прогон всех ее состояний, при этом вероятностные каналы в соответствии с принятой моделью угроз игнорируются. Правила (C-EXT-PRC) и (C-EXT-RET) применяются для процедур, привилегия на выполнение которых (execute) предоставлена не пустому множеству ролей. Для таких процедур статические метки формальных параметров и возвращаемого

значения, представленного через x_{out}, определяются как $\{r_1, \ldots, r_n\}$, где $r_1, \ldots, r_n$ – роли, обладающие привилегией execute. Правила для обработки исключений и выхода из блока обработки исключений: (C-EXC) и (C-END-EXC), задаются аналогично правилам для условного оператора и цикла while. Оставшиеся правила не требуют пояснений.

Докажем ряд утверждений, чтобы показать, что модель параллельных вычислений, прошедшая проверку с использованием описанной ниже процедуры, является безопасной в соответствии с Определением 4.1. Введем обозначения:

$\mathbb{M}$ – множество возможных абстрактных состояний среды вычислений, т.е. функций, задающих отображение множества элементов среды вычислений на множество меток безопасности;

$\mathbb{C}$ – множество возможных глобальных шагов вычислений вида $\langle c_1, \ldots c_n \rangle$ для n сеансов (глобальный шаг определяется текущими состояниями счетчиков команд в активных сеансах). Если k – общее количество хранимых программных блоков, то: $\mathbb{C} = \bigcup_{i=1}^{k^n} C_i^1 \times \ldots C_i^n$, где C_i^m (при $m \in [1, n]$) – множество возможных шагов вычислений конкретной процедуры, выполняемой в сеансе m при i-м размещении (с повторами) k процедур по n сеансам. Отметим, что ситуацию, когда в сеансе завершено выполнение одной процедуры, и еще не начато выполнение другой, можно описать состоянием, в котором этому сеансу ставится в соответствие так называемый load шаг очередной процедуры.

$\mathbb{O} = \mathbb{C} \times \mathbb{M}$ – множество глобальных состояний модели.

Введем отношение частичного порядка на множестве абстрактных состояний среды вычислений: $M_1 \le M_2 \Leftrightarrow Dom(M_1) = Dom(M_2)$ и $\forall x \in Dom(M_1) : M_1[x] \sqsubseteq M_2[x]$, где $Dom(M_n)$ – область определения функции M_n, x – переменная среды вычислений.

Предположение 4.1. *Заданные в виде предикатов состояний инварианты безопасности гарантируют неразличимость эффектов соответствующих команд и выражений в низкоэквивалентных контекстах* (*Условие а*). *Справедливо также и обратное утверждение. Например*, ПЗИН $(x := e)$ *является истинным для начального отображения M множества элементов среды вычислений на множество меток тогда и только тогда*, *когда истинным является инвариант правила* (*C-EXT-ASSIGN*).

Предположение является обоснованным, так как правила безопасных вычислений, взятые за основу в представленной абстрактной семантике, соответствуют известным общепринятым правилам для простого императивного языка [10], для которых, в свою очередь, в [11] была предложена классическая схема ИН.

Лемма 4.1. *Если* $M_1 \le M_2$, *то для* $\forall c_i$ *из* $\langle c_1, \ldots c_i \ldots c_n \rangle \in \mathbb{C}$, *такого*, *что предикат* ПЗИН (c_i) *является истинным при начальном отображении множества переменных на множество меток* M_2 *и конечном отображении* M_2' *будем иметь*, *что предикат* ПЗИН (c_i) *является также истинным для начального отображения* M_1 *и конечного* – M_1'.

Доказательство. Истинность условия б) обеспечивается потокочувствительностью семантики – значения меток безопасности переменных изменяются в процессе вычислений. Докажем истинность условия а). В соответствии с правилами абстрактной семантики внешними эффектами обладают следующие команды и выражения: (E-TR-LIB), (C-EXT-ASSIGN), (C-EXT-SEL), (C-EXT-INS), (C-EXT-UPD), (C-EXT-DEL), (C-EXT-PRC), (C-EXT-RET) и (C-EXT-FETCH). Проведем доказательство для правила (C-EXT-ASSIGN).

Если для вычисления выражения e используется правило (E-CONST), то e обладает меткой $\bot$. Из истинности ПЗИН (c_i) для начального отображения M_2 следует, что $p \sqcup pc_1 \sqcup \ldots pc_n \sqsubseteq p_x$ (по Предположению 4.1), откуда имеем: $pc_1 \sqcup \ldots pc_n \sqsubseteq p_x$, тогда также справедливо и $\bot \sqcup pc_1 \sqcup \ldots pc_n \sqsubseteq p_x$, т.е. Условие а) справедливо для любого начального отображения, в том числе и M_1.

Если для вычисления выражения e используется правило (E-VAR), то из $M_1 \le M_2$ следует, что $p^1 \sqsubseteq p^2$, где $p^1 = M_1[x]$, $p^2 = M_2[x]$ (1). Тогда из истинности ПЗИН (c_i) для начального отображения M_2 следует, что $p^2 \sqcup pc_1 \sqcup \ldots pc_n \sqsubseteq p_x$ (2) (по Предположению 4.1). Из 1 и 2 имеем: $p^1 \sqcup pc_1 \sqcup \ldots pc_n \sqsubseteq p_x$, т.е. Условие а) справедливо для начального отображения M_1.

Теперь если последним правилом является (E-OPER), докажем по индукции на основе правил вычисления выражений, что если $\langle e, M_1 \rangle \Downarrow p^1$ и $\langle e, M_2 \rangle \Downarrow p^2$, то $p^1 \sqsubseteq p^2$ (1). Начальный шаг, когда e представляет собой константу или переменную, очевиден. По предположению индукции для e_1 и e_2 примем: $p_1^1 \sqsubseteq p_1^2$ и $p_2^1 \sqsubseteq p_2^2$, тогда $p_1^1 \sqcup p_2^1 \le p_1^2 \sqcup p_2^2$, то есть $p^1 \sqsubseteq p^2$. Индуктивный шаг доказан. Из $p^2 \sqcup pc_1 \sqcup \ldots pc_n \sqsubseteq p_x$ – следует из истинности ПЗИН (c_i) для начального отображения M_2, и 1 имеем: $p^1 \sqcup pc_1 \sqcup \ldots pc_n \sqsubseteq p_x$, то есть Условие а) справедливо для начального отображения M_1.

Для остальных правил доказательство выглядит аналогично.

Докажем утверждение, определяющие условие безопасной последовательной композиции вычислений в многопоточных системах.

Лемма 4.2. *Если для* c_i *из* $\langle c_1,\dots c_n\rangle_1 \in \mathbb{C}$, c_j *из* $\langle c_1,\dots c_n\rangle_2 \in \mathbb{C}$ *таких, что существуют переходы*: $\langle\langle c_1,\dots c_n\rangle_1, M_1\rangle \to^i_{1/n} \langle\langle c_1,\dots c_n\rangle_2, M_2\rangle$ *и* $\langle\langle c_1,\dots,c_n\rangle_2, M_2\rangle \to^j_{1/n} \langle\langle c_1,\dots,c_n\rangle_3, M_3\rangle$ *выполняется*: ПЗИН $(c_i)_{M_1,M_2}$ *и* ПЗИН $(c_j)_{M_2,M_3}$, *то выполняется также и* ПЗИН $(c_i;c_j)_{M_1,M_3}$.

