МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РФ
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«НОВОСИБИРСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»
(НОВОСИБИРСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ, НГУ)

Факультет информационных технологий
Кафедра компьютерных технологий

Направление подготовки 09.03.01 Информатика и вычислительная техника
Направленность (профиль): Программная инженерия и компьютерные науки

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА БАКАЛАВРА

Гетмановой Анастасии Николаевны

Тема работы:

РАЗРАБОТКА ГЕНЕРАТОРА LTL-ФОРМУЛ ИЗ СПИСКА EDTL-ТРЕБОВАНИЙ

«К защите допущена»
Заведующий кафедрой,
д.т.н., доцент
Зюбин В. Е. /……..
(ФИО) / (подпись)
«31» мая 2022 г.

Руководитель ВКР
к. ф-м. н, доцент
КафКТ ФИТ НГУ
Гаранина Н. О. /……...
(ФИО) / (подпись)
«31» мая 2022 г.

Соруководитель ВКР
к.т.н., доцент
КафКТ ФИТ НГУ
Лях Т. В. /……..
(ФИО) / (подпись)
«31» мая 2022 г..

Новосибирск, 2022

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РФ
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«НОВОСИБИРСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»
(НОВОСИБИРСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ, НГУ)

Факультет информационных технологий

Кафедра компьютерных технологий
(название кафедры)

Направление подготовки 09.03.01 Информатика и вычислительная техника

Направленность (профиль): Программная инженерия и компьютерные науки

УТВЕРЖДАЮ

Зав. кафедрой Зюбин В. Е.
(фамилия, И., О.)

…….
(подпись)

«29» октября 2021 г.

ЗАДАНИЕ
НА ВЫПУСКНУЮ КВАЛИФИКАЦИОННУЮ РАБОТУ БАКАЛАВРА

Студент*у(ке)* Гетмановой Анастасие Николавне, группы 18201
(фамилия, имя, отчество, номер группы)

Тема Разработка генератора LTL-формул из списка EDTL-требований
(полное название темы выпускной квалификационной работы)

утверждена распоряжением проректора по учебной работе от 29.10.2021 №0297

Срок сдачи студентом готовой работы 20 мая 2022 г.

Исходные данные (или цель работы):

Разработать транслятор EDTL требований в формулы логики линейного времени LTL.

Структурные части работы:

Введение, Анализ предметной области, Правила упрощения и алгоритм, классификация EDTL требований и эмпирическая проверка на корпусе EDTL требований.

Консультанты по разделам ВКР (при необходимости, с указанием разделов):

……
(раздел, ФИО)

Руководитель ВКР
к. ф-м. н, доцент
КафКТ ФИТ НГУ
Гаранина Н. О. /……..
(ФИО) / (подпись)

«29» октября 2021 г.

Задание принял к исполнению
Гетманова А. Н. /……..
(ФИО студента) / (подпись)

«29» октября 2021 г.

Соруководитель ВКР
к.т.н., доцент
КафКТ ФИТ НГУ
Лях Т. В. /……..
(ФИО) / (подпись)

«29» октября 2021 г.

СОДЕРЖАНИЕ

ОПРЕДЕЛЕНИЯ, ОБОЗНАЧЕНИЯ И СОКРАЩЕНИЯ

LTL — Linear temporal logic, линейная временная логика. Модальная временная логика с модальностями, относящимися ко времени.

EDTL — Event Driven Temporal Logic, собыийная временная логика.

ПЛК — Программируемый логический контроллер.

Model checking — Верификация моделей, Проверка моделей. Метод автоматической формальной верификации параллельных систем с конечным числом состояний, позволяющий проверить, удовлетворяет ли заданная модель системы формальным спецификациям.

ВВЕДЕНИЕ

При разработке управляющего программного обеспечения важной задачей является формулирование и верификация требований к системе. Стоимость программного обеспечения становится основной частью общих затрат при разработке системы управления. Критические свойства системы, такие как безопасность, корректность, надежность и ремонтопригодность, становятся приоритетом в сфере управляющего программного обеспечения. Поэтому первый шаг к обеспечению качества систем — разработка требований к ним.

Для повышения качества разработки критического управляющего программного обеспечения в Институте автоматики и электрометрии СО РАН были разработаны критерии качественных? требований к программному обеспечению управляющего ПО. Для удовлетворения этих критериев был предложен способ формулировки требований с помощью шаблона на основе управляемой событиями темпоральной логики (Event Driven Temporal Logic, EDTL). Предложенная формулировка требований может быть задана в табличной форме. Формальная семантика EDTL требований, определенных с помощью разработанного шаблона, может быть задана с помощью линейной темпоральной логики (Linear Temporal Logic, LTL), логики предикатов первого порядка и конечного автомата Бюхи. В этой работе была использована LTL семантика EDTL требований.

Для проверки качества программного обеспечения относительно EDTL-требований используются формальные средства верификации, такие как верификация моделей (model checking) и дедуктивная верификация. При верификации методом проверки моделей требования формулируются на языке LTL. Ручной перевод EDTL требований в LTL-формулы повышает риск возникновения ошибок в силу наличия человеческого фактора. Для обеспечения корректности перевода была поставлена цель разработки транслятора EDTL требований в формулы логики линейного времени LTL.

Для достижения этой цели были поставлены следующие задачи:

1. Проанализировать специфику EDTL требований и семантику LTL формул.
2. Сформулировать требования к разрабатываемому транслятору.
3. Определить правила преобразования EDTL требований в упрощенные LTL формулы.
4. Предложить алгоритм трансляции EDTL требований в LTL формулы.
5. Реализовать предложенный алгоритм.
6. Классифицировать EDTL требования на основе результатов работы транслятора.

7. Эмпирически проверить корректность работы транслятора на корпусе EDTL требований.

В результате работы был разработан метод преобразования EDTL-требований в формулы логики линейного времени LTL, реализован алгоритм трансляции EDTL требований в формулы логики линейного времени LTL и получена классификация EDTL требований. Разработанные алгоритм и классификация были применены для анализа требований EDTL, разработанных для управляющего ПО.

Алгоритм трансляции требований EDTL требований в формулы LTL позволит автоматизировать перевод EDTL-требований в LTL-формулы для целей верификации методами проверки моделей и дедуктивной верификации. Позволит создавать бесшовные наборы инструментов, которые минимизируют возможность внесения ошибок при обработке информации пользователем. Классификация, полученная в результате работы алгоритма, может быть использована для других преобразований EDTL требований, например, для преобразования EDTL требований в требования на естественном языке. Результаты классификации также могут быть использованы для диагностики ошибок при определении требований, разработки канонической формы требований и определения значений атрибутов по умолчанию.

Работа изложена в трех главах. В первой главе анализируется инженерия требований, специфика синтаксиса и семантики логик LTL и EDTL. Вторая глава посвящена вопросам реализации алгоритма трансляции EDTL требований в LTL формулы. В третьей главе описываются классификация EDTL требований и анализ требований EDTL, разработанных для управляющего ПО.

ОСНОВНАЯ ЧАСТЬ

1. Анализ предметной области

Для создания надежного управляющего программного обеспечения для киберфизических систем необходима тщательная разработка требований к ним [7]. Как правило, эта разработка должна обеспечивать однозначность и понятность требований для всех участников процесса (заказчиков, разработчиков, инженеров по качеству). И возможность формально выражать требования в логической форме необходима для дальнейшей автоматизированной верификации.

В Институте Автоматики и электрометрии разработан подход к определению таких требований, который позволит формулировать их семантику в темпоральной логике и логике первого порядка и далее применять формальные методы верификации, такие как проверка модели и дедуктивная верификация. На основе этого подхода ведется разработка IDE для работы с EDTL требованиями: редактор EDTL требований, транслятор EDTL требований в LTL, транслятор EDTL требований в естественный язык, транслятор в конечный автомат и графическое представление EDTL формул. Также ведется работа над программными шаблонами для управляющего ПО программируемых логических контроллеров (ПЛК) и шаблонами для требований к ним.

Была предложена идея описания семантики программ ПЛК в виде циклов чтения и записи входных и выходных переменных, при этом внутреннее функционирование рассматривается как черный ящик [4]. В случае паттернов требований было получено табличное представление ключевых аспектов организации промышленной программы и представлено в виде паттерна событийно-управляемой темпоральной логики (EDTL) [9]. Такое представление позволяет разложить сложное требование на части общего шаблона, причем каждая его часть может быть формулой со специальными терминами. Эти специальные термины определяют изменение фронта сигнала и сочетаются с логическими терминами и временными ограничениями.

Формальная семантика EDTL требований в частности может быть задана с помощью линейной темпоральной логики, которая в дальнейшем может быть использована для проверки модели системы управления с помощью метода верификации модели [3].

Далее в этой главе представлены предварительные сведения о линейной темпоральной логике LTL и управляемой событиями темпоральной логике EDTL, а также важное наблюдение, которое легло в основу разработанного алгоритма трансляции.

