УНИВЕРСИТЕТ ИТМО
НАУЧНО-ТЕХНИЧЕСКИЙ ВЕСТНИК ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ, МЕХАНИКИ И ОПТИКИ
май–июнь 2022 Том 22 № 3 http://ntv.ifmo.ru/
SCIENTIFIC AND TECHNICAL JOURNAL OF INFORMATION TECHNOLOGIES, MECHANICS AND OPTICS
May–June 2022 Vol. 22 No 3 http://ntv.ifmo.ru/en/
ISSN 2226-1494 (print) ISSN 2500-0373 (online)
НАУЧНО-ТЕХНИЧЕСКИЙ ВЕСТНИК УНИВЕРСИТЕТ ИТМО
ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ, МЕХАНИКИ И ОПТИКИ

КОМПЬЮТЕРНЫЕ СИСТЕМЫ И ИНФОРМАЦИОННЫЕ ТЕХНОЛОГИИ

COMPUTER SCIENCE

doi: 10.17586/2226-1494-2022-22-3-517-527
УДК 004.421.2:519.17 004.657

Формализация языков частично упорядоченных мультимножеств в системе Coq для спецификации слабых моделей памяти

Евгений Александрович Моисеенко[1]✉, Владимир Петрович Гладштейн[2], Антон Викторович Подкопаев[3], Дмитрий Владимирович Кознов[4]

[1,2,4] Санкт-Петербургский государственный университет, Санкт-Петербург, 199034, Российская Федерация
[1,2,3] JetBrains Research, Санкт-Петербург, 194100, Российская Федерация
[3] Национальный исследовательский университет «Высшая школа экономики», Санкт-Петербург, 194100, Российская Федерация

[1] e.moiseenko@2012.spbu.ru✉, https://orcid.org/0000-0003-2715-1143
[2] vovaglad00@gmail.com, https://orcid.org/0000-0001-9233-3133
[3] apodkopaev@hse.ru, https://orcid.org/0000-0002-9448-6587
[4] d.koznov@spbu.ru, https://orcid.org/0000-0003-2632-3193

Аннотация

Предмет исследования. Модели памяти задают семантику многопоточных программ, работающих с разделяемой памятью. В настоящее время эта область активно развивается, создаются новые модели памяти, востребованы новые средства формализации таких моделей, а также способы и средства доказательства их свойств. В работе рассмотрена задача формальной спецификации моделей памяти в системах интерактивного доказательства теорем. **Метод.** Использован семантический домен языков частично упорядоченных мультимножеств или языков помсетов. Предложен метод кодировки семантического домена с помощью фактор-типов и обсуждены его достоинства и недостатки. **Основные результаты.** Представлена библиотека, разработанная в системе Coq, реализующая предложенный метод. В рамках библиотеки установлена взаимосвязь языков помсетов с традиционной операционной семантикой, заданной в терминах помеченных систем переходов. Это позволило специфицировать в Coq модели памяти, параметризованные операционной семантикой структуры данных, и, таким образом, разделить определения модели памяти и структуры данных. **Практическая значимость.** Предложенная библиотека может быть использована для формальной спецификации и доказательства свойств широкого класса моделей памяти. Чтобы продемонстрировать это, в работе приведена формализация нескольких базовых моделей, а именно, моделей последовательной, причинной и конвейерной согласованности.

Ключевые слова

модели памяти, языки помсетов, формальная семантика, системы интерактивного доказательства теорем

Благодарности

Исследование выполнено при финансовой поддержке РФФИ в рамках научного проекта № 20-31-90088.

Ссылка для цитирования: Моисеенко Е.А., Гладштейн В.П., Подкопаев А.В., Кознов Д.В. Формализация языков частично упорядоченных мультимножеств в системе Coq для спецификации слабых моделей памяти // Научно-технический вестник информационных технологий, механики и оптики. 2022. Т. 22, № 3. С. 517–527. doi: 10.17586/2226-1494-2022-22-3-517-527

Mechanization of pomset languages in the Coq proof assistant for the specification of weak memory models

Evgenii A. Moiseenko[1]✉, Vladimir P. Gladstein[2], Anton V. Podkopaev[3], Dmitry V. Koznov[4]

[1,2,4] Saint Petersburg State University, Saint Petersburg, 199034, Russian Federation
[1,2,3] JetBrains Research, Saint Petersburg, 194100, Russian Federation
[3] HSE University, Saint Petersburg, 194100, Russian Federation

[1] e.moiseenko@2012.spbu.ru✉, https://orcid.org/0000-0003-2715-1143
[2] vovaglad00@gmail.com, https://orcid.org/0000-0001-9233-3133
[3] apodkopaev@hse.ru, https://orcid.org/0000-0002-9448-6587
[4] d.koznov@spbu.ru, https://orcid.org/0000-0003-2632-3193

Abstract
Memory models define semantics of concurrent programs operating on shared memory. Theory of these models is an active research topic. As new models emerge, the problem of providing a rigorous formal specification of these models becomes relevant. In this paper we consider a problem of formalizing memory models in the interactive theorem proving systems. We use semantic domain of pomset languages to formalize memory models. We propose an encoding of pomset languages using quotient types and discuss advantages and shortcomings of this approach. We present a library developed in the Coq proof assistant that implements the proposed approach. As a part of this library, we establish a connection between pomset languages and operational semantics defined by labeled transition systems. With the help of this theory, it becomes possible to define in Coq memory models parameterized by the operational semantics of an abstract datatype, and thus to decouple the definition of a memory model from the definition of the datatype. The proposed library can be used to develop formal machine-checked specifications of a wide class of memory models. In order to demonstrate its applicability, we present specifications of several basic memory models: sequential, causal, and pipelined consistency.

Keywords
memory models, pomset languages, formal semantics, interactive theorem proving

Aknowledgements
The reported study was funded by RFBR, project number № 20-31-90088.

For citation: Moiseenko E.V., Gladstein V.P., Podkopaev A.V, Koznov D.V. Mechanization of pomset languages in the Coq proof assistant for the specification of weak memory models. *Scientific and Technical Journal of Information Technologies, Mechanics and Optics*, 2022, vol. 22, no. 3, pp. 517–527 (in Russian). doi: 10.17586/2226-1494-2022-22-3-517-527

Введение

В последнее время получили широкое распространение многопоточные и распределенные программные системы. Соответственно, актуальной задачей является повышение надежности таких систем. В связи с этим активно развивается теория моделей памяти [1]: появляются спецификации моделей памяти процессоров, например, Intel [2], POWER [3], ARM [4], языков программирования C/C++ [5], Java [6, 7], LLVM [8], JavaScript [9], а также моделей памяти распределенных систем [10–12]. Подобные спецификации известны своей сложностью, а в доказательствах их ключевых свойств «на бумаге» нередко находятся существенные ошибки [13, 14]. По этой причине стандартной практикой становится формализация данных спецификации в интерактивных системах доказательства теорем, таких как Coq[1], Isabelle[2], Agda[3]. Эти системы позволяют сформулировать математические утверждения и их доказательства на специальном языке, а затем автоматически проверить корректность этих доказательств. Данные системы используются как для формализации математических теорий [15], так и для разработки корректного программного обеспечения [16].

При формализации семантики многопоточных программ используют либо подходы, основанные на операционной семантике с чередованием (interleaving semantics), либо подходы на основе так называемых *моделей истинной конкурентности* (*true concurrency*), куда обычно относят такие формализмы, как языки частично упорядоченных мультимножеств (или языки помсетов) [17, 18], трассы Мацуркиевича [19], структуры событий [20], сети Петри [21] и другие. Особенность моделей истинной конкурентности в том, что они позволяют явно выразить причинно-следственные связи между атомарными событиями системы и не рассматривать все возможные чередования параллельных процессов. Благодаря данной особенности в теории моделей памяти в последние годы возобновился интерес к этим формализмам [11, 22, 23]. Отметим, что формализмы активно используются при разработке инструментов верификации многопоточных программ и лежат в основе известной техники оптимизации под названием *редукция частичных порядков* (*partial order reduction*) [24, 25]. Тем не менее существующие работы либо не формализуют спецификации моделей в системах доказательства теорем [10, 11], либо используют для формализации неклассические варианты моделей истинной конкурентности [22], которые не всегда сохраняют их свойства, и не позволяют использовать известные теоретические результаты о моделях и их взаимосвязи [26–28].