Доказательство. Положим для некоторых состояний среды вычислений S_1 и S_2 справедливо: $S_1 \approx_{M_1,L} S_2$ (1), то есть состояния являются низкоэквивалентными относительно уровня конфиденциальности L для отображения M_1. Пусть $\langle\langle c_i;c_j\rangle, S_1\rangle \downarrow \langle S_1'', o_1\rangle$ – при последовательном выполнении двух шагов из начального состояния S_1 система переходит в состояние S_1'', переход сопровождается наблюдаемым поведением o_1 относительно уровня L, а также аналогично: $\langle\langle c_i;c_j\rangle, S_2\rangle \downarrow \langle S_2'', o_2\rangle$. Тогда существуют такие S_1', o_1' и o_1'', что $\langle c_i, S_1\rangle \downarrow \langle S_1', o_1'\rangle$ (2), $\langle c_j, S_1'\rangle \downarrow \langle S_1'', o_1''\rangle$ (3), $o_1 = \langle o_1', o_1''\rangle$ (4). Аналогично существуют такие S_2', o_2' и o_2'', что $\langle c_i, S_2\rangle \downarrow \langle S_2', o_2'\rangle$ (5), $\langle c_j, S_2'\rangle \downarrow \langle S_2'', o_2''\rangle$ (6), $o_2 = \langle o_2', o_2''\rangle$ (7). Из истинности ПЗИН $(c_i)_{M_1,M_2}$, 1, 2, 5 следует, что $S_1' \approx_{M_2,L} S_1'$ (8) и $o_1' \approx_L o_2'$ (9). Из истинности ПЗИН $(c_j)_{M_2,M_3}$, 8, 3, 6 следует, что $S_1'' \approx_{M_3,L} S_2''$ (10) и $o_1'' \approx_L o_2''$ (11). Наконец, из 1, 10, 9, 11 и 4 имеем ПЗИН $(c_i;c_j)_{M_1,M_3}$.

4.1. Этапы проверки выполнения условий безопасности информационных потоков

В общем виде распространение действующей системы управления доступом в программную среду БД для верификации информационных потоков, возникающих в хранимых процедурах и функциях, с точки зрения конфиденциальности (для целостности процедура выглядит аналогично), предлагается осуществлять в три этапа.

4.1.1. Трансляция требований существующей политики безопасности в метки элементов среды вычислений

На этом этапе требуется задать метки безопасности для входных значений, формальных параметров и возвращаемых значений процедур и функций, которые могут явно вызываться пользователями (пользовательскими приложениями). Очевидно, выбор меток должен соответствовать существующей политике управления доступом. Например, если привилегия execute на вызов некоторой процедуры предоставлена ролям: r_1, r_2, при этом существует такая подрешетка $(R_1, \sqsubseteq) : R_1 = \{r_1\}\uparrow$, в которой r_1 является нижней гранью, а T – верхней гранью и подрешетка $(R_2, \sqsubseteq) : R_2 = \{r_2\}\uparrow$, то формальным параметрам процедуры и ее возвращаемому значению присваивается метка $R_1 \cup R_2$. Предполагается, что параметрам и возвращаемым значениям “неразмеченных” программных блоков по умолчанию назначается наименее строгий элемент решетки меток безопасности в предположении, что доступ к таким программным блокам неограничен. Например, в соответствии с расширенной решеткой меток безопасности (рис. 9б) и требованиями политики:

– вызов функции *f_getsubmissions* возможен, если в сеансе установлена роль *manager* или выполнено условие ”time_expired” (в этом случае допускается вызов функции в сеансе любого пользователя);

– вызов процедуры *p_submit_paper* может осуществляться любым пользователем;

– вызов процедуры *p_change_status* может осуществляться пользователем, в сеансе которого установлена роль *reviewer*;

разметка соответствующих программных блоков выглядит, как показано в табл. 4.

Метки входных значений определяются аналитиком исходя из понимания их фактического уровня конфиденциальности. Для доверенных сторонних программных блоков в явном виде задаются метки: формальных параметров, выходного значения и эффекта записи (writing effect). Чистые функции разметки не требуют. Для простоты будем полагать, что внешние программные блоки не выбрасывают не обработанных исключений (достигается универсальным обработчиком WHEN OTHERS null;).

Аналогично задаются метки безопасности для иных элементов среды вычислений: атрибутов отношений, доступ к которым предоставлен явно некоторым пользователям, исключений проверяемых (не сторонних) программных блоков, переменных, выводимых в выходные потоки.

4.1.2. Построение и проверка модели параллельных вычислений с использованием линейной темпоральной логики действий

Далее с использованием *TLA+* спецификаций строится модель выполнения *PL/SQL* блоков в параллельных сеансах пользователей. Модель описывает все возможные переходы системы,

Таблица 4. Пример разметки программных блоков БД

	Входные значения	Формальные параметры	Выходное значение
f_getsubmissions	f_gs_i: $\bot$	f_gs_p1:	f_gs_r:
		$\{m, o, g', r', m', o'\}$	$\{m, o, g', r', m', o'\}$
p_submit_paper	p_sp_i1: $\bot$,	p_sp_p1: $\bot$,	
	p_sp_i2: $\bot$,	p_sp_p2: $\bot$,	
	p_sp_i3: $\bot$,	p_sp_p3: $\bot$,	
	p_sp_i4: $\bot$	p_sp_p4:	
p_change_status	p_cs_i1: $\bot$,	p_cs_p1: $\{r, o, r', o'\}$,	
	p_cs_i2: $\bot$ $\{m, o, g', r', m', o'\}$	p_sp_p2: $\{r, o, r', o'\}$	

описанной в п. 4, для множества начальных состояний $D \times C_{init} \to \{M_{init}\}$ и множества всех состояний $\mathbb{O}$ при условии, что в каждом из n сеансов (задается как параметр модели) выполняется только один программный блок. Заданное условие упрощает модель, при этом за счет композиционной природы свойства ПЗИН (Лемма 4.2) результат успешной верификации можно обобщить до системы, в которой в каждом сеансе пользователя выполняется произвольное количество хранимых процедур и функций. Здесь C_{init} – множество начальных шагов вычислений имеющихся процедур, M_{init} – начальное абстрактное состояние среды вычислений, в котором отображение глобальных переменных и выходных потоков на метки безопасности задается явно, а локальным переменным присваивается минимальная метка $\bot$.

Алгоритм проверки циклический.

Предварительный этап.

Модель инициализируется значениями констант. Они определяют количество параллельных сеансов, множество пользователей, алфавит меток безопасности. В соответствии с имеющимися привилегиями ролей и дополнительными требованиями политики безопасности осуществляется разметка элементов модели, ассоциируемых с проверяемыми программными блоками, а также используемыми в вычислениях доверенными сторонними процедурами. Задаются начальные состояния элементам модели, соответствующим глобальным переменным (таблицам) среды вычислений. Элементам, соответствующим локальным переменным, присваивается минимальная метка безопасности.

