

ПРИМЕНЕНИЕ НЕЙРОСЕТЕВОГО ПОДХОДА И ПАРАЛЛЕЛЬНОЙ ОБРАБОТКИ ИНФОРМАЦИИ ДЛЯ ОПРЕДЕЛЕНИЯ ВЫПОЛНИМОСТИ ФОРМУЛ ЛОГИКИ ВЕТВЯЩЕГОСЯ ВРЕМЕНИ НА СТРУКТУРАХ КРИПКЕ¹

А.П. Еремеев (*eremeev@appmat.ru*)
Н.Ю. Филинов (*filinov.n@yandex.ru*)

Национальный исследовательский университет «МЭИ», Москва

В работе представлен подход к задаче проверки выполнимости формул темпоральной вычислительной древовидной логики CTL на структурах Крипке с использованием графовых нейронных сетей. Выполнимость формулы формализуется как задача бинарной классификации по паре (формула CTL, структура Крипке). Предложена архитектура модели, объединяющая графовую и формульную части с последующей классификацией. Осуществлена генерация синтетического обучающего набора данных с автоматической разметкой с использованием классического алгоритма *model checking*. Представлены результаты экспериментов, подтверждающие высокую точность модели и её преимущество по скорости работы по сравнению с традиционными методами. Работа выполняется в рамках разработки инструментальных средств для интеллектуальных систем поддержки принятия решений реального времени.

Ключевые слова: искусственный интеллект, темпоральная логика, *model checking*, структура Крипке, графовая нейросеть, реальное время.

Введение

Верификация свойств систем с помощью формальных логик остаётся ключевым направлением в теории программирования и автоматической проверке моделей. Одним из наиболее используемых формализмов в этой

¹ Работа выполнена при финансовой поддержке РФН (проект № 24-11-00285), <https://rscf.ru/project/24-11-00285/>.

области является вычислительная древовидная логика (Computational Tree Logic, CTL), формулы которой интерпретируются на структурах Крипке [Clarke et al., 1981], [Kupferman et al., 2000]. Задача проверки выполнимости (satisfiability) формул CTL заключается в определении, существует ли структура Крипке, на которой формула выполняется. Эта задача, несмотря на свою выразительность, остаётся алгоритмически сложной.

С развитием методов машинного обучения, в частности, нейросетевых архитектур, возникает естественный вопрос: можно ли использовать нейросети для решения задач логического вывода, таких как выполнимость формул? Особенно интересным представляется применение графовых нейросетей (Graph Neural Network, GNN), так как структуры Крипке по сути являются помеченными графами.

В данной работе рассматривается подход, при котором задача определения выполнимости CTL-формул моделируется как задача бинарной классификации на графах. Предлагается способ кодирования формул и моделей, рассматриваются возможные архитектуры нейросетей, и приводятся результаты экспериментов, демонстрирующие перспективность подхода.

Данные исследования продолжают исследования и разработки по темпоральной логике ветвящегося времени, описанные в работах [Еремеев и др., 2011; 2017; 2023].

1. Логика CTL

Логика CTL является *темпоральной логикой ветвящегося времени*. В этой логике допустимы только формулы, в которых каждый темпоральный оператор X , U , F и G (характеризующий некоторое вычисление) предворяется квантором пути – A или E , что превращает любую темпоральную формулу пути в формулу, характеризующую состояние. На рис. 1 представлена схема вложенности логик, где линейные темпоральные логики LTL [Pnueli, 1977] и CTL [Ben-Ari et al., 1983], [Clarke et al., 1980] являются подмножеством более широкого класса логики CTL*.

Грамматика CTL:

$$\begin{aligned} \phi ::= & \perp \mid \top \mid p \mid (\neg\phi) \mid (\phi \wedge \phi) \mid (\phi \vee \phi) \mid (\phi \Rightarrow \phi) \mid (\phi \Leftrightarrow \phi) \\ & \mid AX\phi \mid EX\phi \mid AF\phi \mid EF\phi \mid AG\phi \mid EG\phi \mid A[\phi U \phi] \mid E[\phi U \phi]. \end{aligned}$$

Синтаксическое значение кванторов пути в CTL:

- A означает "по всем путям" (неизбежно);
- E означает "по крайней мере (существует) один путь" (возможно).

