MODELING AND ANALYSIS OF INFORMATION SYSTEMS, VOL. 31, NO. 4, 2024
journal homepage: www.mais-journal.ru

THEORY OF COMPUTING

Pattern-based Approach to Automation of Deductive Verification of Process-oriented Programs: Patterns, Lemmas and Algorithms

I. M. Chernenko[1], I. S. Anureev[1]

DOI: 10.18255/1818-1015-2024-4-384-425

[1]Institute of Automation and Electrometry SB RAS, Novosibirsk, Russia

MSC2020: 68Q60
Research article
Full text in Russian

Received November 6, 2024
Revised November 15, 2024
Accepted November 25, 2024

Process-oriented programming is an approach to developing control software in which a program is defined as a set of interacting processes. PoST is a process-oriented language, which is an extension of the ST language from the IEC 61131-3 standard. In the field of control software development, formal verification plays an important role due to the need to ensure high reliability of such software. Deductive verification is a formal verification method in which a program and its requirements are represented as logical formulas, and logical inference is used to prove that the program satisfies the requirements. Control software often has temporal requirements. We formalize such requirements for process-oriented programs as control loop invariants. However, control loop invariants that represent requirements are not sufficient to prove the correctness of the program. Therefore, we add extra invariants containing auxiliary information. This paper considers the problem of automating deductive verification of process-oriented programs. An approach is proposed in which temporal requirements are specified using requirement patterns which are constructed from basic patterns. For each requirement pattern, a corresponding extra invariant pattern and lemmas are defined. In this paper, the proposed approach and schemes of basic and derived requirement patterns are described. The schemes of basic extra invariant patterns, schemes of lemmas defined for basic patterns, and a set of basic patterns and lemmas for them are considered. The scheme of derived extra invariant patterns and schemes of lemmas defined for derived patterns are defined. The algorithms for constructing derived extra invariant patterns and lemmas for them, as well as methods for proving these lemmas are presented. The schemes of proving verification conditions are considered. The proposed approach is demonstrated with an example. The analysis of related works has also been carried out.

Keywords: deductive verification; temporal requirements, requirement pattern; loop invariant; control software; process-oriented programming

INFORMATION ABOUT THE AUTHORS

Chernenko, Ivan M. (corresponding author)	ORCID iD: 0000-0001-7675-8449. E-mail: cheriv98@mail.ru Graduate student
Anureev, Igor S.	ORCID iD: 0000-0001-9574-128X. E-mail: anureev@gmail.com Senior researcher, PhD

Funding: State task IAaE SB RAS, project No. 122031600173-8.

For citation: I. M. Chernenko and I. S. Anureev, "Pattern-based approach to automation of deductive verification of process-oriented programs: patterns, lemmas and algorithms", *Modeling and Analysis of Information Systems*, vol. 31, no. 4, pp. 384–425, 2024. DOI: 10.18255/1818-1015-2024-4-384-425.

МОДЕЛИРОВАНИЕ И АНАЛИЗ ИНФОРМАЦИОННЫХ СИСТЕМ, ТОМ 31, № 4, 2024
сайт журнала: www.mais-journal.ru

THEORY OF COMPUTING

Подход к автоматизации дедуктивной верификации процесс-ориентированных программ, основанный на шаблонах: шаблоны, леммы и алгоритмы

И. М. Черненко[1], И. С. Ануреев[1]

DOI: 10.18255/1818-1015-2024-4-384-425

[1]Институт автоматики и электрометрии СО РАН, Новосибирск, Россия

УДК 004.415.52
Научная статья
Полный текст на русском языке

Получена 6 ноября 2024 г.
После доработки 15 ноября 2024 г.
Принята к публикации 25 ноября 2024 г.

Процесс-ориентированное программирование — это подход к разработке управляющего программного обеспечения, в котором программа определяется как набор взаимодействующих процессов. PoST — это процесс-ориентированный язык, который является расширением языка ST из стандарта IEC 61131-3. В области разработки управляющего программного обеспечения формальная верификация играет важную роль вследствие необходимости обеспечения высокой надежности такого программного обеспечения. Дедуктивная верификация — это метод формальной верификации, в котором программа и требования к ней представляются в виде логических формул, а для доказательства того, что программа удовлетворяет требованиям, используется логический вывод. К управляющему программному обеспечению часто предъявляются темпоральные требования. Мы формализуем такие требования для процесс-ориентированных программ в виде инвариантов цикла управления. Но инварианты цикла управления, представляющие требования, недостаточны для доказательства корректности программы. Поэтому мы добавляем дополнительные инварианты, которые содержат вспомогательную информацию. В данной статье рассматривается проблема автоматизации дедуктивной верификации процесс-ориентированных программ. Предложен подход, в котором темпоральные требования задаются с использованием шаблонов требований, которые строятся из базовых шаблонов. Для каждого шаблона требований определяются соответствующий шаблон дополнительных инвариантов и леммы. В статье описан предлагаемый подход и схемы базовых и производных шаблонов требований. Рассмотрены схемы базовых шаблонов дополнительных инвариантов, схемы лемм, определяемых для базовых шаблонов, а также набор базовых шаблонов и леммы для них. Определены схема производных шаблонов дополнительных инвариантов и схемы лемм, определяемых для производных шаблонов. Представлены алгоритмы построения производных шаблонов дополнительных инвариантов и лемм для них, а также метод доказательства этих лемм. Рассмотрены схемы доказательства условий корректности. Предложенный подход демонстрируется на примере. Также проведен анализ связанных работ.

Ключевые слова: дедуктивная верификация; темпоральные требования; шаблон требований; инвариант цикла; управляющее программное обеспечение; процесс-ориентированное программирование

ИНФОРМАЦИЯ ОБ АВТОРАХ

Черненко, Иван Михайлович (автор для корреспонденции) | ORCID iD: 0000-0001-7675-8449. E-mail: cheriv98@mail.ru
Аспирант

Ануреев, Игорь Сергеевич | ORCID iD: 0000-0001-9574-128X. E-mail: anureev@gmail.com
Старший научный сотрудник, к.ф.-м.н.

Финансирование: Госзадание ИАиЭ СО РАН, проект № 122031600173-8.

Для цитирования: I. M. Chernenko and I. S. Anureev, “Pattern-based approach to automation of deductive verification of process-oriented programs: patterns, lemmas and algorithms”, *Modeling and Analysis of Information Systems*, vol. 31, no. 4, pp. 384–425, 2024. DOI: 10.18255/1818-1015-2024-4-384-425.

Введение

Процесс-ориентированное программирование [1] — перспективный метод разработки управляющего программного обеспечения. Эта парадигма программирования позволяет описывать программу как набор взаимодействующих процессов. Каждый процесс представляет собой расширенный конечный автомат и определяется набором именованных состояний, содержащих программный код. Помимо активных состояний, определенных в коде, каждый процесс имеет два неактивных состояния: состояние нормальной остановки *STOP* и состояние остановки по ошибке *ERROR*. Выполнение программы осуществляется циклически: на каждой итерации цикла управления все процессы программы выполняются последовательно в их текущих состояниях. Длительность нахождения процесса в текущем состоянии контролируется оператором таймаута. С каждым процессом связан таймер для контроля этого времени. Таймер сбрасывается, когда процесс переходит в другое состояние, а также может быть сброшен программно. Процессы имеют возможность запускать и останавливать другие процессы, а также проверять, активен или неактивен другой процесс. При первом запуске процесса он переходит в состояние, которое определено первым в определении этого процесса. При запуске программы запускается ее первый процесс, а все остальные процессы неактивны и находятся в состоянии *STOP*.

Язык poST [2] — это процесс-ориентированный язык, расширяющий язык ST из стандарта IEC 61131-3 [3]. Программа на языке poST состоит из объявлений переменных и определений процессов. Программа или процесс может содержать объявления входных переменных `VAR_INPUT`, значения которых изменяются средой на каждой итерации цикла управления, выходных переменных `VAR_OUTPUT`, которые определяют управляющие сигналы, и локальных переменных `VAR`. Определение процесса содержит последовательность определений состояний.

Управляющее программное обеспечение требует формальной верификации, поскольку имеет высокие требования к надежности. Дедуктивная верификация [4] — один из методов формальной верификации, в котором требования к программе формализуются в виде логических формул, по программе и формализованным требованиям генерируются условия корректности — логические формулы, истинность которых гарантирует корректность программы относительно этих требований, а затем порожденные условия корректности доказываются. Для каждого цикла в программе должен быть указан инвариант цикла — логическая формула, истинная при входе в цикл и после каждой его итерации.

Важным классом требований к управляющему программному обеспечению являются темпоральные требования. Для задания темпоральных требований для процесс-ориентированных программ в [5] предложен подход, в котором требования задаются как инварианты цикла управления. При описании требований программа рассматривается как черный ящик, т. е. требования не содержат информацию о структуре программы (состояниях процессов, значениях таймеров процессов и локальных переменных). Однако такая информация необходима для доказательства условий корректности. Поэтому в этом подходе инвариант цикла управления представляется в виде конъюнкции формализованного требования и дополнительного инварианта, содержащего информацию о структуре программы. Требования и дополнительные инварианты задаются на языке темпоральных требований DV-TRL [6], который является вариантом типизированной логики первого порядка. Этот язык основан на типе данных *состояние изменений* (*state*), значения которого представляют собой истории всех изменений в программе. Специализированные функции над этим типом позволяют использовать в требованиях значения переменных в различные моменты времени. Это дает возможность задавать темпоральные требования.

При дедуктивной верификации полностью автоматизировать можно только решение задачи порождения условий корректности. Задачи синтеза инвариантов циклов и доказательства условий

корректности в общем случае неразрешимы. Тем не менее, существуют подходы к решению этих задач в частных случаях.

Одним из подходов к автоматизации решения этих задач является использование шаблонов. В работе [7] был разработан набор шаблонов темпоральных требований на языке DV-TRL. Для каждого такого шаблона требований были определены соответствующий шаблон дополнительных инвариантов, используемый для задания зависящих от требований инвариантов, и набор лемм, необходимых для доказательства условий корректности, порождаемых для требований и дополнительных инвариантов, удовлетворяющих этим шаблонам. Также был предложен набор шаблонов дополнительных инвариантов, независимых от требований. В совокупности это позволило автоматизировать дедуктивную верификацию процесс-ориентированных программ для достаточно широкого класса требований. Однако этот подход требует разработки новых шаблонов требований и дополнительных инвариантов каждый раз, когда появляется требование, не удовлетворяющее ранее разработанным шаблонам. Было замечено, что ранее разработанные шаблоны и шаблоны, которые могли бы описывать новые классы требований, могут быть составлены по определенным правилам из небольшого числа базовых шаблонов. В этой статье представлена модификация подхода к автоматизации дедуктивной верификации процесс-ориентированных программ, основанного на шаблонах, в которой шаблоны требований могут быть построены путем комбинирования базовых шаблонов, а соответствующие шаблоны дополнительных инвариантов и леммы с их доказательствами могут быть сгенерированы автоматически.

Эта статья имеет следующую структуру. Раздел 1 описывает наш подход к автоматизации дедуктивной верификации, а также общий вид (схемы) базовых и производных шаблонов требований. В разделе 2 дана классификация базовых шаблонов требований на шаблоны будущего и шаблоны прошлого, и в рамках этой классификации приведены схемы базовых шаблонов дополнительных инвариантов, соответствующих базовым шаблонам требований, схемы лемм для пар базовых шаблонов — требований и дополнительных инвариантов, а также конкретные пары таких шаблонов, используемых на практике, и леммы для них. В разделе 3 описываются схемы производных шаблонов дополнительных инвариантов, которые строятся по соответствующим производным шаблонам требований, а также схемы лемм для этих пар шаблонов. В разделе 4 описаны алгоритмы построения производных шаблонов дополнительных инвариантов по соответствующему производному шаблону требований, а также алгоритмы построения лемм для этих пар шаблонов. Раздел 5 демонстрирует наш подход на примере. В разделе 6 приведены схемы доказательства условий корректности. Раздел 7 представляет обзор связанных работ. В заключении подводятся итоги работы.

1. Подход к автоматизации дедуктивной верификации

В этом разделе описывается наш подход к автоматизации дедуктивной верификации процесс-ориентированных программ на основе шаблонов.

1.1. Классификация шаблонов

Мы используем шаблоны требований для задания требований и шаблоны дополнительных инвариантов для задания дополнительных инвариантов. Дополнительные инварианты и их шаблоны разделяются на зависимые и независимые от требований.

Независимые от требований дополнительные инварианты связывают значения переменных, состояния процессов и значения их таймеров и являются бесквантворными формулами. Данные инварианты могут быть найдены с помощью статического анализа кода процесс-ориентированной программы и использоваться для доказательства бесквантворных формул в доказательствах условий корректности автоматическими методами. Независимые от требований инварианты могут использоваться в доказательствах условий корректности для различных требований.

Зависимые от требований инварианты описывают вспомогательные свойства, необходимые для доказательства соответствующих требований. Эти инварианты связывают значения переменных в различные моменты времени с состояниями процессов, значениями их таймеров и других переменных в точке, в которой выполняется инвариант.

Для каждой программы можно определить несколько дополнительных инвариантов, независимых от требований. Для каждого требования определяется один дополнительный инвариант, зависящий от требований.

Шаблоны требований и шаблоны дополнительных инвариантов, зависящих от требований, разделяются на базовые и производные.

Базовые шаблоны требований представляют собой утверждения на формулах, определенных в языке DV-TRL, которые используются для построения производных шаблонов. Базовые шаблоны дополнительных инвариантов — это шаблоны вспомогательных свойств, необходимых для доказательства соответствующих экземпляров базового шаблона требований.

Производный шаблон требований определяет некоторый класс требований и используется для формализации требований путем указания значений параметров шаблона. Производный шаблон дополнительных инвариантов — это шаблон, используемый для задания зависимых от требований дополнительных инвариантов. Зависимый от требований инвариант является усилением соответствующего требования, т. е. утверждает, что выполняется требование, а также некоторое дополнительное свойство.

Каждый базовый (производный) шаблон требований имеет соответствующий базовый (производный) шаблон дополнительных инвариантов и набор лемм, связанных с ним.

Леммы для пары, состоящей из базового шаблона требований и соответствующего шаблона дополнительных инвариантов, сводят доказательство формул определенного вида, содержащих экземпляры этих базовых шаблонов, к доказательству формул, не содержащих эти экземпляры базовых шаблонов за исключением экземпляров, выполняющихся в начале итерации, но содержащих значения их параметров. Эти леммы используются для доказательства лемм для производных шаблонов, содержащих эти базовые шаблоны.

Леммы для пары производных шаблонов сводят доказательство условий корректности и лемм для других производных шаблонов, построенных с помощью этих, к доказательству более простых утверждений, причем для доказательства условий корректности применяются упрощенные леммы, применение которых дает бесквантерные формулы, которые во многих случаях могут быть доказаны автоматически. Это позволяет автоматически строить производный шаблон дополнительных инвариантов и леммы, соответствующие производному шаблону требований, используя определение шаблона требований, а также автоматически доказывать эти леммы.

Для доказательства условий корректности мы определяем скрипты доказательства. Эти скрипты вместе с леммами позволяют автоматически доказывать условия корректности. Связь между различными видами шаблонов и их экземплярами представлена на рисунке 1, а связь лемм и шаблонов представлена на рисунке 2.

1.2. Схема взаимодействия пользователя с инструментом верификации

Возможное взаимодействие пользователя с инструментом верификации, построенным на базе данного подхода, показано на рисунке 3. Сначала пользователь выбирает очередное недоказанное требование и строит производный шаблон требований, которому это требование удовлетворяет. Затем пользователь выбирает шаблон независимых от требований дополнительных инвариантов на основе верифицируемой процесс-ориентированной программы. Эти два шаблона подаются на вход инструмента верификации. Для переданного производного шаблона требований инструмент генерирует соответствующий производный шаблон дополнительных инвариантов и леммы, которые доказываются в Isabelle/HOL [8]. Производный шаблон возвращается пользователю и он

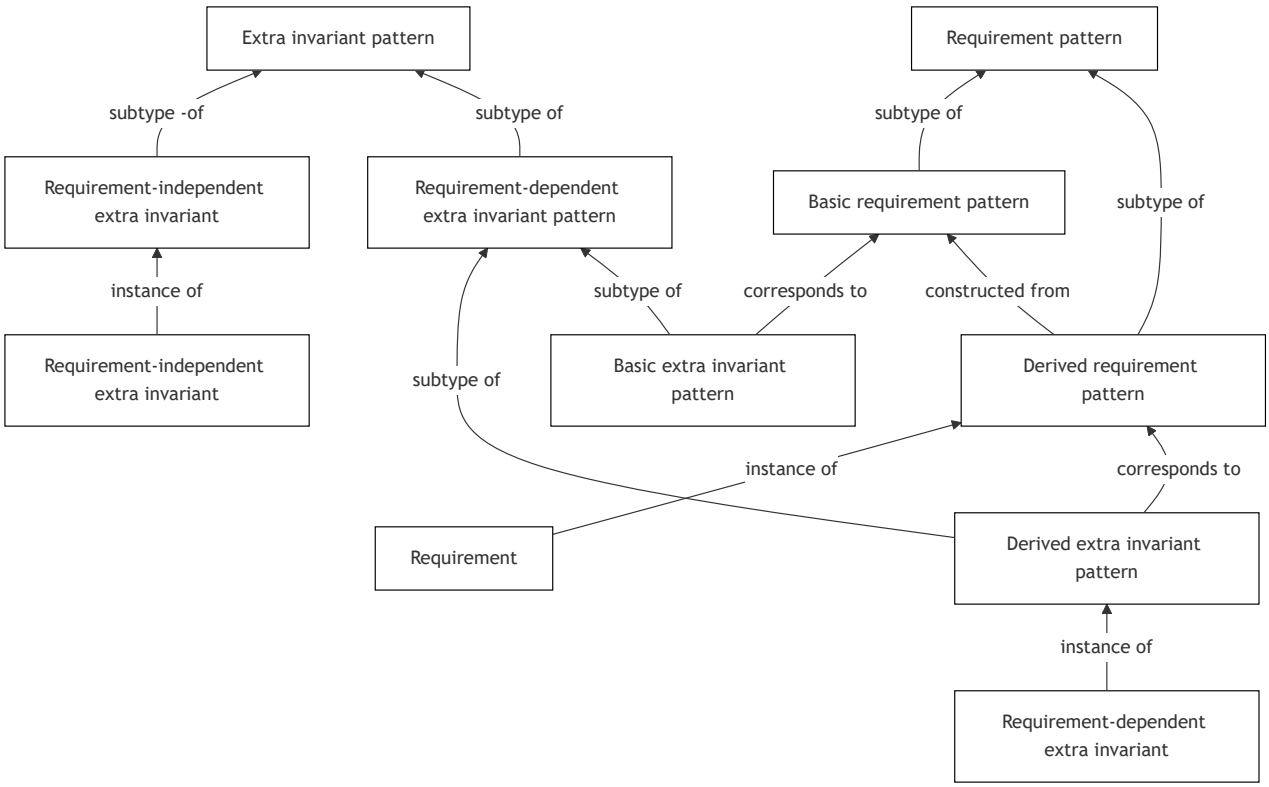

Fig. 1. Relationship between different kinds of patterns and their instances

Рис. 1. Связь между различными видами шаблонов и их экземплярами

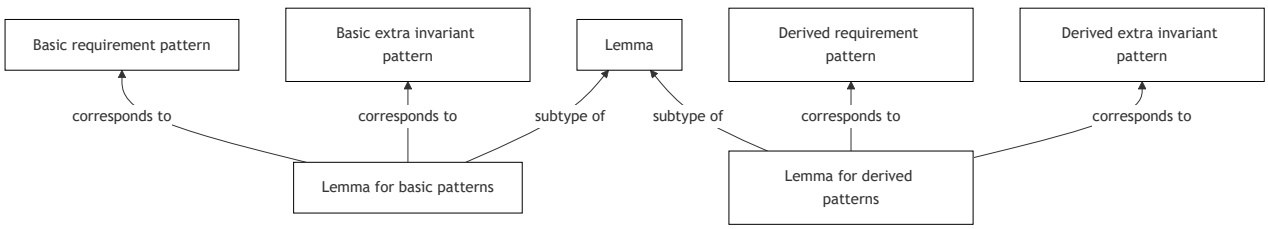

Fig. 2. Relationship between lemmas and patterns

Рис. 2. Связь между леммами и шаблонами

для трех имеющихся шаблонов задает конкретные значения параметров и передает их инструменту верификации. Далее на основе конкретизированных шаблонов и верифицируемой программы генерируются условия корректности на основе процесс-ориентированной программы и скрипты доказательства для доказательства условий корректности. Эти скрипты доказываются в Isabelle/HOL с использованием порожденных выше лемм. Если условия корректности доказаны, пользователь переходит к проверке следующего требования. В противном случае, пользователь уточняет параметры шаблонов, а инструмент верификации переходит к повторной генерации условий корректности. Инструмент верификации сохраняет необходимые пары производных шаблонов требований и порожденных для них шаблонов дополнительных инвариантов с соответствующими леммами в базе знаний для их повторного использования.