Шаг 1. Осуществляется прогон модели. Если нарушений инварианта безопасности не выявлено, осуществляется сопоставление начального и конечного отображения глобальных переменных на метки безопасности G_{init} и G_{fin}. Если начальное и конечное отображение совпадают, алгоритм завершается, рекомендаций по инструментированию кода или изменению привилегий системы разграничения доступа не требуется. В противном случае начальное отображение M_{init} изменяется следующим образом:

$$M_{init}(x) = \begin{cases} G_{fin}(x), & \text{если} \quad x \in Dom(G_{fin}) \\ M_{init}(x), & \text{в противном случае} \end{cases}$$

и шаг 1 повторяется. Если для некоторого состояния выявлено нарушение инварианта, происходит переход к шагу 2.

Шаг 2. Производится анализ трассы, приведшей к возникновению ошибки, в спецификацию вносятся соответствующие изменения, в список рекомендаций добавляется новая рекомендация, производится переход к шагу 1.

Как уже отмечалось, метки глобальных переменных возрастают монотонно, решетка меток конечна, поэтому алгоритм также конечен.

Утверждение 4.1. *Успешное завершение алгоритма гарантирует безопасное функционирование системы в смысле ПЗИН при неограниченном количестве программных блоков, выполняемых в каждом сеансе.*

Доказательство. Положим алгоритм проверки завершен успешно, для некоторого отображения множества глобальных переменных на множество меток безопасности G'. С учетом ранее сформулированной стратегии действий при “простаивании” сеанса после завершения выполнения программного блока (очередным действием сеанса становится load – операция следующего программного блока), а также, принимая во внимание сброс меток безопасности локальных переменных при новом запуске процедур и возможность их инициализации только значениями, извлекаемыми из глобальных переменных, для каждого возможного состояния системы $\langle C_k, M_k \rangle \in \mathbb{O}$, где $C_k = \langle c_1, \ldots c_n \rangle_k$ будем иметь: для $\forall i \in \{1, \ldots n\}$ справедливо, что при переходе $\langle \langle c_1, \ldots c_n \rangle_k, M_k \rangle \to^i_{1/n} \langle \langle c_1, \ldots c_n \rangle_l, M_l \rangle$ инвариант безопасности не нарушается, то есть в соответствии с Предположением 4.1 для c_i выполняет-

ся условие а) Определения 4.1. Истинность условия б) обеспечивается потокочувствительностью абстрактной семантики. Таким образом, для любого контекста $\langle c_i, M_k \rangle$ выполняется условие ПЗИН $(c_i)_{M_k, M_l}$. Истинность утверждения устанавливается по структурной индукции на основе правил переходов системы (см. п. 4). Истинность индуктивного шага следует из Леммы 4.2.

Также отметим, в соответствии с Леммой 4.1 приведение меток таблиц, доступ к которым явно не предоставлен ни одной роли, в соответствие с некоторым конечным отображением множества глобальных переменных на множество меток безопасности G' не требуется.

Справедливость выполнения условий Определения 4.1 в части "расхождения" вычислений для проверенных в соответствии с алгоритмом программ может быть обеспечена исключением самой возможности бесконечных вычислений в рамках выделенного сеанса путем использования так называемых ресурсных ограничений на уровне СУБД.

В данной работе подробно не рассматривается процедура трансформации *PL/SQL* кода в формулы линейной темпоральной логики действий, однако, в Приложении 1 представлен результат трансформации процедуры:

```
1 p_chahge_status (s_id number, stat number)
2 IS
3 BEGIN
4 UPDATE SUBMISSIONS
5 SET STATUS = stat
6 WHERE SUBMISSION_ID = s_id
7 END p_chahge_status;
```

Для генерации спецификаций используется абстрактная семантика (Приложение 2). Проверка выполнения условий прогресс-зависимого информационного невлияния на каждом шаге вычислений осуществляется с помощью инварианта безопасности вида:

$$NonInt\_Inv \triangleq p_{expr} \sqcup p_{pc} \sqsubseteq p_{out}$$

Здесь p_{expr} – метка выражения, которое выводится в выходной поток (присваивается глобальной переменной), p_{pc} – метка текущего счетчика команд, pc_{out} – метка выходного потока.

4.1.3. Выработка рекомендаций по инструментированию кода и изменению общей политики

При проверке модели с помощью model checker могут возникать нарушения инварианта безопасности. В таких случаях с целью устранения запрещенных информационных потоков аналитик исправляет спецификацию, корректируя метки размеченных программных блоков (элементов среды вычислений, доступных внешнему наблюдателю) или используя операторы декласс-ификации. На основе внесенных исправлений формируются рекомендации по инструментированию кода хранимых процедур и функций или изменению правил разграничения доступа.

Пример обработки нарушения инварианта

Пусть в системе реализуется политика безопасности, описанная выше. Исключением является требование по ограничению доступа к функции *f_getsubmissions*. Допустим, в соответствии с политикой вызывать функцию *f_getsubmissions* могут любые пользователи, если выполняется условие "time_expired". Таким образом, для настройки привилегий могут применяться следующие команды:

```
1 grant execute on f_getsubmissions to PUBLIC;
2 grant execute on p_submit_paper_to PUBLIC;
3 grant execute on p_change_status to reviewer;
```

Разметка программных блоков на этапе трансляции требований политики безопасности в метки элементов среды вычислений выполняется, как описано в табл. 4, за исключением формального

Таблица 5. Варианты корректировки модели

	Исправление спецификации	Практические рекомендации
Вариант 1 (*корректировка политики*)	f_gs_p1: $\{m, o\}$,	grant execute on
	f_gs_r: $\{m, o\}$	f_getsubmissions to managers
Вариант 2 (*деклассификация*)	f_gs_p1: $\{g', r', m', o'\}$,	if time_expired(c_id) then ...
	f_gs_r: $\{g', r', m', o'\}$	return v_submissions; else null;
Вариант 3 (*деклассификация, корректировка политики*)	f_gs_p1: $\{m, o, g', r', m', o'\}$	if time_expired(c_id) or session_role.exists('manager') or
	f_gs_r: $\{m, o, g', r', m', o'\}$	session_role.exist('organizer') then ... return v_submissions; else null;
Вариант 4 (*ложное срабатывание – игнорирование*)	(C-EXT-RET): Inv – ignore	

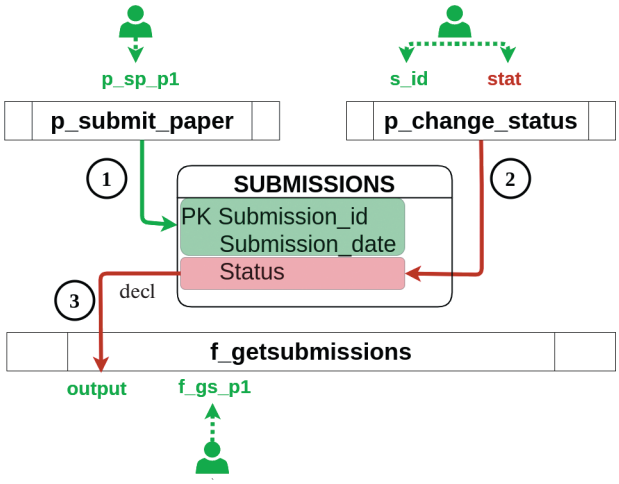


Рис. 10. Нарушение инварианта информационного невлияния.