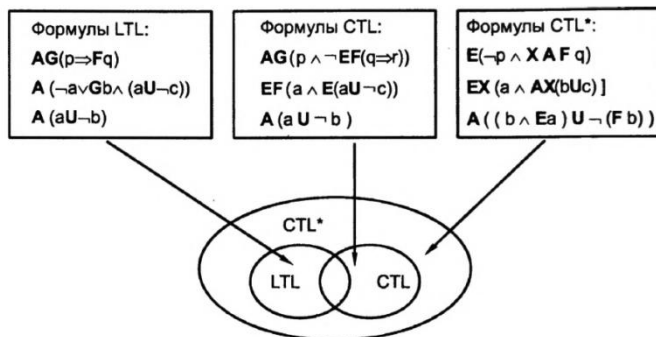


Рис. 1. Общая схема вложенности логик

Кванторы, зависящие от пути:

- φX – Next: φ должно сохраняться в следующем состоянии (этот оператор иногда указывается N вместо X);
- φG – Globally: φ должно сохраняться на всем последующем пути;
- φF – Finally: φ в конечном итоге должно выполняться (где-то на последующем пути);
- $\psi U \varphi$ – Until: φ должно сохраняться *по крайней мере* до тех пор, пока не будет сохранено некоторое положение ψ . Это подразумевает, что ψ будет проверяться в будущем;
- $\psi W \varphi$ – Weak until - φ должно выполняться до тех пор, пока не будет выполнено ψ . Разница с U заключается в том, что нет гарантии, что ψ когда-либо будет проверено. Оператор W иногда называют "unless (пока не)".

2. Структура Крипке

Структура Крипке M – это пятерка где:

- S – конечное непустое множество состояний;
- \subseteq – непустое множество начальных состояний;
- $R \subseteq S \times S$ – тотальное отношение на S , т.е. множество переходов,

удовлетворяющее требованию: (из любого состояния есть переход);

- AP – конечное множество атомарных предикатов;
- L – функция пометок (каждому состоянию отображение сопоставляет множество истинных в нем атомарных предикатов).

В работе используется алгоритм model checking для CTL со сложностью [Еремеев и др., 2023].

3. Постановка задачи машинного обучения

Рассматривается задача определения выполнимости формулы CTL на заданной структуре Крипке. Формально задача может быть сформулирована следующим образом.

Дано:

- формула CTL: φ ;
- структура Крипке:

Требуется:

определить, существует ли состояние $s \in S$, такое что $M, s \models \varphi$, т.е. φ выполнима в M .

В классическом варианте задача решается с помощью алгоритмов model checking, которые проверяют истинность формулы φ в каждом состоянии структуры. Однако в данном исследовании предлагается аппроксимировать эту проверку с помощью машинного обучения, предсказывая выполнимость φ на M как задачу бинарной классификации.

4. Описание предлагаемого подхода

Подготовительный этап. Для применения методов машинного обучения необходимо перевести логическую и графовую информацию в числовое или структурированное представление.

Формула представляется в виде абстрактного синтаксического дерева (Abstract Syntax Tree, AST), которое затем кодируется с использованием одного из следующих подходов:

- Bag-of-Operators: частотный вектор по используемым операторам (EX , AG , \wedge , и т.д.).
- Position encoding: информация о глубине вложенности и структуре формулы.
- Tree embeddings: использование рекурсивных нейросетей или трансформеров для кодирования структуры формулы.

Поскольку структура Крипке – это ориентированный помеченный граф, она кодируется как: граф с вершинами (состояниями), дугами (переходами) и метками (атомарные предикаты); adjacency matrix + node labels: представление через матрицу смежности и признаковые векторы вершин. В случае применения GNN используется n -мерное признаковое пространство на узлах и итеративная агрегация признаков по соседям.

Формулировка в виде задачи классификации. Преобразованная пара (φ, M) подаётся на вход нейросетевой модели (например, GNN + MLP), задача которой – предсказать метку $y \in \{0, 1\}$, где: $y = 1$, если существует состояние s в M , такое что $M, s \models \varphi$ (т.е. φ выполнима); $y = 0$, в противном случае. Таким образом, подход работает как приближённый предсказатель

выполнимости, позволяя быстро фильтровать нерелевантные пары или подсказывать возможные области интереса для дальнейшей формальной проверки.