1.1 Синтаксис и семантика LTL

Linear Temporal logic (LTL), линейная темпоральная логика – это модальная временная логика, с модальностями относящимися ко времени. Она позволяет формулировать свойства исполнимых вычислительных последовательностей системы. LTL состоит из обычной пропозициональной логики, расширенной темпоральными операторами: унарный оператор X (Next, «следующий момент времени») и бинарный U (Until, «пока»).

Пусть AP – множество атомарных предложений. Тогда базовыми формулами, которые могут быть выражены в LTL, являются:

1. p для всех $p \in AP$;
2. если φ – формула, то φ – формула;
3. если φ и ψ – формулы, то $\varphi \vee \psi$ – формула;
4. если φ – формула, то X φ – формула;
5. если φ и ψ – формулы, то φ U ψ – формула.

Множество формул, построенных в соответствии с этими правилами, называется формулами LTL.

Дизъюнктивный базис логики высказываний создают булевы операторы отрицание и дизъюнкция, через них могут быть выражены любые другие булевы операторы. Темпоральный базис LTL образуют пара темпоральных операторов **U**, **X**, через которые можно выразить выводимые операторы **F** (Future, «когда-нибудь») и **G** (Globally, «всегда»). **F**p можно считать сокращением для (*true* **U** p). **G**p можно считать сокращением для ¬**F**(¬p). Так как *true* верно во всех состояниях, формула **F**p в действительности означает, что *p* выполняется в какой-то момент в будущем. Предположим, что нет момента в будущем, когда выполняется *p*. Тогда *p* выполняется все время. Это объясняет определение **G**p.

Более слабый вариант оператора Until, оператор Weak until (unless) **W** утверждает, что p выполняется непрерывно либо до момента, когда впервые будет выполняться q, либо в течение всей последовательности. Оператор **W** определяется следующим образом: p **W** q = **G**p $\wedge$ (p **U** q). Оператор **U** можно выразить через **W** двойственным способом: p **U** q ≡ **F**q $\wedge$ (p **W** q).

Можно утверждать, что формула без темпорального оператора (**X, F, G, U**) на «верхнем уровне» относится к текущему состоянию, формула **X**p – к следующему состоянию, **G**p – ко всем будущим состояниям, **F**p – к некоторому будущему состоянию, а **U** – ко всем будущим состояниям до тех пор, пока определенное условие не станет верным.

Приоритет на темпоральных операторах вводится следующим образом: унарные операции имеют более высокий приоритет, чем бинарные, темпоральные операторы более высокий приоритет, чем булевы операторы.

Семантика формул LTL определяется на вычислениях, то есть формула принимает истинное или ложное значение на бесконечных цепочках состояний, в каждом из которых предикаты имеют конкретное истинностное значение – *true* или *false*. Любая формула линейной темпоральной логики на конкретном вычислении будет либо истинна, либо ложна. Формула считается истинной на вычислении, если она истинна в начальном его состоянии.

Формальный смысл свойств в темпоральной логике определяется в терминах модели.

Пусть AP множество атомарных высказываний (булевых выражений над множеством переменных, констант и предикатных символов). Моделью Крипке назовем четверку $M = (S, I, R, L)$ состоящую из:

1. конечного множества состояний S;
2. множества начальных состояний $I \subseteq S$;
3. отношения перехода$R \subseteq S \times S$, где $\forall s \in S, \exists s' \in S,$ такое что $(s, s') \in S$;
4. функции пометок L: $S \rightarrow 2^{AP}$.

Семантика LTL определяется следующим образом. Пусть p $\in$ AP – атомарное предложение, $M = (S, R, Label)$ структура Крипке, $s \in S$ и φ, ψ – LTL-формулы. Отношение выполнимости формулы на конкретном состоянии модели задается следующим образом:

1. $s \vDash p \leftrightarrow p \in Label(s)$;
2. $s \vDash \neg\psi \leftrightarrow \neg(s \vDash \psi)$;
3. $s \vDash (\varphi \vee \psi) \leftrightarrow (s \vDash \varphi) \vee (s \vDash \psi)$;
4. $s \vDash X\psi \leftrightarrow R(s) \vDash \psi$;
5. $s \vDash (\varphi\ U\ \psi) \leftrightarrow \exists j \geq 0 :\ Rj\ (s) \wedge \vDash \psi (0 \leq k\ <\ j :\ Rk(s) \vDash \varphi)$

Формальная интерпретация остальных связок может быть получена теперь из определений непосредственной подстановкой.

Использованием вышеописанной семантики можно вывести тавтологичность формул LTL, то есть громоздкие формулы линейной темпоральной логики можно свести к более коротким простой подстановкой семантики и преобразованием. Однако манипуляция семантическими формулами вручную трудоемка и ненадежна. Более эффективный путь проверять правильность формул – это использовать их синтаксис вместо их семантики. То есть переписывать LTL-формулы в семантически эквивалентные LTL-формулы на синтаксическом уровне. Семантическая эквивалентность означает, что для всех состояний всех моделей эти правила

выполняются

1.2 Синтаксис и семантика EDTL

EDTL требование – это комбинация событий системы, ограниченных заданными временными взаимосвязями. Эти события являются атрибутами требования EDTL. EDTL требование R это кортеж, состоящий из следующих EDTL атрибутов:

$$R = (trigger, invariant, final, delay, reaction, release) \quad (1.1)$$

- *trigger* – событие, после которого инвариант должен быть истинным, пока не произойдет событие *release* или *reaction*; это событие также является отправной точкой тайм-аутов для создания событий *final/release* (если есть);
- *invariant* – утверждение, которое должно быть истинным с момента возникновения триггерного события до момента *release* или *reaction;*
- *final* – событие, после которого должна произойти реакция с допустимой задержкой; это событие всегда следует за триггером;
- *delay* – лимит времени после финального события, в течение которого должна появиться реакция; событие реакции должно случиться в пределах допустимой задержки с момента финального события;
- *reaction* – требование считается выполненным в момент, когда произошла реакция;
- *release* – событие отмены требования.

Следующее описание семантики EDTL требований на естественном языке соответствует вышеизложенному неформальному определению: после каждого триггерного события инвариант должен оставаться истинным до события отмены или финального события. Инвариант также должен оставаться истинным после финального события до события отмены или реакции, и, кроме того, реакция должна иметь место в пределах указанной допустимой задержки от финального события.

Значение каждого атрибута представляет собой EDTL формулу, построеную из атомарных предложений, стандартных логических операторов и операторов «событий» для описания событий, которые могут происходить или не происходить. EDTL формулы делятся на два класса: формулы состояний и формулы событий. Неформально формулы состояния утверждают о значениях системных переменных в данный момент времени, а формулы событий утверждают о событиях, которые происходят или не происходят, то есть об изменении или сохранении значений переменных с предыдущего момента времени.

Определим *EDTL формулы*. Пусть p — утверждение, ϕ и ψ EDTL-формулы. Тогда:

- формулы состояний:

- true, false и p — атомарные EDTL-формулы;
- $\phi \wedge \psi$ — конъюнкция ϕ и ψ;
- $\phi \vee \psi$ дизъюнкция ϕ и ψ;
- $\neg\phi$ — отрицание ϕ;

- формулы событий с утверждением p:
- \ p — падающий фронт: значение p меняется с истинного на ложное;
- / p — нарастающий фронт: значение p изменяется с ложного на истинное;
- _p — низкое установившееся состояние: значение p остается ложным;;
- ~p — высокое установившееся состояние: значение p остается истинным.

Неформально семантика формул состояния является стандартной семантикой для LTL-формул без темпоральных операторов, но семантика формул событий использует выполнимость атомарного утверждения в предыдущем состоянии системы. Эту семантику можно смоделировать с помощью темпорального оператора $-X$ (предыдущий момент) или путем введения в модель специальной призрачной переменной $prev(p)$, которая сохраняет предыдущее значение утверждения p. В последнем случае семантика формул событий сводится к семантике EDTL-формул состояния следующим образом:

- /p ≡ ¬prev(p) $\wedge$ p;
- \p ≡ prev(p) $\wedge$ ¬p;
- ~ p ≡ prev(p) $\wedge$ p;
- _p ≡ ¬prev(p) $\wedge$ ¬p.

Определим LTL-семантику для EDTL-требований. Требование EDTL R, состоящее из EDTL-атрибутов триггер, инвариант, окончание, задержка, реакция и выпуск, которые являются формулами состояний, выполняется в системе управления C, если и только если следующая формула LTL Φ_Rвыполняется в структуре Крипке MC для каждого начального пути:

$$\Phi R = G(trig \wedge \neg rel \rightarrow inv \wedge (G(inv \wedge \neg fin) \vee$$
$$\vee (inv \wedge \neg fin)\ U\ (rel \vee (fin \wedge \quad (1.1)$$
$$(inv \wedge \neg del)\ U\ (rel \vee rea \wedge inv)))))$$

В этой формуле переменные *trig, inv, fin, del, rea* и *real* являются значениями соответствующих EDTL-атрибутов. EDTL-требование удовлетворяется в системе управления, если его LTL — семантика удовлетворяется в этой модели системы управления. Значения EDTL атрибутов могут принимать тождественно истинное, тождественно ложное значение или иметь не постоянное значение в различных состояниях модели системы управления. Из-за особенностей спецификации управляющего программного обеспечения,

значения всех атрибутов EDTL требования, за исключением реакции, оцениваются сразу после чтения ввода, а значение реакции оценивается непосредственно перед записью вывода.