Цель работы — формализация теории моделей истинной конкурентности [26] в системах интерактивного доказательства теорем. Рассмотрена проблема формализации одной из моделей истинной конкурентности — языков помсетов — в системе Coq. Предложен метод кодировки семантического домена с помощью *фактор-типов* (*quotient types*) [29]. Представлена библиотека[4], реализующая данный метод и разработанная с использованием расширения SSReflect [30] и библиотеки формализованных математических тео-

[1] The Coq Proof Assistant (2021) [Электронный ресурс]. Режим доступа: https://doi.org/10.5281/zenodo.4501022 (дата обращения: 21.03.2022).

[2] Isabelle Proof Assistant [Электронный ресурс]. Режим доступа: https://isabelle.in.tum.de/ (дата обращения: 21.03.2022).

[3] Agda language reference [Электронный ресурс]. Режим доступа: https://agda.readthedocs.io/ (дата обращения: 21.03.2022).

[4] Библиотека доступна по адресу [Электронный ресурс]. Режим доступа: https://github.com/event-structures/event-struct (дата обращения: 21.03.2022).

рий Mathcomp[1]. Выполнено обсуждение ограничений предложенного метода, предложены возможные пути их преодоления.

В рамках разработанной библиотеки раскрыта взаимосвязь домена языков помсетов и традиционных операционных семантик, заданных в терминах помеченных систем переходов. Это позволяет задавать модели памяти, параметризованные операционной семантикой абстрактной структуры данных [10], и отделить определения модели памяти и структуры данных. В качестве примера приведены спецификации трех базовых моделей памяти, имеющих важное практическое значение [10]: последовательной согласованности (sequential consistency), причинной согласованности (causal consistency), и конвейерной согласованности (pipelined consistency). На основе примеров сделан вывод, что широкий класс моделей, которые сохраняют программный порядок [1] и задаются относительно последовательной спецификации структуры данных, может быть выражен в терминах языков помсетов, и что кодирование помсетов с помощью фактор-типов подходит для представления этих моделей в системах доказательства теорем.

Модели памяти и языки помсетов

Формальная семантика многопоточной или распределенной программы, работающей с некоторыми разделяемыми ресурсами, определяется моделью согласованности. Традиционно модели согласованности задаются для абстракции разделяемой памяти [1], т. е. индексированного множества ячеек памяти. В этом контексте модели согласованности обычно называются *моделями памяти* (*memory model*). Рассмотрим подходы к формализации моделей памяти с помощью операционных семантик и языков помсетов.

Операционная семантика. Традиционным способом формального описания поведения программы (в том числе и модели памяти) является использование операционной семантики, заданной в терминах помеченных систем переходов [31]. Система помеченных переходов является тройкой $\Sigma = (S, L, \rightarrow)$, где S — множество состояний; L — множество меток; символ «$\rightarrow$» — отношение перехода или множество троек $(l, s, s') \in L \times S \times S$, обозначаемых как $s \xrightarrow{l} s'$. Для фиксированного начального состояния s_0 система переходов порождает язык, принимаемый в данном состоянии $Lang(\Sigma, s_0)$, т. е. последовательность меток $l_1, \ldots, l_n \in L$, таких, что существует трасса $s_0 \xrightarrow{l_1} s_1 \xrightarrow{l_2} s_2 \xrightarrow{l_3} l_3 \ldots \xrightarrow{l_n} s_n$.

В рамках операционной семантики поведение многопоточной программы, состоящей из нескольких параллельных процессов, может быть определено как поочередное исполнение атомарных действий процессов. Например, на рис. 1 показан пример многопоточной программы, состоящей из четырех потоков, выполняющих операции чтения и записи в разделяемые переменные x и y, а также один из ее возможных сценариев

$$x := 1 \;\Big\|\; y := 1 \;\Big\|\; \begin{matrix} r_1 := x \\ r_2 := y \end{matrix} \;\Big\|\; \begin{matrix} r_3 := y \\ r_4 := x \end{matrix}$$

$$t_1 : W_x1 \rightarrow t_2 : W_y1 \rightarrow t_3 : R_x1 \rightarrow t_3 : R_y1 \rightarrow t_4 : R_y1 \rightarrow t_4 : R_x1$$

Рис. 1. Пример многопоточной программы и ее последовательно согласованного сценария исполнения

Fig. 1. Example of the multithreaded program and its sequentially consistent execution

исполнения. Этот сценарий задается последовательностью меток вида $t{:}l$, где $t \in T$ — идентификатор потока; $l := W_x(v)|R_x(v)$ — метка операции: или операции чтения, или операции записи значения v в разделяемую переменную x.

Модель памяти, которая допускает только сценарии поведения, полученные как результат последовательного поочередного исполнения инструкций процессоров, называется *моделью последовательной согласованности* [32]. Реализация данной модели в многопоточных и распределенных системах приводит к значительным накладным расходам [33]. Потому на практике часто используются более слабые модели, допускающие больше возможных сценариев исполнения программы [34].

Например, в рамках *модели причинной согласованности* [10], которая описывает поведение распределенной системы с обменом сообщениями, для программы на рис. 1 допускается также сценарий поведения, в котором третий и четвертый потоки наблюдают обновление переменных x и y в разном порядке (рис. 2).

Видно, что поток t_3 сначала выполняет операцию чтения R_x1, а затем — R_y0, в то время как поток t_4 сначала выполняет R_y1, а затем R_x0. Дело в том, что модель причинной согласованности определяет порядок «происходит-до» (happens-before) как объединение программного порядка внутри потоков и порядка обмена сообщениями между потоками, а затем предписывает, что только операции записи, связанные отношением «происходит-до», должны наблюдаться всеми потоками в одинаковом порядке. Поскольку потоки t_1 и t_2 не обмениваются сообщениями, то операции записи W_x1 и W_y1 не упорядочены отношением «происходит-до» и могут наблюдаться другими потоками в произвольном порядке.

Поведение программы в рамках модели причинной согласованности также можно определить в терминах системы помеченных переходов, например, с помощью так называемых *векторных часов* (*vector clock*) [34]. Проблема заключается в том, что для кодирования каждой модели согласованности необходимо использовать различные техники и структуры данных, например, буферы операций записи, очереди сообщений и т. п.

Языки помсетов. Чтобы задавать различные модели памяти декларативно и единообразно, используют-

$$t_1 : W_x1 \rightarrow t_2 : W_y1 \rightarrow t_3 : R_x1 \rightarrow t_3 : R_y0 \rightarrow t_4 : R_y1 \rightarrow t_4 : R_x0$$

Рис. 2. Пример причинно согласованного сценария исполнения программы

Fig. 2. Example of the causally consistent execution

[1] Mathematical components [Электронный ресурс]. Режим доступа: https://github.com/math-comp/math-comp/ (дата обращения: 21.03.2022).

ся модели на основе отношения частичного порядка, заданного на атомарных действиях системы [10, 35], например, языки помсетов [17].

Языки помсетов — обобщение понятия обычного «последовательного» языка, т. е. множества слов некоторого алфавита. Обобщение заключается в переходе от линейного порядка символов в рамках слова к частичному порядку. В контексте параллельных вычислений частичный порядок представляет один сценарий исполнения параллельной программы. Носитель порядка состоит из событий — атомарных шагов вычисления. Каждому событию присваивается семантическая метка. Если событие e_1 упорядочено с e_2, т. е. $e_1 \le e_2$, то полагается, что появление события e_2 зависит от появления e_1. Если не верно ни $e_1 \le e_2$, ни $e_2 \le e_1$, тогда события e_1 и e_2 полагаются параллельными, что обозначается как $e_1 \asymp e_2$. Это означает, что события могут быть исполнены последовательным вычислителем в любом порядке.

На рис. 3 показан пример помсета. В данном помсете каждому событию соответствует уникальная метка. События с метками $t_1 : W_x1$ и $t_2 : W_y1$ являются параллельными и оба предшествуют событию с меткой $t_3 : R_x1$, которое, в свою очередь, предшествует $t_3 : R_y1$.

Формально, помеченное частично упорядоченное множество — тройка $(E, \lambda, \le)$ следующего вида:

— E — носитель, и элементы множества в контексте данной работы называются *событиями*;
— λ — функция с сигнатурой $E \to L$, отображающая события в *метки*;
— $\le$ — частичный порядок *причинно-следственной связи* между событиями.

Запись $Poset_L$ обозначает множество всех частично упорядоченных множеств, помеченных метками типа L.

Заметим, что идентификаторы событий сами по себе неважны, а важны лишь их метки и порядок между ними. Таким образом, с помеченными частично упорядоченными множествами работают по модулю переименования событий или, другими словами, с точностью до изоморфизма. Следующие определения формализуют эти идеи.