Основная цель нейросетевого подхода – снижение вычислительной нагрузки на классические алгоритмы *model checking*, особенно в случае больших систем. Модель может использоваться как предварительный фильтр (*pre-check*), приближённый классификатор или компонент гибридной системы верификации.

Целью данного этапа является подготовка качественной обучающей выборки, состоящей из пар (формула CTL, структура Крипке) и метки, определяющей, выполнима ли формула на данной структуре. Поскольку таких данных не существует в открытом доступе, необходима автоматическая генерация и разметка.

Генерация структур Крипке и CTL формул. Для получения разнообразных графов, имитирующих модели программ, используется генерация случайных структур:

- состояния (S): фиксируются или варьируются количество вершин (например, от 5 до 100);
- переходы (R): сгенерированы случайно с гарантией достижимости и связности (например, вероятность ребра p выбирается из диапазона $[0.1, 0.4]$);
- метки (L): каждый узел помечается подмножеством из фиксированного множества атомарных предикатов $AP = \{p, q, r, \dots\}$.

Дополнительно: можно добавлять вариации – циклы, ветвления, сильно связанные компоненты (SCC) – чтобы разнообразить типы систем.

Реализация: с использованием библиотеки *networkx*, можно параллельно генерировать тысячи таких графов.

Формулы CTL генерируются синтаксически с использованием шаблонов:

- используются глубины формулы как параметр сложности: глубина 2–5;
- операторы: EX, AX, EF, AF, EG, AG, логические связки (\wedge , \vee , \rightarrow);
- формулы создаются рекурсивно:
 - базовые случаи: атомарные p , q , $\neg p$, и т.д.;
 - рекурсивные: $EX \phi$, $\phi \wedge \psi$, $AG(\phi \rightarrow EF \psi)$, и т.д.

Формулы можно генерировать с контролем по:

- количеству переменных;
- глубине вложенности;
- частоте использования определённых операторов

Чтобы избежать переобучения, генерация должна обеспечивать структурное разнообразие. При генерации формул используются классовые представления формул из *PyModelChecking* [10].

Разметка данных. Для каждой пары (структура, формула) вычисляется метка. Используется *model checker*, предложенный в [Еремеев и др., 2023].

Если хотя бы в одном состоянии s верно, что $M, s \models \varphi$, то метка есть 1 (выполнима), иначе 0. Для расширенного исследования можно менять метрику выполнимости. Например, можно считать, что формула выполнима на структуре, если формула истинна во всех состояниях S структуры Крипке.

Параллельность. Поскольку каждая пара независима, проверку можно распараллелить по ядрам или в кластере. В языке python данная генерация реализована через Multiprocessing. Распараллеливание этого процесса особенно актуально на данных с большим количеством состояний, переходов в структуре Крипке и подформул формулы CTL.

При генерации формул возникает проблема в ситуации, когда классы сильно разбалансированы. Эту проблему можно решить отбрасыванием некоторой части сгенерированных данных, где превалирует та или иная метка (0 или 1). Для оценки качества перекоса данных при обучении так же используется метрика ROC-AUC [Hanley et al., 1982].

5. Архитектура модели GNN

Выбрана параллельная архитектура. Модель объединяет в себе два потока данных – графовую составляющую и формульную составляющую.

Графовая составляющая. Структура Крипке представляется графом, узлы которого имеют бинарные признаки — наличие или отсутствие атомарных предикатов. Эта часть обрабатывается двумя слоями GCNConv и агрегируется через global mean pool, формируя глобальное представление графа.

Global mean pool – это агрегирующая операция, используемая в сетях (GNN), в частности, в библиотеке PyTorch Geometric [PyG Documentation, 2024], для преобразования признаков узлов графа в один вектор фиксированной длины для каждого графа в батче. Эта операция является одним из простых эффективных способов в своем роде.

Формульная составляющая. Формула токенизируется и передается через слой Embedding, затем поступает в длительную кратковременную память (Long Short-Term Memory, LSTM). Последнее скрытое состояние LSTM служит эмбедингом формулы. Затем два эмбединга – графа и формулы – объединяются конкатенацией и передаются в классификатор на полносвязных слоях для предсказания результата.