Было обнаружено, что при подстановке постоянных значений атрибутов в эту формулу, она значительно сокращается за счет применения правил эквивалентности для LTL формул. Более того, одним и тем же результирующим формулам могут соответствовать различные комбинации возможных вариантов значений атрибутов. Это наблюдение лежит в основе алгоритма трансляции EDTL требований в LTL формулы.

1.3 Требования к транслятору EDTL требований в LTL формулы

В результате анализа предметной области, были выдвинуты следующие требования к разрабатываемому транслятору:

- Должна быть обеспечена возможность интеграции в IDE работы с EDTL требованиями
- При разработке прототипа транслятора необходимо обеспечить:
  - Использование консольного режима
  - Описание EDTL требования в формате .csv
  - Фиксацию результата в формате .csv
- Трансляция должна основываться на LTL семантике EDTL требований
- В случае константных логических значений атрибутов EDTL требований транслятор должен генерировать упрощенную LTL формулу.

2. Правила упрощения и алгоритм

2.1 Алгоритм трансляции

Значения атрибутов EDTL требования могут быть тождественно истиными, тождественно ложными или непостоянными в разных состояниях модели системы управления. Было замечено, что при постоянных значениях атрибутов EDTL требования, LTL семантика требования значительно упрощается. Идея алгоритма заключается в упрощении LTL семантики EDTL требования за счет подстановки постоянных значений атрибутов.

На вход алгоритму подается набор значений атрибутов, которые могут быть тождественно истинными, тождественно ложными или иметь не постоянное значение. Результатом работы алгоритма является упрощенная LTL семантика с этими входными значениями атрибутов. Алгоритм подставляет значения входных атрибутов в основную формулу ΦR семантики EDTL, а затем уменьшает эту формулу, применяя стандартные и специально разработанные правила эквивалентного упрощения LTL формул. Это упрощение идет шаг за шагом от семантической формулы изнутри, используя упрощенные подформулы для сокращения их надформул.

Основная функция алгоритма представляет собой поочередный вызов функций операторов LTL семантики EDTL требования 1.2. Промежуточные функции алгоритма являются набором функций этих операторов: дизъюнкция, конъюнкция, импликация, отрицание, темпоральные операторы Until, Globally, Future. Функции операторов вычисляют значение подформулы семантики, применяя тот или иной оператор к значениям переменных. Это вычисление основано на правилах упрощения, описанных в следующем разделе. Каждая функция оператора реализует набор правил упрощения, срабатывающих в случае вхождения определенных типов переменных. Результатом работы функции оператора является упрощенная подформула LTL семантики EDTL требования.

```
     EDTL2LTL(trig, rel, fin, del, inv, rea) = (
2 let f0 = con(invariant, reaction);
3 let f1 = dis(release, f0);
4 let f2 = con(invariant, no(delay));
5 let f3 = Until(f2, f1);
6 let f4 = con(final, f3);
7 let f5 = dis(rel, f4);
8 let f6 = con(invariant, no(final));
9 let f7 = Until(f6, f5);
10 let f8 = Globally(invariant, no(final));
```

```
let f9 = dis(f8, f7);
let f10 = con(inv, f9);
let f11 = con(trigger, no(release));
let f12 = impl(f11, f10);
Globally(f12)
)
```

Листинг 2.1 – Псевдокод основной функции алгоритма.

2.2 Правила упрощения

Формальными основаниями для правил упрощения являются эквивалентности формул линейной темпоральной логики. Две формулы LTL эквивалентны, если они имеют одинаковую семантику. Правило может быть использовано, если на текущем шаге упрощения подформула семантики соответствует левой части этой эквивалентности. В этом случае алгоритм упрощения следует правилу и использует правую часть этой эквивалентности для следующего шага упрощения (если таковой имеется). Например, если в функцию оператор Globally входит переменная *true*, то функция возвращает значение оператора **G**(*true*) равное *true*.

Все эквивалентности, используемые для правил упрощения в программе, представлены в Таблицах 2.1 и 2.2. Таблица 2.1 содержит стандартные правила упрощения для формул и констант, извлеченные из литературы [1, 2]. При этом, перечислены только те из них, которые действительно упрощают семантику EDTL Ф*R*. Эти правила были дополнены специально разработанными правилами упрощения LTL семантики EDTL требования.

Правила для констант		**Правила для формул**
$\varphi \vee true \equiv true$	false **U** $\psi \equiv \psi$	$\varphi \vee \varphi \equiv \varphi$
$\varphi \wedge true \equiv \varphi$	true **U** $\psi \equiv$ F ψ	$\varphi \vee (\psi \wedge \varphi) \equiv \varphi$
$\varphi \vee false \equiv \varphi$	φ **U** false ≡ false	**G**(**G**(φ)) ≡ **G**(φ)
$\varphi \wedge false \equiv false$	φ **U** true ≡ true	$\varphi \vee (\psi \wedge (\varphi \vee \xi)) \equiv \varphi \vee (\psi \wedge \xi)$
G true ≡ true	false **U** true ≡ true	$\varphi \vee (\varphi \vee \psi) \equiv \varphi$ V ψ
G false ≡ false	true **U** false ≡ false	φ **U** $\varphi \equiv \varphi$
F false ≡ false	false **U** false≡ false	$\neg\varphi$ **U** $\varphi \equiv$ **F**(φ)
F true ≡ true	true **U** true ≡ true	$(\varphi \wedge \psi)$ **U** $\varphi \equiv \varphi$

Таблица 2.1 – Стандартные эквивалентности

В таблице 2.2 перечислены эквивалентности, которые являются особыми для упрощения формулы семантики EDTL. Специально разработанные правила упрощения представляют собой упрощения, в которых несколько шагов преобразования объединены в один шаг, или специальные преобразования основанные на специфике EDTL требования. Например, формула $(\varphi \vee F(\varphi))$ в действительности означает, что φ выполняется или выполняется в какой-то момент в будущем. Такая формулировка не обязательна, достаточно знать, что в какой-то момент в будущем выполняется φ. Поэтому этой формуле в соответствие ставится $F(\varphi)$. Все специальные правила могут быть доказаны с помощью семантики логики линейного времени LTL.

Утверждение 1. *Для каждой LTL-формулы* φ*, каждой структуры Крипке* $M = (S, S0, R, L)$ *и каждого пути* π *из этой структуры выполняется следующее:*

$M, \pi \models \varphi \vee \mathbf{F}(\varphi)$ iff $M, \pi \models \mathbf{F}(\varphi)$.

Доказательство:

$M, \pi \models \varphi \Longrightarrow M, \pi \models \mathbf{F}(\varphi)$ и $M, \pi \models \mathbf{F}(\varphi) \Longrightarrow M, \pi \models \varphi \vee \mathbf{F}(\varphi)$.

Следовательно, $M, \pi \models \varphi \vee \mathbf{F}(\varphi) \Longleftrightarrow M, \pi \models \mathbf{F}(\varphi)$.■

Разработанные правила
G(G(φ) ∨ (φ U ψ)) ≡ **G**(φ ∧ **F**(ψ))
φ ∨ (ψ **U** φ) ≡ (ψ **U** φ)
φ ∨ (ψ **U** (φ ∨ ξ)) ≡ (ψ **U** (φ ∨ ξ))
G(¬φ) ∨ **F**(φ) ≡ true
G(¬φ) ∨ (**F**(φ) ∨ (c)) ≡ (c)
¬φ **U** (ψ ∨ φ) ≡ **F**(φ) ∨ (¬φ **U** ψ)
(φ ∧ ψ) **U** (ξ ∨ φ) ≡ φ ∨ ((φ ∧ ψ) **U** ξ)
φ **U** (ψ ∨ φ) ≡ φ ∨ (φ **U** ψ)
(φ ∧ ¬ψ) **U** (ψ ∧ φ) ≡ φ **U** (ψ ∧ φ)
φ ∨ **F**(φ ∨ ψ) ≡ **F**(φ ∨ ψ)
G(**G**(φ ∧ ¬ψ) ∨ (φ **U** (ψ ∧ φ))) ≡ **G**(φ)
φ ∨ **F**(φ) ≡ **F**(φ)
G((φ ∧ ψ) **U** (φ ∧ ξ)) ≡ **G**(φ ∧ (ψ **U** ξ))
G(**G**(φ ∧ ¬ψ) ∨ ((φ ∧ ¬ψ) **U** (ψ ∧ (φ ∧ ξ)))) ≡ **G**(φ ∧ (**G**(¬ψ) ∨ **F**(ψ ∧ ξ)))
G(**G**(φ ∧ ¬ψ) ∨ ((φ ∧ ¬ψ) **U** (ψ ∧ (φ **U** (φ ∧ ξ))))) ≡ **G**(φ ∧ (**G**(¬ψ) ∨ **F**(ψ ∧ **F**(ξ))))

Таблица 2.2 – Специально разработанные эквивалентности для упрощения LTL-семантики EDTL требования

2.3 Реализация алгоритма

Реализация алгоритма трансляции EDTL требований в LTL формулы представляет собой функцию операторного представления LTL семантики EDTL требования и набор функций, которые возвращают результат пропозициональных и темпоральных операторов, с применением правил преобразования, описанных выше. Программа написана на языке программирования Python, так как он поддерживает функциональное программирование и является высокоуровневым языком общего назначения с динамической строгой типизацией, что позволяет заранее не указывать типы входящих в функции и выходящих из них переменных.