Пусть $p, q \in Poset_L$, и дана функция $f : E_p \to E_q$.

Введем следующие свойства:

— если $\lambda_q(f(e)) = \lambda_p(e)$, то будем называть такую функцию *сохраняющей метки* (*label preserving*);
— если $e_1 \le_p e_2$ влечет $f(e_1) \le_q f(e_2)$, то будем называть такую функцию *сохраняющей порядок* (*order preserving*);
— если $e_1 \le_p e_2$ тогда и только тогда, когда $f(e_1) \le_q f(e_2)$, то будем называть такую функцию *вкладывающей порядок* (*order embedding*).

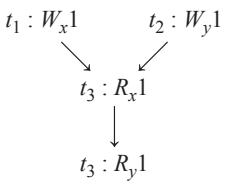

Рис. 3. Пример помсета
Fig. 3. Example of the pomset

Функция, сохраняющая и метки, и порядок, называется *гомоморфизмом*. Биективная функция, сохраняющая метки и вкладывающая порядок называется *изоморфизмом*. Будем писать $p \cong q$, если существует изоморфизм между p и q. Классы эквивалентности помеченных частичных порядков по модулю изоморфизма называются *помсетами*: $Pomset_L \triangleq Poset_L/\cong$. Таким образом, *язык помсетов* — множество помсетов: $Pomlang_L \triangleq \mathcal{P}(Pomset_L)$.

Линеаризация помсетов. В простейшем случае язык помсетов P можно связать с последовательным языком, рассмотрев множество линеаризаций частичных порядков $Lin(P)$, т. е. их дополнений до полных порядков. Будем говорить, что помсет p *поглощается* (*subsumed by*) q, что обозначается как $p \sqsubseteq q$, если существует биективный гомоморфизм из q в p. Интуитивно это означает, что множество p более упорядоченно по сравнению с q. Язык P *поглощается* Q, т. е. $P \sqsubseteq Q$, если для каждого $p \in P$ существует $q \in Q$, который его поглощает ($p \sqsubseteq q$). Тогда помсет q является линеаризацией p, т. е. $q \in Lin(p)$, если он является полностью упорядоченным и поглощается p. Множество линеаризаций языка помсетов определяется как объединение линеаризаций всех помсетов, принадлежащих данному языку: $Lin(P) \triangleq \bigcup_{p \in P} Lin(p)$. Таким образом, имея последовательную спецификацию структуры данных $\Sigma = (S, L, \to)$ с начальным состоянием $s_0 \in S$, можно сказать, что язык помсетов P соответствует Σ, если $Lin(P) = Lang(\Sigma, s_0)$.

На рис. 4 показан пример линеаризации помсета. Пунктирными стрелками продемонстрирован гомоморфизм – функция, отображающая события правого помсета в события левого помсета. Можно видеть, что помсет слева полностью упорядочен, а отображение событий биективно, сохраняет метки и порядок. Следовательно, помсет слева действительно является линеаризацией помсета справа.

Модели памяти. Модель памяти задается относительно последовательной структуры данных путем ослабления требования на множество линеаризаций языка помсетов. Обычно требуется только существование некоторой линеаризации, принадлежащей языку структуры данных. Будем говорить, что язык P *сводится* к языку Q, и обозначается это как $P \hookrightarrow Q$, если для каждого $p \in P$ существует $q \in Q$, такой что $q \sqsubseteq p$. Тогда требование о том, чтобы для каждого $p \in P$ существова-

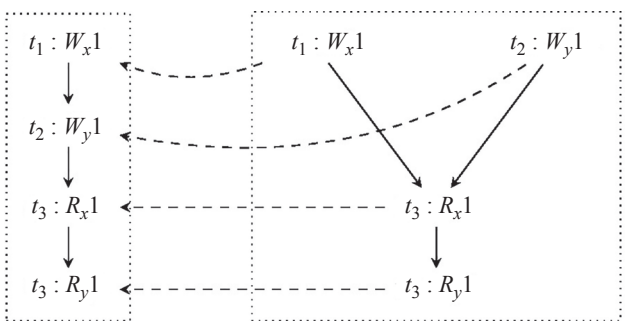

Рис. 4. Пример линеаризации помсета
Fig. 4. Example of the pomset linearization

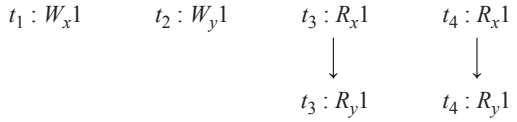

Рис. 5. Пример отношения программного порядка
Fig. 5. Example of the program order

ла линеаризация, принадлежащая языку системы переходов, выглядит следующим образом: $P \hookrightarrow Lang(\Sigma, s_0)$.

Например, в определении уже упомянутой модели последовательной согласованности рассматриваются линеаризации только *программного порядка* — порядка выполнения операций внутри потока. Помсет считается последовательно-согласованным, если существует хотя бы одна такая линеаризация, принадлежащая языку структуры данных.

Рассмотрим помсет (рис. 5), который задает программный порядок, соответствующий программе, приведенной на рис. 1. Видно, что последовательность меток, показанная на рис. 1, является линеаризацией данного помсета — последовательность упорядочена согласно отношению программного порядка. Данная последовательность меток является корректной с точки зрения абстракции разделяемой памяти. Следовательно, приведенный помсет — последовательно согласован.

Выполним формализацию в системе Coq понятия системы помеченных переходов, языков помсетов и их взаимосвязь. Полученная в результате библиотека позволяет специфицировать модели памяти, которые выражаются с помощью линеаризации языков помсетов. Приведем технические детали реализации библиотеки и примеры спецификаций моделей памяти, формализованных с ее помощью.

Формализация языков помсетов в Coq

Несмотря на относительно простое определение языков помсетов, их формализация в системе Coq не так тривиальна. Проблема заключается в особенностях представления фактор-множеств в интенсиональных теориях типов [36], к которым относится и теория типов, лежащая в основе Coq. В системе Coq существует два подхода к формализации фактор-множеств: с применением *сетоидов* [37] и с помощью *фактор-типов* [38].

Сетоид — тип, оснащенный отношением эквивалентности. В системе Coq реализована поддержка переписывания термов сетоид-типа по отношению эквивалентности. Недостаток подхода — переписывание по отношению эквивалентности работает существенно медленнее, чем по встроенному пропозициональному равенству [39], что приводит к замедлению времени проверки доказательств.

Фактор-тип — такой тип, в котором каждый населяющий его терм соответствует единственному классу эквивалентности, и, наоборот, каждому классу эквивалентности соответствует некоторый терм фактор-типа. Преимущество использования фактор-типов — задача переписывания по отношению эквивалентности может быть сведена к задаче переписывания по пропозициональному равенству на соответствующем фактор-типе. Таким образом, можно обойти проблему производительности переписывания по отношению эквивалентности.

К сожалению, в интенсиональных теориях типов невозможно в общем случае построить фактор-тип для заданного отношения эквивалентности. Обойти это ограничение можно, добавив в систему аксиомы, необходимые для построения фактор-типа, но это приведет к потери свойства *каноничности*. Последнее означает, что все замкнутые термы могут быть редуцированы к нормальной форме.

В работе [29] представлена библиотека (часть библиотеки MathComp), позволяющая в некоторых частных случаях построить фактор-тип без добавления в систему новых аксиом. Опишем ограничения, необходимые для конструктивного построения фактор-типов, и способ авторов настоящей работы кодировки помеченных отношений частичного порядка, который позволяет удовлетворить этим ограничениям и таким образом построить фактор-тип, соответствующий помсетам.

Конструктивные фактор типы. Рассмотрим тип `T` и отношение эквивалентности на этом типе `equiv`, по которому требуется построить фактор-тип. Для конструктивного построения фактор-типа необходимо выполнение следующих условий.

— Тип `T` должен быть оснащен *оператором выбора* `xchoose`. Наличие данного оператора позволяет построить терм, обладающий заданным свойством, имея доказательство существования такого терма. Другими словами, данный оператор имеет следующий тип `exists x : T, P x -> T`. Заметим, что Coq основан на интуиционистской [36], а не классической логике, поэтому оператор `xchoose` не может быть определен для произвольного типа без использования дополнительных аксиом.
— Отношение эквивалентности `equiv` должно быть разрешимо, т. е. оно должно иметь тип `T -> T -> bool`.