Общий план forward-прохода следующий:

- $x = \text{ReLU}(\text{GCN1}(x)) \rightarrow \text{ReLU}(\text{GCN2}(x)) \rightarrow \text{global_mean_pool} \rightarrow \text{g_embed}$;
- $\text{formula} \rightarrow \text{Embedding} \rightarrow \text{Packed LSTM} \rightarrow \text{f_embed}$;
- $\text{combined} = \text{concat}(\text{g_embed}, \text{f_embed}) \rightarrow \text{classifier} \rightarrow \text{output}$.

Предложенный подход имеет следующие преимущества. Модель обучается в end-to-end режиме. Это значит, что модель учится одновременно распознавать структуры графов и понимать логическую формулу с учётом

взаимосвязи между ними. Это ключевое преимущество end-to-end подхода. Разделение графа и формулы позволяет гибко подбирать архитектуры для каждого потока. LSTM хорошо справляется с переменной длиной формул, особенно в сочетании с `pack_padded_sequence`. Функция `pack_padded_sequence` нужна для игнорирования паддинга, чтобы обрабатывались только непустые токены.

Архитектура GNN с визуализацией процесса представлена на рис. 2.

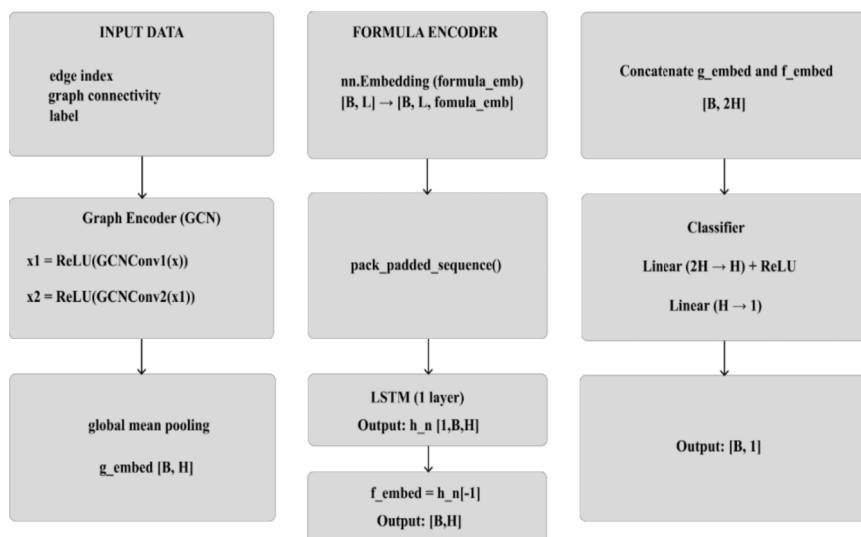


Рис. 2. Архитектура GNN

6. Обучение

Для обучения сети и запуска тестовых примеров использовалась следующая конфигурация ПК.

- Processor 12th Gen Intel(R) Core(TM) i7-12700H 2.30 GHz;
- Installed RAM 32,0 GB (31,7 GB usable);
- System type 64-bit operating system, x64-based processor.

Обучение проводится с использованием функции потерь BCEWithLogitsLoss, дополнительно используется параметр `pos_weight` для компенсации дисбаланса классов. Оценка проводится по метрикам Accuracy (сравнение предсказанных и истинных значений один к одному) и ROC-AUC (площадь под кривой ошибок). ROC-AUC нужна, чтобы оценить качество бинарного классификатора в случае, если присутствует большой перекоп классов в исходных данных.

Процесс обучения проведен на разных датасетах. Результаты приведены в табл. 1. Заметим, что с ростом глубины генерации формулы увеличивается сложность задачи и время обучения. Это связано с тем, что формула глубины, например, 40, может иметь около 50000 токенов, что является объемным для вычислений. В целом результаты обучения хорошие, есть возможность разнообразить датасеты и увеличить количество эпох для увеличения точности. Наглядно результаты представлены на рис. 3.