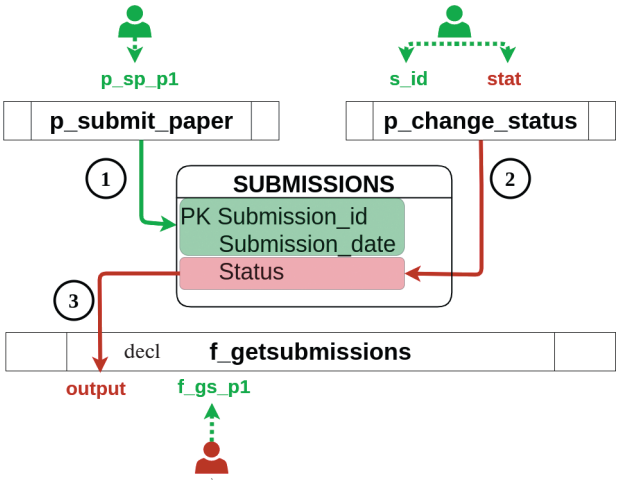


Рис. 11. Деклассификация – рекомендации по изменению кода.

параметра и возвращаемого значения функции *f_getsubmissions*, для них выбирается минимальная метка $\perp$. В этом случае при прогоне модели возникает нарушение инварианта. Трасса переходов, предшествующих ошибке выглядит так:

```
FATAL InvariantViolation:
    Error validating NonInt_Inv:
    {m,o,g',r',m',o'} is greater than bot
    bob → p_f_getsubmissions9
    bob → p_f_getsubmissions6
    mattew → p_change_status_exit
    mattew → p_change_status4
    bob → p_f_getsubmissions_load
    mattew → p_change_status_load
    alex → p_submit_paper_exit
    alex → p_submit_paper4
    alex → p_submit_paper_load
```

Запрещенный информационный поток возникает вследствие переноса входного значения *stat* (p_cs_i2) функции *f_getsubmissions*, обладающего меткой $\{m,o,g',r',m',o'\}$, в выходную переменную *out* (f_gs_p1), обладающую меткой $\perp$ (рис. 10).

В табл. 5 приведены возможные варианты исправления модели и выработки рекомендаций для разработчиков (администраторов безопасности).

Первый вариант предполагает ужесточение глобальной политики управления доступом – рис. 11, второй вариант основан на процедуре деклассификации данных, которая может быть реализована путем внесения в соответствующие участки кода проверок выполнения условий деклассификации – рис. 12. Третий – наиболее гибкий вариант, предполагает внесение минимальных ограничений, обеспечивающих устранение запрещенного информационного потока. Поскольку при построении модели приходится прибегать к абстрагированию, точность анализа теряется, и в определенных случаях нарушение инварианта может классифицироваться как ложное срабатывание (Вариант 4). Например, в описанной абстрактной семантике источником неточности, как отмечалось выше, могут выступать правила: (C-IF) и (C-WHILE).

5. ЗАКЛЮЧЕНИЕ

Основной вклад данной работы в развитие теории КИП заключается в разработке подхода к разделению функций написания кода и описания политики безопасности прикладного уровня. Формализована процедура распространения требований ролевого разграничения доступа в про-

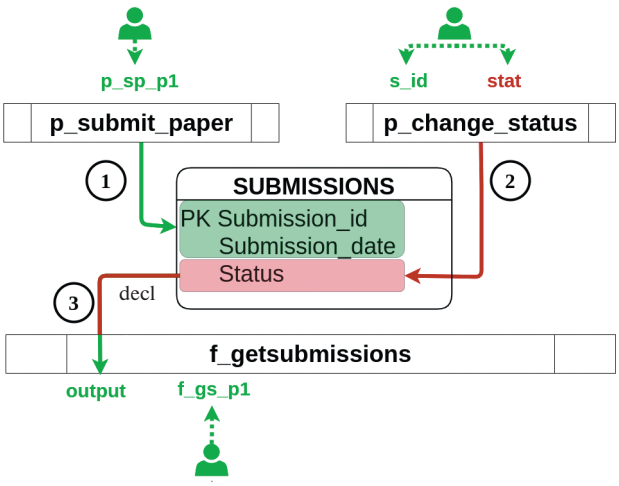


Рис. 12. Изменение глобальной политики.

граммную среду БД, предложен алгоритм проверки модели вычислений и подготовки рекомендаций по инструментированию кода (изменению политики). Сформулированы условия миграции механизма проверки условий безопасности вычислений из языковой среды в область формальной верификации.

Очевидно, реализация КИП на языковом уровне не имеет альтернативы с точки зрения гарантий безопасности. Объективно, в силу влияния человеческого фактора на процесс написания (исправления) кода, предложенный подход не может полностью исключить появления в программном обеспечении запрещенных информационных потоков. Кроме того, абстрагирование, которое применяется для сокращения числа состояний до приемлемого, понижает точность анализа и повышает критичность задачи выявления "ложных" срабатываний. Тем не менее предложенный подход на практике может оказаться более жизнеспособным с учетом тенденций развития отрасли.

К перспективным направлениям дальнейших исследований целесообразно отнести: оптимизацию разработанной абстрактной модели параллельного выполнения хранимых программных блоков, разработку утилиты разметки и трансформации *PL/SQL* кода в спецификации *TLA+*, выработку рекомендаций по встраиванию вызовов проверенных процедур и функций в прикладное ПО.

ПРИЛОЖЕНИЕ 1.
ПРИМЕР ТРАНСФОРМАЦИИ PL/SQL2TLA+

```
p_change_status_load(id) ≜
 IF XLocks = Undef
    THEN
    ∧ XLocks' = id
    ∧ Sessions' = [Sessions EXCEPT ! [id ]["SessionM"] = Sessions[id]["SessionM"] ∘
                  ⟨min; {⟨u1; ⟨[t_expire ↦ {NONE}]; [guest ↦ {NONE};
                        reviewer ↦ {NONE}; manager ↦ {u1}; organizer ↦ {NONE}⟩⟩;
                  ⟨u1; ⟨[t_expire ↦ {}]; [guest ↦ {NONE}; reviewer ↦ {NONE};
                        manager ↦ {NONE}; organizer ↦ {NONE}⟩⟩}⟩]
    ∧ New2Old' = ⟨⟨p_cs_p_s_id(id):policy; p_cs_p_stat(id).policy⟩,
                  ⟨min; {⟨u1; ⟨[t_expire ↦ {NONE}]; [guest ↦ {NONE};
                        reviewer ↦ {NONE}; manager ↦ {u1}; organizer ↦ {NONE}]⟩⟩;
                  ⟨u1; ⟨[t expire ↦ {}]; [guest ↦ {NONE}; reviewer ↦ {NONE};
                        manager ↦ {NONE}; organizer ↦ {NONE}]⟩⟩}⟩⟩
    ∧ Ignore' = 1
    ∧ StateE' = SLocks'[id]
    ∧ UNCHANGED ⟨VPol; SLocks⟩
 ELSE UNCHANGED vars

p_change_status4(id) ≜
    ∧ update(id; ⟨"col_submissions_status"⟩, ⟨load(id; p_cs_p_stat(id))⟩;
              LUB(VPol["col_submissions_submission_id"].policy, load(id; p_cs_p_s_id(id)));
                   ⟨"p_change_status"; "exit"⟩
    ∧ Trace' = Append(Trace; ⟨id; "p_change_status4"⟩)
    ∧ Ignore' = 0
    ∧ StateE' = SLocks'[id]
    ∧ UNCHANGED ⟨XLocks; SLocks⟩

p_change_status_exit(id) ≜
    ∧ IF Len(Sessions[id]["StateRegs"]) = 1
       THEN XLocks' = Undef
       ELSE XLocks' = XLocks
    ∧ Sessions' =
      [Sessions EXCEPT
```