На вход алгоритму подается таблица формата .csv всех возможных наборов значений атрибутов EDTL требования, которая представлена в [11]. При этом было задано три вида значений переменных: *true*, *false* и некоторое строковое значение в зависимости от атрибута, которому оно присвоено — ‘rel’, ‘del’, ‘fin’, ‘rea’, ‘inv’. Значение атрибута *trigger* присваивается в программе. Программа считывает значения набора атрибутов из таблицы и инициализирует переменные: *release*, *delay*, *final*, *reaction*, *invariant*. Программа присваивает атрибуту *trigger* значение ‘trig’ и вызывает основную функцию алгоритма. Затем программа присваивает атрибуту trigger значение true и вызывает основную функцию алгоритма. И, наконец, программа присваивает атрибуту *trigger* значение *false* и задает значение результирующей переменной – *true*, поскольку при *trigger* равном *false* EDTL требование всегда тождественно истинно. Затем программа записывает в таблицу набор значений атрибутов EDTL требования и следующие три столбца заполняются соответствующими LTL формулами, упрощенными алгоритмом, при *trigger* равном ‘trig’, *trigger* равном *true* и *trigger* равном *false*. Далее программа считывает из таблицы, поданной на вход, очередной набор атрибутов требования и проделывает все то же самое с ним. Цикл продолжается до конца таблицы.

Результатом работы программы является таблица, в строках которой содержатся все возможные наборы значений атрибутов EDTL требования с соответствующей LTL семантикой, упрощенной алгоритмом, при *trigger* равном ‘trig’, *trigger* равном *true* и *trigger* равном *false*.

Как уже было сказано в предыдущем разделе, основная функция алгоритма представляет собой поочередный вызов функций для темпоральных и логических операторов из LTL семантики EDTL требования 1.2.

Каждая такая функция представляет собой набор правил упрощения, срабатывающих в случае вхождения определенных типов переменных. На вход этой функции подаются переменные значений атрибутов, необходимых для выполнения операции. Выходом функции является упрощенная промежуточная LTL формула. На Листинге 2 можно увидеть пример функции для пропозиционального оператора конъюнкция.

```
    def conSimpl(a, b):
if (a == False) or (b == False): return False;
elif (a == True): return b;
elif (b == True): return a;
else: return '('+a+' ∧ '+b+')';
```

Листинг 2.2 – Оператор конъюнкция

Функция оператора конъюнкция содержит только стандартные правила для констант.

На Листинге 3 приведен отрывок кода оператора Until с применением как стандартных, так и специальных правил преобразования $a\ \mathbf{U}\ a = a$, $\neg a\ \mathbf{U}\ a = \mathbf{F}(a)$ и $\neg a\ \mathbf{U}\ (b \vee a) = \mathbf{F}(a) \vee (\neg a\ \mathbf{U}\ b)$.

```
    def UntilSimpl(a, b):
  <...>
#a U a = a
elif (a == b):
  rules[34][1] +=1;
  return a;
# ¬a U a = F(a)
elif((a.find('¬') == 0) and (b == a[1:])):
  rules[35][1] +=1;
  return FutureSimpl(b);
#¬a U (b ∨ a) = F(a) ∨ (¬a U b)
elif ((a[0] == '¬') and (b.find ('∨ ' + a[1:]) > 0)):
  rules[42][1]+=1;
  b1 = b[1:(b.find ('∨ ' + a[1:]))-1];
  return disSimpl(FutureSimpl(a[1:]), UntilSimpl(a, b1));
  <...>
else: return '('+a+' U '+b+')'
```

Листинг 2.3 – Отрывок кода оператора Until

В процессе работы алгоритма подсчитывалась частота использования правил упрощения и была получена статистика их применения (Таблица 2.3). Изначально было разработано больше правил упрощения. Такая статистика помогла исключить неиспользуемые правила. Полный листинг программы представлен в Приложении А.

Правило упрощения	**частота**
G(G(a) ∨ (a U b)) = G(a ∧ F(b))	28
a ∨ a = a	22
a ∨ (b ∧ a) = a	22
a ∨ (b U a) = (b U a)	20
G(G(a)) = G(a)	15
a ∨ (b U (a ∨ c)) = (b U (a ∨ c))	12
a ∨ (b ∧ (a ∨ c)) = a ∨ (b ∧ c)	6
G(¬a) ∨ F(a) = true	6
a ∨ (a ∨ b) = a V b	6
a U a = a	6
¬a U a = F(a)	6
G(¬a) ∨ (F(a) ∨ (c)) = (c)	6
¬a U (b ∨ a) = F(a) ∨ (¬a U b)	6
(a ∧ b) U (c ∨ a) = a ∨ ((a ∧ b) U c)	6
(a ∧ b) U a = a	6
a U (b ∨ a) = a ∨ (a U b)	6
(a ∧ ¬b) U (b ∧ a) = a U (b ∧ a)	6
a ∨ F(a ∨ b) = F(a ∨ b)	4
G(F(a)) = GF(a)	3
G(G(a ∧ ¬b) ∨ (a U (b ∧ a))) = G(a)	3
a ∨ F(a) = F(a)	2
G((a ∧ b) U (a ∧ c)) = G(a ∧ (b U c))	1
G(G(a ∧ ¬b) ∨ ((a ∧ ¬b) U (b ∧ (a U (a ∧ c))))) = G(a ∧ (G(¬b) ∨ F(b ∧ F(c))))	1
G(G(a ∧ ¬b) ∨ ((a ∧ ¬b) U (b ∧ (a ∧ c)))) = G(a ∧ (G(¬b) ∨ F(b ∧ c)))	1
G(G(a) ∨ (a U b)) = G(a ∧ F(b))	28

Таблица 2.3 – Частота применения правил упрощения для LTL семантики EDTL требования

3. Классификация EDTL требований и эмпирическая проверка на корпусе EDTL требований

3.1 Классификация EDTL требований

Алгоритм трансляции был использован для классификации требований. Требования относятся к одному классу если их LTL семантика, упрощенная алгоритмом, является одной и той же формулой LTL. В таблице 3.1 приведен пример различных требований, попадающих в один класс. Этот пример показывает, что приведенные требования не зависят от атрибута *delay*.

trig	rel	del	fin	rea	inv	LTL формула
trig	rel	true	fin	rea	inv	G(trig → (¬fin U rel))
trig	rel	false	fin	rea	inv	G(trig → (¬fin U rel))
trig	rel	del	fin	rea	inv	G(trig → (¬fin U rel))

Таблица 3.1 – Требования попадающие в один класс

Классификация содержит 80 одноэлементных классов (Приложение В) и 15 многоэлементных классов (Таблица 3.2) , а также классы с тривиальной семантикой *true* и *false.* Детали классификации можно посмотреть в [11]. Вес класса представляет собой количество требований входящих в данный класс. На Рисунке 3.1 представлено распределение классов с нетривиальной семантикой.