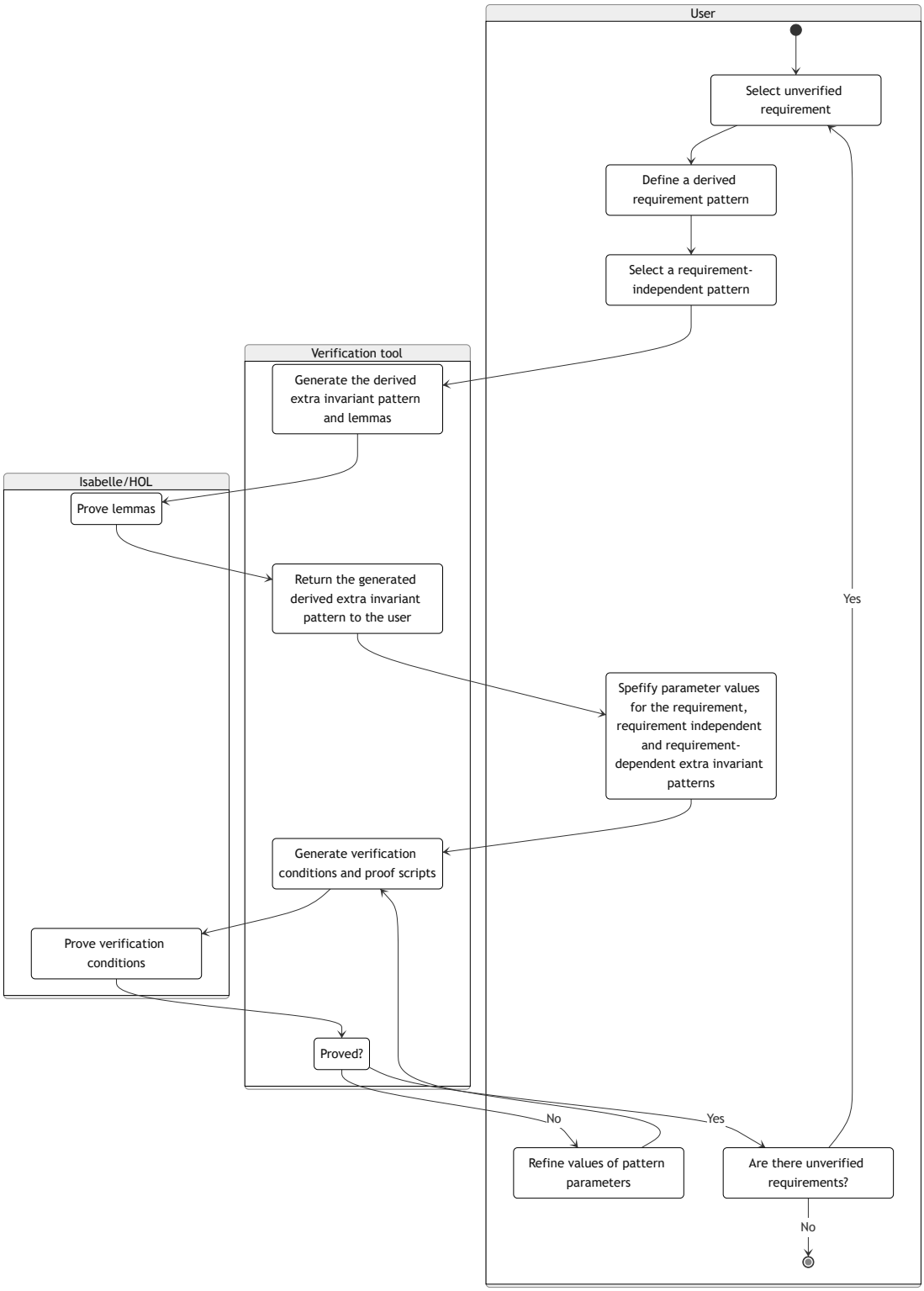

Fig. 3. User interaction with the verification tool

Рис. 3. Взаимодействие пользователя с инструментом верификации

1.3. Формат базовых и производных шаблонов

В шаблонах, обсуждаемых ниже, используются следующие обозначения:

- $s, s_1, \ldots, s_n, r, r_1, \ldots, r_n$ — состояния;
- $A_1, \ldots, A_n, B_1, \ldots, B_n, C_1, \ldots, C_n, D_1, \ldots, D_n$ — произвольные логические формулы;
- $p(s)$ возвращает предыдущее *внешнее* состояние. *Внешнее* состояние — это состояние в точке передачи значений переменных от контроллера к объекту управления в цикле управления. В противном случае, состояние является *внутренним*;
- $s \leqslant r$ возвращает true, если $s = r$, или $s \leqslant p(r)$;
- $s_1 \leqslant \ldots \leqslant s_n$ — сокращение для $s_1 \leqslant s_2 \wedge \ldots \wedge s_{n-1} \leqslant s_n$;
- $s < r$ возвращает true, если $s \leqslant r$ и $s \neq r$;
- $e(s)$ возвращает true, если состояние s является внешним;
- $n(s_1, s_2)$ возвращает количество внешних состояний между состояниями s_1 и s_2, включая s_2, но не включая s_1;
- $s[x]$ — значение программной переменной x в состоянии s;
- $getPstate(s, p)$ — состояние процесса p в состоянии изменений s;
- $ltime(s, p)$ — значение таймера процесса p в состоянии s;
- $consecutive(s_1, s_2)$ возвращает *true*, если $e(s_1) \wedge e(s_2) \wedge s_1 \leqslant s_2 \wedge n(s_1, s_2) = 1$.

В темпоральной логике формулы можно преобразовать в положительную нормальную форму (positive normal form) [9] (глава 4), в которой отрицания применяются только к атомарным формулам, при наличии двойственных операторов. В нашей работе формулы также представляются в положительной нормальной форме — в качестве логических связок используются только конъюнкция, дизъюнкция и отрицание, причем отрицание применяется только к атомарным формулам. Поэтому мы рассматриваем только комбинирование шаблонов требований с использованием конъюнкции, дизъюнкции и композиции шаблонов.

Каждый базовый шаблон параметризуется двумя состояниями изменений: s_1 (состояние изменений, в котором выполняется экземпляр шаблона) и s (состояние изменений, в котором выполняется инвариант цикла).

Определение 1. (Базовый шаблон требований)
Базовый шаблон требований — шаблон требований, определение которого имеет вид:

$$R \equiv \lambda s.\lambda(s_1, p_1, \ldots, p_m, A_1, \ldots, A_n).R'(s_1, s, p_1, \ldots, p_m, A_1(s, r_{j_1}), \ldots, A_n(s, r_{j_n}))\ ,$$

где значениями параметров $p_1, \ldots, p_m$ (называемых константными параметрами или с-параметрами) могут быть только константы базовых типов (целочисленных и булевского), R' — произвольная параметризованная формула языка DV-TRL, в которой параметры $A_1, \ldots, A_n$ (называемые формульными параметрами или fm-параметрами) не содержатся в посылках импликаций, r_{j_i} (i= 1, . . . , n) — переменные, обозначающие состояния изменений, связанные кванторами в R' такие, что $r_{j_i} \leqslant s$, а значения fm-параметров $A_1, \ldots, A_n$ являются формулами в положительной нормальной форме и принадлежат множеству F_r, определяемому следующим образом: 1) атомарные формулы и их отрицания, параметризованные двумя состояниями изменений, принадлежат F_r, 2) если B_1 и B_2 принадлежат F_r, то $\lambda(s, s_1).B_1(s, s_1) \wedge B_2(s, s_1)$ и $\lambda(s, s_1).B_1(s, s_1) \vee B_2(s, s_1)$ принадлежат F_r, 3) если P — шаблон требований с m' с-параметрами и n' fm-параметрами, $q_1, \ldots, q_{m'}$ — значения с-параметров, и $B_1, \ldots, B_{n'}$ — значения fm-параметров, принадлежащие F_r, то $\lambda(s, s_1).P(s, s_1, p_1, \ldots, p_{m'}, B_1, \ldots, B_{n'})$ принадлежит F_r.

Определение 2. (Производный шаблон требований)
Производный шаблон требований — шаблон требований, определение которого имеет следующий вид:

$$R \equiv \lambda s.\lambda(p_1, \ldots, p_m, A_1, \ldots, A_u, B_1, \ldots, B_v).R'(s, p_1, \ldots, p_m, A_1, \ldots, A_u, B_1(s), \ldots, B_v(s))\ ,$$

где s — состояние изменений, в котором должно быть выполнено требование; $p_1, \ldots, p_m$ — с-параметры; B_1,.., B_v — fm-параметры, значения которых принадлежат множеству F_r; $A_1, \ldots, A_u$ — fm-параметры, значения которых принадлежат множеству F_p, задаваемому следующим образом: 1) атомарные формулы, параметризованные одним состоянием изменений, и их отрицания принадлежат F_p, 2) если C_1 и C_2 принадлежат F_p, то $\lambda s_1.C_1(s_1) \wedge C_2(s_1)$ и $\lambda s_1.C_1(s_1) \vee C_2(s_1)$ принадлежат F_p; R' — параметризованная формула, являющаяся булевой комбинацией формул вида $P(s, q_1, \ldots, q_{m'}, D_1, \ldots, D_{v'})$ и $P(s, q_1, \ldots, q_{m'}, C_1, \ldots, C_{u'}, D_1, \ldots, D_{v'})$, где P — это базовый или производный шаблон требований с m' с-параметрами, u' fm-параметрами со значениями из F_p (отсутствуют у базового шаблона) и v' fm-параметрами со значениями из F_r, $q_1, \ldots, q_{m'}$ — значения с-параметров, а $C_1, \ldots, C_{u'}$ и $D_1, \ldots, D_{v'}$ — значения соответствующих fm-параметров.

2. Базовые шаблоны дополнительных инвариантов и леммы

В этом разделе представлены схемы базовых шаблонов дополнительных инвариантов, а также схемы лемм, определяемых для каждой пары, состоящей из базового шаблона требований и соответствующего базового шаблона дополнительных инвариантов, и примеры таких пар и соответствующих лемм. Леммы для базовых шаблонов используются для доказательства лемм для производных шаблонов, которые в свою очередь используются для доказательства условий корректности, порожденных для требований и дополнительных инвариантов, соответствующих этим производным шаблонам.

Базовые шаблоны требований делятся на шаблоны будущего и шаблоны прошлого. Базовый шаблон является шаблоном будущего, если $s_1 \leqslant r_{j_i}$ для всех i и j. Базовый шаблон является шаблоном прошлого, если $r_{j_i} \leqslant s_1$ для всех i и j. Базовый шаблон дополнительного инварианта относится к тому же времени, что и соответствующий ему базовый шаблон требований. Определим для каждой пары таких шаблонов соответствующие им леммы отдельно для шаблонов будущего и прошлого.

Отметим следующую особенность используемого подхода, связанную с тем, что требования задаются в виде инвариантов циклов, описывающих истории значений переменных. В используемом языке темпоральных требований не могут быть заданы требования, утверждающие что-либо о будущем относительно момента времени, когда выполняется инвариант цикла управления. Поэтому будущее в шаблонах будущего ограничено состоянием изменений s, в котором выполняется инвариант цикла, и экземпляры шаблонов будущего всегда вложены в шаблоны прошлого или в производные шаблоны, задающие утверждения о прошлом относительно состояния, в котором выполняется инвариант. То, что будущее в данном подходе является ограниченным, приводит к невозможности задавать некоторые утверждения о будущем, которые могут быть выражены в логиках, в которых будущее является неограниченным. К таким свойствам относятся свойства вида: «Событие должно произойти в будущем». Некоторые другие утверждения о будущем можно задать в используемом языке темпоральных требований, так как их можно свести к эквивалентным утверждениях об ограниченном будущем. К таким требованиям относятся утверждения, задающие максимальное время, в течение которого должно произойти событие, а также утверждения о том, что какое либо условие должно выполняться, пока не произойдет некоторое событие. Также в используемом языке требований не могут быть заданы требования, для формализации которых необходима некоторая логика ветвящегося времени. Это связано с тем, что все состояния изменений, входящие в требование соответствуют моментам времени, предшествующим моменту времени, когда выполняется инвариант, и принадлежат одному пути исполнения программы.

2.1. Шаблоны будущего

Определение 3. Шаблон дополнительных инвариантов будущего — шаблон дополнительных инвариантов, определение которого имеет следующий вид:

$$I \equiv \lambda s.\lambda(s_1, p_1, \ldots, p_m, ep_1, \ldots, ep_k, A'_1, \ldots, A'_n).I'(s_1, s, p_1, \ldots, p_m, ep_1(s), \ldots, ep_k(s), A'_1(s, r_{j_1}), \ldots, A'_n(s, r_{j_n}))\ ,$$

где $p_1, \ldots, p_m$ — c-параметры из базового шаблона требований, $ep_1, \ldots, ep_k$ — функции (называемые функциональными параметрами или fn-параметрами), возвращающие для состояния значения базовых типов, значения этих параметров содержат дополнительную информацию о структуре программы, необходимую для доказательства, I' — параметризованная формула языка DV-TRL, в которой fm-параметры $A'_1, \ldots, A'_n$ определяются по соответствующим fm-параметрам базового шаблона требований, r_{j_i} (i=1, …, n) — связанные кванторами в I' переменные, обозначающие состояния изменений такие, что $s_1 \leqslant r_{j_i} \leqslant s$, значения $A'_1, \ldots, A'_n$ принадлежат множеству F_e, определяемому следующим образом: 1) атомарные формулы и их отрицания принадлежат F_e, 2) если B_1 и B_2 принадлежат F_e, то $\lambda(s, s_1).B_1(s, s_1) \wedge B_2(s, s_1)$ и $\lambda(s, s_1).B_1(s, s_1) \vee B_2(s, s_1)$ также принадлежат F_e, 3) если P — шаблон дополнительных инвариантов будущего с m' c-параметрами, k' fn-параметрами и n' fm-параметрами, $q_1, \ldots, q_{m'}$ — значения базовых типов, $ep_1, \ldots, ep_{k'}$ — функции соответствующих типов, и $B_1, \ldots, B_{n'}$ принадлежат F_e, то $\lambda(s, s_1).P(s, s_1, q_1, \ldots, q_{m'}, ep_1, \ldots, ep_{k'}, B_1, \ldots, B_{n'})$ принадлежит F_e, 4) если P — шаблон требований прошлого с m' c-параметрами и n' fm-параметрами, $q_1, \ldots, q_{m'}$ — значения базовых типов, $B_1, \ldots, B_{n'}$ принадлежат F_e, то $\lambda(s, s_1).P(s, s_1, q_1, \ldots, q_{m'}, B_1, \ldots, B_{n'})$ принадлежит F_e.

Леммы для любой пары, состоящей из базового шаблона требований будущего и соответствующего базового шаблона дополнительных инвариантов, должны удовлетворять следующим схемам:

1. LS_1:

$$
\begin{aligned}
&consecutive(s, s') \longrightarrow \\
&cond_{true}(p_1, \ldots, p_m) \vee \\
&(\forall s_1.e(s_1) \wedge s_1 \leqslant s \wedge A'_1(s, s_1) \longrightarrow A'_1(s', s_1)) \wedge \ldots \\
&(\forall s_1.e(s_1) \wedge s_1 \leqslant s \wedge A'_n(s, s_1) \longrightarrow A'_n(s', s_1)) \wedge \\
&cond(p_1, \ldots, p_m, ep_1(s), \ldots, ep_k(s), ep_1(s'), \ldots, ep_k(s'), A'_1(s', s'), \ldots, A'_n(s', s')) \longrightarrow \\
&\forall s_1.e(s_1) \wedge s_1 \leqslant s \wedge I(s)(s_1, p_1, \ldots, p_m, ep_1, \ldots, ep_k, A'_1, \ldots, A'_n) \longrightarrow I(s')(s_1, p_1, \ldots, p_m, ep_1, \ldots, ep_k, A'_1, \ldots, A'_n)
\end{aligned}
$$

   Леммы, удовлетворяющие этой схеме, используются для доказательства лемм, применяемых для доказательства условий корректности для дополнительных инвариантов. Здесь и в других леммах, связывающих дополнительные инварианты до и после итерации цикла управления, переменная s обозначает состояние изменений до итерации, а s' обозначает состояние изменений после итерации. Посылка леммы, удовлетворяющей этой схеме, утверждает, что если условие $cond_{true}$, при котором требование тождественно истинно, ложно, то для каждого состояния изменений вплоть до s, если выполняется формула $A'_i(s)$, то формула $A'_i(s')$ выполняется в том же состоянии, $i = 1, \ldots, n$, и выполняется формула $cond$, выражающая связь между $A'_i(s', s')$, p_i, $ep_i(s)$ и $ep_i(s')$. Заключение леммы утверждает, что если была выполнена подформула дополнительного инварианта с начальным состоянием s_1 и конечным состоянием s, то также выполняется подформула дополнительного инварианта с конечным состоянием s'. Заметим, что формула в заключении может рассматриваться как формула посылки, если вместо $A'_i(s, s_1)$ используется $I(s, s_1, p_1, \ldots, p_m, ep_1, \ldots, ep_k, A_1, \ldots, A_n)$, а вместо $A'_i(s', s_1)$ используется $I(s', s_1, p_1, \ldots, p_m, ep_1, \ldots, ep_k, A'_1, \ldots, A'_n)$. Это позволяет определять и доказывать леммы для производных шаблонов индукцией по построению этих шаблонов.

2. LS_2:

$$e(s) \longrightarrow cond(p_1, \ldots, p_m, ep_1(s), \ldots, ep_k(s), A'_1(s, s), \ldots, A'_n(s, s)) \longrightarrow I(s, s, p_1, \ldots, p_m, ep_1, \ldots, ep_k, A'_1, \ldots, A'_n)$$

   Леммы, удовлетворяющие этой схеме, используются в доказательствах лемм, применяемых для доказательства условий корректности для дополнительных инвариантов, в случае, когда начальное и конечное состояния изменений совпадают. Условие $cond$ выражает связь между $A'_1(s, s), \ldots, A'_n(s, s)$, $p_1, \ldots, p_m$, $ep_1(s), \ldots, ep_k(s)$.

3. LS_3:

$e(s) \longrightarrow$
$(\forall s_1.e(s_1) \wedge s_1 \leqslant s \wedge A'_1(s, s_1) \longrightarrow A_1(s, s_1)) \wedge \ldots \wedge$
$(\forall s_1.e(s_1) \wedge s_1 \leqslant s \wedge A'_n(s, s_1) \longrightarrow A_n(s, s_1)) \wedge$
$cond(p_1, \ldots, p_m, ep_1(s), \ldots, ep_k(s)) \longrightarrow$
$\forall s_1.e(s_1) \wedge s_1 \leqslant s \wedge I(s, s_1, p_1, \ldots, p_m, ep_1, \ldots, ep_k, A'_1, \ldots, A'_n) \longrightarrow R(s, s_1, p_1, \ldots, p_m, A_1, \ldots, An)$

Леммы, удовлетворяющие этой схеме, используются для доказательства лемм, применяемых для доказательства условий корректности для требований с использованием дополнительных инвариантов. Посылка *cond* лемм, удовлетворяющих этой схеме, связывает значения $p_1, \ldots, p_m$ и $ep_1(s), \ldots, ep_k(s)$.