Таким образом, чтобы закодировать помсеты с помощью фактор-типов, необходимо сначала закодировать все возможные помеченные частично упорядоченные множества типом с оператором выбора, а отношение изоморфизма — разрешимым отношением. Для выполнения условия необходимо наложить существенное ограничение и рассмотреть только конечные отношения частичного порядка.

Кодирование помсетов. Для кодирования помеченных отношений частичного порядка использован тип функций с конечным носителем $A \underset{fin}{\rightarrow} B$ из библиотеки MathComp. Функция $f\colon A \underset{fin}{\rightarrow} B$ отображает все элементы вне конечного носителя $\mathcal{A} \subseteq A$ на некоторое значение по умолчанию $b \in B$: $\notin \mathcal{A} \Rightarrow f(a) = b$. Тип функций $A \underset{fin}{\rightarrow} B$ уже оснащен оператором выбора (при условии, что типы A и B также оснащены оператором выбора). Таким образом, использование этих функций для представления помеченных частично упорядоченных множеств позволяет удовлетворить первое ограничение на построение конструктивных фактор-типов.

Конечное отношение частичного порядка кодируется с помощью функции с конечным носителем следую-

щим образом. Для отношения частичного порядка $\leq$ его покрытие $\lessdot$ определяется как: $x \lessdot y \triangleq x < y \wedge \nexists z.\, x < z < y$. Отношение R называется *префикс-конечным*, если для любого x множество $\{y|(y, x) \in R\}$ конечно. Префикс-конечное отношение $R \subseteq X \times X$ можно представить с помощью функции $f : X \underset{fin}{\rightarrow} \mathcal{P}(X)$, такой, что $y \in f(x) \Leftrightarrow$ $\Leftrightarrow (y, x) \in R$. Для конечного отношения частичного порядка его покрытие является префикс-конечным, и, следовательно, может быть представлено с помощью подобной функции. Такое представление также оказывается удобно при кодировании операционной семантики для построения частично упорядоченного множества путем добавления нового события на каждом шаге.

Более детально, для представления частично упорядоченного множества используется функция $f : E \underset{fin}{\rightarrow} L \times \mathcal{P}(E)$. Данная функция ставит в соответствие событию e пару $(l, \mathfrak{E})$, где l — метка, а $\mathfrak{E} \subseteq E$ — конечное множество непосредственных предков события e. Все элементы вне носителя функции отображаются в значение по умолчанию, которое является парой $(\bot, \emptyset)$, где $\bot$ является выделенной меткой, обозначающей неопределенный элемент. На функцию f накладывается дополнительное ограничение: отношение, индуцируемое данной функцией, должно быть ацикличным. Таким образом, гарантируется, что рефлексивно-транзитивное замыкание этого отношения действительно является отношением частичного порядка.

Обозначим как *lposet E L* тип, кодирующий помеченные частично упорядоченные множества с типом E и типом меток L описанным выше способом. Рассмотрим два отношения частичного порядка — p, q : *lposet E L* и функцию h: $E \rightarrow E$. Так как p и q конечны, то можно определить разрешимый предикат, который проверяет, что функция h удовлетворяет свойствам изоморфизма. Более того, доказывается, что существует только конечное число изоморфизмов между парой p и q. Таким образом, с помощью квантора существования по конечному множеству можно проверить, существует ли изоморфизм между p и q — построить разрешимое отношение изоморфизма $\cong$. Это позволяет удовлетворить второе ограничение на построение конструктивных фактор-типов. Наконец, тип помсетов *pomset E L* определяется взятием фактор-типа по отношению изоморфизма $\cong$.

Чтобы формализовать взаимосвязь языков помсетов с системами помеченных переходов, также определяется множество линеаризаций языка помсетов. Для этого вводится отношение поглощения $p \sqsubseteq q$ на помсетах. Доказывается, что данное отношение образует частичный порядок. Заметим, что это доказательство существенно опирается на два следующих факта. Во-первых, помсеты конечны (в противном случае, $p \sqsubseteq q$ и $q \sqsubseteq p$ не влечет $p \cong q$. Во-вторых, отношение поглощения задано на типе помсетов *pomset E L*, а не типе отношений частичного порядка *lposet E L* (иначе $p \cong q$ не влечет $p = q$). При кодировании в Coq линеаризации представляются как списки меток, что позволяет сразу использовать их в контексте систем переходов и последовательных языков. Затем множества линеаризаций помсета и языка помсетов задаются как предикаты на списках меток типа L при помощи отношения поглощения.

Таким образом, с помощью формализации взаимосвязи языков помсетов и операционных семантик в дальнейшем становится возможным задать спецификацию модели памяти в терминах линеаризаций помсетов.

Ограничения. Как уже упоминалось, на данный момент предложенная библиотека поддерживает только конечные помсеты. Тем не менее заметим, что параллельная семантика программы все еще может быть выражена как бесконечное множество (язык) конечных помсетов, аналогично тому, как в последовательном случае семантика может быть выражена бесконечным языком конечных слов. Кроме того, использование конечных помсетов достаточно для доказательства *свойств безопасности* программы (*safety properties*) [40]. Тем не менее, иногда бывает нужно также рассматривать бесконечные помсеты, например, для доказательства *свойств живучести* программы (*liveness properties*) [41].

Обойти ограничение на конечность помсетов можно ценой потери каноничности, с помощью добавления в систему новых аксиом. В частности, можно воспользоваться библиотекой MathComp-Analysis [42], предоставляющей некоторые леммы для работы с классической логикой в Coq. Данная библиотека добавляет в систему несколько аксиом классической логики, в том числе:

— *аксиому конструктивного неопределенного описания* (*constructive indefinite description*), которая позволяет построить терм, обладающий заданным свойством, имея доказательство существования такого терма; другими словами, данная аксиома позволяет оснастить произвольный тип оператором выбора `exists x : T, P x -> T`;
— *аксиому пропозициональной экстенсиональности*, которая позволяет вывести равенство двух утверждений из доказательства их эквивалентности. Более формально данная аксиома имеет тип `forall (P Q : Prop), (P <-> Q)-> P = Q`.

По теореме Диаконеску [43] эти две аксиомы влекут *закон исключенного третьего в информативной форме* (*excluded middle informative*), который имеет тип `forall (P : Prop), {P} + ~{P}`. С помощью этого закона по любому отношению можно построить его разрешимый аналог.

Таким образом, используя данные законы классической логики, тип бесконечных помеченных отношений частичного порядка можно оснастить оператором выбора и задать на нем разрешимое отношение изоморфизма, а затем построить фактор-тип для представления бесконечных помсетов.

Спецификация моделей памяти

Модель памяти задает семантику многопоточных программ, оперирующих с разделяемой структурой данных. Поведение разделяемой структуры данных и отдельных потоков часто задается «последовательной» семантикой [10, 11], в терминах помеченных систем переходов. Определение модели памяти в терминах языков помсетов, таким образом, связывает язык помсетов с «последовательными» системами переходов

структуры данных и отдельных потоков при помощи понятия линеаризации помсетов.

Формализация языков помсетов и их взаимосвязи с системами переходов в рамках предложенной библиотеки позволяет специфицировать в Coq модели памяти в таком стиле. Чтобы продемонстрировать применимость библиотеки для решения данной задачи, приведем примеры спецификаций нескольких моделей памяти, а именно модели *последовательной согласованности*, *причинной согласованности* и *конвейерной согласованности*. Данные модели определены согласно работе [10].

Спецификация структуры данных. Последовательная спецификации типа данных задается как помеченная система переходов $\Sigma_D = (S_D, L, \underset{D}{\rightarrow})$ с выделенным начальным состоянием $d_0 \in S_D$. Для спецификации «строгих» моделей, например модели последовательной согласованности, достаточно задать только систему переходов и начальное состояние. Более слабые модели (модель причинной согласованности), по-разному трактуют операции записи и чтения в структуре данных, поэтому для спецификации этих моделей необходимо задать дополнительные ограничения на структуру данных.

Существуют различные способы формально определить операции записи и чтения в структуре данных [10, 11, 35]. Допустим, что на множестве меток задана *структура коммуникации*, которая моделирует передачу информации в структуре данных и позволяет вывести определение подмножества операций записи и операций чтения. Данная структура задается отношением $\Lambda \vdash l$, где $\Lambda \subseteq L$ и $l \in L$. Будем говорить, что помсет p *уважает отношение коммуникации*, $p \in \mathrm{COM}(L)$, если выполняется следующее условие:

$$\forall e.\, \exists e_1 \ldots e_n.\, \{\lambda(e_1), \ldots, \lambda(e_n)\} \vdash \lambda(e) \wedge \forall i.\, e_i \leq e.$$

Данное условие гарантирует, что появление в помсете p события e с меткой $\lambda(e)$ должно быть обосновано наличием предшествующих e событий $e_1 \ldots e_n$ с метками $\lambda(e_1), \ldots, \lambda(e_n)$ находящимися в отношении коммуникации с $\lambda(e)$.