! [*id*][“StageRegs”] = Tail (*Sessions*[*id*][“StateRegs”]) ∘ ⟨⟩;
! [*id*][“SessionM”] = *SubSeq*(*Sessions*[*id*][“SessionM”]; 1; *Len*(*Sessions*[*id*][“SessionM”]) − 2)]
∧ *Trace'* = *Append*(*Trace*; ⟨*id* ; “p_change_status_exit”⟩)
∧ *Ignore'* = 1
∧ *StateE'* = *SLocks'*[*id*]
∧ UNCHANGED ⟨*New2Old; VPol; SLocks*⟩

p_change_status(*id*; *st*) ≜
CASE *Head*(*st*):*pc* [2] = *“lbl_*4” → *p_change_status*4(*id*)
□ *Head*(*st*):*pc* [2] = *“exit”* → *p_change_status_exit*(*id*)
□ OTHER → UNCHANGED *vars*

ПРИЛОЖЕНИЕ 2.
АБСТРАКТНАЯ СЕМАНТИКА ИНФОРМАЦИОННЫХ ПОТОКОВ

EXPRESSIONS:

$$\textbf{(E-CONST)}\frac{}{\langle v, M\rangle \Downarrow \bot} \quad \textbf{(E-VAR)}\frac{}{\langle x, M\rangle \Downarrow M[x]} \quad \textbf{(E-OPER)}\frac{\langle e_1, M\rangle \Downarrow p_1 \quad \langle e_2, M\rangle \Downarrow p_2}{\langle e_1 \odot e_2, M\rangle \Downarrow p_1 \sqcup p_2}$$

$$\textbf{(E-REC-FLD)}\frac{\langle x, M\rangle \Downarrow p}{\langle x_r.x, M\rangle \Downarrow p} \quad \textbf{(E-VAR-REC)}\frac{\langle x_r.x_1, M\rangle \Downarrow p_1 \ldots \langle x_r.x_n, M\rangle \Downarrow p_n}{\langle x_r, M\rangle \Downarrow p_1 \sqcup p_2 \ldots \sqcup p_n}$$

$$x_{cur} \triangleq \begin{array}{l}\textbf{select } e_1,\ldots,e_n \\ \textbf{from } x_{t_1},\ldots,x_{t_n}\end{array} \textbf{where } cnd$$

$$\textbf{(E-TAB-COL)}\frac{}{\langle x_t.c, M\rangle \Downarrow M[x_t.c]} \quad \textbf{(E-CUR)}\frac{\langle e_1, M\rangle \Downarrow p_1 \ldots \langle x_n, M\rangle \Downarrow p_n \quad \langle cnd, M\rangle \Downarrow p_{cnd}}{\langle x_{cur}, M\rangle \Downarrow p_1 \sqcup p_2 \ldots \sqcup p_n \sqcup p_{cnd}}$$

$$\textbf{(E-VAR-ARR)}\frac{\langle x_a[1], M\rangle \Downarrow p_{x_a[1]} \ldots \langle x_a[n], M\rangle \Downarrow p_{x_a[n]} \quad \langle n, M\rangle \Downarrow p_{len}}{\langle x_a, M\rangle \Downarrow p_{x_a[1]} \sqcup p_{x_a[2]} \ldots \sqcup p_{x_a[n]} \sqcup p_{len}}$$

$$\textbf{(E-CUR-FND)}\frac{\langle x, M\rangle \Downarrow p_x}{\langle x\%\ \text{found}, M\rangle \Downarrow p_x} \quad \textbf{(E-PUR-LIB)}\frac{x_f - pure \quad \langle e_1, M\rangle \Downarrow p_1 \ldots \langle e_n, M\rangle \Downarrow p_n}{\langle PC, x_f(x_1 \to e_1, \ldots x_n \to e_n), M\rangle \Downarrow p_1 \sqcup \ldots p_n}$$

$$\textbf{(E-TR-LIB)}\frac{\begin{array}{r}x_f - trusted \quad \langle e_1, M\rangle \Downarrow p_1 \ldots \langle e_m, M\rangle \Downarrow p_m \quad \langle f_{out}, M\rangle \Downarrow p_{out} \quad x_f \rightsquigarrow p_w \\ \langle x_1, M\rangle \Downarrow p_{x_1} \ldots \langle x_m, M\rangle \Downarrow p_{x_m} \quad \textbf{inv} : \wedge p_1 \sqsubseteq p_{x_1} \ldots p_m \sqsubseteq p_{x_m} \\ \wedge pc_1 \sqcup pc_2 \ldots pc_n \sqsubseteq p_w\end{array}}{\langle PC, x_f(x_1 \to e_1, \ldots x_m \to e_m), M\rangle \Downarrow p_{out}}$$

$$\textbf{(E-C-OPER)}\frac{\langle e_1, M\rangle \Downarrow p_1 \quad \langle e_2, M\rangle \Downarrow p_2}{\langle e_1 \times e_2, M\rangle \Downarrow p_1 \sqcup p_2} \quad \textbf{(E-L-OPER)}\frac{\langle cnd_1, M\rangle \Downarrow p_1 \quad \langle cnd_2, M\rangle \Downarrow p_2}{\langle cnd_1 \star cnd_2, M\rangle \Downarrow p_1 \sqcup p_2}$$