Значения параметров $cond_{true}$ и *cond* в леммах, удовлетворяющих схемам лемм LS_1, LS_2 и LS_3, являются бесквантормыми формулами, не содержащими шаблонов.

Далее рассмотрим конкретные шаблоны требований будущего и соответствующие им базовые шаблоны дополнительных инвариантов и леммы.

2.1.1. Шаблон FRP1

Первый шаблон требований будущего FRP1, определенный ниже, утверждает, что не позднее, чем через время t после начала отсчета времени в состоянии s_1 произойдет событие A_2, а с начала отсчета времени до возникновения события A_2 выполняется условие A_1.

$FRP1(s, s_1, t, A_1, A_2) \equiv$
$n(s_1, s) \geqslant t \longrightarrow$
$(\exists r_2.e(r_2) \wedge s_1 \leqslant r_2 \leqslant s \wedge n(s_1, r_2) \leqslant t \wedge A_2(s, r_2) \wedge$
$\quad (\forall r_1.(e(r_1) \wedge s_1 \leqslant r_1 \leqslant r_2 \wedge r_1 \neq r_2 \longrightarrow A_1(s, r_1))))$

В этом определении s и s_1 являются состояниями изменений, в которых выполняются инвариант цикла управления и экземпляр шаблона, соответственно, t является с-параметром, а A_1 и A_2 являются fm-параметрами. Неравенство $n(s_1, s) \geqslant t$ в посылке необходимо, поскольку если время t не прошло с начала отсчета времени в состоянии s_1, то событие A_2 может не произойти до конечного состояния s. Это определение утверждает, что между состояниями s_1 и s существует внешнее состояние r_2 такое, что время, прошедшее от состояния s_1 до состояния r_2, не больше, чем t, и A_2 выполняется в состоянии r_2. Более того, для каждого внешнего состояния r_1 между состояниями s_1 и r_2, включая s_1, но исключая r_2, выполняется условие A_1.

Шаблон дополнительных инвариантов FIP1, соответствующий базовому шаблону требований FRP1, определяется следующим образом:

$FIP1(s, s_1, t, t_1, A'_1, A'_2)$
$(\exists r_2.e(r_2) \wedge s_1 \leqslant r_2 \wedge r_2 \leqslant s \wedge n(s_1, r_2) \leqslant t \wedge A'_2(s, r_2) \wedge (\forall r_1.e(r_1) \wedge s_1 \leqslant r_1 \wedge r_1 < r_2 \longrightarrow A'_1(s, r_1))) \vee$
$n(s_1, s) < t_1(s) \wedge (\forall r_1.e(r_1) \wedge s_1 \leqslant r_1 \wedge r_1 \leqslant s \longrightarrow A'_1(s, r_1))$

Экземпляр этого шаблона дополнительных инвариантов утверждает, что соответствующий экземпляр шаблона требований выполнен, но он также дополнительно утверждает, что максимальное время ожидания события A_2 в состоянии s равно $t_1(s)$. В этом определении s и s_1 являются состояниями изменений, в которых выполняются инвариант цикла управления и экземпляр шаблона, соответственно, t является с-параметром, его значение в экземпляре шаблона FIP1 равно значению с-параметра t в соответствующем экземпляре шаблона FRP1, t_1 является дополнительным fn-параметром, зависящим от s, A_1 и A_2 являются fm-параметрами, их значения в общем случае не равны, но связаны со значениями fm-параметров A_1 и A_2 в соответствующем экземпляре шаблона FRP1. Это определение утверждает, что либо существует внешнее состояние r_2 между состояниями s_1 и s такое, что время, прошедшее от состояния s_1 до состояния r_2, не превышает t, A_2 выполняется в состоянии r_2 и для каждого внешнего состояния r_1 между состояниями s_1 и r_2, включая s_1, но исключая r_2, выполняется условие A_1, либо время $t_1(s)$ не прошло после начала отсчета времени в состоянии s_1 и для каждого внешнего состояния r_1 между s_1 и s, условие A_1 выполняется в r_1.

Условия в леммах для этих шаблонов следующие:

$cond_{true} \equiv False$; $cond_{LS_1} \equiv t_1(s) = 0 \vee A'_2(s', s') \wedge t_1(s) \leqslant t \vee A'_1(s', s') \wedge t_1(s) < t_1(s')$;
$cond_{LS_2} \equiv A'_2(s, s) \vee A'_1(s, s) \wedge t_1(s) > 0$; $cond_{LS_3} \equiv t_1 \leqslant t$

Условие $cond_{true} = False$ указывает на то, что не существует такого значения t, при котором шаблоны требований и дополнительных инвариантов тождественно истинны для всех A_1, A_2. Условие $cond_{LS_1}$ утверждает, что выполняется по крайней мере одно из следующих условий: 1) значение дополнительного fn-параметра t_1 в инварианте перед итерацией цикла равно нулю, что означает, что событие A'_2 произошло не позднее, чем когда программа находилась в состоянии изменений s, 2) событие A'_2 произошло в состоянии изменений s' и время ожидания события A'_2 не превышено, или 3) условие A'_1 выполняется в состоянии изменений s' и время ожидания увеличилось не менее чем на 1. Во второй лемме условие $cond_{LS_2}$ утверждает, что произошло событие A'_2 или условие A'_1 истинно и событие A'_2 ожидается в будущем. В третьей лемме условие $cond_{LS_3}$ утверждает, что максимальное время ожидания не превышает значения с-параметра t.

2.1.2. Шаблон FRP2

Второй шаблон требований будущего FRP2, определенный ниже, утверждает, что условие A_1 должно выполняться по крайней мере, пока не будет достигнуто время t.

$FRP2(s, s_1, t, A_1) \equiv \forall r_1.e(r_1) \wedge s_1 \leqslant r_1 \leqslant s \wedge n(s_1, r_1) < t \longrightarrow A_1(s, r_1)$

Это определение утверждает, что во всех состояниях изменений r_1 между состояниями s_1 и s таких, что время между состояниями s_1 и r_1 меньше t итераций, выполняется условие A_1.

Шаблон дополнительных инвариантов FIP2, соответствующий базовому шаблону требований FRP2, определяется следующим образом:

$FIP2(s, s_1, t, t_1, A'_1) \equiv t = 0 \vee n(s_1, s) \geqslant t_1(s) \wedge (\forall r_1.e(r_1) \wedge s_1 \leqslant r_1 \leqslant s \wedge n(s_1, r_1) < t \longrightarrow A'_1(s, r_1))$

Экземпляр этого шаблона дополнительных инвариантов утверждает, что, если $t \neq 0$ (при $t = 0$ шаблон требований и шаблон дополнительных инвариантов истинны при любых значениях других параметров), то выполняется соответствующий экземпляр шаблона требований, то есть во всех внешних состояниях изменений r_1 между состояниями s_1 и s таких, что время между состояниями s_1 и r_1 меньше t, выполняется условие A_1, и от начала отсчета времени в состоянии s_1 до текущего момента (состояния s) прошло время по крайней мере $t_1(s)$, то есть fn-параметр t_1 определяет минимальное время, в течение которого выполнялось условие A'_1.

Условия в леммах для этих шаблонов следующие:

$cond_{true} \equiv t = 0$; $cond_{LS_1} \equiv (t_1(s) + 1 \geqslant t \vee A'_1(s's') \wedge t_1(s') \leqslant t_1(s) + 1$; $cond_{LS_2} \equiv t = 0 \vee t_1(s) = 0 \wedge (A'_1(s, s))$;
$cond_{LS_3} \equiv True$

Условие $cond_{true} \equiv t = 0$ утверждает, что при $t = 0$ шаблоны FRP2 и FIP2 истинны при любых значениях других параметров. Условие $cond_{LS_1}$ утверждает, что значение fn-параметра t_1 перед итерацией цикла не менее t или меньше t на 1, что означает, что условие A'_1 выполнялось по крайней мере в течение времени $t - 1$, или в конце итерации выполняется условие A'_1, то есть, если еще не прошло время t, в течение которого должно выполняться A_1, A_1 продолжает выполняться. Кроме того, условие $cond_{LS_1}$ утверждает, что значение fn-параметра t_1 на каждой итерации увеличивается не более, чем на 1. Условие $cond_{LS_2}$ утверждает, что либо $t = 0$, что означает, что экземпляр шаблона дополнительных инвариантов истинен при любых значениях других параметров, либо $t_1(s) = 0$, что означает, что начинается отсчет времени, и выполняется условие A_1. Условие $cond_{LS_3} = True$ утверждает, что при любом значении дополнительного fn-параметра t_1 из истинности экземпляра шаблона FIP2 следует истинность соответствующего экземпляра шаблона FRP2.

2.1.3. Шаблон FRP3

Третий шаблон требований будущего FRP3, определенный ниже, утверждает, что условие A_1 выполняется, пока не произойдет событие A_2.

$$FRP3(s, s_1, A_1, A_2) \equiv$$
$$(\exists r_2.e(r_2) \wedge s_1 \leqslant r_2 \leqslant s \wedge A_2(s, r_2) \wedge (\forall r_1.e(r_1) \wedge s_1 \leqslant r_1 < r_2 \longrightarrow A_1(s, r_1))) \vee (\forall r_1.e(r_1) \wedge s_1 \leqslant r_1 \leqslant s \longrightarrow A_1(s, r_1))$$

Это определение утверждает, что либо существует внешнее состояние изменений r_2 между состояниями s_1 и s, в котором выполняется условие A_2, и при этом во всех внешних состояниях r_1 между s_1 и r_2, включая s_1, но не включая r_2, выполняется условие A_1, либо условие A_1 выполняется во всех внешних состояниях r_1 между s_1 и s.

Шаблон дополнительных инвариантов FIP3, соответствующий базовому шаблону требований FRP3, определяется следующим образом:

$$FIP3(s, s_1, w, A'_1, A'_2) \equiv$$
$$(\exists r_2.e(r_2) \wedge s_1 \leqslant r_2 \leqslant s \wedge A'_2(s, r_2) \wedge (\forall r_1.e(r_1) \wedge s_1 \leqslant r_1 < r_2 \longrightarrow A'_1(s, r_1))) \vee w(s) \wedge (\forall r_1.e(r_1) \wedge s_1 \leqslant r_1 \leqslant s \longrightarrow A'_1(s, r_1))$$

Экземпляр шаблона дополнительных инвариантов утверждает, что выполняется соответствующий экземпляр шаблона требований, а также, если значение дополнительного fn-параметра $w(s)$ равно *False*, то событие A_2 должно произойти до текущего состояния s. Это определение, как и определение шаблона требований FRP3, имеет вид диъюнкции, но в этом определении ко второму дизъюнкту добавлен дополнительный fn-параметр $w(s)$ в качестве конъюнкта. Если формула $w(s)$ истинна, то экземпляр шаблона FIP3 эквивалентен соответствующему экземпляру шаблона FRP3, если $A'_1 = A_1$ и $A'_2 = A_2$. Если формула $w(s)$ ложна, то второй дизъюнкт ложен, и, следовательно, выполняется первый, то есть существует внешнее состояние изменений r_2 между состояниями s_1 и s, в котором выполняется условие A_2, и при этом во всех внешних состояниях r_1 между s_1 и r_2, включая s_1, но не включая r_2, выполняется условие A_1.

Условия в леммах для этих шаблонов следующие:

$$cond_{true} \equiv False; \quad cond_{LS_1} \equiv \neg w(s) \vee A'_2(s', s') \vee w(s') \wedge A'_1(s', s'); \quad cond_{LS_2} \equiv A'_2(s, s) \vee w(s) \wedge A'_1(s, s); \quad cond_{LS_3} \equiv True$$

Условие $cond_{LS_1}$ утверждает, что либо формула $w(s)$ ложна, что означает, что событие A'_2 произошло до текущей итерации, либо в конце текущей итерации произошло A'_2, либо выполняется $w(s')$, что означает, что A'_2 не обязательно должно произойти до окончания текущей итерации, и в конце текущей итерации выполняется A'_1. Условие $cond_{LS_2}$ утверждает, что либо произошло событие A'_2, либо формула $w(s)$ истинна, что означает, что A'_2 не обязательно должно произойти в состоянии s, и при этом выполняется условие A'_1. Условие $cond_{LS_3} = True$ утверждает, что при любом значении fn-параметра w из истинности экземпляра шаблона дополнительных инвариантов следует истинность соответствующего экземпляра шаблона требований.

2.1.4. Шаблон FRP4

Четвертый шаблон требований будущего, определенный ниже, утверждает, что условие A_1 выполняется по крайней мере в течение времени t или пока не произойдет событие A_2.

$$FRP4(s, s_1, t, A_1, A_2) \equiv$$
$$(\exists r_2.e(r_2) \wedge s_1 \leqslant r_2 \leqslant s \wedge n(s_1, r_2) \leqslant t \wedge A_2(s, r_2) \wedge (\forall r_1.e(r_1) \wedge s_1 \leqslant r_1 < r_2 \longrightarrow A_1(s, r_1))) \vee$$
$$(\forall r_1.e(r_1) \wedge s_1 \leqslant r_1 \leqslant s \wedge n(s_1, r_1) \leqslant t \longrightarrow A_1(s, r_1))$$

Это определение утверждает, что либо существует внешнее состояние r_2 между состояниями s_1 и s такое, что время между состояниями s_1 и r_2 не превышает t итераций, и в состоянии r_2 произошло событие A_2, и во всех внешних состояниях r_1 между s_1 и r_2, включая s_1, но не включая r_2, выполняется условие A_1, либо A_1 выполняется во всех внешних состояниях r_1 между s_1 и s таких, что время между состояниями s_1 и r_1 не превосходит t итераций.

Шаблон дополнительных инвариантов FIP4, соответствующий базовому шаблону требований FRP4, определяется следующим образом:

$FIP4(s, s_1, t, t_1, A_1, A_2) \equiv$
$(\exists r_2.e(r_2) \wedge s_1 \leqslant r_2 \leqslant s \wedge n(s_1, r_2) \leqslant t \wedge A_2(s, r_2) \wedge (\forall r_1.e(r_1) \wedge s_1 \leqslant r_1 < r_2 \longrightarrow A_1(s, r_1)))\vee$
$n(s_1, s) \geqslant t_1(s) \wedge (\forall r_1.e(r_1) \wedge s_1 \leqslant r_1 \leqslant s \wedge n(s_1, r_1) \leqslant t \longrightarrow A_1(s, r_1))$

Экземпляр шаблона FIP4 утверждает, что выполняется соответствующий экземпляр шаблона FRP4, а также, если от начала отсчета времени в состоянии s_1 до состояния s прошло время, меньшее, чем t_1, то не позднее, чем через время t, отсчитываемое от состояния s_1, происходит событие A_2. Следовательно, если от начала отсчета событие A_2 не произошло, то время между состояниями s_1 и s составляет по крайней мере $t_1(s)$ итераций. Если событие A_2 произошло, то в качестве значения fn-параметра $t_1(s)$ можно взять значение t. Таким образом, значение fn-параметра t_1 показывает минимальное время, в течение которого выполнялось условие A_1, если событие A_2 не произошло, и равно t, если событие A_2 произошло.

Условия в леммах для этих шаблонов следующие:

$cond_{true} \equiv False$;
$cond_{LS_1} \equiv t_1(s) < t \wedge (A_2(s', s') \wedge t_1(s') \leqslant t + 1 \vee t_1(s') \leqslant t_1(s) + 1 \wedge A_1(s', s')) \vee t_1(s) \geqslant t \wedge t_1(s') \leqslant t_1(s) + 1)$;
$cond_{LS_2} \equiv A_2(s, s) \vee t_1(s) = 0 \wedge A_1(s, s)$; $cond_{LS_3} \equiv True$

Условие $cond_{LS_1}$ утверждает следующее. Если $t_1(s) < t$, что означает, что событие A_2 не произошло до начала текущей итерации, то либо в конце итерации произошло событие A_2, и в этом случае значение fn-параметра t_1 превышает значение c-параметра t не более, чем на 1, либо в конце итерации выполняется условие A_1, и в этом случае значение fn-параметра t_1 увеличивается в течение итерации не более, чем на 1. Если $t_1(s) \geqslant t$, что означает, что событие A_2 произошло до начала текущей итерации, или условие A_1 выполнялось по крайней мере в течение времени t, то значение fn-параметра t_1 в течение итерации увеличивается не более, чем на 1.

2.2. Шаблоны прошлого

Определение 4. (Шаблон дополнительных инвариантов прошлого)
Шаблон дополнительных инвариантов прошлого — шаблон дополнительных инвариантов, определение которого имеет следующий вид:

$$I \equiv \lambda s.\lambda(p_1, \ldots, p_m, ep_1, \ldots, ep_k, A'_1, \ldots, A'_n).I'(s, p_1, \ldots, p_m, ep_1(s), \ldots, ep_k(s), A'_1(s, r_{j_1}), \ldots, A'_n(s, r_{j_n}))\,,$$

где r_{j_i} $(i = 1, \ldots, n)$ — связанные кванторами в I' переменные, обозначающие состояния изменений, такие, что $r_{j_i} \leqslant s$, остальные параметры имеют то же значение, что и в шаблонах будущего.

Леммы для любой пары, состоящей из базового шаблона требований прошлого и соответствующего базового шаблона дополнительных инвариантов, должны удовлетворять следующим схемам:

1. LS_4:

   $consecutive(s, s') \longrightarrow$
   $(\forall s_1.e(s_1) \wedge s_1 \leqslant s \wedge A'_1(s, s_1) \longrightarrow A'_1(s', s_1)) \wedge \ldots \wedge (\forall s_1.e(s_1) \wedge s_1 \leqslant s \wedge A'_n(s, s_1) \longrightarrow A'_n(s', s_1)) \longrightarrow$
   $(\forall s_1.e(s_1) \wedge s_1 \leqslant s \wedge R(s, s_1, p_1, \ldots, p_m, A'_1, \ldots, A'_n) \longrightarrow R(s', s_1, p_1, \ldots, p_m, A'_1, \ldots, A'_n)$

   Леммы, удовлетворяющие этой схеме, используются для доказательства лемм, применяемых для доказательства условий корректности для дополнительных инвариантов. Посылка этой леммы утверждает, что для каждого состояния изменений вплоть до s, если формула $A'_i(s)$ $(i = 1, \ldots, n)$ выполняется, то $A'_i(s')$ выполняется в том же состоянии, Заключение леммы утверждает, что если подформула дополнительного инварианта, выполняющегося в состоянии s, была выполнена, то подформула дополнительного инварианта, выполняющегося в состоянии s', выполняется в том же состоянии. Заметим, что формула в заключении может рассматриваться как формула посылки, если вместо $A'_i(s)$ используется $R(s, s_1, p_1, \ldots, p_m, A'_1, \ldots, A'_n)$, а вместо $A'_i(s', s_1)$ используется $R(s', s_1, p_1, \ldots, p_m, A'_1, \ldots, A'_n)$. Это позволяет определять и доказывать леммы для производных шаблонов индукцией по построению этого шаблона.

2. LS_5:

$$e(s) \longrightarrow$$
$$(\forall s_1.e(s_1) \wedge s_1 \leqslant s \wedge A'_1(s, s_1) \longrightarrow A_1(s, s_1)) \wedge \ldots \wedge (\forall s_1.e(s_1) \wedge s_1 \leqslant s \wedge A'_n(s, s_1) \longrightarrow A_n(s, s_1)) \longrightarrow$$
$$(\forall s_1.e(s_1) \wedge s_1 \leqslant s \wedge R(s, s_1, p_1, \ldots, p_m, A'_1, \ldots, A'_n) \longrightarrow R(s, s_1, p_1, \ldots, p_m, A_1, \ldots, A_n)$$

Эта схема похожа на предыдущую, но леммы, удовлетворяющие этой схеме, используются для доказательства лемм, применяемых для доказательства условий корректности для требований. Посылка этой леммы утверждает, что для каждого состояния изменений вплоть до s, если формула $A'_i(s)$ $(i = 1, \ldots, n)$ выполняется, то $A_i(s)$ выполняется в том же состоянии.