Предположим, что язык системы переходов $Lang(\Sigma_D, s_0)$ уважает отношение коммуникации.

Метка $w \in L$ считается операцией записи $w \in W$, если существуют метки $\Lambda \subseteq L$ и метка $r \in L$, такие что $w \in \Lambda$ и $\Lambda \vdash r$. Метка $r \in L$ считается операцией чтения $r \in R$, если существуют метки $\Lambda \subseteq L$, такие что $\Lambda \vdash r$. Частично определенная функция к возвращает тип метки:

$$\kappa(l) = \begin{cases} R, l \in R \backslash W \\ W, l \in W \backslash R \\ RW, l \in W \cap R \\ \bot, \text{иначе.} \end{cases}$$

Примем, что на метках задано отношение конфликта #. Положим, что неконфликтующие метки коммутируют, т. е. для любой пары $a, b \in L$, такой что $\neg(a \# b)$, и для любых строк $u, v \in L^*$ справедливо следующее:

$$uabv \in Lang(\Sigma_D, d_0) \Leftrightarrow ubav \in Lang\,(\Sigma_D, d_0).$$

Например, рассмотрим спецификацию разделямой памяти — базового типа данных, для которого традиционно задаются модели памяти. Система переходов выглядит следующим образом:

$$\frac{m' = m[x \mapsto v]}{m \xrightarrow{W_x v} m'}\,(Write)\;\frac{m(x) = v}{m \xrightarrow{R_x v} m}\,(Read).$$

Состояние системы переходов — отображение из адреса переменной в ее значение, которое моделируется функцией с конечным носителем m: $\mathcal{A} \underset{fin}{\rightarrow} \mathcal{V}$. Отношение коммуникации (часто называемое «читает-из») и отношение конфликта заданы по следующим правилам:

$$\overline{\{W_x v\} \vdash R_x v} \quad \overline{W_x v_1 \,\#\, W_x v_2} \quad \overline{W_x v_1 \,\#\, R_x v_2}.$$

Программный порядок. Для того чтобы моделировать потоки параллельной программы, во-первых, полагается, что спецификация потоков также задана как система помеченных переходов над тем же множеством меток, что и у спецификации структуры данных $\Sigma_T = (S_T, L, \underset{T}{\rightarrow})$. Во-вторых, множество меток дополнительно индексируется множеством идентификаторов потоков T, т. е. пара (t, l) обозначает, что поток $t \in T$ выполняет операцию с меткой l. Рассмотрим помсет p над множеством меток $T \times L$: $p \in Pomset_L$. Обозначим как τ и λ функции, возвращающие идентификатор потока и метку операции соответственно.

На помсете p можно задать отношение эквивалентности, связывающее события из одного потока: $e_1 =_\tau e_2 \triangleq \tau(e_1) = \tau(e_2)$. Пересечение отношений частичного порядка p и $=_\tau$ образует частичный порядок, и, следовательно, помсет. Этот порядок называется *программным порядком*, соответствующий помсет обозначим как $po(p)$. Пусть $t \in T$ — идентификатор потока. Тогда можно рассмотреть сужение помсета p на класс эквивалентности t по отношению $=_\tau$, обозначим его как p_t. Этот помсет соответствует потоку с идентификатором t. Для всех рассматриваемых далее моделей потребуем, чтобы для каждого t помсет p_t принадлежал языку Σ_T для некоторого начального состояния $s_0 \in S_T$:

$$\forall t \in T.\, \exists s_0 \in \Sigma_T.\, p_t \in Lang(\Sigma_T, s_0).$$

Заметим, что это влечет к тому, что p_t должен быть полностью упорядочен. Это свойство позволяет связать помсет с последовательной семантикой, задающей поведение потоков.

Модель последовательной согласованности предписывает, что результат каждого сценария исполнения программы мог быть объяснен как результат поочередного выполнения инструкций потоков. Другими словами, для каждого сценария исполнения программы должен существовать способ полностью упорядочить события данного сценария, сохраняя программный порядок.

Обозначим как $SegCst(\Sigma_D, d_0)$ язык помсетов, которые являются последовательно согласованными относительно структуры данных Σ_D с начальным со-

стоянием d_0. Помсет p является *последовательно согласованным*, если выполняется следующее условие:

$$p \in SegCst(\Sigma_D, d_0) \triangleq \{po(p)\} \hookrightarrow Lang(\Sigma_D, d_0).$$

Модель причинной согласованности определяется более сложным образом с помощью порядка «происходит-до» (happens-before), который должен включать программный порядок. Требуется, чтобы все потоки наблюдали операции записи, упорядоченные отношением «происходит-до», согласно этому порядку. В рамках модели последовательной согласованности каждое событие обладает знанием только о предшествующих событиях из своего потока, а также о предшествующих операциях записи из других потоков.

Чтобы формализовать определение причинной согласованности, введем понятие префикса события и перемаркировки помсета. Ограничение помсета на префикс события позволяет рассматривать только те события, которые происходили до данного события. Перемаркировка помсета позволяет изменить метки операций чтения в других потоках, чтобы отразить тот факт, что поток обладает знанием только об операциях записи.

Префикс события e относительно помсета p это множество $\lceil e \rceil \triangleq \{e' | e' \leq e\}$. Запись $p_{\lceil e \rceil}$ обозначает сужение p на префикс события e. Пусть $f\colon L \to L$ это функция на метках. Тогда перемаркировка помсета p определяется как $f(p) \triangleq (E_p, \tau, f \circ \lambda, \leq_p)$. Будем говорить, что функция $f\colon L \to L$ обладает следующими свойствами:

— *сохраняет тип операций*, если $\kappa_{f(p)}(e) = \kappa_p(e)$;
— *сохраняет отношение конфликта*, если $e_1 \#_{f(p)} e_2 \Leftrightarrow$ $\Leftrightarrow e_1 \#_p e_2$;
— *сохраняет метки по модулю подмножества событий* $\mathcal{E}$, если $e \notin \mathcal{E}$ влечет $\lambda_{f(p)}(e) = \lambda_p(e)$.

Обозначим как $Relab(\mathcal{E})$ множество функций, которые удовлетворяют всем этим условиям относительно подмножества событий $\mathcal{E}$. Для потока $t \in T$ обозначим как $R_p^{\neq t}$ множество операций чтений, принадлежащих другим потокам:

$$R_p^{\neq t} \triangleq \{e \in E_p | \tau(e) \neq t \wedge \kappa(e) = R\}.$$

Определим посредством $HB(\Sigma_D, d_0)$ язык помсетов, который содержит все возможные варианты отношения «происходит-до» относительно структуры данных Σ_D и начального состояния d_0. Помсет q принадлежит этому языку, если для каждого события $e \in E_q$ сужение помсета q на префикс события e может быть линеаризировано к последовательности, принадлежащей языку структуры данных, по модулю перемаркировки некоторых операций чтения из других потоков:

$$q \in HB(\Sigma_D, d_0) \triangleq \forall e \in E_q.\ \exists f \in Relab(R_p^{\neq \tau(e)}).$$
$$\{f(q_{\lceil e \rceil})\} \hookrightarrow Lang(\Sigma_D, d_0).$$

Помсет p является *причинно-согласованным*, если существует $q \in HB(\Sigma_D, d_0)$, такой что $q \sqsubseteq po(p)$:

$$p \in CausalCst(\Sigma_D, d_0) \triangleq \{po(p)\} \hookrightarrow HB(\Sigma_D, d_0).$$

Модель конвейерной согласованности ослабляет модель последовательной согласованности. Требуется, чтобы все потоки наблюдали в одинаковом порядке только те операции записи, которые упорядочены отношением программного порядка (а не отношением «происходит-до»).

Помсет p является *конвейерно-согласованным*, если для каждого потока существует линеаризация программного порядка, принадлежащей языку структуры данных, по модулю перемаркировки некоторых операций чтения из других потоков:

$$p \in PipeCst(\Sigma_D, d_0) \triangleq \forall t \in T.\ \exists f \in Relab(R_p^{\neq t}).$$
$$\{f(po(p))\} \hookrightarrow Lang(\Sigma_D, d_0).$$

Выводы и ограничения. При помощи разработанной библиотеки были формализованы три базовые модели консистентности. Кроме перечисленных выше моделей памяти, ожидается, что другие модели, которые выражаются в терминах линеаризаций помсетов и сохраняют программный порядок [1], также могут быть формализованы с помощью разработанной библиотеки. Этот класс моделей включает в том числе различные варианты *модели последовательной согласованности* (*causal consistency*) [10] и *модели согласованности в конечном счете* (*eventual consistency*) [11].