Номер класса	Формула класса	Вес класса
#1	G(rel)	29
#2	G(inv)	15
#3	G(¬fin)	3
#4	G(¬trig)	33
#5	G(trig → rel)	29
#6	G(¬fin U rel)	3
#7	G(inv ∧ ¬fin)	3
#8	G(trig → inv)	3
#9	G(inv ∧ F(rel))	9
#10	G(trig → G(inv))	9
#11	G(trig → G(¬fin))	3
#12	G(trig → (¬fin U rel))	3
#13	G(trig → G(inv ∧ ¬fin))	3
#14	G(trig → (G(inv) ∨ (inv U rel)))	9
#15	G(trig → (G(inv ∧ ¬fin) ∨ (inv U (fin ∧ inv))))	3

Таблица 3.2. – Нетривиальные классы

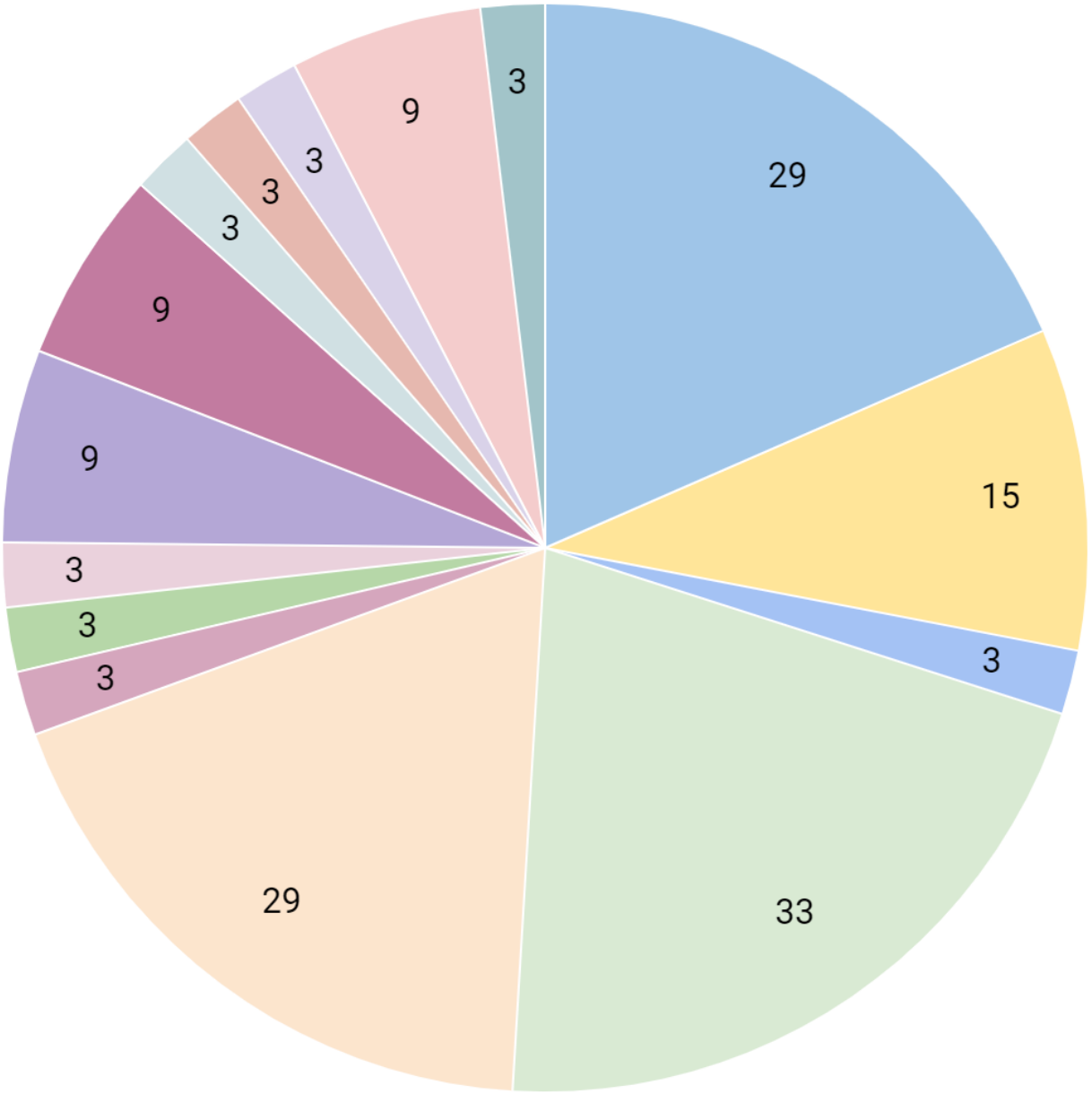

Рисунок 3.1 – Распределение многоэлементных классов

Самыми большими классами являются классы с тривиально истинной и тривиально ложной семантикой. Класс с тривиально истинной семантикой содержит 459 представителей, а класс с тривиально ложной семантикой — 33 представителя. Эти классы включают в себя тождественно истинные или тождественно ложные требования, которые можно считать бессодержательными. Они не зависят от значений не константных атрибутов. Планируется, что IDE для EDTL требований, разрабатываемая в Институте Автоматики и Электрометрии, будет выдавать предупреждение, если сформулированное требование относится к этим классам.

Самый большой значимый класс #4 с семантикой, представленной формулой LTL, G(¬trig) содержит 33 представителя. Требования этого класса описывают инварианты поведения системы управления.

Семантика классов #1 G(rel), #2 G(inv), #3 G(¬fin) и несколько других имеют одинаковую темпоральную структуру. Следовательно, эти классы также могут задавать инварианты поведения. Поэтому можно объединить их в один класс относительно темпоральной структуры их семантики. Это уточнение может быть полезно для определения канонической формы требований EDTL.

Например, определение инвариантных свойств системы выглядит более естественным с требованиями класса #2, имеющими семантику G(inv).

Большое количество одноэлементных классов (80 элементов) можно объяснить сложностью семантической формулы LTL. Однако уточнение текущей классификации относительно темпоральной структуры семантики может уменьшить это число.

Также уточнить классификацию можно с помощью следующих Логических правил. Пусть непостоянные значения атрибутов требования совпадают или отрицают друг друга. В этом случае семантические формулы разных классов текущей классификации могут быть одинаковыми. Например, если trig = inv, то требования класса #8 G(trig → inv) сливаются с тривиальный класс # 18 (true), и если trig = ¬inv, то они сливаются с нетривиальным классом # 2 (G(inv)).

Результаты классификации могут быть использованы для диагностики ошибок при определении требований, разработки канонической формы требований, определения значений атрибутов по умолчанию и разработки методов для объяснения требований EDTL на естественном языке.

3.2 Эмпирическая проверка корректности работы транслятора на корпусе EDTL требований

Достаточность правил упрощения и корректность работы алгоритма трансляции, тестировались посредством генерации упрощенных LTL формул для EDTL требований из корпуса EDTL требований.

Проверка работоспособности алгоритма проводилась на следующих тестовых примерах систем управления: трехэтажный лифт, санитайзер, сушилка для рук, турникет, шлюз для воды и робот-пылесос. Было проанализировано около 40 требований [11].

Тестовый пример представляет собой таблицу с набором требований на естественном языке и в виде EDTL требования (Таблица 3.3). Для каждого требования были:

- сгенерирована упрощенная LTL формула с помощью алгоритма трансляции,
- поставлен в соответствие класс, согласно полученной классификации,

- представлен общий вид формулы класса (Таблица 3.4).

NL	trigger event	release event	final event	allowable delay	invariant condition	reaction
R1	*H.RE* ∧ ¬*D*	false	true	true	true	*D′*
R2	*H* ∧ *D*	*D.H >= #T1h*	true	true	true	*D′*
R3	¬H ∧ ¬D	false	true	true	true	¬D'
R4	*D* ∧ *H.FE*	*H*	*passed(1s)*	true	*D*	¬D'
R5	*D.RE*	¬D	*passed(1h)*	true	true	¬D'

Таблица 3.3 – Тестовый пример Сушилка для рук

R1: If the dryer (D) was not turned on and hands (H) appeared, it will turn on ASAP.

R2: If the hands (H) are present and the dryer (D) is on, it will not turn off.

R3: If there is no hands (H) and the dryer (D) is not turned on, the dryer will not turn on until the hands appear.

R4: If the dryer (D) is on, then it turns off after no hands (H) are present for 1 second.

R5: The time of continuous work of dryer (D) is no more than an hour.

NL	LTL formula	общий вид формулы класса	класс
R1	G((H.RE ∧ ¬D) → D*)	G(trig → rea)	#1A
R2	G((H ∧ D) → ((D.H >= #T1h) ∨ D*))	G(trig → (rel ∨ rea))	#5A
R3	G((¬H ∧ ¬D) → ¬D*)	G(trig → rea)	#1A
R4	G((D ∧ H.FE) → (G(D ∧ ¬passed(1s)) ∨ ((D ∧ ¬passed(1s)) U (H ∨ (passed(1s) ∧ (D ∧ ¬D*)))))))	G(trig → (G(inv ∧ ¬fin) ∨ ((inv ∧ ¬fin) U (rel ∨ (fin ∧ (inv ∧ rea))))))	#16A
R5	G(D.RE → (G(¬passed(1h)) ∨ (¬passed(1h) U (¬D ∨ (passed(1h) ∧ ¬D*))))))	G(trig → (G(¬fin) ∨ (¬fin U (rel ∨ (fin ∧ rea)))))	#34A

Таблица 3.4 – Результат работы алгоритма для тестового примера Сушилка для рук

В результате эмпирической проверки корректности работы транслятора на корпусе EDTL требований было выявлено, что LTL формула, полученная в результате работы алгоритма, семантически согласована с требованием на естественном языке, что говорит о корректности трансляции и достаточности правил упрощения. Также была подсчитана частота встречаемости требований определенного класса (Таблица 3.5).