3. LS_6:

$$consecutive(s, s') \longrightarrow$$
$$cond(p_1, \ldots, p_m, ep_1(s), \ldots, ep_k(s), ep_1(s'), \ldots, ep_k(s'), A_1(s', s'), \ldots, A_n(s', s'), I(s, p_1, \ldots, p_m, ep_1, ep_k, A'_1, \ldots, A'_n),$$
$$(\forall s_1.e(s_1) \wedge s_1 \leqslant s \wedge A'_1(s, s_1) \longrightarrow A'_1(s', s_1)), \ldots, (\forall s_1.e(s_1) \wedge s_1 \leqslant s \wedge A'_n(s, s_1) \longrightarrow A'_n(s', s_1))) \longrightarrow$$
$$I(s', p_1, \ldots, p_m, ep_1, \ldots, ep_k, A'_1, \ldots, A'_n)$$

Леммы, удовлетворяющие этой схеме, используются для доказательства дополнительного конъюнкта дополнительного инварианта в доказательствах лемм, применяемых для доказательства условий корректности для дополнительных инвариантов. Посылка этой леммы утверждает, что выполняется некоторое условие *cond*, связывающее $A_1(s', s')$,..., $A_n(s', s')$, значения с-параметров p_1,..., p_m, значения fn-параметров в состояниях s и s' $ep_1(s)$,..., $ep_k(s)$, $ep_1(s')$,..., $ep_k(s')$, экземпляр шаблона дополнительных инвариантов, выполняющийся в начале итерации в состоянии s, а также формулы $(\forall s_1.e(s_1) \wedge s_1 \leqslant s \wedge A'_i(s, s_1) \longrightarrow A'_i(s', s_1))$, $i = 1, \ldots, n$, утверждающие, что если $A_i(s)$ выполняется в состоянии s_1, то формула $A'_i(s')$ также выполняется в состоянии s_1. Заключение леммы — экземпляр шаблона дополнительных инвариантов, который выполняется в конце итерации в состоянии s'.

4. LS_7:

$$consecutive(s, s') \longrightarrow$$
$$cond(p_1, \ldots, p_m, ep_1(s), \ldots, ep_k(s), ep_1(s'), \ldots, ep_k(s'), A'_1(s', s'), \ldots, A'_n(s', s'), I(s, p_1, \ldots, p_m, ep_1, \ldots, ep_k, A'_1, \ldots, A'_n),$$
$$(\forall s_1.e(s_1) \wedge s_1 \leqslant s \wedge A'_1(s, s_1) \longrightarrow A'_1(s', s_1)), \ldots, (\forall s_1.e(s_1) \wedge s_1 \leqslant s \wedge A'_n(s, s_1) \longrightarrow A'_n(s', s_1))) \longrightarrow$$
$$R(s', s', p_1, \ldots, p_m, A'_1, \ldots, A'_n)$$

Леммы, удовлетворяющие этой схеме, используются для доказательства лемм, применяемых для доказательства условий корректности для дополнительных инвариантов. Посылки леммы утверждают, что выполняется условие *cond*, связывающее $A'_1(s', s')$, ..., $A'_n(s', s')$, значения с-параметров $p_1, \ldots, p_m$, значения fn-параметров в состояниях s и s' $ep_1(s), \ldots, ep_k(s)$, $ep_1(s'), \ldots, ep_k(s')$, экземпляр шаблона дополнительных инвариантов, выполняющийся в начале итерации, а также формулы $(\forall s_1.e(s_1) \wedge s_1 \leqslant s \wedge A'_i(s, s_1) \longrightarrow A'_i(s', s_1))$, $i = 1, \ldots, n$, утверждающие, что если $A_i(s)$ выполняется в состоянии s_1, то формула $A'_i(s')$ также выполняется в состоянии s_1. Заключение леммы утверждает, что в конце итерации выполняется подформула дополнительного инварианта $R(s', s', p_1, \ldots, p_m, A'_1, \ldots, A'_n)$, являющаяся экземпляром шаблона требований прошлого.

Далее рассмотрим конкретные шаблоны требований прошлого и соответствующие им базовые шаблоны дополнительных инвариантов и леммы.

2.2.1. Шаблон PRP1

Шаблон требований прошлого PRP1, определенный ниже, утверждает, что условие A_1 всегда должно быть истинным между итерациями цикла управления вплоть до текущего состояния s_1.

$$PRP1(s, s_1, A_1) \equiv \forall r_1.e(r_1) \wedge r_1 \leqslant s_1 \longrightarrow A_1(s, r_1)$$

В этом определении s и s_1 являются состояниями изменений, в которых выполняются инвариант цикла управления и экземпляр шаблона, соответственно, A_1 является fm-параметром. Это опреде-

ление утверждает, что условие A_1 выполняется в каждом внешнем состоянии изменений вплоть до состояния s_1.

Базовый шаблон дополнительных инвариантов PIP1, соответствующий базовому шаблону требований PRP1, совпадает с PRP1 при $s_1 = s$, т. е. *BIP2* определяется следующим образом:

$$PIP1(s, A'_1) \equiv \forall r_1.e(r_1) \wedge r_1 \leqslant s \longrightarrow A'_1(s, r_1)$$

Этот шаблон параметризован одним состоянием изменений s, поскольку экземпляр этого шаблона является дополнительным инвариантом, и он выполняется в состоянии s. Значение fm-параметра A_1 в шаблоне PIP1 не равно в общем случае, но связано со значением fm-параметра A_1 в соответствующем экземпляре шаблона PRP1.

Условия в леммах для этих шаблонов следующие:

$$cond_{LS_6} \equiv (\forall s_1.e(s_1) \wedge A'_1(s, s_1) \longrightarrow A'_1(s', s_1)) \wedge A'_1(s', s'); \quad cond_{LS_7} \equiv (\forall s_1.e(s_1) \wedge A'_1(s, s_1) \longrightarrow A'_1(s, s_1)) \wedge A'_1(s', s')$$

Условия $cond_{LS_6}$ и $cond_{LS_7}$ совпадают, так как при $s_1 = s$ совпадают определения шаблонов PRP1 и PRP2. Они утверждают, что, если будущее ограничено состоянием s и в некотором состоянии s_1 выполняется fm-параметр A'_1, то, если будущее ограничено состоянием s', A'_1 также выполняется в том же состоянии s_1. Кроме того A'_1 выполняется в состоянии s'.

2.2.2. Шаблон PRP2

Шаблон требований прошлого PRP2, определенный ниже, утверждает, что существует предыдущее внешнее состояние изменений, и в этом состоянии выполнялось условие A_1. Предыдущее внешнее состояние не существует, если состояние s_1, в котором выполняется экземпляр шаблона, соответствует точке входа в цикл управления.

$$PRP2(s, s_1, A_1) \equiv \exists r_1.consecutive(r_1, s_1) \wedge A(s, r_1)$$

Шаблон дополнительных инвариантов PIP2, соответствующий базовому шаблону требований PRP2, определяется следующим образом:

$$PIP2(s, A_1) \equiv A_1(s, s)$$

Экземпляр шаблона утверждает, что в состоянии s выполняется условие A_1. То есть вспомогательным утверждением для доказательства экземпляра шаблона PRP2 является его fm-параметр A_1.

Условия в леммах для этих шаблонов следующие:

$$cond_{LS_6} \equiv A'_1(s', s'); \quad cond_{LS_7} \equiv PIP2(s, A'_1)$$

Условие $cond_{LS_6}$ утверждает, что в конце итерации в состоянии s' выполняется условие A_1. Значение $cond_{LS_6}$ совпадает с определением шаблона PIP2, выполняющимся в состоянии s' в конце итерации. Условие $cond_{LS_7}$ утверждает, что в состоянии s выполняется экземпляр шаблона PIP2, что означает, что в начале итерации в состоянии s должно выполняться условие A_1. Это гарантирует, что в конце итерации выполняется экземпляр шаблона PRP2.

2.2.3. Шаблон PRP3

Шаблон требований прошлого PRP3, определенный ниже, утверждает, что, если предыдущее внешнее состояние существует, то в нем выполняется условие A_1.

$$PRP3(s, s_1, A_1) \equiv \forall r_1.consecutive(r_1, s_1) \longrightarrow A_1(s, r_1)$$

Если состояние s_1 соответствует моменту времени, до которого была выполнена по крайней мере одна итерации цикла управления, экземпляр этого шаблона эквивалентен экземпляру шаблона PRP2 с теми же значениями параметров. Если состояние s_1 соответствует точке входа в цикл управления, то экземпляр шаблона PRP3 является истинным, в то время как экземпляр шаблона PRP2 является ложным. Заметим, что шаблон PRP3 является двойственным к шаблону PRP2, то есть

для всех s, s_1 и A_1 выполняется равенство

$\neg PRP2(s, s_1, A_1) = PRP3(s, s_1, (\lambda\ s\ r_1.\neg A_1(s, r_1)))$.

Шаблон дополнительных инвариантов PIP3, соответствующий шаблону требований PRP3 совпадает с шаблоном дополнительных инвариантов PIP2, соответствующим шаблону требований PRP2. Условия $cond_{LS_6}$ и $cond_{LS_7}$ в леммах для шаблона PRP3 также совпадают с соответствующими условиями в леммах для шаблона PRP2.

2.2.4. Шаблоны PRP4 и PRP5

Шаблон требований прошлого PRP4, определенный ниже, утверждает, что в прошлом произошло событие A_2, причем от наступления A_2 прошло как минимум время t, и после наступления A_2 и до текущего момента выполняется A_1.

$PRP4(s, s_1, t, A_1, A_2) \equiv \exists r_1.e(r_1) \wedge r_1 \leqslant s_1 \wedge n(r_1, s_1) \geqslant t \wedge A_2(s, r_1) \wedge (\forall r_2.e(r_2) \wedge r_1 < r_2 \leqslant s_1 \longrightarrow A_1(s, r_2))$

Это определение утверждает, что существует внешнее состояние r_1, которое было до состояния s_1 или совпадает с s_1, такое, что время между r_1 и s_1 составляет не менее t итераций, и в состоянии r_1 выполняется A_2, и при этом во всех состояниях r_2 между r_1 и s_1, не включая r_1, но включая s_1, выполняется A_1.

При задании требований из нашей коллекции этот шаблон используется в посылках импликаций. Для приведения таких требований к положительной нормальной форме необходимо определить двойственный шаблон для шаблона PRP4. Шаблон PRP5, являющийся двойственным для шаблона PRP4, определяется следующим образом:

$PRP5(s, s_1, t, A_1, A_2) \equiv \forall r_1.e(r_1) \wedge r_1 \leqslant s_1 \wedge n(r_1, s_1) \geqslant t \longrightarrow A_2(s, r_1) \vee (\exists r_2.e(r_2) \wedge r_1 < r_2 \leqslant s_1 \wedge A_1(s, r_2))$

Этот шаблон утверждает, что, если в некоторый момент времени в прошлом в состоянии r_1, после которого прошло по крайней мере время t, не выполнялось условие A_2, то после r_1 до текущего момента (состояния s_1), не включая r_1, но включая текущий момент, должно произойти событие A_1.

Шаблон дополнительных инвариантов PIP5, соответствующий базовому шаблону требований PRP5, определяется следующим образом:

$PIP5(s, t, t_1, A_1, A_2) \equiv \forall r_1.e(r_1) \wedge r_1 \leqslant s \wedge n(r_1, s) \geqslant t_1(s) \longrightarrow A_2(s, r_1) \vee (\exists r_2.e(r_2) \wedge r_1 < r_2 \leqslant s \wedge A_1(s, r_2)) \equiv PRP5(s, s, t_1(s), A_1, A_2)$

Экземпляр шаблона утверждает, что, если в некоторый момент времени в прошлом в состоянии r_1, после которого прошло по крайней мере время $t_1(s)$, не выполнялось условие A_2, то после r_1 до текущего момента (состояния s_1), не включая r_1, но включая текущий момент, должно произойти событие A_1. Значение fn-параметра t_1 определяет максимальное время ожидания события A_1 после того, как не выполнилось условие A_2. В состоянии s экземпляр шаблона FIP5 эквивалентен соответствующему экземпляру шаблона FRP5, в котором константа t заменена термом $t_1(s)$. Параметр шаблона FRP5 t не используется в определении шаблона FIP5.

Условия в леммах для этих шаблонов следующие:

$cond_{LS_6} \equiv (t_1(s') > 0 \vee A'_2(s', s')) \wedge (A'_1(s', s') \vee PIP5(s, t, t_1, A'_1, A'_2) \wedge (\forall s_1.e(s_1) \wedge s_1 \leqslant s \wedge A'_1(s, s_1) \longrightarrow A'_1(s', s_1)) \wedge$
$(\forall s_1.e(s_1) \wedge s_1 \leqslant s \wedge A'_2(s, s_1) \longrightarrow A'_2(s', s_1)) \wedge t_1(s) < t_1(s'))$;
$cond_{LS_7} \equiv (t > 0 \vee A'_2(s', s')) \wedge (PIP5(s, t, t_1, A'_1, A'_2) \wedge (\forall s_1.e(s_1) \wedge s_1 \leqslant s \wedge A'_1(s, s_1) \longrightarrow A'_1(s', s_1)) \wedge$
$(\forall s_1.e(s_1) \wedge s_1 \leqslant s \wedge A'_2(s, s_1) \longrightarrow A'_2(s', s_1)) \wedge t_1(s) < t \vee A'_1(s', s'))$

Условие $cond_{LS_6}$ утверждает, что выполняются следующие два условия: 1) если значение fn-параметра t_1 в конце итерации равно 0, то в конце итерации выполняется условие A'_2, 2) Либо в конце итерации произошло событие A'_1, либо а) в начале итерации выполнялся экземпляр шаблона дополнительных инвариантов, б) для fm-параметров A'_i ($i \in \{1; 2\}$) выполняется следующее условие: если будущее было ограничено состоянием s и в некотором состоянии s_1 выполнялось условие A'_i, то, если будущее ограничено состоянием s', то A'_i также выполняется в том же состоянии s_1, и в) время ожидания события A'_1 увеличилось по крайней мере на 1. Условие $cond_{LS_7}$ почти совпадает с условием

$cond_{LS_6}$, но условие 1) утверждает, что условие A'_2 выполняется в конце итерации, если значение с-параметра t равно 0, а условие 2в) утверждает, что время ожидания t_1 в начале итерации меньше t.

3. Производные шаблоны требований и дополнительных инвариантов

В этом разделе описываются схемы производных шаблонов дополнительных инвариантов, которые строятся по соответствующим производным шаблонам требований, а также схемы лемм для этих пар шаблонов.

Определение 5. (Производный шаблон дополнительных инвариантов)
Производный шаблон дополнительных инвариантов — шаблон дополнительных инвариантов, определение которого имеет следующий вид:

$$I \equiv \lambda s.\lambda(p_1, \ldots, p_m, ep_1, \ldots, ep_k, b_1, \ldots, b_l, A_1, \ldots, A_u, B_1, \ldots, B_v.\\ I'(s, p_1, \ldots, p_m, ep_1(s), \ldots, ep_k(s), b_1(s), \ldots, b_l(s), A_1, \ldots, A_u, B_1(s), \ldots, B_v(s)$$

где s — состояние изменений, в котором дополнительный инвариант должен быть истинным; p_1, $\ldots$, p_m — с-параметры, значения которых в дополнительном инварианте совпадают со значениями соответствующих параметров в требовании; $b_1, \ldots, b_l$ — fn-параметры, введенные для экземпляров шаблонов прошлого, значениями этих параметров являются функции, принимающие в качестве аргумента состояние изменений и возвращающие логическое значение, показывающее, должен ли соответствующий экземпляр шаблона дополнительных инвариантов прошлого выполняться в этом состоянии; $ep_1, \ldots, ep_k$ — fn-параметры, значения которых совпадают со значениями fn-параметров базовых шаблонов дополнительных инвариантов, включенных в I'; $B_1, \ldots, B_v$ — fm-параметры, значениями которых являются формулы из множества F_e, определяющие вспомогательные утверждения, необходимые для доказательства формул, являющихся, соответственно, значениями параметров $B_1,\ldots, B_v$ шаблона требований; $A_1, \ldots, A_u$ — fm-параметры, значения которых совпадают со значениями соответствующих параметров шаблона требований; I' — параметризованная формула, имеющая вид $P(s, p_1, \ldots, p_{m'}, ep_1, \ldots, ep_{k'}, A_1, \ldots, A_{n'}) \wedge (b(s)_1 \longrightarrow I_1) \wedge \ldots \wedge (b_l(s) \longrightarrow I_l)$, где P — производный шаблон дополнительных инвариантов, значения fm-параметров которого содержат параметризованные экземпляры шаблона требований прошлого $R_1, \ldots, R_l$, $I_1, \ldots, I_l$ — параметризованные дополнительные инварианты, соответствующие $R_1, \ldots, R_l$, соответственно.

Первый конъюнкт $P(s, p_1, \ldots, p_{m'}, ep_1, \ldots, ep_{k'}, A_1, \ldots, A_{n'})$ определяет дополнительный инвариант, необходимый для доказательства требования. Остальные конъюнкты являются дополнительными конъюнктами. Инварианты I_i $(i = 1, \ldots, l)$ необходимы для доказательства формул о прошлом R_i, входящих в требование и в дополнительный инвариант. Но в некоторых состояниях первый конъюнкт I' или I_i может быть доказан даже если инвариант I_i не выполняется на предыдущей итерации. Поэтому дополнительные конъюнкты являются импликациями, посылки которых $b_i(s)$ определяют условия, при которых должны выполняться соответствующие инварианты I_i.

Для каждой пары, состоящей из производного шаблона требований и соответствующего шаблона дополнительных инвариантов определяются 4 леммы. Первые две леммы удовлетворяют следующим схемам:

1. LS_8:

$$I(s, p_1, \ldots, p_m, ep_1, \ldots, ep_k, b_1, \ldots, b_l, A_1, \ldots, A_u, B'_1, \ldots, B'_v) \longrightarrow\\ consecutive(s, s') \longrightarrow\\ cond(p_1, \ldots, p_m, ep_1(s), \ldots, ep_k(s), b_1(s), \ldots, b_l(s), ep_1(s'), \ldots, ep_k(s'), b_1(s'), \ldots, b_l(s'), A_1(s'), \ldots, A_u(s'), B'_1(s', s'), \ldots,\\ B'_v(s', s'),\\ (\forall s_1.e(s_1) \wedge s_1 \leqslant s \wedge B'_1(s, s_1) \longrightarrow B'_1(s', s_1)), \ldots, (\forall s_1.e(s_1) \wedge s_1 \leqslant s \wedge B'_v(s, s_1) \longrightarrow B'_v(s', s_1))) \longrightarrow\\ I(s', p_1, \ldots, p_m, ep_1, \ldots, ep_k, b_1, \ldots, b_l, A_1, \ldots, A_u, B'_1, \ldots, B'_v)$$

Посылки леммы, удовлетворяющей этой схеме, утверждают, что 1) дополнительный инвариант выполняется в начале итерации цикла управления в состоянии s, 2) выполняется

условие *cond*, связывающее $A'_i(s')$, $B'_i(s', s')$, значения с-параметров $p_1, \ldots, p_m$ и fn-параметров $ep_1(s), \ldots, ep_k(s)$, $b_1(s), \ldots, b_l(s)$, $ep_1(s'), \ldots, ep_k(s')$ и $b_1(s'), \ldots, b_l(s')$ и формулы вида $(\forall s_1.e(s_1) \wedge s_1 \leqslant s \wedge B'_1(s, s_1) \longrightarrow B'_1(s', s_1))$, утверждающие, что, если будущее ограничено состоянием s и B'_i выполняется в некотором состоянии s_1, то B'_i будет выполняться в том же состоянии s_1 при условии, что будущее ограничено состоянием s'. Формула $consecutive(s, s')$ в посылке 2 утверждает, что s и s' являются состояниями изменений до итерации и после итерации цикла, соответственно. Условие *cond* в посылке 3 не содержит формул с кванторами кроме формул вида $(\forall s_1.e(s_1) \wedge s_1 \leqslant s \wedge B'_i(s, s_1) \longrightarrow B'_i(s', s_1))$ и задает дополнительное условие, при котором инвариант выполняется в конце итерации, если он выполнялся в начале итерации. Заключение леммы утверждает, что дополнительный инвариант выполняется в конце итерации в состоянии s'.