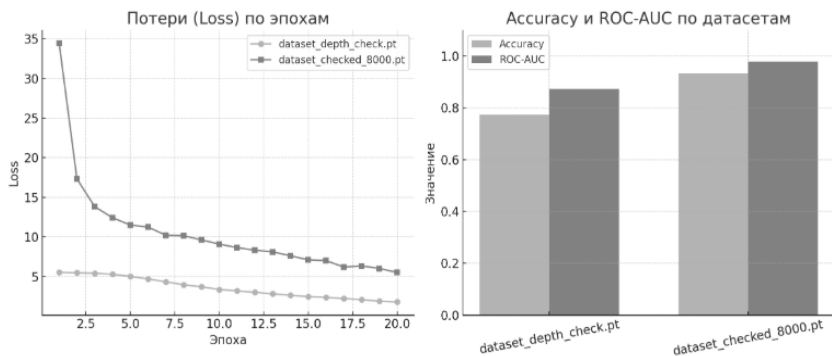


Рис. 3. Результаты обучения GNN

7. Анализ результатов

Будем сравнивать результаты работы алгоритма из [PyModelChecking's, 2024] с результатами GNN на одних и тех же наборах данных.

Предложенная графовая нейронная сеть GNN уверенно обходит классический подход по скорости. Даже при длинных формулах её время редко превышает 1–1.2 секунды. Однако оно также растёт с увеличением размера входных данных. Результаты сравнения даны в табл. 2.

Подход с использованием GNN в целом оказался эффективным – даже простые модели могут достигать высоких результатов и обходить по скорости классические алгоритмы модельной проверки, особенно, если использовать распараллеливание по ветвям. Отметим, что нейросетевой метод дает приближенные результаты и не гарантирует строгой корректности. Для минимизации рисков предлагается встраивать в систему многократный запуск метода и использовать, например, среднее среди всех полученных результатов. В случаях, когда метод работаеткратно быстрее классических алгоритмов, такой подход может быть целесообразен.

Таблица 1

Результаты обучение GNN на разных датасетах

Параметр	Датасет 1	Датасет 2
Количество данных	1000	5160
Глубина формулы при генерации	40	20
Баланс классов	318 / 318	2580 / 2580
Эпох обучения	20	20
Потери (Loss) на 1-й эпохе	5.5344	34.4617
Потери (Loss) на 20-й эпохе	1.7991	5.5247
Ассурасу на тесте	0.7734	0.9471
ROC-AUC на тесте	0.8728	0.9787
Время обучения (сек)	257.39	45.70
Примечание	Глубокие формулы, меньше данных	Много данных, формулы проще

Таблица 2

Сравнение времени работы: Model Checking vs GNN

Кол-во узлов	Глубина формулы	Длина формулы (токенов)	Model Checking (c)	GNN (c)
60	40	~5000	0.110	0.030
80	40	~5000	0.263	0.042
120	40	~5000	1.170	0.077
40	80	19512	5.078	1.036
80	80	—	12.481	1.229

Чем длиннее формула, тем эффективнее оказывается применение GNN по сравнению с обычным алгоритмом. Результаты приведены в табл. 3. При этом точность остаётся довольно высокой (около 90-92%), что делает GNN особенно удобными для задач, где важна скорость и обрабатывается большой поток формул. Заметим, что с увеличением глубины формулы значительно растёт нагрузка на память и вычисления, так как мы храним каждый токен формулы и при случайной генерации можем получить массивы порядка элементов. LSTM обрабатывает последовательность за $O(N)$ шагов (все токены параллельно через матричные операции). Проведённое сравнение демонстрирует практическое преимущество ML-подходов в задачах приближённой проверки CTL-формул. Хотя они не дают строгой гарантии корректности, но высокая точность и большой выигрыш по времени делают их подходящими применения в интеллектуальных системах реального времени (ИС РВ) (типа ИС РВ для поддержки принятия решений, ИСППР РВ), особенно когда требуется масштабируемость и оперативность.