класс	общий вид формулы класса	частота
#1A	G(trig → rea)	11
#34A	G(trig → (¬fin ∧ F(rel ∨ (fin ∧ rea))))	5
#2	G(inv)	4
#8	G(trig → inv)	3
#37A	G(trig → (¬del U rea))	2
#23A	G(trig → (G(inv ∧ ¬fin) ∨ ((inv ∧ ¬fin) U (fin ∧ (inv ∧ rea)))))	2
#17A	G(trig → (G(inv ∧ ¬fin) ∨ ((inv ∧ ¬fin) U (rel ∨ (fin ∧ (inv ∧ rea))))))	2
true	*true*	1
#6A	G(trig → (rel ∨ inv))	1
#5A	G(trig → (rel ∨ rea))	1
#4	G(¬trig)	1
#38A	G(trig → (¬del U (rel ∨ rea)))	1
#27A	G(trig → (G(¬fin) ∨ F(fin ∧ rea)))	1
#24A	G(trig → (G(inv ∧ ¬fin) ∨ ((inv ∧ ¬fin) U (fin ∧ (inv ∧ (¬del U rea))))))	1
#13A	G(trig → (inv ∧ rea))	1

Таблица 3.5 – Частота встречаемости классов в корпусе требований.

Самый часто встречаемый класс в корпусе требований — класс #1A с семантическим представлением G(trig → rea). Это наиболее распространенный тип требований, представляющий собой требование немедленного реагирования на триггерное событие.

На данный момент классификация основана только на общем отображении требований в формулы логики линейного времени LTL. Это может привести к тому, что одно и то же требование на естественном языке, выраженное с помощью разных EDTL требований, помещается в разные классы, хотя результирующие LTL формулы одинаковы. Например, следующее требование на естественном языке иллюстрирует этот случай:

NL	Должна быть исключена одновременная подача сигналов на перемещение лифта вверх и вниз.					
LTL	G(¬(Up ∧ Down))					
EDTL	**trigger**	**release**	**final**	**delay**	**invariant**	**reaction**
	Up ∧ Down	false	true	true	true	false
Общая формула класса						
	G(¬trig)					
Класс	#4					

EDTL	**trigger**	**release**	**final**	**delay**	**invariant**	**reaction**
	true	false	true	true	¬(Up ∧ Down)	true
Общая формула класса						

	G(inv)
Класс	#2

В обоих случаях алгоритм упрощения возвращает LTL-формулу следующего вида: G(¬(Up ∧ Down)). Однако, в первом случае эта формула попадает в класс #4 с семантикой G(¬trig), а во втором в класс #2 с семантикой G(inv). Добавление Логических правил упрощения к текущему набору правил могут устранить этот недостаток классификации.

ЗАКЛЮЧЕНИЕ

При выполнении данной работы была проанализирована специфика EDTL требований и семантика LTL формул, были сформулированы требования к разрабатываемому транслятору, были определены правила преобразования EDTL требований в упрощенные LTL формулы, был предложен алгоритм трансляции EDTL требований в LTL формулы, был реализован предложенный алгоритм трансляции, была получена классификация EDTL требований на основе результатов работы транслятора, была эмпирически проверена корректность работы транслятора на корпусе EDTL требований.

По итогам работы были получены следующие результаты:

- определены правила преобразования EDTL требований в упрощенные LTL формулы;
- разработан и реализован алгоритм трансляции;
- получена классификация EDTL требований;
- корпус требований дополнен представлением требований в LTL.

В дальнейшем планируется уточнение классификации с учетом темпоральной структуры EDTL требований и соотношения логических значения их атрибутов, а также использование в алгоритме трансляции EDTL формул в качестве значений EDTL атрибутов.

Результаты работы могут быть использованы для формальной верификации управляющего программного обеспечения, генерации тестовых сценариев, разработки методов преобразования EDTL требований в требования на естественном языке, диагностики ошибок при определении требований, разработки канонической формы требований и определении значений атрибутов по умолчанию.

Результаты работы были представлены на Международной научной студенческой конференции 2022 в секции «Информационные технологии» в подсекции «Инструментальные и прикладные программные системы», тезисы доклада приняты в печать. На основе результатов работы была подготовлена и принята публикация «Semantic Classification of Event Driven Temporal Logic Requirements» на Международную конференцию молодых специалистов по электронным приборам и материалам IEEE EDM 2022.

Выпускная квалификационная работа выполнена мной самостоятельно и с соблюдением правил профессиональной этики. Все использованные в работе материалы и заимствованные принципиальные положения (концепции) из опубликованной научной литературы и других источников имеют ссылки на них. Я несу ответственность за приведенные данные и сделанные выводы.

Я ознакомлен с программой государственной итоговой аттестации, согласно которой обнаружение плагиата, фальсификации данных и ложного цитирования является основанием

для не допуска к защите выпускной квалификационной работы и выставления оценки «неудовлетворительно».

Гетманова Анастасия Николаевна _____
ФИО студента *Подпись студента*

« ____ »_____20 __г. *(заполняется от руки)*

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ И ЛИТЕРАТУРЫ

1. Вельдер С. Э. Верификация автоматных программ. С. Э. Вельдер, М. А. Лукин, А. А. Шалыто, Б. Р. Яминов — СПб: Наука, 2011. — 244 с.
2. Карпов Ю. Г. MODEL CHECKING. Верификация параллельных и распределенных программных систем. – БХВ-Петербург, 2010.
3. Clarke E. M. et al. (ed.). Handbook of model checking. – Cham : Springer, 2018. – Т. 10.
4. Garanina N. O. et al. A Temporal Logic for Programmable Logic Controllers //Automatic Control and Computer Sciences. – 2021. – Т. 55. – №. 7. – С. 763-775.
5. Gamma E. et al. Design patterns: elements of reusable object-oriented software. – Pearson Deutschland GmbH, 1995.
6. Holzmann G. J. The model checker SPIN //IEEE Transactions on software engineering. – 1997. – Т. 23. – №. 5. – С. 279-295.
7. Möller D. P. F. Systems and software engineering //Guide to Computing Fundamentals in Cyber-Physical Systems. – Springer, Cham, 2016. – С. 235-305.
8. Pnueli A. The temporal logic of programs //18th Annual Symposium on Foundations of Computer Science (sfcs 1977). – ieee, 1977. – С. 46-57.
9. Zyubin V. et al. Event-Driven Temporal Logic Pattern for Control Software Requirements Specification //International Conference on Fundamentals of Software Engineering. – Springer, Cham, 2021. – С. 92-107.
10. Zyubin V. E. et al. poST: A Process-Oriented Extension of the IEC 61131-3 Structured Text Language //IEEE Access. – 2022. – Т. 10. – С. 35238-35250.
11. А. Гетманова, Translator-EDTL-LTL-Diplom, 2022. [Online]. Доступно: https://github.com/agetmanova/Translator-EDTL-LTL-Diplom