2. LS_9:

$$
\begin{aligned}
&I(s, p_1, \ldots, p_m, ep_1, \ldots, ep_k, b_1, \ldots, b_l, A_1, \ldots, A_u, B'_1, \ldots, B'_v) \longrightarrow \\
&e(s) \longrightarrow \\
&cond(p_1, \ldots, p_m, ep_1(s), \ldots, ep_k(s), b_1(s), \ldots, b_l(s), (\forall s_1.e(s_1) \wedge s_1 \leqslant s \wedge B'_1(s, s_1) \longrightarrow B_1(s, s_1)), \ldots, \\
&\qquad (\forall s_1.e(s_1) \wedge s_1 \leqslant s \wedge B'_v(s, s_1) \longrightarrow B_v(s, s_1))) \longrightarrow \\
&R(s, p_1, \ldots, p_m, A_1, \ldots, A_u, B_1, \ldots, B_v)
\end{aligned}
$$

Посылки леммы, удовлетворяющей этой схеме, утверждают, что 1) дополнительный инвариант выполняется в состоянии s, 2) выполняется условие *cond*, связывающее значения с-параметров $p_1, \ldots, p_m$ и fn-параметров $ep_1(s), \ldots, ep_k(s)$ и $b_1(s), \ldots, b_l(s)$, а также формулы вида $(\forall s_1.e(s_1) \wedge s_1 \leqslant s \wedge B'_i(s, s_1) \longrightarrow B_i(s, s_1))$, утверждающие, что, если в некотором состоянии s_1 выполняется формула $B'_i(s)$, то в том же состоянии s_1 выполняется другая формула $B_i(s)$. Заключение леммы утверждает, что в состоянии s выполняется требование.

Назовем рассмотренные выше производные шаблоны общими шаблонами. Их можно использовать для определения новых производных шаблонов. Но в случае, когда fm-параметры шаблона не содержат вложенных шаблонов, леммы, удовлетворяющие схемам LS_8 и LS_9, можно упростить так, чтобы они не содержали формул с кванторами. Поэтому для каждого общего шаблона требований или дополнительных инвариантов можно определить соответствующий частный шаблон, fm-параметры которого не могут содержать вложенных шаблонов.

4. Построение шаблонов дополнительных инвариантов и лемм для производных шаблонов требований

В этом разделе описаны алгоритмы построения производных шаблонов дополнительных инвариантов по соответствующему производному шаблону требований, а также алгоритмы построения лемм для этих пар шаблонов.

Упрощенные леммы для частных шаблонов строятся из лемм для общих шаблонов, соответствующих схемам лемм LS_8 и LS_9 путем сопоставления каждого A'_i с соответствующим A_i $(i = 1, \ldots, n)$ и исключения посылок вида $\forall s_1.(e(s_1) \wedge s_1 \leqslant s \wedge A_i(s_1) \longrightarrow A_i(s_1))$. Леммы для общего шаблона требований R и соответствующего шаблона дополнительных инвариантов используются в доказательствах аналогичных лемм, определяемых для комбинаций шаблона требований R с другими шаблонами, а также в доказательствах лемм для соответствующих частных шаблонов. Леммы для частных шаблонов используются для доказательства условий корректности. Леммы для частных шаблонов могут быть доказаны в Isabelle/HOL автоматически с помощью метода доказательства `auto`. Для доказательства лемм для общих шаблонов мы определяем методы доказательства `proveOuter` и `prove`. Метод `proveOuter` используется для доказательства цели, которая является экземпляром шаблона или булевой комбинацией экземпляров шаблона, которая является подформулой заключения доказываемой леммы и не вложена в другие шаблоны. Метод `prove` используется для доказательства формул, полученных при доказательстве леммы после применения некоторой леммы.

Далее мы опишем алгоритм построения шаблона дополнительных инвариантов, генерации лемм, соответствующих схемам LS_8 и LS_9, а также методы `prove` и `proveOuter` для конъюнкции, дизъюнкции, вложения шаблона будущего и вложения шаблона прошлого.

4.1. Построение шаблонов дополнительных инвариантов

Определим функцию *invPattern*, которая принимает в качестве аргумента формулу R', являющуюся подформулой определения производного шаблона требований, и возвращает кортеж, первым элементом которого является подформула шаблона дополнительных инвариантов, соответствующая формуле R', а второй элемент является списком формул, которые добавляются к шаблону дополнительных инвариантов в качестве дополнительных конъюнктов. В псевдокоде данной функции конъюнкция является левоассоциативной, в то время как в Isabelle/HOL она является правоассоциативной. Это позволяет упростить доказательство, так как конъюнкция представляется в виде набора конъюнктов до того, как первый конъюнкт отделяется от дополнительных. Псевдокод функции *invPattern* представлен на листинге 1:

Листинг 1: Algorithm for generation of derived extra invariant patterns. Алгоритм генерации производных шаблонов дополнительных инвариантов

```
fun invPattern(R) = match(R) {
    boolean(op, A, B) → {
      (A', A'') = invPattern(A);
      (B', B'') = invPattern(B);
      (boolean(op, A', B'), A'' + B'');
    } |
    future(P, p_1, ..., p_m, A_1, ..., A_n) → {
      [(A'_1, A''_1),..., (A'_n, A''_n)] = map(invPattern, [A_1,..., A_n]);
      (basicInvPattern(P, p_1, ..., p_m, A'_1, ..., A'_n), A''_1 + ... + A''_n)
    } |
    past(P, p_1, ..., p_m, A_1, ..., A_n) → {
       [(A'_1, A''_1),..., (A'_n, A''_n)] = map(invPattern, [A_1,..., A_n]);
       R' = past(P, p_1, ..., p_m, A'_1, ..., A'_n);
       b = newParam();
       associate(R', b);
       I = basicInvPattern(P, p_1, ..., p_m, A'_1, ..., A'_n);
       associate (I, b);
      (R', A''_1 + ... + A''_n + [b ⟶ I])
    } |
    derived(P, p_1, ..., p_m, A_1, ..., A_u, B_1, ..., B_v) →
      [(B'_1, B''_1),..., (B'_v, B''_v)] = map(invPattern, [B_1,..., B_v]);
      (extended_inv(derivedInvPattern(P, p_1, ..., p_m, A_1, ..., A_u, B'_1, ..., B'_v),
        B''_1 + ... + B''_v), []) |
    A → (A, []) }
```

В алгоритме используется сопоставление с образцом. Проверяется соответствие формулы R' следующим типам формул:

1) $boolean(op, A, B)$ обозначает формулу $\lambda ss_1.A(s, s_1)\ op\ B(s, s_1)$, $\lambda s_1.A(s_1)\ op\ B(s_1)$ или $AopB$ в зависимости от типа формул A и B, где op — конъюнкция или дизъюнкция;
2) $future(P, p_1, \ldots, p_m, A_1, \ldots, A_n)$ — формула $\lambda s, s_1.P(s, s_1, p_1, \ldots, p_m, A_1, \ldots, A_n)$, где P — шаблон будущего;
3) $past(P, p_1, \ldots, p_m, A_1, \ldots, A_n)$ — формула $\lambda ss_1.P(s, s_1, p_1, \ldots, p_m, A_1, \ldots, A_n)$, где P — шаблон прошлого;
4) $derived(P, p_1, \ldots, p_m, A_1, \ldots, A_u, B_1, \ldots, B_v)$ — формула $\lambda s.P(s, p_1, \ldots, p_m, A_1, \ldots, A_u, B_1, \ldots, B_v)$, где P — производный шаблон.

5) *extended_inv*$(I, [I_1, \ldots, I_n])$ — формула $I \wedge I_1 \wedge \ldots \wedge I_n$, где I — экземпляр производного шаблона дополнительных инвариантов, $I_1, \ldots, I_n$ — дополнительные конъюнкты.

В определении функции *invPattern* используются функции *basicInvPattern*, *derivedInvPattern*, *associate* и *newParam*, а также операция конкатенации списков +. Функция *associate*(P, b) связывает с экземпляром P шаблона требований или дополнительных инвариантов прошлого логический параметр b, введенный для этого шаблона. Функция *basicInvPattern* принимает в качестве аргументов базовый шаблон требований и значения его параметров и возвращает соответствующий базовый шаблон дополнительных инвариантов, в который подставлены значения параметров базового шаблона требований, причем дополнительные параметры базового шаблона дополнительных инвариантов переименованы так, чтобы в производном шаблоне дополнительных инвариантов отсутствовали конфликты имен параметров. Функция *derivedInvPattern* аналогична функции *basicInvPattern*, но принимает в качестве аргумента производный шаблон требований и возвращает шаблон дополнительных инвариантов для него. Функция *newParam* принимает в качестве аргумента шаблон дополнительных инвариантов, создает и возвращает новый логический fn-параметр, который становится дополнительным параметром производного шаблона дополнительных инвариантов. Кроме того используется функция *map*(f, l), которая применяет функцию f к каждому элементу списка l и возвращает список результатов.

Если формула R' является атомарной формулой или fm-параметром, первым элементом кортежа является эта формула или fm-параметр. Атомарные формулы в требовании и в дополнительном инварианте совпадают. Вторым элементом кортежа является пустой список, так как для атомарных формул и fm-параметров не вводятся дополнительные конъюнкты.

Если формула R' имеет вид A *op* B, где *op* является конъюнкцией или дизъюнкцией, то первым элементом кортежа, возвращаемого функцией *invPattern* является формула, полученная из R' заменой подформул A и B первыми элементами кортежей, вычисленных функцией *invPattern* для A и B. То есть дополнительный инвариант для конъюнкции и дизъюнкции требований является конъюнкцией (дизъюнкцией, соответственно) соответствующих дополнительных инвариантов. Вторым элементом кортежа, возвращаемого функцией *invPattern*, является конкатенация вторых элементов кортежей, вычисленных функцией *invPattern* для A и B. То есть дополнительными конъюнктами дополнительного инварианта для булевой комбинации формул являются дополнительные конъюнкты, необходимые как для первой, так и для второй формулы.

Если R' является экземпляром шаблона будущего, то первым элементом кортежа, возвращаемого функцией *invPattern*, является экземпляр шаблона дополнительных инвариантов будущего, в котором значения fm-параметров являются первыми элементами кортежей, вычисленных функцией *invPattern* для значений fm-параметров базового шаблона требований. То есть шаблон требований будущего в дополнительном инварианте заменяется соответствующим шаблоном дополнительных инвариантов будущего, а значения параметров шаблона будущего преобразуются согласно тому же алгоритму рекурсивными вызовами функции *invPattern*. Вторым элементом кортежа, возвращаемого функцией *invPattern* для шаблона будущего, является конкатенация вторых элементов, вычисленных функцией *invPattern* для параметров шаблона будущего. То есть дополнительными конъюнктами дополнительного инварианта для шаблона будущего является все дополнительные конъюнкты, необходимые для каждого параметра шаблона.

Если R' является экземпляром шаблона требований прошлого, то первым элементом кортежа, возвращаемого функцией *invPattern*, является экземпляр того же шаблона требований прошлого, в котором значениями fm-параметров являются первые компоненты кортежей, вычисленных функцией *invPattern* для значений fm-параметров этого базового шаблона в определении производного шаблона требований. То есть шаблон требований прошлого в дополнительном инварианте не заменяется, но его параметры преобразуются в соответствии с алгоритмом рекурсивными вызовами

функции *invPattern*. Вторым элементом кортежа, возвращаемого функцией *invPattern*, является список, полученный в результате конкатенации вторых элементов кортежей, вычисленных функцией *invPattern* для параметров шаблона требований прошлого, и добавления импликации, посылкой которой является логический fn-параметр производного шаблона дополнительных инвариантов, а заключением — экземпляр соответствующего шаблона дополнительных инвариантов прошлого, значениями fm-параметров в котором являются первые элементы кортежей, вычисленных функцией *invPattern* для значений fm-параметров шаблона требований прошлого в определении производного шаблона требований. То есть дополнительными конъюнктами дополнительного инварианта для экземпляра шаблона прошлого являются все дополнительные конъюнкты необходимые для его параметров, и дополнительный конъюнкт, определяемый непосредственно для экземпляра шаблона прошлого. Этот дополнительный конъюнкт является импликацией. Посылкой этой импликации является дополнительный логический fn-параметр производного шаблона дополнительных инвариантов (один из fn-параметров $b_1, \ldots, b_l$ в схеме, введенной в разделе 3), возвращаемый функцией *newParam*. Значение этого параметра определяет условие, при котором должен выполняться экземпляр шаблона дополнительных инвариантов прошлого. Заключением импликации является экземпляр шаблона дополнительных инвариантов прошлого. Значениями параметров этого шаблона являются значения параметров шаблона требований прошлого в определении производного шаблона требований, преобразованные в соответствии с алгоритмом рекурсивными вызовами функции *invPattern*.

Если R' является экземпляром производного шаблона, то первым элементом кортежа, возвращаемого функцией *invPattern*, является конъюнкция экземпляра соответствующего шаблона дополнительных инвариантов и формул содержащихся во вторых элементах кортежей, вычисленных функцией *invPattern* для значений fm-параметров этого шаблона в определяемом шаблоне требований. Значения fm-параметров в этом экземпляре шаблона дополнительных инвариантов являются первыми элементами кортежей. То есть производный шаблон требований в дополнительном инварианте заменяется соответствующим шаблоном дополнительных инвариантов, а значения его параметров преобразуются в соответствии с алгоритмом рекурсивными вызовами функции *invPattern*, и к нему добавляются дополнительные конъюнкты. Второй элемент кортежа для производных шаблонов не используется, так как, дополнительные конъюнкты добавляются только для шаблонов прошлого, вложенных в другие шаблоны. Производным шаблоном дополнительных инвариантов является первый элемент кортежа, возвращаемого функцией *invPattern* для производного шаблона требований.

4.2. Построение лемм

Первые две посылки и заключение лемм, соответствующих схемам LS_8 и LS_9, определяются схемами лемм и шаблонами, для которых эти леммы строятся. Параметры схемы содержатся только в третьей посылке Определим функции *genL8* и *genL9*, вычисляющие третьи посылки лемм, удовлетворяющих схемам LS_8 и LS_9, соответственно. В псевдокоде используются следующие функции:

- *getInvPattern*(I) — возвращает производный шаблон дополнительных инвариантов, экземпляром которого является инвариант I;
- *operands*(A) — возвращает список операндов конъюнкции или дизъюнкции A;
- *params*(A) — возвращает значения параметров в экземпляре шаблона A в виде ассоциативного массива, в котором ключами являются параметры, а значениями– значения параметров;
- *substitute*(A, a) — возвращает формулу, полученную из A заменой всех вхождений ключей из ассоциативного массива a их значениями;
- *premise*(L) — возвращает посылку леммы, не являющуюся дополнительным инвариантом, а также не являющуюся формулой *consecutive*(s, s') и $e(s)$. Если лемма L соответствует схеме LS_8 или LS_9, возвращается третья посылка. Для других лемм возвращается вторая посылка;

- *lemmaForPattern*(P, LS) — возвращает лемму, связанную с шаблоном требований или дополнительных инвариантов P, которая удовлетворяет схеме LS;
- *getBoolParameter*(P)— возвращает логический fn-параметр, связанный с экземпляром P шаблона требований или дополнительных инвариантов прошлого с помощью функции *associate*, описанной в разделе 4.1.

Псевдокод функции *genL8* представлен на листинге 2.

Листинг 2: Algorithm for constructing a lemma satisfying the scheme LS_8. Алгоритм построения леммы, соответствующей схеме LS_8.

```
fun genL8(I) = match(I) {
    boolean(op, I1, I2) → genL8(I1) ∧ genL8(I2) |
    extended_inv(I0, [I1,..., In]) → {
      P = getInvPattern(I0);
      L8 = lemmaForPattern(P, LS8);
      p = premise(L8);
      p' = substitute(p, params(I0)) ∧ I1 ∧ ... ∧ In;
      replacePatternsL8(p');
    }
 }
```

Для определения этой функции также используется сопоставление с образцом.

Если производный шаблон дополнительных инвариантов и соответствующий ему шаблон требований являются булевыми комбинациями (конъюнкциями или дизъюнкциями), то возвращаемая функцией *genL8* формула является конъюнкцией формул, возвращаемых этой функцией для каждого конъюнкта или дизъюнкта шаблона дополнительных инвариантов. То есть посылка леммы для булевой комбинации шаблонов является конъюнкцией посылок соответствующих лемм для этих шаблонов.

Если шаблон дополнительных инвариантов соответствует шаблону требований, являющемуся экземпляром некоторого другого производного шаблона требований R, то шаблон дополнительных инвариантов представляет собой конъюнкцию экземпляра шаблона дополнительных инвариантов I_0 и дополнительных конъюнктов $I_1,\ldots, I_n$. В этом случае сначала с помощью функции *lemmaForPattern* определяется лемма *L8*, соответствующая шаблону P, которому удовлетворяет I_0. Затем используется функция *premise* для получения третьей посылки p этой леммы. Далее выполняется подстановка значений параметров шаблона P в экземпляре I_0 в формулу p, а также добавление дополнительных конъюнктов. В результате этого получается формула p'. Посылкой, возвращаемой функцией *genL8*, в этом случае является результат преобразования формулы p' функцией *replacePatternsL8*. Функция *replacePatternsL8* принимает в качестве параметра подформулу посылки леммы и заменяет в ней подформулы, содержащие шаблоны, формулами, не содержащими шаблонов, согласно следующему алгоритму, представленному на листинге 3.

Листинг 3: Definition of the function *replacePatternsL8*. Определение функции *replacePatternsL8*.

```
fun replacePatternsL8(p) = match(p) {
    boolean(op, A1, A2) → Boolean(op, replacePatternsL8(A1), replacePatternsL8(A2))
    |
    always_imp(s, A, A') → replacePatternsForImpL8(A, A') |
    future_inv(P, p1,..., pm, ep1,..., epk, A1,..., An)(s', s') → {
      prem = premise(lemmaForPattern(P, LS2));
      prem' = substitute(prem, params(p));
      replacePatternsL8(prem')
    } |
    past(P, p1,..., pm, A1,..., An)(s',s') → {
      prem = premise(lemmaForPattern(P, LS7));
      prem' = substitute(prem, params(p));
```

```
      replacePatternsL8(prem')
    } |
    past_inv(P, p_1,..., p_m, ep_1,..., ep_k, A_1,..., A_n)(s') → {
      prem = premise(lemmaForPattern(P, LS_6));
      prem' = substitute(prem, params(p));
      replacePatternsL8(prem')
    } |
    past_inv(P, p_1,..., p_m, ep_1,..., ep_k, A_1,..., A_n)(s) → getBoolParameter(p)(s) |
    A → A
}
```

В псевдокоде функции *replacePatternsL8* помимо типов формул, определенных в разделе 4.1, используются следующие типы формул:

- $always\_imp(s, A, A')$ — формула $\forall s_1.e(s_1) \wedge s_1 \leqslant s \wedge A(s_1) \longrightarrow A'(s_1)$;
- $future\_inv(P, p_1, \ldots, p_m, ep_1, \ldots, ep_k, A_1, \ldots, A_n$ — формула $\lambda ss_1.P(s, s_1, p_1, \ldots, p_m, ep_1, \ldots, ep_k, A_1, \ldots, A_n)$, где P — некоторый шаблон дополнительных инвариантов будущего;
- $past\_inv(P, p_1, \ldots, p_m, ep_1, \ldots, ep_k, A_1, \ldots, A_n)$ — формула $\lambda s.P(s, p_1, \ldots, p_m, ep_1, \ldots, ep_k, A_1, \ldots, A_n)$, где P — некоторый шаблон дополнительных инвариантов прошлого.

Если исходная формула, переданная в качестве параметра, является конъюнкцией или дизъюнкцией формул A_1 и A_2, то возвращаемая в качестве результата преобразованная формула является, соответственно, конъюнкцией или дизъюнкцией формул, полученных в результате преобразования A_1 и A_2. То есть выполняется замена подформул, содержащих экземпляры шаблонов, в каждом конъюнкте или дизъюнкте.