Отметим, что не все структуры данных могут иметь последовательную спецификацию. В качестве примера в работе [35] приведена структура обменника (exchanger), которая позволяет двум потокам атомарно обменяться парой значений. На данный момент подобные структуры данных не могут быть специфицированы с помощью библиотеки.

Кроме того, не все модели сохраняют программный порядок [9]. К таким моделям относятся модели памяти некоторых процессоров, например ARMv8 [3], и некоторые продвинутые модели для языков программирования, например модель Weaksetmo [22] и модель PwP [23]. В будущем будет интересно расширить библиотеку, чтобы также покрыть более широкий класс моделей.

Связанные работы

Спецификации моделей памяти, параметризованные операционной семантикой структуры данных, были представлены в работах [10, 11, 44]. Ни одна из этих работ не рассматривает задачу формализации данных модели памяти в системах доказательства теорем.

В работе [35] авторы представили метод для формализации моделей памяти и параллельных структур данных, и верифицировали некоторые спецификации в системе Coq. В этой работе также, как и в разработанной в настоящей работе библиотеке, поддерживаются только модели, сохраняющие программный порядок. Представленный метод основан на использовании *множеств графов сценариев исполнения* как семантического домена. Семантика структуры данных задается как набор ограничений на множество графов. Достоинством данного метода является то, что он позволяет формализовать структуры данных, которые не имеют последовательной спецификации, например, структуры обменника (exchanger). Отличие предложенной в настоящей работе библиотеки — взаимосвязь

спецификаций в терминах графов и операционных семантик не формализуются.

В работе для спецификации моделей памяти используется домен языков помсетов, что позволяет применить известные теоретические результаты [17, 18, 28] к определениям моделей памяти. Также, по мнению авторов, в данной работе впервые реализован метод представления помсетов на основе использования конструктивных фактор-типов.

Заключение

В работе представлен метод кодирования семантического домена языков конечных помсетов с использованием конструктивных фактор-типов и библиотека для системы Coq, реализующая данный метод. Использование фактор-типов позволяет в доказательствах проводить рассуждения по модулю изоморфизма помсетов, не прибегая к использованию сетоидов. Ограничение на конечность помсетов позволяет оставаться в рамках конструктивной логики и не прибегать к использованию аксиом классической логики.

Библиотека может быть использована для формализации и доказательства свойств семантики параллельных и распределенных систем и языков программирования. В частности, в качестве примера были формализованы модели памяти последовательной, причинной и конвейерной согласованности.

В будущем планируется добавить поддержку бесконечных помсетов, а также расширить библиотеку, добавив формализацию других моделей «истинной конкурентности» и установив взаимосвязи между ними.

Литература

1. Moiseenko E., Podkopaev A., Koznov D. A survey of programming language memory models // Programming and Computer Software. 2021. V. 47. N 6. P. 439–456. https://doi.org/10.1134/S0361768821060050
2. Sewell P., Sarkar S., Owens S., Nardelli F.Z., Myreen M.O. x86-TSO: A rigorous and usable programmer's model for x86 multiprocessors // Communications of the ACM. 2010. V. 53. N 7. P. 89–97. https://doi.org/10.1145/1785414.1785443
3. Sarkar S., Sewell P., Alglave J., Maranget L., Williams D. Understanding POWER multiprocessors // Proc. of the 32nd ACM SIGPLAN Conference on Programming Language Design and Implementation. 2011. P. 175–186. https://doi.org/10.1145/1993316.1993520
4. Pulte C., Flur S., Deacon W., French J., Sarkar S., Sewell P. Simplifying ARM concurrency: multicopy-atomic axiomatic and operational models for ARMv8 // Proceedings of the ACM on Programming Languages. 2018. V. 2. P. 19. https://doi.org/10.1145/3158107
5. Batty M., Owens S., Sarkar S., Sewell P., Weber T. Mathematizing C++ concurrency // ACM SIGPLAN Notices. 2011. V. 46. N 1. P. 55–66. https://doi.org/10.1145/1925844.1926394
6. Manson J., Pugh W., Adve S.V. The Java memory model // ACM SIGPLAN Notices. 2005. V. 40. N 1. P. 378–391. https://doi.org/10.1145/1047659.1040336
7. Bender J., Palsberg J. A formalization of Java's concurrent access modes // Proceedings of the ACM on Programming Languages. 2019. V. 3. N OOPSLA. P. A142. https://doi.org/10.1145/3360568
8. Chakraborty S., Vafeiadis V. Formalizing the concurrency semantics of an LLVM fragment // Proc.of the IEEE/ACM International Symposium on Code Generation and Optimization (CGO). 2017. P. 100–110. https://doi.org/10.1109/CGO.2017.7863732
9. Watt C., Pulte C., Podkopaev A., Barbier G., Dolan S., Flur S., Pichon-Pharabod J., Guo S.-Y. Repairing and mechanising the JavaScript relaxed memory model // Proc. of the 41st ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI). 2020. P. 346–361. https://doi.org/10.1145/3385412.3385973
10. Perrin M., Mostéfaoui A., Jard C. Causal consistency: beyond memory // Proc. of the 21st ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming (PPoPP). 2016. P. 26. https://doi.org/10.1145/2851141.2851170
11. Jagadeesan R., Riely J. Eventual consistency for CRDTs // Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics). 2018. V. 10801. P. 968–995. https://doi.org/10.1007/978-3-319-89884-1_34
12. Anceaume E., Del Pozzo A., Ludinard R., Potop-Butucaru M., Tucci-Piergiovanni S. Blockchain abstract data type // Proc. of the 31st ACM Symposium on Parallelism in Algorithms and Architectures. 2019. P. 349–358. https://doi.org/10.1145/3323165.3323183
13. Batty M., Memarian K., Nienhuis K., Pichon-Pharabod J., Sewell P. The problem of programming language concurrency semantics // Lecture Notes in Computer Science (including subseries Lecture

References

1. Moiseenko E., Podkopaev A., Koznov D. A survey of programming language memory models. *Programming and Computer Software*, 2021, vol. 47, no. 6, pp. 439–456. https://doi.org/10.1134/S0361768821060050
2. Sewell P., Sarkar S., Owens S., Nardelli F.Z., Myreen M.O. x86-TSO: A rigorous and usable programmer's model for x86 multiprocessors. *Communications of the ACM*, 2010, vol. 53, no. 7, pp. 89–97. https://doi.org/10.1145/1785414.1785443
3. Sarkar S., Sewell P., Alglave J., Maranget L., Williams D. Understanding POWER multiprocessors. *Proc. of the 32nd ACM SIGPLAN Conference on Programming Language Design and Implementation*, 2011, pp. 175–186. https://doi.org/10.1145/1993316.1993520
4. Pulte C., Flur S., Deacon W., French J., Sarkar S., Sewell P. Simplifying ARM concurrency: multicopy-atomic axiomatic and operational models for ARMv8. *Proceedings of the ACM on Programming Languages*, 2018, vol. 2, pp. 19. https://doi.org/10.1145/3158107
5. Batty M., Owens S., Sarkar S., Sewell P., Weber T. Mathematizing C++ concurrency. *ACM SIGPLAN Notices*, 2011, vol. 46, no. 1, pp. 55–66. https://doi.org/10.1145/1925844.1926394
6. Manson J., Pugh W., Adve S.V. The Java memory model. *ACM SIGPLAN Notices*, 2005, vol. 40, no. 1, pp. 378–391. https://doi.org/10.1145/1047659.1040336
7. Bender J., Palsberg J. A formalization of Java's concurrent access modes. *Proceedings of the ACM on Programming Languages*, 2019, vol. 3, no. OOPSLA, pp. A142. https://doi.org/10.1145/3360568
8. Chakraborty S., Vafeiadis V. Formalizing the concurrency semantics of an LLVM fragment. *Proc.of the IEEE/ACM International Symposium on Code Generation and Optimization (CGO)*, 2017, pp. 100–110. https://doi.org/10.1109/CGO.2017.7863732
9. Watt C., Pulte C., Podkopaev A., Barbier G., Dolan S., Flur S., Pichon-Pharabod J., Guo S.-Y. Repairing and mechanising the JavaScript relaxed memory model. *Proc. of the 41st ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI)*, 2020, pp. 346–361. https://doi.org/10.1145/3385412.3385973
10. Perrin M., Mostéfaoui A., Jard C. Causal consistency: beyond memory. *Proc. of the 21st ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming (PPoPP)*, 2016, pp. 26. https://doi.org/10.1145/2851141.2851170
11. Jagadeesan R., Riely J. Eventual consistency for CRDTs. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 2018, vol. 10801, pp. 968–995. https://doi.org/10.1007/978-3-319-89884-1_34
12. Anceaume E., Del Pozzo A., Ludinard R., Potop-Butucaru M., Tucci-Piergiovanni S. Blockchain abstract data type. *Proc. of the 31st ACM Symposium on Parallelism in Algorithms and Architectures*, 2019, pp. 349–358. https://doi.org/10.1145/3323165.3323183
13. Batty M., Memarian K., Nienhuis K., Pichon-Pharabod J., Sewell P. The problem of programming language concurrency semantics.