ПРИЛОЖЕНИЕ А

Полный листинг программы

```
    def conSimpl(a, b):
  if (a == False) or (b == False): return False;
  elif (a == True): return b;
  elif (b == True): return a;

  else: return '('+a+' ∧ '+b+')'

def no(a):
  if (type(a) == type(True)): return not a;

  else: return "¬" + a;

def implSimpl(a, b):
  if ((type(a) == type(True)) or (type(b) == type(True))):
return disSimpl(no(a),b);

  else:  return '('+a+' → '+b+')'\

def FutureSimpl(a):
  if (type(a) == type(True)): return a;
  elif (a[0] != '('): return 'F('+a+')';
  else: return 'F'+a+''

def GloballySimpl(a):
  if (type(a) == type(True)): return a;
  #G(F(a)) = GF(a)
  elif ((a[0] == 'F') and (a[1] == '(') and (a[len(a)-1] ==
')')):
    rules[32][1]+=1;
    return 'G'+a;
  #G(G(a)) = G(a)
  elif ((a[0] == 'G')and (a[1] == '(') and (a[len(a)-1] ==
')')):
    rules[33][1]+=1;
    return a;

  #51 G(G(a ∧ ¬b) ∨ (a U (b ∧ a))) = G(a)
  elif((a[:3] == '(G(') and (a.find(' ∧ ¬') > 0) and (a.find(')
∨ (') > a.find(' ∧ ¬'))
        and (a.find(a[3:a.find(' ∧ ¬')] + ' U (') > 0)
        and (a.find(a[a.find(' ∧ ¬')+4:a.find(') ∨ (')] + ' ∧
```

```
' + a[3:a.find(' ∧ ¬')]) > 0)):
    rules[51][1] +=1;
    return 'G(' + a[3:a.find(' ∧ ¬')] + ')';

  #52 G(G(a ∧ ¬b) ∨ ((a ∧ ¬b) U (b ∧ (a U (a ∧ c))))) = G(a
∧ (G(¬b) ∨ F(b ∧ F(c))))
  elif((a[:3] == '(G(') and (a.find(' ∧ ¬') > 0) and (a.find(')
∨ (') > a.find(' ∧ ¬'))
        and (a.find(a[3:(a.find(') ∨ (('))] + ') U (' +
a[a.find(' ∧ ¬')+4:a.find(') ∨ (')] +
                    ' ∧ (' + a[3:a.find(' ∧ ¬')] + ' U (' +
a[3:a.find(' ∧ ¬')] + ' ∧ ') > a.find(') ∨ (')) ):
    a1 = a[3:a.find(' ∧ ¬')];
    nb1 = a[a.find(' ∧ ¬')+3:a.find(') ∨ (')]
    b1 = a[a.find(' ∧ ¬')+4:a.find(') ∨ (')]

    n ='G(' + a[3:a.find(' ∧ ¬')] + ' ∧ ¬' + a[a.find(' ∧
¬')+4:a.find(') ∨ (')] + ') ∨ ((' + a[3:(a.find(') ∨ (('))] +
') U (' + a[a.find(' ∧ ¬')+4:a.find(') ∨ (')] + ' ∧ (' +
a[3:a.find(' ∧ ¬')] + ' U (' + a[3:a.find(' ∧ ¬')] + ' ∧ '
    t = a.find(n)
    tlen = len(n)
    c = a[tlen + 1:-5]

    rules[52][1] +=1;
    return 'G' + conSimpl(a1, disSimpl(GloballySimpl(nb1),
FutureSimpl(disSimpl(b1, FutureSimpl(c)))))

  #53 G(G(a ∧ ¬b) ∨ ((a ∧ ¬b) U (b ∧ (a ∧ c)))) = G(a ∧
(G(¬b) ∨ F(b ∧ c)))
  elif((a[:3] == '(G(') and (a.find(' ∧ ¬') > 0) and (a.find(')
∨ (') > a.find(' ∧ ¬'))
        and (a.find(a[3:(a.find(') ∨ (('))] + ') U (' +
a[a.find(' ∧ ¬')+4:a.find(') ∨ (')] +
                    ' ∧ (' + a[3:a.find(' ∧ ¬')] + ' ∧ ') >
a.find(') ∨ (')) ):
    a1 = a[3:a.find(' ∧ ¬')];
    nb1 = a[a.find(' ∧ ¬')+3:a.find(') ∨ (')]
    b1 = a[a.find(' ∧ ¬')+4:a.find(') ∨ (')]

    n ='G(' + a[3:a.find(' ∧ ¬')] + ' ∧ ¬' + a[a.find(' ∧
¬')+4:a.find(') ∨ (')] + ') ∨ ((' + a[3:(a.find(') ∨ (('))] +
') U (' + a[a.find(' ∧ ¬')+4:a.find(') ∨ (')] + ' ∧ (' +
a[3:a.find(' ∧ ¬')] + ' ∧ '
```

```
    t = a.find(n)
    tlen = len(n)
    c = a[tlen + 1:-4]

    rules[53][1] +=1;
    return 'G' + conSimpl(a1, disSimpl(GloballySimpl(nb1),
FutureSimpl(disSimpl(b1, c))));

  #44 G(G(a) ∨ (a U b)) = G(a ∧ F(b))
  elif ((a[0:3] == '(G(') and (a.find(') ∨ (') > 0) and
(a.find(a[3:a.find(') ∨ (')] + ' U '))):
    rules[44][1] +=1;
    return GloballySimpl(conSimpl(a[3:a.find(') ∨ (')],
FutureSimpl(a[a.find(' U ')+3:-2])))

  #45 G((a ∧ b) U (a ∧ c)) = G(a ∧ (b U c))
  elif ((a[0:2] == '((') and (a.find(' ∧ ') > 0) and (a.find(')
U (') > a.find(' ∧ '))
          and (a[2:a.find(' ∧ ')] == a[a.find(') U
(')+5:a.find(' ∧ ', a.find(' ∧ ') + 1)])):
    rules[45][1] +=1;
    b = a[a.find(' ∧ ') + 3: a.find(') U (')]
    c = a[a.find(' ∧ ', a.find(' ∧ ') + 1) + 3:-2]
    a1 = a[2:a.find(' ∧ ')]
    return GloballySimpl(conSimpl(a1, UntilSimpl(b,c)));

  elif (a[0] != '('): return 'G('+a+')';
  else: return 'G'+a;

def UntilSimpl(a, b):
  if (b == False): return False;
  elif (b == True): return True;
  elif ((a == False) and (type(b) != type(True))): return b;
  elif ((a == True) and (type(b) != type(True))): return
FutureSimpl(b);

  #a U a = a
  elif (a == b):
    rules[34][1] +=1;
    return a;

  # ¬a U a = F(a)
  elif((a.find('¬') == 0) and (b == a[1:])):
    rules[35][1] +=1;
```

```
    return FutureSimpl(b);

  #¬a U (b ∨ a) = F(a) ∨ (¬a U b)
  elif ((a[0] == '¬') and (b.find ('∨ ' + a[1:]) > 0)):
    rules[42][1]+=1;
    b1 = b[1:(b.find ('∨ ' + a[1:]))-1];
    return disSimpl(FutureSimpl(a[1:]), UntilSimpl(a, b1));

  #46 (a ∧ b) U (c ∨ a) = a ∨ ((a ∧ b) U c)
  elif ((a.find(' ∧ ') > 0) and (b.find(' ∨ ') > 0) and
(a[1:a.find(' ∧ ')] == b[b.find(' ∨ ')+3:-1])):
    rules[46][1] +=1;
    c = UntilSimpl(a, b[1:b.find(' ∨ ')]);
    return disSimpl(a[1:a.find(' ∧ ')], c);

  #47 (a ∧ b) U a = a
  elif(a.find('(' + b + ' ∧ ') == 0):
    rules[47][1] +=1;
    return b;

  #48 a U (b ∨ a) = a ∨ (a U b)
  elif (b.find(' ∨ ' + a + ')') > 0):
    rules[48][1] +=1;
    return disSimpl(a, UntilSimpl(a, b[1:b.find(' ∨ ')]));

  #50 (a ∧ ¬b) U (b ∧ a) = a U (b ∧ a)
  elif ((a.find(' ∧ ') > 0) and (b.find(' ∧ ') > 0)
        and (a[1:a.find(' ∧ ')] == b[b.find(' ∧ ')+3:-1])
        and (a[a.find(' ∧ ')+3] == '¬')
        and (b[1:b.find(' ∧ ')] == a[a.find(' ∧ ')+4:-1])):
    rules[50][1] +=1;
    return UntilSimpl(a[1:a.find(' ∧ ')], b)

  else: return '('+a+' U '+b+')'

import pandas as pd
import numpy as np
from matplotlib import pyplot as plt

#data_frame = pd.read_csv('data.csv', encoding='windows-1251')
data_frame = pd.read_csv('data_work5.csv', encoding='utf8')

def konv(a_txt):
  if (a_txt == 'true*'): return True;
```

```
  elif (a_txt == 'false*'): return False;
  else: return a_txt;
def konv2(a_txt):
  if (a_txt == 'TRUE'): return True;
  elif (a_txt == 'FALSE'): return False;
  elif (a_txt == True): return True;
  elif (a_txt == False): return False;
  else: return a_txt;
def BoolToString(a):
  if (type(a) == type(True)):
    if (a == True): return "true";
    elif (a == False): return "false";
  else: return a;
a = 12
mas = []
for i1 in range(243):
    mas.append([0] * a)


for i in data_frame.index:
  release_txt = data_frame['release'][i]
  delay_txt = data_frame['delay'][i]
  final_txt =  data_frame['final'][i]
  reaction_txt =  data_frame['reaction'][i]
  invariant_txt =  data_frame['invariant'][i]

  triggerVAR_txt = data_frame['trigerVAR'][i];
  triggerTrue_txt = data_frame['trigerTRUE'][i];

  release = konv(release_txt);
  delay = konv(delay_txt);
  final = konv(final_txt);
  reaction = konv(reaction_txt);
  invariant = konv(invariant_txt);

  trigger = 'trig'
  x1 = con(invariant, reaction);
  x2 = dis(release, x1);
  x3 = con(invariant, no(delay));
  x4 = Until(x3, x2);
  x5 = con(final, x4);
  x6 = dis(release, x5)
  x7 = con(invariant, no(final));
  x8 = Until(x7, x6);
```

```
x9 = con(invariant, no(final));
x10 = Globally(x9);
x11 = dis(x10, x8);
x12 = impl(trigger, x11);
x13 = Globally(x12);


k1 = conSimpl(invariant, reaction);
k2 = disSimpl(release, k1);
k3 = conSimpl(invariant, no(delay));
k4 = UntilSimpl(k3, k2);
k5 = conSimpl(final, k4);
k6 = disSimpl(release, k5);
k7 = conSimpl(invariant, no(final));
k8 = UntilSimpl(k7, k6);

k9 = conSimpl(invariant, no(final));
k10 = GloballySimpl(k9);
k11 = disSimpl(k10, k8);
k12 = implSimpl(trigger, k11);
k13 = GloballySimpl(k12);


trigger = True

y1 = con(invariant, reaction);
y2 = dis(release, y1);
y3 = con(invariant, no(delay));
y4 = Until(y3, y2);
y5 = con(final, y4);
y6 = dis(release, y5)
y7 = con(invariant, no(final));
y8 = Until(y7, y6);

y9 = con(invariant, no(final));
y10 = Globally(y9);
y11 = dis(y10, y8);
y12 = impl(trigger, y11);
y13 = Globally(y12);


l1 = conSimpl(invariant, reaction);
l2 = disSimpl(release, l1);
```

```
  l3 = conSimpl(invariant, no(delay));
  l4 = UntilSimpl(l3, l2);
  l5 = conSimpl(final, l4);
  l6 = disSimpl(release, l5);
  l7 = conSimpl(invariant, no(final));
  l8 = UntilSimpl(l7, l6);

  l9 = conSimpl(invariant, no(final));
  l10 = GloballySimpl(l9);
  l11 = disSimpl(l10, l8);
  l12 = implSimpl(trigger, l11);
  l13 = GloballySimpl(l12);

  trigger = False
  z1 = True;

  mas[i][0]=BoolToString(release);
  mas[i][1]=BoolToString(delay);
  mas[i][2]=BoolToString(final);
  mas[i][3]=BoolToString(reaction);
  mas[i][4]=BoolToString(invariant);
  mas[i][5]=BoolToString(k13);
  mas[i][6]=BoolToString(l13);
  mas[i][7]=BoolToString(z1);
for i in range(0, len(mas)):
    for i2 in range(0, len(mas[i])):
        print(mas[i][i2], end=' ')
    print()


from openpyxl import Workbook
wb = Workbook()
ws = wb.active


for subarray in mas:
    ws.append(subarray)
wb.save('./result.xlsx')
```