Если исходная формула имеет вид: $\forall s_1.e(s_1) \wedge s_1 \leqslant s \wedge A(s_1) \longrightarrow A'(s_1)$, то она заменяется формулой, не содержащей шаблоны, вычисляемой функцией *replacePatternsForImpL8*, псевдокод которой представлен на листинге 4. В качестве параметров этой функции передаются формулы A и A'.

Если исходная формула является экземпляром шаблона дополнительных инвариантов будущего, выполняющегося в конце итерации, в котором совпадают значения параметров s_1 и s, т. е. будущее ограничено состоянием, в котором выполняется экземпляр шаблона, то берется третья посылка *prem* леммы для этого шаблона, удовлетворяющей схеме LS_2, в эту посылку подставляются значения параметров шаблона, и выполняется рекурсивный вызов функции *replacePatternsL8* для преобразования полученной формулы. То есть экземпляр шаблона дополнительных инвариантов будущего сводится к формуле, которая является достаточным условием его истинности (если экземпляр шаблона выполняется во внешнем состоянии), а затем в этой формуле выполняется замена подформул, содержащих вложенные шаблоны.

Если исходная формула является экземпляром шаблона требований или дополнительных инвариантов прошлого, выполняющимся в конце итерации, то выполняются аналогичные преобразования, но используется лемма, удовлетворяющая схеме LS_7 или LS_6, соответственно.

Если исходная формула является экземпляром шаблона дополнительных инвариантов прошлого, выполняющимся в начале итерации, он заменяется логическим параметром, введенных для этого экземпляра в производном шаблоне дополнительных инвариантов, для которого строится лемма.

В остальных случаях исходная формула не содержит экземпляров шаблонов и возвращается без преобразования.

Листинг 4: Definition of the function *replacePatternsForImpL8*. Определение функции *replacePatternsForImpL8*.

```
fun replacePatternsForImpL8(A, A') = match(A) {
    boolean(op, B_1, B_2) → {
      [B'_1, B'_2] = operands(A');
      prem = Boolean((∧), always_imp(s, B_1, B'_1), always_imp(s, B_2, B'_2))
```

```
      replacePatternsL8(prem);
      } |
    future_inv(I, p1,..., pm, ep1,..., epk, A'1,..., A'n)(s) → {
      prem = premise(lemmaForPatternL8(I, LS1));
      prem' = substitute(prem, params(A));
      replacePatternsL8(prem')
      } |
    past(P, p1,..., pm, A'1,..., A'n)(s) → {
      prem = premise(lemmaForPattern(P, LS4));
      prem' = substitute(prem, params(A));
      replacePatternsL8(prem')
    } |
    B → if A == A' then True else always_imp(s, A, A')
}
```

В определении функции *replacePatternsForImpL8* выполняется сопоставление с образцом значения первого параметра.

Если значение первого параметра имеет вид B_1 *op* B_2, где *op* — конъюнкция или дизъюнкция, то значение второго параметра также является конъюнкцией или дизъюнкцией, соответственно, т. е. имеет вид B'_1 *op* B'_2. В этом случае функция возвращает формулу, полученную из формулы $(\forall s_1.e(s_1) \wedge s_1 \leqslant s \wedge B_1(s_1) \longrightarrow B'_1(s_1)) \wedge (\forall s_1.e(s_1) \wedge s_1 \leqslant s \wedge B_2(s_1) \longrightarrow B'_2(s_1))$ заменой подформул, содержащих шаблоны, функцией *replacePatternsL8*. То есть каждая подформула вида $(\forall s_1.e(s_1) \wedge s_1 \leqslant s \wedge (B_1(s_1) \wedge B_2(s_1)) \longrightarrow (B'_1(s_1) \wedge B'_2(s_1)))$ или $(\forall s_1.e(s_1) \wedge s_1 \leqslant s \wedge (B_1(s_1) \vee B_2(s_1)) \longrightarrow (B'_1(s_1) \vee B'_2(s_1)))$ в посылке леммы заменяется конъюнкцией формул $(\forall s_1.e(s_1) \wedge s_1 \leqslant s \wedge B_1(s_1) \longrightarrow B'_1(s_1))$ и $(\forall s_1.e(s_1) \wedge s_1 \leqslant s \wedge B_2(s_1) \longrightarrow B'_2(s_1))$.

Если значение первого параметра является экземпляром шаблона дополнительных инвариантов будущего или шаблона требований прошлого, то значение второго параметра является экземпляром того же шаблона, в котором состояние s заменено состоянием s'. В этом случае во вторую посылку леммы для этого шаблона, удовлетворяющей схеме LS_1 или LS_4, соответственно подставляются параметры шаблона, и полученная формула преобразуется функцией *replacePatternsL8*. То есть каждая формула вида

$$(\forall s_1.e(s_1) \wedge s_1 \leqslant s \wedge P(s, s_1, p_1, \ldots, p_m, ep_1, \ldots, ep_k, A_1, \ldots, A_n) \longrightarrow P(s', s_1, p_1, \ldots, p_m, ep_1, \ldots, ep_k, A_1, \ldots, A_n)),$$

где P — шаблон дополнительных инвариантов будущего, а также каждая формула вида

$$(\forall s_1.e(s_1) \wedge s_1 \leqslant s \wedge P(s, s_1, p_1, \ldots, p_m, A_1, \ldots, A_n) \longrightarrow P(s', s_1, p_1, \ldots, p_m, A_1, \ldots, A_n)),$$

где P — шаблон требований прошлого, заменяется формулой, являющейся достаточным условием истинности исходной формулы (при условии, что выполняется $consecutive(s, s')$), и в полученной формуле также заменяются подформулы, содержащие шаблоны.

В остальных случаях исходная формула не содержит вложенных шаблонов и возвращается без преобразований.

Для построения леммы, удовлетворяющей схеме LS_9 определяются функции *replacePatternsL9*, *genL9* и *replacePatternsForImpL9*, аналогичные функциям *replacePatternsL8*, *genL8* и *replacePatternsForImpL8*, соответственно. Здесь не приводится псевдокод этих функций. Рассмотрим отличия этих функций от соответствующих функций построения лемм по схеме LS_8.

- В функции *genL9* берется лемма, соответствующая схеме LS_9, а не схеме LS_8.
- В функции *genL9* к посылке леммы p' не добавляются дополнительные конъюнкты, так как заключение леммы является требованием, а не дополнительным инвариантом, а для замены подформул p', содержащих шаблоны, используется функция *replacePatternsL9*.
- В функции *replacePatternsL9* выполняется сопоставление с образцами *boolean* и *always_imp*, а в остальных случаях возвращается исходная формула без преобразования. Сопоставление

с образцами *future_inv*, *past* и *past_inv* не выполняется, так как исходная формула не содержит экземпляров шаблонов, выполняющихся в состоянии s, в котором выполняется инвариант цикла. Это связано с тем, что условие *cond* в леммах, удовлетворяющих LS_9 не зависит от fm-параметров $A_1,\ldots,A_n$, выполняющихся в состоянии s. В случае, если исходная формула имеет вид $\forall s_1.e(s_1) \wedge s_1 \leqslant s \wedge A(s_1) \longrightarrow A'(s_1)$, эта формула преобразуется функцией *replacePatternsForImpL9*.

- В функции *replacePatternsForImpL9* вместо леммы, соответствующей схеме LS_1 используется лемма, соответствующая схеме LS_3, а вместо леммы, соответствующей схеме LS_4, используется лемма, соответствующая схеме LS_5.
- В функции *replacePatternsForImpL9* в посылку леммы подставляются значения параметров как в формуле A, так и в формуле A', та как эти формулы являются экземплярами различных шаблонов. После подстановки значений дальнейшее преобразование формулы выполняется функцией *replacePatternsL9*.

4.3. Доказательство лемм

В доказательствах лемм для производных шаблонов кроме лемм для базовых шаблонов используются следующие две леммы:

1. L_{conj}:

$$e(s) \longrightarrow$$
$$(\forall s_1.e(s_1) \wedge s_1 \leqslant s \wedge A(s_1) \longrightarrow A'(s_1)) \wedge (\forall s_1.e(s_1) \wedge s_1 \leqslant s \wedge B(s_1) \longrightarrow B'(s_1)) \longrightarrow$$
$$(\forall s_1.e(s_1) \wedge s_1 \leqslant s \wedge A(s_1) \wedge B(s_1) \longrightarrow A'(s_1) \wedge B'(s_1))$$

2. L_{disj}:

$$e(s) \longrightarrow$$
$$(\forall s_1.e(s_1) \wedge s_1 \leqslant s \wedge A(s_1) \longrightarrow A'(s_1)) \wedge (\forall s_1.e(s_1) \wedge s_1 \leqslant s \wedge B(s_1) \longrightarrow B'(s_1)) \longrightarrow$$
$$(\forall s_1.e(s_1) \wedge s_1 \leqslant s \wedge (A(s_1) \vee B(s_1)) \longrightarrow (A'(s_1) \vee B'(s_1))$$

Эти леммы используются для доказательства формул вида: $(\forall s_1.e(s_1) \wedge s_1 \leqslant s \wedge A(s_1) \longrightarrow A'(s_1))$, где A и A' являются конъюнкциями или дизъюнкциями, соответственно. Формулы A и A' могут содержать свободную переменную s или s'. В леммах для производных шаблонов эти переменные обозначают состояния изменений до и после итерации цикла, соответственно.

Мы определяем методы доказательства для доказательства лемм для производных шаблонов. Эти методы доказательства были реализованы на языке описания методов доказательства Eisbach [10] системы Isabelle/HOL.

Для работы этих методов сначала с помощью команды `named_theorems` определяется набор динамических фактов. Для каждого такого факта Isabelle/HOL автоматически создает соответствующий атрибут, который добавляется к лемме, чтобы добавить ее в коллекцию фактов. Атрибут `patternintro` добавляется к леммам L_{conj} и L_{disj}, ко всем леммам, определяемым для шаблонов будущего, а также к леммам для шаблонов прошлого, соответствующим схеме LS_7. Атрибут `pastinv` добавляется к леммам для шаблонов прошлого, соответствующим схеме LS_6. Атрибут `inv_req` добавляется к леммам для шаблонов прошлого, соответствующим схеме LS_5. Атрибут `invsaving` добавляется к леммам для шаблонов прошлого, соответствующим схеме LS_4. Факт `add_rules` является пустым до вызова методов доказательства, но он является параметром методов. При вызове метода в качестве значения параметра `add_rules` передается факт `invsaving`, если доказывается лемма, соответствующая схеме LS_8, или `inv_req`, если доказывается лемма, соответствующая схеме LS_9. Атрибут `elims` добавляется к леммам для производных шаблонов, соответствующим схемам LS_8 и LS_9. Факт `einvs` является пустым до вызова методов. Во время выполнения методов доказательства в него добавляются некоторые подформулы дополнительных инвариантов, в частности, дополнительные конъюнкты, необходимые для доказательства леммы.

```
method prove declares add_rules  =
(*formula without requirement and extra
invariant patterns*)
assumption |
(*conjunction*)
match conclusion in "?P ∧ ?Q" ⇒
‹erule conjE;
match premises in e[thin]: _ (cut) ⇒
‹match premises in p1[thin]: _ (cut) ⇒
‹match premises in p2[thin]: _ (cut) ⇒
‹rule conjI,
(insert e p1)[1], prove,
(insert e p2)[1], prove›››› |
(*disjunction*)
match conclusion in "?P ∨ ?Q" ⇒
‹erule disjE,
rule disjI1, prove, rule disjI2, prove› |
(*implication*)
match conclusion in "?b ⟶ ?P" ⇒
‹rule impI, rule impE, assumption,
rotate_tac,
match premises in p[thin]: _ (cut) ⇒
‹rotate_tac, prove›› |
(*∀ s1. e(s1) ∧ s1 ≤ s ∧ A(s1) ⟶ A'(s1)*)
match conclusion in "always_imp ?s ?A ?A'"
⇒ ‹fact always_imp_refl› |
(*pattern*)
rule patternintro add_rules,
(assumption | simp), prove |
(*past extra invariant pattern*)
rotate_tac;
match premises in p[thin]: b (cut) for b ⇒
‹match einvs in i: "b ⟶ ?P" ⇒
‹fact mp[OF i p]››
```

Fig. 4. Definition of the method *prove*

Рис. 4. Определение метода *prove*

Сначала мы определяем вспомогательный метод `prove`, который используется для доказательства формул, не являющихся подформулами заключения доказываемой леммы и появляющихся при доказательстве после применения других лемм. Этот метод определяется следующим образом (рисунок 4).

Сначала этот метод делает попытку доказать методом `assumption`. Формулы, не содержащие шаблонов требований и дополнительных инвариантов, доказываются методом `assumption`, поскольку посылки леммы содержат формулу, совпадающую с текущей подцелью. Если цель не может быть доказана методом `assumption`, метод делает попытку применить другие альтернативы в следующем порядке.

Если заключение текущей цели является конъюнкцией, цель содержит две посылки: первая имеет вид $e(s)$ или $consecutive(s, s')$, а вторая является конъюнкцией. В этом случае к посылке применяется правило удаления конъюнкции *conjE* . Затем расщепляется конъюнкция в заключении. Для доказательства каждого конъюнкта используются первая посылка e и соответствующий конъюнкт второй посылки (соответственно, *p1* и *p2* для первого и второго конъюнкта) исходной цели. Метод `prove` рекурсивно вызывается для доказательства каждого конъюнкта.

Если заключение текущей цели является дизъюнкцией, цель содержит две посылки: первая имеет вид $e(s)$ или $consecutive(s, s')$, а вторая является дизъюнкцией. Для доказательства этой цели выполняется разбор случаев. Если истинен первый дизъюнкт в посылке, доказывается первый дизъюнкт в заключении, иначе доказывается второй дизъюнкт в заключении. Метод `prove` рекурсивно вызывается для доказательства каждого дизъюнкта.

Если заключение текущей цели является импликацией, цель содержит две посылки: первая имеет вид $e(s)$ или $consecutive(s, s')$, а вторая является импликацией, посылка которой совпадает с посылкой импликации в заключении цели. В этом случае к посылке цели применяется правило удаления импликации *impE*, и посылка импликации в посылке цели доказывается с помощью посылки импликации в заключении цели. Затем эта посылка удаляется из цели и заключение полученной подцели, которое является следствием импликации в заключении исходной цели, доказывается рекурсивным вызовом метода `prove`.

Если заключение текущей цели имеет вид $\forall s_1.e(s_1) \wedge s_1 \leqslant s \wedge A(s_1) \longrightarrow A(s_1)$, применяется лемма *always_imp_refl*, имеющая следующий вид:

$$\forall s_1.e(s_1) \wedge s_1 \leqslant s \wedge A(s_1) \longrightarrow A(s_1)\,.$$

Если заключение текущей цели является экземпляром некоторого шаблона, но не экземпляром шаблона дополнительных инвариантов прошлого, применяется соответствующая лемма из коллекции `patternintro` или `add_rules`. Применение леммы создает две подцели. Первая подцель имеет вид $e(s)$, $e(s')$ или $consecutive(s, s')$. Доказательство этой подцели может быть выполнено методом `assumption` или `simp`. Затем метод `prove` рекурсивно вызывается для доказательства второй подцели. Коллекция фактов *add_rules* является параметром метода, значение которого зависит от того, доказывается ли лемма, удовлетворяющая схеме лемм LS_8 или LS_9.

В остальных случаях заключение текущей цели является экземпляром шаблона дополнительных инвариантов прошлого I, выполняющегося в начале итерации цикла в состоянии s. В этом случае исходная цель имеет две посылки: первая имеет вид $consecutive(s, s')$, а вторая — формула вида $b_i(s)$, где b_i — дополнительный логический fn-параметр производного шаблона дополнительных инвариантов. Для доказательства этой подцели выполняются следующие действия. Вторая посылка уделяется из подцели, для нее вводится имя p, а переменная b связывается с $b_i(s)$. Затем в коллекции фактов `einvs`, хранящей, в частности, дополнительные конъюнкты производного шаблона дополнительных инвариантов, находится импликация i вида $b_i(s) \longrightarrow P$. Затем для доказательства применяется правило вывода modus ponens с посылками i и p.

Затем мы определяем основной метод `proveOuter`, который используется для доказательства лемм, удовлетворяющих LS_8 и LS_9. Этот метод определяется следующим образом (рисунок 5).

Цель, к которой применяется метод `proveOuter`, имеет три посылки.

Если заключение текущей цели является конъюнкцией, одна из посылок имеет вид $e(s)$ или $consecutive(s, s')$, а две другие являются конъюнкциями, причем одна из них является дополнительным инвариантом или его частью, а другая является третьей посылкой доказываемой леммы или ее частью. Для доказательства этой цели к посылкам применяется правило удаления конъюнкции *conjE*. Затем расщепляется конъюнкция в заключении цели. Для доказательства каждого конъюнкта используется посылка вида $e(s)$ или $consecutive(s, s')$ и соответствующие конъюнкты двух других посылок исходной цели (*i1* и *p1* для доказательства первого конъюнкта и *i2* и *p2* для доказательства второго конъюнкта). Доказательство каждого конъюнкта выполняется рекурсивным вызовом метода `proveOuter`.

Если заключение текущей цели является дизъюнкцией, цель имеет следующие посылки: 1) $e(s)$ или $consecutive(s, s')$, 2) дизъюнкцию, которая является дополнительным инвариантом или его частью, 3) конъюнкцию, которая является третьей посылкой доказываемой леммы или ее частью. Порядок первых двух посылок может быть любым. В этом случае для доказательства к третьей посылке применяется правило удаления конъюнкции *conjE*. Затем посылкам подцели задаются имена и конъюнкты третьей посылки удаляются из подцели. Далее выполняется разбор случаев. Если истинен первый дизъюнкт в посылке, то доказывается первый дизъюнкт в заключении подцели с использованием первого конъюнкта *p3* третьей посылки. Если истинен второй дизъюнкт в посылке, доказывается второй дизъюнкт в заключении подцели с использованием второго конъюнкта *p4* третьей посылки исходной цели. В каждом случае для доказательства соответствующего дизъюнкта рекурсивно вызывается метод `proveOuter`.

Если заключение текущей цели — импликация, оно является дополнительным конъюнктом дополнительного инварианта. В этом случае первая посылка текущей цели имеет вид $e(s)$ или $consecutive(s, s')$, а вторая и третья посылки являются импликациями, причем первая импликация является дополнительным конъюнктом дополнительного инварианта, выполняющегося в начале итерации, а вторая импликация является частью третьей посылки доказываемой леммы, и ее посылка совпадает с посылкой импликации в заключении текущей цели и имеет вид $b(s')$ — логический fn-параметр производного шаблона дополнительных инвариантов. Доказательство подцели выполняется следующим образом. Сначала из подцели удаляется первая импликация (конъюнкт дополнительного инварианта). Затем к последней импликации применяется правило удаления импликации *impE*, ее посылка доказывается с помощью посылки $b(s')$ в заключении цели. Заключение полученной после этого подцели является экземпляром шаблона дополнительных инвариантов прошлого, выполняющимся в конце итерации в состоянии s'. Для его доказательства из подцели удаляется посылка $b(s')$, не используемая далее, и применяется подходящая лемма из коллекции `pastinv`. В результате применения леммы получается две подцели. Первая подцель имеет вид $consecutive(s, s')$ и доказывается методом `assumption`. Вторая подцель доказывается методом `prove`.