STATEMENTS:

$$\textbf{(C-ASSIGN)}\frac{\langle e, M\rangle \Downarrow p}{\langle PC, x := e, M\rangle \to \langle PC, \textbf{null}, M[x \mapsto p \sqcup pc_1 \ldots pc_n]\rangle}$$

$$\textbf{(C-CLOSE)}\frac{}{\langle PC, \textbf{close } x, M\rangle \to \langle PC, \textbf{null}, M\rangle} \quad \textbf{(C-OPEN)}\frac{}{\langle PC, \textbf{open } x, M\rangle \to \langle PC, \textbf{null}, M\rangle}$$

$$\textbf{(C-EXT-ASSIGN)}\frac{\langle e, M\rangle \Downarrow p \quad \langle x, M\rangle \Downarrow p_x \quad \textbf{inv}{:}p \sqcup pc_1 \ldots pc_n \sqsubseteq p_x}{\langle PC, x := e, M\rangle \to \langle PC, \textbf{null}, M\rangle}$$

$$\textbf{(C-IF)}\frac{\langle e, M\rangle \Downarrow p \quad \textbf{inv}{:}e - local}{\langle PC, \textbf{if } e \textbf{ then } c_1 \textbf{ else } c_2, M\rangle \to \langle p :: PC, c_1 \vee c_2, M\rangle}$$

$$\textbf{(C-END-IF)}\ \frac{}{\langle p :: PC, \textbf{endif}, M\rangle \to \langle PC, \textbf{null}, M\rangle}$$

$$\textbf{(C-WHILE)}\ \frac{\langle e, M\rangle \Downarrow p \quad \textbf{inv}{:}e - local}{\langle PC, \textbf{while}\ e\ \textbf{do}\ c, M\rangle \to \langle p :: PC, c \vee \textbf{null}, M\rangle}$$

$$\textbf{(C-COMMIT)}\ \frac{}{\langle PC, \textbf{commit}, M\rangle \to \langle PC, \textbf{null}, M\rangle}$$

$$\textbf{(C-END-WHILE)}\ \frac{}{\langle p :: PC, \textbf{endwhile}, M\rangle \to \langle PC, \textbf{null}, M\rangle}$$

$$\textbf{(C-ROLLBACK)}\ \frac{}{\langle PC, \textbf{rollback}, M\rangle \to \langle PC, \textbf{null}, M\rangle}$$

$$\textbf{(C-SEL)}\ \frac{\langle e_1, M\rangle \Downarrow p_1 \ldots \langle e_m, M\rangle \Downarrow p_m \quad \langle cnd, M\rangle \Downarrow p_{cnd}}{\left\langle PC, \begin{array}{c}\textbf{select}\ e_1,\ldots,e_m\ \textbf{into}\ x_1,\ldots,x_m \\ \textbf{from}\ x_{t_1},\ldots,x_{t_k}\ \textbf{where}\ cnd\end{array}, M\right\rangle \to \left\langle PC, \textbf{null}, M\begin{bmatrix} x_1 \mapsto p_1 \sqcup p_{cnd} \sqcup pc_1 \ldots pc_n, \\ \ldots \\ x_m \mapsto p_m \sqcup p_{cnd} \sqcup pc_1 \ldots pc_n \end{bmatrix}\right\rangle}$$

$$\textbf{(C-EXT-SEL)}\ \frac{\begin{array}{l}\langle e_1, M\rangle \Downarrow p_1 \ldots \langle e_m, M\rangle \Downarrow p_m \quad \langle cnd, M\rangle \Downarrow p_{cnd} \quad \textbf{inv}{:} \wedge\, p_1 \sqcup p_{cnd} \sqcup pc_1 \ldots px_1 \sqsubseteq p_{x_1} \wedge \ldots \\ \langle x_1, M\rangle \Downarrow p_{x_1} \ldots \langle e_m, M\rangle \Downarrow p_{x_m} \qquad\qquad \wedge\, p_m \sqcup p_{cnd} \sqcup pc_1 \ldots px_m\end{array}}{\left\langle PC, \begin{array}{c}\textbf{select}\ e_1,\ldots,e_m\ \textbf{into}\ x_1,\ldots,x_m \\ \textbf{from}\ x_{t_1},\ldots,x_{t_k}\ \textbf{where}\ cnd\end{array}, M\right\rangle \to \langle PC, \textbf{null}, M\rangle}$$

$$\textbf{(C-INS)}\ \frac{\langle e_1, M\rangle \Downarrow p_1 \ldots \langle e_m, M\rangle \Downarrow p_m}{\left\langle PC, \begin{array}{c}\textbf{insert into}\ x_t(c_1,\ldots,c_m) \\ \textbf{values}\ (e_1,\ldots,e_m)\end{array}, M\right\rangle \to \left\langle PC, \textbf{null}, M\begin{bmatrix} x_t.c_1 \mapsto M[x_t.c_1] \sqcup p_1 \sqcup pc_1 \ldots pc_n, \\ \ldots \\ x_t.c_m \mapsto M[x_t.c_m] \sqcup p_m \sqcup pc_1 \ldots pc_n \end{bmatrix}\right\rangle}$$

$$\textbf{(C-EXT-INS)}\ \frac{\begin{array}{l}\langle e_1, M\rangle \Downarrow p_1 \ldots \langle e_m, M\rangle \Downarrow p_m \quad \textbf{inv}{:} \wedge\, p_1 \sqcup pc_1 \ldots pc_n \sqsubseteq M[x_t.c_1] \wedge \ldots \\ \qquad\qquad \wedge p_m \sqcup pc_1 \ldots pc_n \sqsubseteq M[x_t.c_m]\end{array}}{\left\langle PC, \begin{array}{c}\textbf{insert into}\ x_t(x_t.c_1,\ldots,x_t.c_m) \\ \textbf{values}\ (e_1,\ldots,e_m)\end{array}, M\right\rangle \to \langle PC, \textbf{null}, M\rangle}$$

$$\textbf{(C-UPD)}\ \frac{\langle e_1, M\rangle \Downarrow p_1 \ldots \langle e_m, M\rangle \Downarrow p_m \quad \langle cnd, M\rangle \Downarrow p_{cnd}}{\left\langle \begin{array}{c}\textbf{update}\ x_t \\ PC, \textbf{set}\ x_t.c_1 = e_1,\ldots, M \\ \textbf{where}\ cnd\end{array}\right\rangle \to \left\langle PC, \textbf{null}, M\begin{bmatrix} x_t.c_1 \mapsto M[x_t.c_1] \sqcup p_1 \sqcup p_{cnd} \sqcup pc_1 \ldots pc_n, \\ \ldots \\ x_t.c_m \mapsto M[x_t.c_m] \sqcup p_m \sqcup p_{cnd} \sqcup pc_1 \ldots pc_n \end{bmatrix}\right\rangle}$$

$$\textbf{(C-EXT-UPD)}\ \frac{\begin{array}{l}\langle e_1, M\rangle \Downarrow p_1 \ldots \langle e_m, M\rangle \Downarrow p_m \quad \textbf{inv}{:} \wedge\, p_1 \sqcup p_{cnd} \sqcup pc_1 \ldots pc_n \sqsubseteq M[x_t.c_1] \wedge \ldots \\ \langle cnd, M\rangle \Downarrow p_{cnd} \qquad\qquad \wedge p_m \sqcup p_{cnd} \sqcup pc_1 \ldots pc_n \sqsubseteq M[x_t.c_m]\end{array}}{\left\langle PC, \begin{array}{c}\textbf{update}\ x_t \\ \textbf{set}\ x_t.c_1 = e_1,\ldots\textbf{where}\ cnd\end{array}, M\right\rangle \to \langle PC, \textbf{null}, M\rangle}$$