Notes in Artificial Intelligence and Lecture Notes in Bioinformatics). 2015. V. 9032. P. 283–307. https://doi.org/10.1007/978-3-662-46669-8_12
14. Lahav O., Vafeiadis V., Kang J., Hur C.-K., Dreyer D. Repairing Sequential Consistency in C/C++11 // ACM SIGPLAN Notices. 2017. V. 52. N 6. P. 618–632. https://doi.org/10.1145/3062341.3062352
15. Gonthier G. Formal proof — the four-color theorem // Notices of the AMS. 2008. V. 55. N 11. P. 1382–1393.
16. Leroy X. A formally verified compiler back-end // Journal of Automated Reasoning. 2009. V. 43. N 4. P. 363–446. https://doi.org/10.1007/s10817-009-9155-4
17. Pratt V. The pomset model of parallel processes: Unifying the temporal and the spatial // Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics). 1985. V. 197. P. 180–196. https://doi.org/10.1007/3-540-15670-4_9
18. Gischer J.L. The equational theory of pomsets // Theoretical Computer Science. 1988. V. 61. N 2-3. P. 199–224. https://doi.org/10.1016/0304-3975(88)90124-7
19. Mazurkiewicz A. Trace theory // Advances in Petri nets 1986, part II on Petri nets: applications and relationships to other models of concurrency. Springer, 1986. P. 278–324.
20. Winskel G. Event structures // Advances in Petri nets 1986, part II on Petri nets: applications and relationships to other models of concurrency. Springer, 1986. P. 325–392.
21. Peterson J.L. Petri nets // ACM Computing Surveys. 1977. V. 9. N 3. P. 223–252. https://doi.org/10.1145/356698.356702
22. Chakraborty S., Vafeiadis V. Grounding thin-air reads with event structures // Proceedings of the ACM on Programming Languages. 2019. V. 3. N POPL. P. 70. https://doi.org/10.1145/3290383
23. Jagadeesan R., Jeffrey A., Riely J. Pomsets with preconditions: a simple model of relaxed memory // Proceedings of the ACM on Programming Languages. 2020. V. 4. N OOPSLA. P. 194. https://doi.org/10.1145/3428262
24. Abdulla P., Aronis S., Jonsson B., Sagonas K. Optimal dynamic partial order reduction // Proc. of the 41st Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages (POPL). 2014. P. 373–384. https://doi.org/10.1145/2535838.2535845
25. Kokologiannakis M., Lahav O., Sagonas K., Vafeiadis V. Effective stateless model checking for C/C++ concurrency // Proceedings of the ACM on Programming Languages. 2017. V. 2. N POPL. P. 17. https://doi.org/10.1145/3158105
26. Nielsen M., Sassone V., Winskel G. Relationships between models of concurrency // Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics). 1994. V. 803. P. 425–476. https://doi.org/10.1007/3-540-58043-3_25
27. Vaandrager F.W. Determinism → (event structure isomorphism = step sequence equivalence) // Theoretical Computer Science. 1991. V. 79. N 2. P. 275–294. https://doi.org/10.1016/0304-3975(91)90333-W
28. Sassone V., Nielsen M., Winskel G. Deterministic behavioural models for concurrency // Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics). 1993. V. 711. P. 682–692. https://doi.org/10.1007/3-540-57182-5_59
29. Cohen C. Pragmatic quotient types in Coq // Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics). 2013. V. 7998. P. 213–228. https://doi.org/10.1007/978-3-642-39634-2_17
30. Gonthier G., Mahboubi A., Tassi E. A small scale reflection extension for the Coq system. Inria Saclay Ile de France, 2016.
31. Aceto L., Fokkink W., Verhoef C. Structural operational semantics // Handbook of Process Algebra. Elsevier, 2001. P. 197–292. https://doi.org/10.1016/B978-044482830-9/50021-7
32. Lamport L. How to make a multiprocessor computer that correctly executes multiprocess programs // IEEE Transactions on Computers. 1979. V. 28. N 9. P. 690–691. https://doi.org/10.1109/TC.1979.1675439
33. Singh A., Narayanasamy S., Marino D., Millstein T., Musuvathi M. End-to-end sequential consistency // Proc. of the 39th Annual International Symposium on Computer Architecture (ISCA). 2012. P. 524–535. https://doi.org/10.1109/ISCA.2012.6237045
34. Fidge C.J. Timestamps in message-passing systems that preserve the partial ordering // Australian Computer Science Communications. 1988. V. 10. N 1. P. 56–66.

Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics), 2015, vol. 9032, pp. 283–307. https://doi.org/10.1007/978-3-662-46669-8_12
14. Lahav O., Vafeiadis V., Kang J., Hur C.-K., Dreyer D. Repairing Sequential Consistency in C/C++11. *ACM SIGPLAN Notices*, 2017, vol. 52, no. 6, pp. 618–632. https://doi.org/10.1145/3062341.3062352
15. Gonthier G. Formal proof — the four-color theorem. *Notices of the AMS*, 2008, vol. 55, no. 11, pp. 1382–1393.
16. Leroy X. A formally verified compiler back-end. *Journal of Automated Reasoning*, 2009, vol. 43, no. 4, pp. 363–446. https://doi.org/10.1007/s10817-009-9155-4
17. Pratt V. The pomset model of parallel processes: Unifying the temporal and the spatial. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 1985, vol. 197, pp. 180–196. https://doi.org/10.1007/3-540-15670-4_9
18. Gischer J.L. The equational theory of pomsets. *Theoretical Computer Science*, 1988, vol. 61, no. 2-3, pp. 199–224. https://doi.org/10.1016/0304-3975(88)90124-7
19. Mazurkiewicz A. Trace theory. *Advances in Petri nets 1986, part II on Petri nets: applications and relationships to other models of concurrency*. Springer, 1986, pp. 278–324.
20. Winskel G. Event structures. *Advances in Petri nets 1986, part II on Petri nets: applications and relationships to other models of concurrency*. Springer, 1986, pp. 325–392.
21. Peterson J.L. Petri nets. *ACM Computing Surveys*, 1977, vol. 9, no. 3, pp. 223–252. https://doi.org/10.1145/356698.356702
22. Chakraborty S., Vafeiadis V. Grounding thin-air reads with event structures. *Proceedings of the ACM on Programming Languages*, 2019, vol. 3, no. POPL, pp. 70. https://doi.org/10.1145/3290383
23. Jagadeesan R., Jeffrey A., Riely J. Pomsets with preconditions: a simple model of relaxed memory. *Proceedings of the ACM on Programming Languages*, 2020, vol. 4, no. OOPSLA, pp. 194. https://doi.org/10.1145/3428262
24. Abdulla P., Aronis S., Jonsson B., Sagonas K. Optimal dynamic partial order reduction. *Proc. of the 41st Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages (POPL)*, 2014, pp. 373–384. https://doi.org/10.1145/2535838.2535845
25. Kokologiannakis M., Lahav O., Sagonas K., Vafeiadis V. Effective stateless model checking for C/C++ concurrency. *Proceedings of the ACM on Programming Languages*, 2017, vol. 2, no. POPL, pp. 17. https://doi.org/10.1145/3158105
26. Nielsen M., Sassone V., Winskel G. Relationships between models of concurrency. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 1994, vol. 803, pp. 425–476. https://doi.org/10.1007/3-540-58043-3_25
27. Vaandrager F.W. Determinism → (event structure isomorphism = step sequence equivalence). *Theoretical Computer Science*, 1991, vol. 79, no. 2, pp. 275–294. https://doi.org/10.1016/0304-3975(91)90333-W
28. Sassone V., Nielsen M., Winskel G. Deterministic behavioural models for concurrency. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 1993, vol. 711, pp. 682–692. https://doi.org/10.1007/3-540-57182-5_59
29. Cohen C. Pragmatic quotient types in Coq. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 2013, vol. 7998, pp. 213–228. https://doi.org/10.1007/978-3-642-39634-2_17
30. Gonthier G., Mahboubi A., Tassi E. *A small scale reflection extension for the Coq system*. Inria Saclay Ile de France, 2016.
31. Aceto L., Fokkink W., Verhoef C. Structural operational semantics. *Handbook of Process Algebra*. Elsevier, 2001, pp. 197–292. https://doi.org/10.1016/B978-044482830-9/50021-7
32. Lamport L. How to make a multiprocessor computer that correctly executes multiprocess programs. *IEEE Transactions on Computers*, 1979, vol. 28, no. 9, pp. 690–691. https://doi.org/10.1109/TC.1979.1675439
33. Singh A., Narayanasamy S., Marino D., Millstein T., Musuvathi M. End-to-end sequential consistency. *Proc. of the 39th Annual International Symposium on Computer Architecture (ISCA)*, 2012, pp. 524–535. https://doi.org/10.1109/ISCA.2012.6237045
34. Fidge C.J. Timestamps in message-passing systems that preserve the partial ordering. *Australian Computer Science Communications*, 1988, vol. 10, no. 1, pp. 56–66.