Таблица 3

Влияние сложности формулы на время (фиксированное число узлов 40)

Макс. длина формулы	Длина токенов	Model Checking (с)	GNN (с)
120	8836	1.378	0.422
120	27040	3.970	1.271
1200	9826	1.514	0.459
1200	19830	3.007	0.890
2000	13629	2.082	0.617
2000	25416	3.781	1.226
5000	6036	0.926	0.270
5000	19804	3.005	0.886
5000	19292	2.927	0.866

Заключение

Предложен новый подход к задаче проверки выполнимости формул темпоральной логики CTL на структурах Крипке с использованием методов машинного обучения. Рассмотрена формализация задачи как бинарной классификации, где входом модели является пара (формула CTL, структура Крипке), а выходом – предсказание её выполнимости. Разработаны методы представления входных данных: формулы кодируются с помощью LSTM на основе токенизации; структуры Крипке – с помощью графовых признаков и обрабатываются графовой нейронной сетью (GNN). Предложена end-to-end архитектура, объединяющая оба потока данных. Предложена методика генерации синтетических датасетов – случайная генерация структур Крипке и формул CTL с последующей разметкой через классический алгоритм model checking. Это позволило получить объёмную и разнообразную обучающую выборку. Для ускорения реализована система параллельной генерации и верификации.

На экспериментальных данных продемонстрировано, что предложенная модель способна эффективно предсказывать выполнимость формул, достигая высоких значений метрик качества (Accuracy и ROC-AUC), и показывает значительное преимущество по времени работы по сравнению с классическим алгоритмом model checking, особенно при увеличении длины формулы или размера графа. Показана перспективность применения нейросетевых архитектур в задачах приближённой проверки логических свойств. Такие модели могут использоваться в качестве предварительных фильтров, ускоряющих классическую верификацию. Данные исследования и разработки выполняются в плане реализации базовых инструментальных (математических и программных) средств для конструирования ИС/ИСППР РВ для диагностики и мониторинга сложных технических (технологических) и организационных объектов и процессов.

Список литературы

- [Еремеев и др., 2011] Еремеев А.П., Куриленко И.Е. Темпоральные модели на основе логики ветвящегося времени в интеллектуальных системах // Искусственный интеллект и принятие решений. – 2011. – № 1. – С. 14-26.
- [Еремеев и др., 2017] Еремеев А.П., Куриленко И.Е. Реализация вывода в темпоральных моделях ветвящегося времени // Известия РАН. Теория и системы управления. – 2017. – № 1. – С. 107-127. – ISSN 0002-3388.
- [Еремеев и др., 2023] Еремеев А.П., Филинов Н.Ю. Реализация алгоритма темпоральной ветвящейся логики в рамках инструментальных средств построения интеллектуальных систем поддержки принятия решений реального времени // Труды Межд. научно-техн. конг. «Интеллектуальные системы и информационные технологии – 2023 (IS&IT'23)». Научн. изд. в 2-х т. Т. 1. – Таганрог: Изд-во Ступина С.А., 2023. – С. 320-330. – ISBN 978-5-6050434-1-6 (Т. 1).
- [Ben-Ari et al., 1983] Ben-Ari M., Manna Z., Pnueli A. The temporal logic of branching time // Acta Informatica. – 1983. – 20 pp. – P. 207-220.
- [Clarke et al., 1980] Clarke E.M., Emerson E.A. Characterizing correctness properties of parallel programs using fixpoints // In Automata, Languages, and Programming. LNCS 85. – Springer, 1980. – P. 169-181.
- [Clarke et al., 1981] Clarke E.M., Emerson E.A. Design and synthesis of synchronization skeletons using branching time temporal logic. In Logic of Programs: Workshop. LNCS 131. – Springer, 1981.
- [Hanley et al., 1982] Hanley J.A., & McNeil B.J. The meaning and use of the area under a receiver operating characteristic (ROC) curve // Radiology. – 1982. – 143(1). – P. 29-36.
- [Kupferman et al., 2000] Kupferman O., Vardi M.Y. and Wolper P. An Automata-Theoretic Approach to Branching-Time Model Checking // In Journal of the ACM. – March 2000. – Vol. 47, No. 2. – P. 312-360.
- [Pnueli, 1977] Pnueli. A. The Temporal Logic of Programs // In Proceedings of the 18th IEEE Symposium on Foundations of Computer Science, Providence, 31 October-2 November 1977. – P. 46-67.
- [PyG Documentation, 2024] PyG Documentation. – <https://pytorch-geometric.readthedocs.io/en/latest/>.
- [PyModelChecking's, 2024] PyModelChecking's,documentation. – <https://pymodelchecking.readthedocs.io/en/latest/>.