$$\textbf{(C-DEL)}\ \frac{}{\left\langle PC, \begin{array}{c}\textbf{delete from}\ x_t \\ \textbf{where}\ cnd\end{array}, M\right\rangle \to \left\langle PC, \textbf{null}, M\begin{bmatrix} x_t.c_1 \mapsto M[x_t.c_1] \sqcup p_{cnd} \sqcup pc_1 \ldots pc_n, \\ \ldots \\ x_t.c_m \mapsto M[x_t.c_m] \sqcup p_{cnd} \sqcup pc_1 \ldots pc_n \end{bmatrix}\right\rangle}$$

$$\textbf{(C-EXT-DEL)}\frac{\begin{array}{c}\textbf{inv}: \wedge\ p_{cnd} \sqcup pc_1 \ldots pc_n \sqsubseteq M[x_t.c_1] \wedge \ldots \\ \wedge\ p_{cnd} \sqcup pc_1 \ldots pc_n \sqsubseteq M[x_t.c_m]\end{array}}{\left\langle PC, \begin{array}{c}\textbf{delete from}\ x_t \\ \textbf{where}\ cnd\end{array}, M\right\rangle \to \langle PC, \textbf{null}, M\rangle}$$

$$\textbf{(C-PROC)}\frac{\langle e_1, M\rangle \Downarrow p_1 \ldots \langle e_m, M\rangle \Downarrow p_m}{\langle PC, x_f(x_1 \to e_1, \ldots x_m \to e_m), M\rangle \to \left\langle PC, \textbf{null}, M\begin{bmatrix} x_1 \mapsto p_1 \sqcup pc_1 \ldots pc_n, \\ \ldots \\ x_m \mapsto p_m \sqcup pc_1 \ldots pc_n \end{bmatrix}\right\rangle}$$

$$\textbf{(C-EXT-PRC)}\frac{\langle e_1, M\rangle \Downarrow p_1 \ldots \langle e_n, M\rangle \Downarrow p_n \quad \langle x_1, M\rangle \Downarrow p_{x_1} \ldots \langle x_n, M\rangle \Downarrow p_{x_n} \quad \textbf{inv}{:}p_1 \sqsubseteq p_{x_1} \ldots p_n \sqsubseteq p_{x_n}}{\langle PC, x_f(x_1 \to e_1, \ldots x_n \to e_n), M\rangle \to \langle PC, \textbf{null}, M[x_1 \mapsto p_1, \ldots x_n \mapsto p_n]\rangle}$$

$$\textbf{(C-RET)}\frac{\langle e, M\rangle \Downarrow p_1}{\langle PC, \textbf{return}(x_{out} \to e), M\rangle \to \langle PC, \textbf{null}, M[x_{out} \mapsto p_1 \sqcup pc_1 \ldots pc_n]\rangle}$$

$$\textbf{(C-EXT-RET)}\frac{\langle e, M\rangle \Downarrow p_1 \quad \langle x_{out}, M\rangle \Downarrow p_2 \quad \textbf{inv}{:}p_1 \sqcup pc_1 \ldots pc_n \sqsubseteq p_2}{\langle PC, \textbf{return}(x_{out} \to e), M\rangle \to \langle PC, \textbf{null}, M\rangle}$$

$$\textbf{(C-FETCH)}\frac{\langle x_{cur}, M\rangle \Downarrow p}{\langle PC, \textbf{fetch}\ x_{cur}\ \textbf{into}\ x, M\rangle \to \langle PC, \textbf{null}, M[x \mapsto p \sqcup pc_1 \ldots pc_n]\rangle}$$

$$\textbf{(C-EXT-FETCH)}\frac{\langle x_{cur}, M\rangle \Downarrow p \quad \langle x, M\rangle \Downarrow p_x \quad \textbf{inv}{:}p \sqcup pc_1 \ldots pc_n \sqsubseteq p_x}{\langle PC, \textbf{fetch}\ x_{cur}\ \textbf{into}\ x, M\rangle \to \langle PC, \textbf{null}, M\rangle}$$

$$\textbf{(C-OR)}\frac{i \in 1,2}{\langle PC, c_1 \vee c_2\rangle \to_{1/2} \langle PC, c_i, M\rangle}$$

$$\textbf{(C-EXC)}\frac{\langle x_{exc}, M\rangle \Downarrow p}{\langle PC, \textbf{when}\ x_{exc}\ \textbf{then}\ c, M\rangle \to \langle p :: PC, c, M\rangle} \qquad \textbf{(C-NULL)}\frac{}{\langle PC, \textbf{null}, M\rangle \to \langle PC, M\rangle}$$

$$\textbf{(C-END-EXC)}\frac{}{\langle p :: PC, \textbf{endexc}, M\rangle \to \langle PC, \textbf{null}, M\rangle}$$

$$\textbf{(C-THROW)}\frac{}{\langle PC, \textbf{throw}\ x_{exc}, M\rangle \to \langle PC, \textbf{null}, M\rangle}$$

$$\textbf{(C-SEQ-1)}\frac{\langle PC, c_1, M\rangle \to \langle PC, c_1', M\rangle}{\langle PC, c_1; c_2, M\rangle \to \langle PC, c_1'; c_2, M\rangle} \qquad \textbf{(C-SEC-2)}\frac{\langle PC, c_1, M\rangle \to \langle PC, M\rangle}{\langle PC, c_1; c_2, M\rangle \to \langle PC, c_2, M\rangle}$$

GLOBAL RULES:

$$\textbf{(GLOB-1)}\frac{O(d) = c_i \quad \langle PC_i, c_i, M\rangle \to \langle PC_k, c_k, M'\rangle}{\langle\langle PC_1, c_1\rangle \ldots \langle PC_n, c_n\rangle\rangle, M\rangle \to^i_{1/n} \langle\langle\langle PC_1, c_1\rangle \ldots \langle PC_{i-1}, c_{i-1}\rangle\langle PC_k, c_k\rangle \ldots \langle PC_n, c_n\rangle\rangle, M'\rangle}$$

$$\textbf{(GLOB-2)}\frac{O(d) = c_i \quad \langle PC_i, c_i, M\rangle \to \langle PC_i, M\rangle}{\langle\langle PC_1, c_1\rangle \ldots \langle PC_n, c_n\rangle\rangle, M\rangle \to^i_{1/n} \langle\langle\langle PC_1, c_1\rangle \ldots \langle PC_{i-1}, c_{i-1}\rangle\langle PC_{i+1}, c_{i+1}\rangle \ldots \langle PC_n, c_n\rangle\rangle, M\rangle}$$