В остальных случаях текущая цель является экземпляром шаблона требований или дополнительных инвариантов. В этом случае текущая цель имеет следующие посылки: 1) $e(s)$ или $consecutive(s, s')$, 2) дополнительный инвариант или его часть, соответствующая заключению текущей подцели, 3) третья посылка доказываемой леммы или ее часть. Порядок первых двух посылок может быть любым. При этом, если доказывается лемма, соответствующая схеме LS_9, заключение цели является экземпляром шаблона требований, а посылка 2 может быть как экземпляром шаблона дополнительных инвариантов, так и конъюнкцией, то есть содержать дополнительные конъюнкты, которые не используются для доказательства текущей цели. Поэтому эти конъюнкты, если таковые имеются, удаляются. Затем применяется подходящая лемма из коллекции `elims`, в результате

```
method proveOuter declares add_rules =
(*conjunction*)
match conclusion in "?P ∧ ?Q" ⇒
‹erule conjE; erule conjE;
match premises in e[thin]: _ (cut) ⇒
‹match premises in i1[thin]: _ (cut) ⇒
‹match premises in i2[thin, einvs]: _ (cut) ⇒
‹match premises in p1[thin]: _ (cut) ⇒
‹match premises in p2[thin]: _ (cut) ⇒
‹rule conjI, (insert e i1 p1)[1], proveOuter,
(insert e i2 p2)[1], proveOuter›››››› |
(*disjunction*)
match conclusion in "?P ∨ ?Q" ⇒
‹erule conjE; rotate_tac 2;
match premises in p1[thin]: _ (cut) ⇒
‹match premises in p2[thin]: _ (cut) ⇒
‹erule disjE,
(insert p1)[1], rule disjI1, proveOuter,
(insert p2)[1], rule disjI2, proveOuter››› |
(*implication: additional conjunct*)
match conclusion in "?b ⟶ ?P" ⇒
‹rotate_tac;
match premises in i[thin]: _ (cut) ⇒
‹rule impI, erule impE, assumption, rotate_tac,
match premises in b[thin]: _ (cut) ⇒
‹rotate_tac, rule pastinv, assumption, prove›››
|
(*pattern*)
rotate_tac -1;
match premises in p[thin]: _ (cut) ⇒
‹(simp only: conj_assoc)?;
(erule conjE; rotate_tac -1;
match premises in ac[thin]: _ (cut) ⇒
‹succeed›)?;
erule elims, assumption, (insert p)[1], prove›
```

Fig. 5. Definition of the method *proveOuter*

Рис. 5. Определение метода *proveOuter*

чего получается две подцели. Первая подцель имеет вид $e(s)$ или $consecutive(s, s')$ и доказывается методом `assumption`. Вторая подцель доказывается методом `prove`.

5. Пример

В этом разделе мы рассмотрим пример построения производного шаблона требований и соответствующего ему шаблона дополнительных инвариантов и задания требования и дополнительных инвариантов по шаблонам.

Рассмотрим в качестве примера программу управления сушилкой для рук. Сушилка для рук включает в себя датчик наличия рук и тепловентилятор. Программа получает входной сигнал от датчика и в зависимости от входного сигнала управляет тепловентилятором. Если руки появляются, то включается тепловентилятор. Если руки убрали, то через определенное время тепловентилятор выключается.

Программа управления сушилкой для рук представлена на рисунке 6.

В программе объявлены две переменные: входная переменная `hands`, которая показывает наличие рук под тепловентилятором, и выходная переменная `dryer`, которая определяет, включен ли тепловентилятор. Определен один процесс `Ctrl`. Он имеет два состояния `waiting` и `drying`. В состоянии `waiting` проверяется наличие рук. Если руки есть, то тепловентилятор включается и процесс `Ctrl` переходит в состояние `drying`. Если руки отсутствуют, то тепловентилятор выключается. В состоянии `drying` также проверяется наличие рук. Если руки есть, то таймер процесса сбрасывается. В этом состоянии устанавливается таймаут. Через 1 секунду процесс `Ctrl` переходит в состояние `waiting`.

Необходимо проверить следующие требования к программе управления сушилкой для рук:

1. Если тепловентилятор не включен и руки отсутствуют, то он не включится
2. Если рук нет, то тепловентилятор должен выключиться не более чем через 1 секунду, если за это время руки не появятся снова

Опишем верификацию программы по шагам.

Первым шагом проверки является определение дополнительных инвариантов, независимых от требований. Один из шаблонов дополнительных инвариантов, независимых от требований, утверждает, что если процесс p находится в состоянии q, то переменная x имеет значение v. Этот шаблон определяется следующим образом:

$$\lambda s.getPstate(s, p) = q \longrightarrow s[x] = v$$

Следующее свойство программы управления сушилкой для рук можно задать с помощью этого шаблона: «Если процесс `Ctrl` находится в состоянии `drying`, переменная `dryer` имеет значение `TRUE`». Параметры имеют следующие значения:

```
p ≡ Ctrl; q ≡ drying; x ≡ dryer; v ≡ True.
```

Далее выполняется верификация сформулированных выше требований.

Первое требование удовлетворяет следующему производному шаблону требований: «Если выполнялось условие A_1, и на следующей итерации выполняется условие A_2, то в конце второй итерации выполняется условие A_3». С помощью базовых шаблонов этот производный шаблон определяется следующим образом:

$$DRP1(s, A_1, A_2, A_3) \equiv PRP1(s, s, (\lambda r_2 r_1.PRP3(r_2, r_1, (\lambda r_4 r_3.\neg A_1(r_3))) \vee \neg A_2(r_1) \vee A_3(r_2, r_1))).$$

Здесь s — это состояние изменений, в котором должно быть выполнено требование, A_1, A_2 и A_3 — fm-формулы, значения A_1 и A_2 не могут содержать вложенных шаблонов, а значение A_3 может. В этом определении используются два базовых шаблона требований `PRP1` и `PRP3`, определенные в базе знаний.

```
PROGRAM Controller
  VAR_INPUT
    hands : BOOL;
  END_VAR
  VAR_OUTPUT
    dryer : BOOL;
  END_VAR
  PROCESS Ctrl
    STATE waiting
      IF hands  THEN
        dryer := TRUE;
        SET NEXT;
      ELSE
        dryer := FALSE;
      END_IF
    END_STATE
    STATE drying
      IF hands  THEN
        RESET TIMER;
      END_IF
      TIMEOUT T#1s THEN
        SET STATE waiting;
      END_TIMEOUT
    END_STATE
  END_PROCESS
END_PROGRAM
```

Fig. 6. Hand dryer control program in poST language

Рис. 6. Программа управления сушилкой для рук на языке poST

В определении производного шаблона требования DRP1 экземпляр шаблона PRP1 выполняется в состоянии s, поскольку экземпляр является инвариантом цикла управления. Связанные переменные r_2 и r_1 соответствуют состояниям изменений, в которых выполняются инвариант цикла управления и fm-параметр A_1 шаблона PRP1, соответственно. Значение fm-параметра A_1 базового шаблона PRP1 является дизъюнкцией. Первый дизъюнкт является экземпляром шаблона PRP3. Значением fm-параметра A_1 шаблона PRP3 является отрицание fm-параметра A_1 производного шаблона требований. Второй дизъюнкт является отрицанием fm-параметра A_2 производного шаблона требований. Третьим дизъюнктом является fm-параметр A_3 производного шаблона требований.

После того, как пользователь определил производный шаблон требований, создается соответствующий производный шаблон дополнительных инвариантов DIP1. Он определяется следующим образом:

$$DIP1(s, A_1, A_2, A_3) \equiv PIP1(s, (\lambda r_2 r_1.PRP3(r_2, r_1, (\lambda r_4 r_3.\neg A_1(r_3))) \vee \neg A_2(r_1) \vee A_3(r_2, r_1))) \wedge (b(s) \longrightarrow PIP3(s, (\lambda r_2 r_1.\neg A_1(r_1))))$$

Значения fm-параметров A_1 и A_2 в экземпляре шаблона DIP1 равны значениям соответствующих fm-параметров в экземпляре шаблона DRP1. Значение fm-параметра A_3 в экземпляре шаблона DIP1 в общем случае не равно значению fm-параметра A_3 в соответствующем экземпляре шаблона DRP1. Определение шаблона DIP1 является конъюнкцией. Первый конъюнкт является экземпляром шаблона PIP1, выполняющимся в состоянии s, в котором выполняется инвариант цикла. Значени-

ем fm-параметра A_1 шаблона PIP1 является дизъюнкция. Первым дизъюнктом является экземпляр шаблона требований прошлого PRP3. Значением fm-параметра A_1 шаблона PRP3 является отрицание fm-параметра A_1 производного шаблона DIP1. Вторым дизъюнктом является отрицание fm-параметра A_2 производного шаблона DIP1. Третьим дизъюнктом является fm-параметр A_3 производного шаблона DIP1. Второй конъюнкт в определении DIP1 является импликацией. Посылкой импликации является дополнительный fn-параметр b шаблона DIP1. Заключением импликации является экземпляр шаблона дополнительных инвариантов прошлого PIP3. Значение fm-параметра A_1 шаблона PIP3 является отрицанием fm-параметра A_1 производного шаблона DIP1.

После определения производных шаблонов требований и дополнительных инвариантов пользователь указывает следующие параметры шаблона:

$$A_1 \equiv \lambda(s, r_1).r_1[dryer] = False; \quad A_2 \equiv \lambda(s, r_2).r_2[hands] = False; \quad A_3 \equiv \lambda(s, r_2).r_2[dryer] = False.$$

Значение дополнительного fn-параметра b шаблона дополнительных инвариантов DIP1 не определяется, так как для доказательства условий корректности для первого требования не используется зависящий от требований дополнительный инвариант (см. раздел 6).

Второе требование удовлетворяет следующему производному шаблону требований: «Если произошло событие A_1, то событие A_3 должно произойти не позднее, чем через время t, причем после наступления A_1 и до наступления A_3 должно выполняться условие A_2». Этот шаблон определяется следующим образом:

$$DRP2(s, t, A_1, A_2, A_3) \equiv PRP1(s, s, (\lambda r_2\ r_1.\neg A_1(r_1) \vee FRP1(r_2, r_1, t, A_2, A_3))).$$

Здесь значение A_1 не может содержать вложенных шаблонов, а значения A_2 и A_3 могут. В этом определении используются два базовых шаблона требований PRP1 и FRP1, определенные в базе знаний.

В определении производного шаблона требования DRP2 экземпляр шаблона PRP1 выполняется в состоянии s, поскольку экземпляр является инвариантом цикла управления. Связанные переменные r_2 и r_1 соответствуют состояниям изменений, в которых выполняются инвариант цикла управления и fm-параметр A_1 шаблона PRP1, соответственно. Значение fm-параметра A_1 базового шаблона PRP1 является дизъюнкцией. Его первый дизъюнкт является отрицанием fm-параметра A_1 производного шаблона требования. Второй дизъюнкт — это экземпляр шаблона FRP1, который выполняется в состоянии изменений r_1, значения параметров t, A_1 и A_2 которого являются параметрами t, A_2 и A_3 производного шаблона, соответственно.

После того, как пользователь определил производный шаблон требований, создается соответствующий производный шаблон дополнительных инвариантов DIP2. Он определяется следующим образом:

$$DIP2(s, t, t_1, A_1, A_2, A_3) \equiv PIP1(s, (\lambda r_2 r_1.\neg A_1(s_1) \vee FIP1(r_2, r_1, t, t_1, A_2, A_3)))$$

В этом определении s — состояние изменений, в котором выполняется дополнительный инвариант, t — с-параметр, его значение равно значению параметра t в соответствующем экземпляре шаблона DRP2, t_1 — дополнительный fn-параметр, зависящий от состояния изменений s, A_1, A_2 и A_3 — fm-параметры, причем значение параметра A_1 равно значению параметра A_1 в соответствующем экземпляре шаблона DRP2, а значения A_2 и A_3 в общем случае не равны, но связаны со значениями параметров A_2, A_3, соответственно, в соответствующем экземпляре шаблона DRP2. Экземпляр шаблона PIP1 выполняется в состоянии s. Связанные переменные r_2 и r_1 соответствуют состояниям изменений, в которых выполняются инвариант цикла управления и параметр A_1 шаблона PIP1, соответственно. Значение параметра A_1 в шаблоне PIP1 является дизъюнкцией. Первый дизъюнкт является отрицанием параметра A_1 производного шаблона. Второй дизъюнкт является экземпляром шаблона FIP1, который выполняется в состоянии r_1. Значения параметров t, t_1, A_1 и A_2 в шаблоне FIP1 являются параметрами t, t_1, A_2 и A_3 в производном шаблоне DIP2.

После определения производных шаблонов требований и дополнительных инвариантов пользователь указывает следующие параметры шаблонов:

$A_1 \equiv \lambda(s, r_1).r_1[hands] = False$; $A_2 \equiv \lambda(s, r_2).r_2[dryer] = True \wedge r_2[hands] = False$;
$A_3 \equiv \lambda(s, r_3).r_3[dryer] = False \vee r_3[hands] = True$; $t \equiv 10$;
$t_1 \equiv \lambda s.\ \mathtt{if}\ getPstate(s, Ctrl) = drying\ \mathtt{then}\ ltime(s, Ctrl)\ \mathtt{else}\ 10.$

Далее генерируются леммы для этих производных шаблонов, которые затем доказываются в Isabelle/HOL, генерируются условия корректности и скрипты доказательства для них. Эти скрипты доказательства выполняются в Isabelle/HOL, и все условия корректности доказываются. Поскольку этот производный шаблон требования является универсальным, он сохраняется в базе знаний вместе с соответствующим производным шаблоном дополнительных инвариантов и леммами.

6. Схемы доказательства условий корректности

Ранее в [11] были разработаны схемы доказательства условий корректности для требований, соответствующих ранее разработанным шаблонам требований. Для каждого шаблона требований была определена схема доказательства, позволяющая доказывать в системе Isabelle/HOL условия корректности для требований, соответствующих этим шаблонам. Но в той работе отсутствовало разделение шаблонов требований на базовые и производные. Для некоторых шаблонов схемы доказательства различались только применяемой леммой и могли быть обобщены на новые классы требований путем замены леммы, но для других шаблонов применялись специализированные схемы доказательства. Для доказательства условий корректности в рамках подхода, в котором пользователь может определять производные шаблоны и задавать требования с помощью этих шаблонов, необходимы схемы доказательства, применимые ко всем классам требований, но, возможно, имеющие параметры, значения которых зависят от типа требования, для которого доказывается условие корректности. Кроме того, в ранее разработанных схемах доказательства в некоторых случаях необходимо было выполнять разбор случаев по некоторому терму, например, состоянию процесса или значению локальной переменной в начале итерации цикла. Определение необходимости выполнения разбора случаев и подходящего терма требовало анализа доказываемого условия корректности. Поэтому этот терм был параметром схемы доказательства и указывался вручную. Необходимость разбора случаев была связана с тем, что зависящий от требований дополнительный инвариант имел вид конъюнкции импликаций, посылки которых являются утверждениями о состояниях процессов, значениях локальных и выходных переменных и таймеров в состоянии изменений, в котором выполняется инвариант, и для доказательства используется один из конъюнктов в зависимости от этих значений. Но эти значения и состояния процессов могут быть неизвестны. Чтобы устранить необходимость разбора случаев, в данном разделе предлагается новый вид зависящих от требований дополнительных инвариантов и новые схемы доказательства условий корректности. Будем определять дополнительный инвариант в виде конъюнкции независимой от требований части и экземпляра производного шаблона дополнительных инвариантов. При этом значения дополнительных fn-параметров шаблона $ep_1, \ldots, ep_k, b_1, \ldots, b_l$ имеют вид:

$\lambda s.\ \mathtt{if}\ P_1(s)\ \mathtt{then}\ v_1(s)\ \mathtt{else}\ \mathtt{if}\ P_2(s)\ \mathtt{then}\ v_2(s)\ \mathtt{else}\ \ldots\ \mathtt{if}\ P_n(s)\ \mathtt{then}\ v_n(s)\ \mathtt{else}\ v_{n+1}(s),\ n \geqslant 0,$

где P_i — некоторые утверждения о состояниях процессов, значениях переменных и таймеров, которые раньше были посылками импликаций в конъюнктах дополнительного инварианта, $v_i(s)$ — термы вида c или $c + ltime(s, p)$, c — константа, p — процесс. Пусть L_8 — лемма для частных шаблонов, с помощью которых заданы требование и дополнительный инвариант, полученная упрощением леммы для общих шаблонов, соответствующей схеме лемм LS_8, L_9 — лемма для частных шаблонов, полученная упрощением леммы для общих шаблонов, соответствующей схеме лемм LS_9. Так как при использовании этого подхода дополнительный инвариант содержит единственный конъюнкт, соответствующий производному шаблону, можно применить лемму L_8 для доказательства

условий корректности для дополнительного инварианта и лемму L_9 для доказательства условий корректности для расширенного инварианта, являющегося конъюнкцией дополнительного инварианта и требования. После применения этих лемм получаются цели, не содержащие шаблонов, но содержащие условные выражения. Полученная цель может быть доказана в Isabelle/HOL с помощью расщепления условных выражений, для выполнения которого используется правило упрощения *if_splits* из стандартной библиотеки Isabelle/HOL, и применения автоматических методов доказательства, например, `auto`.

Рассмотрим схемы доказательства условий корректности. Доказательство условия корректности, соответствующего пути инициализации программы, выполняется путем раскрытия всех определений и применения метода `auto`. Далее будут описаны схемы доказательства условий корректности, соответствующие другим путям. Сначала в виде вспомогательной теоремы *cei* доказывается условие корректности для дополнительного инварианта, не зависящего от требований. Затем эта лемма используется при доказательстве условий корректности для дополнительных инвариантов и требований. Данная лемма доказывается путем раскрытия определений условий корректности и дополнительного инварианта, не зависящего от требований, и применения автоматического метода доказательства `force`.

Далее рассмотрим схемы доказательства условий корректности для дополнительных инвариантов, зависящих от требований, и для расширенных инвариантов. Для большинства классов требований применяются универсальные схемы доказательства условий корректности, параметризованные леммами, зависящими от классов требований. Для доказательства условий корректности для дополнительного инварианта схема доказательства имеет следующий вид:

```
theorem extra: "VC extraInv env s input_values"
apply(unfold VC_def extraInv_def)
 apply(rule impI)
  apply(rule conjI)
    using cei apply((auto simp add: VC_def)[1];fastforce)
  apply(unfold commonExtraInv_def)
  apply(erule conjE)+
  apply(erule L8)
   apply(auto split: if_splits)
  done
```

В этой лемме *VC* — доказываемое условие корректности, *extraInv* — дополнительный инвариант, для которого доказывается условие корректности, *env* — ограничение на значения входных переменных, *s* — переменная, обозначающая состояние изменений в начале итерации цикла, *input_values* — переменные в теореме, обозначающие значения входных переменных программы.

В доказательстве сначала раскрываются определения условия корректности и дополнительного инварианта. Первый конъюнкт в заключении условия корректности представляет собой независимый от требований дополнительный инвариант, выполняющийся в конце итерации. Этот конъюнкт доказывается с использованием леммы *cei*. Сначала применяется метод `auto`, затем доказательство завершается методом `fastforce` Второй конъюнкт в заключении условия корректности представляет собой дополнительный инвариант, зависящий от требований. Для его доказательства раскрывается определение дополнительного инварианта, не зависящего от требований, расщепляется конъюнкция в посылке условия корректности, применяется лемма L_8, и доказательство завершается применением метода `auto` с использованием правила расщепления *if_splits*.

Доказательство условий корректности для требований выполняется с помощью следующей схемы:

```
theorem  "VC inv env s0 input_values"
  apply(unfold VC_def inv_def R_def)
  apply(rule impI)
```

```
  apply(rule context_conjI)
  using extra apply((auto simp add: VC_def)[1];fastforce)
  apply(rule conjI)
   apply simp
  apply(unfold extraInv_def commonExtraInv_def)
  apply(erule conjE)+
  apply(auto simp add: L9)
  done
```

В теореме *inv* — инвариант цикла управления, представляющий собой конъюнкцию дополнительного инварианта и требования ($inv(s) \equiv extraInv(s) \wedge R(s)$), R — требование, для которого доказывается условие корректности. Значения других параметров аналогичны значениям одноименных параметров в других леммах.

В доказательстве сначала раскрываются определения условия корректности, инварианта и требования. Затем расщепляется конъюнкция в заключении условия корректности (в инварианте), причем второй конъюнкт доказывается с использованием первого. Доказательство первого конъюнкта, представляющего собой дополнительный инвариант, выполняется с помощью ранее доказанной леммы *extra*, в которой доказано условие корректности для дополнительного инварианта. Сначала применяется метод `auto`, затем доказательство завершается методом `fastforce`. Второй конъюнкт (требование) является конъюнкцией, в которой первый конъюнкт $e(s')$, где s' — состояние изменений в конце итерации, доказывается автоматически методом `simp`. Для доказательства второго конъюнкта, представляющего собой экземпляр некоторого шаблона требований, раскрываются определения дополнительного инварианта *extraInv* и независимого от требований дополнительного инварианта *commonExtraInv*, расщепляется конъюнкция в посылке условия корректности, а затем применяется метод `auto`, которому в качестве дополнительного правила упрощения передается лемма L_9.