ПРИЛОЖЕНИЕ В

Классификация. Одноэлементные классы

№	A
1	G(trig → rea)
2	G(trig → F(rel))
3	G(trig → F(rel ∨ rea))
4	G(trig → F(rea))
5	G(trig → (rel ∨ rea))
6	G(trig → (rel ∨ inv))
7	G(trig → (rel ∨ (inv ∧ rea)))
8	G(trig → (inv U rel))
9	G(trig → (inv U (rel ∨ (inv ∧ rea))))
10	G(trig → (inv U (inv ∧ rea)))
11	G(trig → (inv ∨ (inv U rel)))
12	G(trig → (inv ∨ ((inv ∧ ¬del) U rel)))
13	G(trig → (inv ∧ rea))
14	G(trig → (G(inv ∧ ¬fin) ∨ ((inv ∧ ¬fin) U rel)))
15	G(trig → (G(inv ∧ ¬fin) ∨ ((inv ∧ ¬fin) U (rel ∨ (fin ∧ inv)))))
16	G(trig → (G(inv ∧ ¬fin) ∨ ((inv ∧ ¬fin) U (rel ∨ (fin ∧ (inv ∧ rea))))))
17	G(trig → (G(inv ∧ ¬fin) ∨ ((inv ∧ ¬fin) U (fin ∧ (inv U rel)))))
18	G(trig → (G(inv ∧ ¬fin) ∨ ((inv ∧ ¬fin) U (fin ∧ (inv U (rel ∨ (inv ∧ rea)))))))
19	G(trig → (G(inv ∧ ¬fin) ∨ ((inv ∧ ¬fin) U (fin ∧ (inv U (inv ∧ rea))))))
20	G(trig → (G(inv ∧ ¬fin) ∨ ((inv ∧ ¬fin) U (fin ∧ (inv ∨ (inv U rel))))))
21	G(trig → (G(inv ∧ ¬fin) ∨ ((inv ∧ ¬fin) U (fin ∧ (inv ∨ ((inv ∧ ¬del) U rel))))))
22	G(trig → (G(inv ∧ ¬fin) ∨ ((inv ∧ ¬fin) U (fin ∧ (inv ∧ rea)))))
23	G(trig → (G(inv ∧ ¬fin) ∨ ((inv ∧ ¬fin) U (fin ∧ ((inv ∧ ¬del) U rel)))))
24	G(trig → (G(inv ∧ ¬fin) ∨ ((inv ∧ ¬fin) U (fin ∧ ((inv ∧ ¬del) U (rel ∨ (inv ∧ rea)))))))
25	G(trig → (G(inv ∧ ¬fin) ∨ ((inv ∧ ¬fin) U (fin ∧ ((inv ∧ ¬del) U (inv ∧ rea))))))
26	G(trig → (G(¬fin) ∨ (¬fin U rel)))
27	G(trig → (G(¬fin) ∨ (¬fin U (rel ∨ (fin ∧ rea)))))
28	G(trig → (G(¬fin) ∨ (¬fin U (rel ∨ (fin ∧ F(rel))))))
29	G(trig → (G(¬fin) ∨ (¬fin U (fin ∧ rea))))
30	G(trig → (G(¬fin) ∨ (¬fin U (fin ∧ F(rel ∨ rea)))))
31	G(trig → (G(¬fin) ∨ (¬fin U (fin ∧ F(rea)))))
32	G(trig → (G(¬fin) ∨ (¬fin U (fin ∧ (¬del U rel)))))
33	G(trig → (G(¬fin) ∨ (¬fin U (fin ∧ (¬del U rea)))))
34	G(trig → (G(¬fin) ∨ (¬fin U (fin ∧ (¬del U (rel ∨ rea))))))
35	G(trig → (¬del U rel))
36	G(trig → (¬del U rea))
37	G(trig → (¬del U (rel ∨ rea)))
38	G(trig → ((inv ∧ ¬del) U rel))

№	A
39	G(trig → ((inv ∧ ¬del) U (rel ∨ (inv ∧ rea))))
40	G(trig → ((inv ∧ ¬del) U (inv ∧ rea)))

№	B
1	G(rea)
2	GF(rel)
3	GF(rel ∨ rea)
4	GF(rea)
5	G(rel ∨ rea)
6	G(rel ∨ inv)
7	G(rel ∨ (inv ∧ rea))
8	G(inv U rel)
9	G(inv U (rel ∨ (inv ∧ rea)))
10	G(inv U (inv ∧ rea))
11	G(inv ∨ (inv U rel))
12	G(inv ∨ ((inv ∧ ¬del) U rel))
13	G(inv ∧ rea)
14	G(inv ∧ ¬fin ∧ F(rel))
15	G(inv ∧ ¬fin ∧ F(rel ∨ (fin ∧ inv)))
16	G(inv ∧ ¬fin ∧ F(rel ∨ (fin ∧ (inv ∧ rea))))
17	G(inv ∧ ¬fin ∧ F(fin ∧ (inv U rel)))
18	G(inv ∧ ¬fin ∧ F(fin ∧ (inv U (rel ∨ (inv ∧ rea)))))
19	G(inv ∧ (G(¬fin) ∨ F(fin ∨ F(rea))))
20	G(inv ∧ ¬fin ∧ F(fin ∧ (inv ∨ (inv U rel))))
21	G(inv ∧ ¬fin ∧ F(fin ∧ (inv ∨ ((inv ∧ ¬del) U rel))))
22	G(inv ∧ (G(¬fin) ∨ F(fin ∨ rea)))
23	G(inv ∧ ¬fin ∧ F(fin ∧ ((inv ∧ ¬del) U rel)))
24	G(inv ∧ ¬fin ∧ F(fin ∧ ((inv ∧ ¬del) U (rel ∨ (inv ∧ rea)))))
25	G(inv ∧ ¬fin ∧ F(fin ∧ ((inv ∧ ¬del) U (inv ∧ rea))))
26	G(¬fin ∧ F(rel))
27	G(¬fin ∧ F(rel ∨ (fin ∧ rea)))
28	G(¬fin ∧ F(rel ∨ (fin ∧ F(rel))))
29	G(¬fin ∧ F(fin ∧ rea))
30	G(¬fin ∧ F(fin ∧ F(rel ∨ rea)))
31	G(¬fin ∧ F(fin ∧ F(rea)))
32	G(¬fin ∧ F(fin ∧ (¬del U rel)))
33	G(¬fin ∧ F(fin ∧ (¬del U rea)))
34	G(¬fin ∧ F(fin ∧ (¬del U (rel ∨ rea))))
35	G(¬del U rel)
36	G(¬del U rea)
37	G(¬del U (rel ∨ rea))

№	B
38	G((inv ∧ ¬del) U rel)
39	G((inv ∧ ¬del) U (rel ∨ (inv ∧ rea)))
40	G(inv ∧ (¬del U rea))