35. Raad A., Doko M., Rožić L., Lahav O., Vafeiadis V. On library correctness under weak memory consistency: specifying and verifying concurrent libraries under declarative consistency models // Proceedings of the ACM on Programming Languages. 2019. V. 3. N POPL. P. 68. https://doi.org/10.1145/3290381
36. Martin-Löf P., Sambin G. Intuitionistic Type Theory. V. 9. Bibliopolis, 1984.
37. Barthe G., Capretta V., Pons O. Setoids in type theory // Journal of Functional Programming. 2003. V. 13. N 2. P. 261–293. https://doi.org/10.1017/S0956796802004501
38. Chicli L., Pottier L., Simpson C. Mathematical quotients and quotient types in Coq // Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics). 2003. V. 2646. P. 95–107. https://doi.org/10.1007/3-540-39185-1_6
39. Gross J.S. Performance Engineering of Proof-Based Software Systems at Scale: Ph. D. thesis / Massachusetts Institute of Technology. 2021.
40. Lamport L. Proving the correctness of multiprocess programs // IEEE Transactions on Software Engineering. 1977. V. SE-3. N 2. P. 125–143. https://doi.org/10.1109/TSE.1977.229904
41. Lahav O., Namakonov E., Oberhauser J., Podkopaev A., Vafeiadis V. Making weak memory models fair // Proceedings of the ACM on Programming Languages. 2021. V. 5. N OOPSLA. P. 98. https://doi.org/10.1145/3485475
42. Affeldt R., Cohen C., Rouhling D. Formalization techniques for asymptotic reasoning in classical analysis // Journal of Formalized Reasoning. 2018. V. 11. N 1. P. 43–76. https://doi.org/10.6092/issn.1972-5787/8124
43. Diaconescu R. Axiom of choice and complementation // Proceedings of the American Mathematical Society. 1975. V. 51. N 1. P. 176–178. https://doi.org/10.1090/S0002-9939-1975-0373893-X
44. Doherty S., Dongol B., Wehrheim H., Derrick J. Making linearizability compositional for partially ordered executions // Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics). 2018. V. 11023. P. 110–129. https://doi.org/10.1007/978-3-319-98938-9_7

35. Raad A., Doko M., Rožić L., Lahav O., Vafeiadis V. On library correctness under weak memory consistency: specifying and verifying concurrent libraries under declarative consistency models. *Proceedings of the ACM on Programming Languages*, 2019, vol. 3, no. POPL, pp. 68. https://doi.org/10.1145/3290381
36. Martin-Löf P., Sambin G. *Intuitionistic Type Theory*. V. 9. Bibliopolis, 1984.
37. Barthe G., Capretta V., Pons O. Setoids in type theory. *Journal of Functional Programming*, 2003, vol. 13, no. 2, pp. 261–293. https://doi.org/10.1017/S0956796802004501
38. Chicli L., Pottier L., Simpson C. Mathematical quotients and quotient types in Coq. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 2003, vol. 2646, pp. 95–107. https://doi.org/10.1007/3-540-39185-1_6
39. Gross J.S. *Performance Engineering of Proof-Based Software Systems at Scale*. Ph. D. thesis. Massachusetts Institute of Technology, 2021.
40. Lamport L. Proving the correctness of multiprocess programs. *IEEE Transactions on Software Engineering*, 1977, vol. SE-3, no. 2, pp. 125–143. https://doi.org/10.1109/TSE.1977.229904
41. Lahav O., Namakonov E., Oberhauser J., Podkopaev A., Vafeiadis V. Making weak memory models fair. *Proceedings of the ACM on Programming Languages*, 2021, vol. 5, no. OOPSLA, pp. 98. https://doi.org/10.1145/3485475
42. Affeldt R., Cohen C., Rouhling D. Formalization techniques for asymptotic reasoning in classical analysis. *Journal of Formalized Reasoning*, 2018, vol. 11, no. 1, pp. 43–76. https://doi.org/10.6092/issn.1972-5787/8124
43. Diaconescu R. Axiom of choice and complementation. *Proceedings of the American Mathematical Society*, 1975, vol. 51, no. 1, pp. 176–178. https://doi.org/10.1090/S0002-9939-1975-0373893-X
44. Doherty S., Dongol B., Wehrheim H., Derrick J. Making linearizability compositional for partially ordered executions. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 2018, vol. 11023, pp. 110–129. https://doi.org/10.1007/978-3-319-98938-9_7

Авторы

Моисеенко Евгений Александрович — аспирант, Санкт-Петербургский государственный университет, Санкт-Петербург, 199034, Российская Федерация; исследователь, JetBrains Research, Санкт-Петербург, 194100, Российская Федерация, SC 57219492685, https://orcid.org/0000-0003-2715-1143, e.moiseenko@2012.spbu.ru

Гладштейн Владимир Петрович — студент, Санкт-Петербургский государственный университет, Санкт-Петербург, 199034, Российская Федерация; исследователь, JetBrains Research, Санкт-Петербург, 194100, Российская Федерация, https://orcid.org/0000-0001-9233-3133, vovaglad00@gmail.com

Подкопаев Антон Викторович — кандидат наук, доцент, Национальный исследовательский университет «Высшая школа экономики», Санкт-Петербург, 194100, Российская Федерация; исследователь, JetBrains Research, Санкт-Петербург, 194100, Российская Федерация, SC 56875418900, https://orcid.org/0000-0002-9448-6587, apodkopaev@hse.ru

Кознов Дмитрий Владимирович — доктор наук, доцент, профессор, Санкт-Петербургский государственный университет, Санкт-Петербург, 199034, Российская Федерация, SC 8885649400, https://orcid.org/0000-0003-2632-3193, d.koznov@spbu.ru

Authors

Evgenii A. Moiseenko — PhD Student, Saint Petersburg State University, Saint Petersburg, 199034, Russian Federation; Researcher, JetBrains Research, Saint Petersburg, 194100, Russian Federation, SC 57219492685, https://orcid.org/0000-0003-2715-1143, e.moiseenko@2012.spbu.ru

Vladimir P. Gladstein — Student, Saint Petersburg State University, Saint Petersburg, 199034, Russian Federation; Researcher, JetBrains Research, Saint Petersburg, 194100, Russian Federation, https://orcid.org/0000-0001-9233-3133, vovaglad00@gmail.com

Anton V. Podkopaev — PhD, Associate Professor, HSE University, Saint Petersburg, 194100, Russian Federation; Researcher, JetBrains Research, Saint Petersburg, 194100, Russian Federation, SC 56875418900, https://orcid.org/0000-0002-9448-6587, apodkopaev@hse.ru

Dmitry V. Koznov — D. Sc., Associate Professor, Professor, Saint Petersburg State University, Saint Petersburg, 199034, Russian Federation, SC 8885649400 https://orcid.org/0000-0003-2632-3193, d.koznov@spbu.ru

Статья поступила в редакцию 28.01.2022
Одобрена после рецензирования 24.03.2022
Принята к печати 15.05.2022

Received 28.01.2022
Approved after reviewing 24.03.2022
Accepted 15.05.2022