Таким образом, все значения параметров предлагаемых схем доказательства однозначно определяются доказываемым условием корректности, требованием и дополнительным инвариантом.

Описанные схемы доказательства условий корректности являются универсальными и могут применяться для различных классов требований. Однако, для доказательства требований, являющихся экземплярами шаблона DRP1, описанного в разделе 5, которые не содержат вложенных шаблонов, нет необходимости определять зависящий от требования дополнительный инвариант, а если такой инвариант не задан, необходимо использовать другую схему доказательства. Рассмотрим схему доказательства для требований класса DRP1.

```
theorem  "VC inv env s input_values"
  apply(unfold VC_def inv_def R_def DRP1_def PRP1_def PRP3_def)
  apply(rule impI)
  apply(rule conjI)
  subgoal
   apply(unfold commonExtraInv_def)[1]
    apply(erule conjE)+
    apply auto
    using substate_toEnvNum_id[of _ s] apply (force+)?
    done
    using cei apply((simp add: VC_def);blast)
    done
```

В данной теореме инвариант *inv* представляет собой конъюнкцию требования и дополнительного инварианта, не зависящего от требований.

В доказательстве сначала раскрываются определения условия корректности, инварианта и требования, а также шаблонов требований. В доказательстве первого конъюнкта заключения условия корректности (инварианта), представляющего собой требование, раскрывается определение допол-

нительного инварианта, не зависящего от требований, и расщепляется конъюнкция в посылке условия корректности. Затем сначала применяется метод `auto`. Но `auto` не доказывает случай, когда события A_2 и A_3 происходят в конце итерации в состоянии s', так как в этом случае необходимо доказать, что событие A_1 произошло в начале итерации в состоянии s. Поэтому далее применяется метод `force` с леммой *substate_toEnvNum_id* из общей теории состояний изменений. Эта лемма имеет вид:

$$s_1 \leqslant s_2 \wedge n(s_1, s_2) = 0 \wedge e(s_1) \wedge e(s_2) \longrightarrow s_1 = s_2$$

и утверждает, что, если после одного из двух состояний изменений, происходящих на одном пути исполнения программы и являющихся точками выхода из итерации цикла управления, произошедшего не позднее второго, до второго прошло время, равное 0, то эти состояния совпадают.

Второй конъюнкт в заключении условия корректности представляет собой независимый от требований дополнительный инвариант, выполняющийся в конце итерации, и доказывается с помощью теоремы *cei*. Сначала применяется метод `simp`, а затем доказательство завершается методом `blast`.

7. Обзор связанных работ

Существует множество методов поиска инвариантов циклов. К ним относятся абстрактная интерпретация [12], метод индукции-итерации [13], методы на основе шаблонов [14], рекуррентный анализ [15], использование неудачных попыток доказательства [16], усиление инвариантов по требованию [17], динамический анализ [18] и машинное обучение [19]. Абстрактная интерпретация и методы на основе шаблонов являются наиболее распространенными методами статического синтеза инвариантов цикла [20]. Рассмотрим наиболее близкие к этой работы по автоматической генерации инвариантов циклов.

Методы генерации инвариантов циклов на основе шаблонов наиболее успешно применяются в области линейной арифметики [21]. В [14] предлагается метод генерации линейных инвариантов циклов на основе шаблонов. Инварианты имеют вид линейных неравенств. Авторы генерируют ограничения на параметры шаблона, которые являются коэффициентами в неравенстве. Эти ограничения гарантируют, что инвариант истинен, когда программа входит в цикл и после итерации, если он был истинен до итерации. Для генерации ограничений используется лемма Фаркаша. Полученные ограничения можно удовлетворить путем элиминации кванторов. Но поскольку элиминация кванторов является вычислительно сложным процессом, авторы упрощают ограничения, используя различные методы. В нашей работе дополнительные инварианты — это не линейные неравенства, связывающие значения переменных программы в одной точке, а формулы, связывающие значения переменных в разные моменты времени и содержащие кванторы по состояниям изменений (в шаблонах). В настоящее время значения параметров шаблона указываются вручную, но наши леммы могут использоваться для генерации ограничений на параметры наших шаблонов. В этом случае ограничения будут бескванторными. Мы планируем исследовать проблему генерации ограничений в дальнейшем.

Для поиска значений параметров шаблона можно использовать SMT-решатели. В [22] предлагается подход, основанный на шаблонах, определяемых пользователями. Авторы сводят задачу поиска значений параметров шаблона к задаче выполнимости формул, что позволяет использовать готовые решатели. Значения параметров шаблона являются не константами, а выражениями и предикатами. Чтобы найти такие значения, авторы делают предположения, которые позволяют им свести эту задачу к поиску констант. Для получения этих значений используется SMT-решатель. В нашей работе значения параметров шаблона являются условными выражениями, включающими не только константы, но и термы, содержащие таймеры процессов. Чтобы свести задачу поиска

таких выражений к поиску констант, мы могли бы рассмотреть экземпляры ограничений для различных путей в программе.

В STeP [23] используются два подхода к генерации инвариантов: восходящий подход, в котором инварианты генерируются с помощью статического анализа программы, и нисходящий подход. В нисходящем подходе для усиления инвариантов используются недоказанные условия корректности. Если какое-либо условие корректности не может быть доказано, вычисляется слабейшее предусловие относительно инварианта, который должен быть доказан, и перехода, которому соответствует условие корректности. Усиленный инвариант является конъюнкцией исходного инварианта, который должен быть доказан, и вычисленного слабейшего предусловия. В нашей работе мы также используем как восходящие, так и нисходящие (т. е. зависящие от требований) инварианты. Мы также могли бы использовать усиление инвариантов. Этот подход необходимо использовать вместе с эвристикой замены константы термом. Однако было отмечено, что дополнительные инварианты, необходимые для доказательства требований, удовлетворяющих одному и тому же шаблону, похожи и также могут быть описаны шаблоном.

Статья [24] посвящена дедуктивной верификации программ, написанных на языке LD из стандарта IEC 61131-3. Темпоральные требования к LD-программам задаются с помощью временных диаграмм. Процесс верификации использует систему дедуктивной верификации Why3. Авторы формализовали инструкции LD как функции в Why3. В этом подходе событие и последующее стабильное состояние временной диаграммы моделируются как цикл в Why3, где тело цикла соответствует одной итерации цикла управления программы на LD, а условие цикла представляет собой условие, которому должен соответствовать вход в момент события и во время стабильного состояния. Проверяемые требования задаются как инварианты этого цикла. Для моделирования последовательностей событий фиксированной длительности вводится счетчик времени, который увеличивается на каждой итерации. Если определенные условия корректности не могут быть доказаны, генерируются контрпримеры для дальнейшего анализа. Поскольку инвариантов недостаточно, авторы используют автоматическую генерацию дополнительных инвариантов цикла. Авторы используют метод абстрактной интерпретации для автоматического создания инвариантов циклов. Ранее разработанный прототип системы Why3 не поддерживает булевы переменные, которые встречаются в программах, представляющих LD-программы и временные диаграммы в Why3. Авторы кодируют булевы переменные как целочисленные переменные с ограничениями, которые позволяют использовать существующие методы для синтеза инварианта цикла с булевыми переменными. В нашей работе мы верифицируем программы на более выразительных процесс-ориентированных языках. Для задания требований мы используем логику первого порядка вместо временных диаграмм. Мы представляем программу как один бесконечный цикл управления, а не как несколько циклов. Мы также отметили, что можно определить шаблоны дополнительных инвариантов для задания вспомогательных свойств и использовать эти шаблоны вместо абстрактной интерпретации.

В статье [25] исследуется метод автоактивной верификации для автоматизации дедуктивной верификации. Этот подход требует от пользователей предоставления дополнительных направляющих аннотаций таких, как утверждения, призрачный код и функции-леммы, для достижения более высокой степени автоматизации доказательства. В результате он позволяет использовать автоматические решатели в случаях, где традиционно использовались интерактивные инструменты доказательства. Авторы реализуют автоактивную верификацию для программ на языке C в рамках Frama-C. В нашем исследовании мы не используем призрачный код и функции-леммы. Однако мы можем включать формализованные требования в качестве аннотаций в программу. Например, инвариант цикла управления *INV* в начале итерации может быть выражен с помощью аннотации `ASSUME INV`, а инвариант в конце итерации может быть представлен с помощью `ASSERT INV`. Кроме

того, мы можем использовать аннотацию ASSERT для добавления дополнительных утверждений в любой точке программы.

В [26] предлагается подход, который позволяет сделать спецификации требований повторно используемыми с помощью объектно-ориентированных концепций. В этом подходе, кроме декларативных спецификаций для задания требований используется подмножество языка программирования. Для задания темпоральных требований используются циклы с инвариантами и оценочными функциями. Авторы выбрали Eiffel в качестве языка программирования. Их подход основан на драйверах спецификаций, которые являются подпрограммами с контрактами, и задают некоторые поведенческие свойства своих формальных параметров через контракты. Затем авторы описывают шаблоны требований, используя классы, называемые бесшовными объектно-ориентированными шаблонами требований. Каждый такой класс содержит драйвер спецификации и отложенные свойства, соответствующие параметрам шаблона требований. Чтобы задать требование с помощью шаблона, создается класс, представляющий требование и называемый «бесшовным объектно-ориентированным требованием», который наследуется от класса, представляющего шаблон требований и реализующий отложенные свойства. В нашей работе мы верифицируем программы на процесс-ориентированных языках, а не на объектно-ориентированных языках. Мы также определяем шаблоны требований, но определяем их в типизированной логике первого порядка и не используем язык программирования для задания требований. Мы задаем темпоральные требования как инварианты цикла управления, который является понятием управляющего программного обеспечения.

В [27] метод на основе шаблонов сочетается с абстрактной интерпретацией и динамическим анализом для генерации инвариантов циклов. Шаблон является булевой комбинацией линейных неравенств. Каждый путь в программе удовлетворяет следующему условию: если соответствующий экземпляр шаблона истинен в начале пути, соответствующий экземпляр шаблона должен быть истинен в конце пути. Это условие транслируется в ограничение. Ограничения являются нелинейными и их трудно решать. Поэтому используется статический и динамический анализ. Тестовые сценарии для динамического анализа генерируются другими инструментами. В динамическом анализе может использоваться конкретное или символическое выполнение. В динамическом анализе переменные программы в шаблонах заменяются их значениями. Это позволяет получить линейные ограничения. Сначала применяется абстрактная интерпретация для генерации некоторых инвариантов, которые являются недостаточными, а затем генерируются другие инварианты нисходящим методом. В нашей работе мы используем только метод на основе шаблонов, не комбинируя его с другими методами. Подобно этой работе, мы используем как независимые от требований инварианты, так и зависимые от требований инварианты.

В статье [28] представлен основанный на шаблонах подход к генерации инвариантов циклов в объединенной теории линейной арифметики и неинтерпретированных функциональных символов. Сначала авторы применяют очистку (purification), т. е. заменяют подтермы, которые являются применением неинтерпретированной функции к выражениям, на новую переменную с сохранением определения этой переменной. Затем генерируются и удовлетворяются ограничения. Мы используем инварианты в теории состояний изменений, возможно, объединенной с теорией арифметики. Используя наши леммы и некоторые эвристики, которые мы планируем разработать, мы могли бы генерировать ограничения, которые не содержат состояний изменений.

Заключение

В этой статье мы представили подход к дедуктивной верификации процесс-ориентированных программ, в котором темпоральные требования задаются с помощью комбинирования базовых шаблонов. В этом подходе определяется набор базовых шаблонов требований. Для каждого такого базового шаблона определяются соответствующий базовый шаблон дополнительных инвариантов

и леммы. Затем базовые шаблоны требований могут комбинироваться для определения производных шаблонов требований. Для каждого производного шаблона требований строятся соответствующий шаблон дополнительных инвариантов и леммы для доказательства условий корректности.

Предложенный в этой статье подход позволит автоматически определять шаблон дополнительных инвариантов и леммы, необходимые для доказательства условий корректности для заданного требования, и доказывать эти леммы. Обобщив ранее разработанные стратегии доказательства условий корректности так, чтобы стратегии были параметризованы соответствующей леммой, мы можем автоматизировать доказательство условий корректности. Таким образом, единственная задача, которая еще не была автоматизирована, — это нахождение значений параметров шаблонов дополнительных инвариантов.

В настоящее время наш набор шаблонов требований содержит 9 базовых шаблонов. Используя их, мы определили 11 универсальных производных шаблонов, определяющих классы, включающие по крайней мере два требования, и 4 специальных производных шаблона. Эти шаблоны позволили нам специфицировать и верифицировать все требования из нашей коллекции, содержащей 76 требований к 12 системам управления. Однако наша система шаблонов в настоящее время не позволяет задавать некоторые классы требований, например, требования, утверждающие, что некоторое событие должно или не должно произойти в течение временного интервала после или до некоторого другого события, которые рассматриваются в [29], а также требования, утверждающие, что некоторое событие не должно произойти после (до) некоторой задержки после (до) некоторого другого события. Также требования, утверждающие, что событие должно произойти k раз, рассматриваемые в [30], не могут быть заданы. Мы планируем расширить нашу систему шаблонов в дальнейшем, чтобы покрыть эти классы требований.

В дальнейшем мы планируем разработать инструменты для генерации производных шаблонов дополнительных инвариантов и лемм, а также скриптов для доказательства условий корректности. Мы также планируем разработать эвристический алгоритм для нахождения значений параметров шаблонов дополнительных инвариантов.

References

[1] V. E. Zyubin, “Hyper-automaton: A model of control algorithms”, in *Proceedings of the Siberian Conference on Control and Communications*, IEEE, 2007, pp. 51–57. DOI: 10.1109/SIBCON.2007.371297.

[2] V. E. Zyubin, A. S. Rozov, I. S. Anureev, N. O. Garanina, and V. Vyatkin, “poST: A process-oriented extension of the IEC 61131-3 structured text language”, *IEEE Access*, vol. 10, pp. 35 238–35 250, 2022.

[3] IEC, *IEC 61131-3: 2013 programmable controllers-Part 3: Programming languages*, https://webstore.iec.ch/publication/4552, International Standard, 2013.

[4] R. Hähnle and M. Huisman, “Deductive software verification: From pen-and-paper proofs to industrial tools”, *Computing and Software Science: State of the Art and Perspectives*, pp. 345–373, 2019.

[5] I. Anureev, N. Garanina, T. Liakh, A. Rozov, V. Zyubin, and S. Gorlatch, “Two-step deductive verification of control software using reflex”, in *Perspectives of System Informatics*, 2019, pp. 50–63. DOI: 10.1007/978-3-030-37487-7_5.

[6] I. Chernenko, I. S. Anureev, N. O. Garanina, and S. M. Staroletov, “A temporal requirements language for deductive verification of process-oriented programs”, in *Proceedings of the IEEE 23rd International Conference of Young Professionals in Electron Devices and Materials (EDM)*, IEEE, 2022, pp. 657–662.

[7] I. Chernenko, “Requirements patterns in deductive verification of process-oriented programs and examples of their use”, *System Informatics*, no. 22, pp. 11–20, 2023.

[8] L. C. Paulson, T. Nipkow, and M. Wenzel, “From LCF to Isabelle/HOL”, *Formal Aspects of Computing*, vol. 31, pp. 675–698, 2019.

[9] E. M. Clarke, T. A. Henzinger, H. Veith, R. Bloem, *et al.*, *Handbook of model checking*. Springer, 2018, vol. 10.

[10] D. Matichuk, T. Murray, and M. Wenzel, “Eisbach: A proof method language for Isabelle”, *Journal of Automated Reasoning*, vol. 56, no. 3, pp. 261–282, 2016. DOI: 10.1007/s10817-015-9360-2.

[11] I. M. Chernenko, I. S. Anureev, and N. O. Garanina, “Requirement patterns in deductive verification of poST programs”, *Modeling and Analysis of Information Systems*, vol. 31, no. 1, pp. 6–31, 2024, in Russian.

[12] P. Cousot and R. Cousot, “Abstract interpretation: A unified lattice model for static analysis of programs by construction or approximation of fixpoints”, in *Proceedings of the 4th ACM SIGACT-SIGPLAN Symposium on Principles of Programming Languages*, 1977, pp. 238–252.

[13] N. Suzuki and K. Ishihata, “Implementation of an array bound checker”, in *Proceedings of the 4th ACM SIGACT-SIGPLAN Symposium on Principles of Programming Languages*, 1977, pp. 132–143.

[14] M. A. Colón, S. Sankaranarayanan, and H. B. Sipma, “Linear invariant generation using non-linear constraint solving”, in *Computer Aided Verification*, Springer, 2003, pp. 420–432.

[15] L. Kovács, “Reasoning algebraically about P-solvable loops”, in *International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, Springer, 2008, pp. 249–264.

[16] J. Stark and A. Ireland, “Invariant discovery via failed proof attempts”, in *International Workshop on Logic Programming Synthesis and Transformation*, Springer, 1998, pp. 271–288.

[17] K. R. M. Leino and F. Logozzo, “Loop invariants on demand”, in *Asian Symposium on Programming Languages and Systems*, Springer, 2005, pp. 119–134.

[18] M. D. Ernst, J. Cockrell, W. G. Griswold, and D. Notkin, “Dynamically discovering likely program invariants to support program evolution”, in *Proceedings of the 21st International Conference on Software engineering*, 1999, pp. 213–224.

[19] X. Si, H. Dai, M. Raghothaman, M. Naik, and L. Song, “Learning loop invariants for program verification”, *Advances in Neural Information Processing Systems*, vol. 31, 2018.

[20] C. A. Furia, B. Meyer, and S. Velder, “Loop invariants: Analysis, classification, and examples”, *ACM Computing Surveys (CSUR)*, vol. 46, no. 3, pp. 1–51, 2014.

[21] J. Breck, J. Cyphert, Z. Kincaid, and T. Reps, “Templates and recurrences: Better together”, in *Proceedings of the 41st ACM SIGPLAN Conference on Programming Language Design and Implementation*, 2020, pp. 688–702.

[22] S. Srivastava, S. Gulwani, and J. S. Foster, “Template-based program verification and program synthesis”, *International Journal on Software Tools for Technology Transfer*, vol. 15, pp. 497–518, 2013.

[23] Z. Manna *et al.*, “STeP: The Stanford temporal prover”, in *Proceedings of the TAPSOFT’95: Theory and Practice of Software Development*, Springer, 1995, pp. 793–794.

[24] C. Belo Lourenço, D. Cousineau, F. Faissole, C. Marché, D. Mentré, and H. Inoue, “Automated formal analysis of temporal properties of Ladder programs”, *International Journal on Software Tools for Technology Transfer*, vol. 24, no. 6, pp. 977–997, 2022.

[25] A. Blanchard, F. Loulergue, and N. Kosmatov, “Towards full proof automation in Frama-C using auto-active verification”, in *NASA Formal Methods Symposium*, Springer, 2019, pp. 88–105.

[26] A. Naumchev, "Seamless object-oriented requirements", in *Proceedings of the International Multi-Conference on Engineering, Computer and Information Sciences (SIBIRCON)*, IEEE, 2019, pp. 0743–0748.

[27] A. Gupta and A. Rybalchenko, "Invgen: An efficient invariant generator", in *Proceedings of the Computer Aided Verification*, Springer, 2009, pp. 634–640.

[28] D. Beyer, T. A. Henzinger, R. Majumdar, and A. Rybalchenko, "Invariant synthesis for combined theories", in *International Workshop on Verification, Model Checking, and Abstract Interpretation*, Springer, 2007, pp. 378–394.

[29] A. Mekki, M. Ghazel, and A. Toguyeni, "Patterns-based assistance for temporal requirement specification", in *Proceedings of the International Conference on Software Engineering Research and Practice (SERP)*, 2011, p. 40 893 006.

[30] M. B. Dwyer, G. S. Avrunin, and J. C. Corbett, "Patterns in property specifications for finite-state verification", in *Proceedings of the 21st International Conference on Software Engineering*, 1999, pp. 411–420.