СПИСОК ЛИТЕРАТУРЫ

1. *Fischer P., Katkalov K., Stenzel K., Reif W.* Formal Verification of Information Flow Secure Systems with IFlow, 2012
2. *Graf J., Hecker M., Mohr M., Snelting G.* Checking Applications using Security APIs with JOANA, 2015. http://www.dsi.unive.it/ focardi/ASA8/.
3. *Myers C.A., Liskov B.* A decentralized model for information flow control // ACM SIGOPS Operating Systems Review. 1997. № 5. P. 129–142.
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. *Pottier F., Simonet V.* Information Flow Inference for ML // ACM Transactions on Programming Languages and Systems, 2003. P. 117–158.
6. *Лесько С.А.* Модели и сценарии реализации угроз для интернет-ресурсов // Российский технологический журнал. 2020. Т. 8. № 6. С. 9–33.
7. *Goguen A.J., Meseguer J.* Security policies and security models // 1982 IEEE Symposium on Security and Privacy. IEEE, 1982. P. 11–11.
8. *Hedin D., Sabelfeld A.* A Perspective on Information-Flow Control // Software Safety and Security. 2012. P. 319–347.
9. *Bell D., LaPadula Lj J.* Secure computer systems: Mathematical foundations // Data Base. 1973. MTR-2547. V. I. P. 513–523.
10. *Denning E.D., Denning J.P.* Certification of Programs for Secure Information Flow // Communications of the ACM. 1977. № 7. P. 504–513.
11. *Volpano D.I., Cynthia S.G.* Sound type system for secure flow analysis // Journal of Computer Security. 1996. № 2–3. P. 167–187.
12. *Cohen S.E.* Information transmission in sequential programs // Foundations of Secure Computation. 1978. P. 297–335.
13. *Denning E.D.* A lattice model of secure information flow // Communications of the ACM. 1976. № 5. P. 236–243.
14. *Broberg N., Sands D.* Flow locks: Towards a core calculus for dynamic flow policies // Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics). 2006. P. 180–196.
15. *Bart V.D., Broberg N., Sands D.* A Datalog semantics for Paralocks // Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics). 2013. P. 305–320.
16. *Hedin D., Sabelfeld A.* A Perspective on Information-Flow Control // Software Safety and Security. 2012. P. 319–347.
17. *Sabelfeld A., Sands D.* Probabilistic noninterference for multi-threaded programs // Proceedings 13th IEEE Computer Security Foundations Workshop. CSFW-13. IEEE, 2000. P. 200–214.
18. *Mantel H., Sabelfeld A.* A Generic Approach to the Security of Multi-Threaded Programs // CSFW. Citeseer, 2001. P. 126–142.
19. *Mantel H., Sabelfeld A.* A unifying approach to the security of distributed and multi-threaded programs // Journal of Computer Security. 2003. № 4. P. 615–676.
20. *Askarov A., Sabelfeld A.* Gradual release: Unifying declassification, encryption and key release policies // Proceedings – IEEE Symposium on Security and Privacy, 2007. P. 207–221.
21. *Broberg N.* Thesis for the Degree of Doctor of Engineering Practical, Flexible Programming with Information Flow Control. 2011.
22. *Sabelfeld A., Russo A.* From dynamic to static and back: Riding the roller coaster of information-flow control research // International Andrei Ershov Memorial Conference on Perspectives of System Informatics. Spinger, 2009. P. 352–365.
23. *Shroff P., Smith S., Thober M.* Dynamic dependency monitoring to secure information flow // 20th IEEE Computer Security Foundations Symposium (CSF'07). 2007. P. 203–217.
24. *Mantel H., Sudbrock H.* Types vs. pdgs in information flow analysis // International Symposium on Logic-Based Program Synthesis and Transformation. 2012. P. 106–121.
25. *Schoepe D., Murray T., Sabelfeld A.* VERONICA: expressive and precise concurrent information flow security (extended version with technical appendices) // arXiv preprint arXiv:2001.11142. 2020.
26. *Müller C., Seidl H., Z*ā linescu E. Inductive invariants for noninterference in multi-agent workflows // 2018 IEEE 31st Computer Security Foundations Symposium (CSF). 2018. P. 247–261.
27. *Finkbeiner B., Muller C., Seidl H., Z*ā linescu E. Verifying security policies in multi-agent workflows with loops // Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security. 2017. P. 633–645.
28. *Девянин П.Н., Леонова М.А.* Применение подтипов и тотальных функций формального метода Event-B для описания и верификации МРОСЛ ДП-модели // Программная инженерия. 2020. Т. 11. № 4. С. 230–241.
29. *Balliu M., Liebe B., Schoepe D., Sabelfeld A.* Jslinq: Building secure applications across tiers // Proceedings of the Sixth ACM Conference on Data and Application Security and Privacy. 2016. P. 307–318.
30. *Huang Y.-W., Yu F., Hang C., Tsai C.-H., Lee D.-T., Kuo S.-Y.* Securing web application code by static analysis and runtime protection // Proceedings of the 13th international conference on World Wide Web. 2004. P. 40–52.
31. *Chlipala A.* Ur/Web: A simple model for programming the web // Proceedings of the 42nd Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages. 2015. P. 153–165.
32. *Arden O., George M.D., Liu J., Vikram K., Askarov A., Myers A.C.* Sharing mobile code securely with information flow control // 2012 IEEE Symposium on Security and Privacy. 2012. P. 191–205.
33. *Schoepe D., Hedin D., Sabelfeld A.* SeLINQ: tracking information across application-database boundaries // Proceedings of the 19th ACM SIGPLAN international conference on Functional programming. 2014. P. 25–38.

34. *Schultz D., Liskov B.* IFDB: decentralized information flow control for databases // Proceedings of the 8th ACM European Conference on Computer Systems. 2013. P. 43–56.

35. *Guarnieri M., Balliu M., Schoepe D., Basin D., Sabelfeld A.* Information-Flow Control for Database-backed Applications // 2019 IEEE European Symposium on Security and Privacy (EuroS&P). 2019. P. 79–94.

36. *Spearritt J.* The Applicability of Information Flow to Secure Java Development. 2017.

37. *Methni A., Lemerre M., Hedia B.B., Barkaoui K., Haddad S.* An Approach for Verifying Concurrent C Programs // 8th Junior Researcher Workshop on Real-Time Computing. 2014. P. 33–36.

38. *Timakov A.* PLIF. 2021. GitHub. https://github.com/timimin/plif.

39. *Lamport L.* Specifying systems. B.: Addison-Wesley, 2002. 388 p.

40. *Konnov I., Kukovec J., Tran T.-H.* TLA+ model checking made symbolic // Proceedings of the ACM on Programming Languages. 2019. V. 3. OOPSLA. P. 1–30.

41. *Smith G., Volpano D.* Secure information flow in a multi-threaded imperative language // Conference Record of the Annual ACM Symposium on Principles of Programming Languages. 1998. P. 355–364.

42. *Methni A., Lemerre M., Hedia B.B., Haddad S., Barkaoui K.* Specifying and verifying concurrent C programs with TLA+ // Communications in Computer and Information Science. 2015. V. 476. P. 206–222.

43. *Harrison M.A ., Ruzzo W.L., Ullman J.D.* Protection in operating systems // Communications of the ACM. 1976. V. 19. № 8. P. 461–471.

44. *Volpano D., Smith G.* Probabilistic noninterference in a concurrent language // Journal of Computer Security. 1999. V. 7. № 2. P. 231–253.

45. *Sabelfeld A., Sands D.* Probabilistic noninterference for multi-threaded programs // Proceedings 13th IEEE Computer Security Foundations Workshop. CSFW-13. 2000. P. 200–214.