МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ
РОССИЙСКОЙ ФЕДЕРАЦИИ
ТОМСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

МАТЕРИАЛЫ

IX-й Международной научной конференции

«МАТЕМАТИЧЕСКОЕ И ПРОГРАММНОЕ ОБЕСПЕЧЕНИЕ ИНФОРМАЦИОННЫХ, ТЕХНИЧЕСКИХ И ЭКОНОМИЧЕСКИХ СИСТЕМ»

Томск, 26–28 мая 2022 г.

Под общей редакцией
кандидата технических наук И.С. Шмырина

Томск
Издательство Томского государственного университета
2022

Операции над связями между узлами	+	+

Заключение

В работе предлагается метод статического анализа для обнаружения утечек ресурсов в динамических списочных структурах, в частности, односвязных и двухсвязных списках. В методе предлагается моделирование кучи с помощью трех множеств: память, рабочая среда и результат. На основании исходного кода программы строится графовая структура, позволяющая отследить состояние связей в процессе моделирования работы программы и по ее завершении. После того как анализ динамической структуры завершится и найдутся вершины во вспомогательном графе, которые не имеют связей, следующим этапом можно будет отследить историю изменений состояния вершин в модели работы динамической памяти. Благодаря данному анализу результатом будут строки, где были совершены утечки.

ЛИТЕРАТУРА

1. *Кулямин В.В.* Методы верификации программного обеспечения. – М.: Институт системного программирования РАН, 2008. – 111. с

2. Static Code Analysis. In: van Tilborg H.C.A., Jajodia S. (eds) Encyclopedia of Cryptography and Security. Springer, Boston, MA. https://doi.org/10.1007/978-1-4419-5906-5_1371

3. *Кормен Т., Лейзерсон Ч., Ривест Р., Штайн К.* Алгоритмы. Построение и анализ». – Вильямс, 2013. – 1324 с.

4. *Sai-ngern S.* An address mapping approach for test data generation of dynamic linked structures // Information and Software Technology. – 2005. – V. 47. – № 3. – P. 199–214.

DOI: 10.17223/978-5-907572-27-0-2022-14

ОПТИМИЗАЦИЯ РЕШЕНИЯ АЛГОРИТМА CDCL, ПРЕДСТАВЛЕННОГО В ВИДЕ ТРОИЧНОГО ВЕКТОРА

Иванкин Д.И. Андреева В.В.

Томский государственный университет

ivankin00@list.ru

Введение

В работе рассматривается SAT-проблема, суть которой заключается в нахождении набора переменных, для которых булева формула принимает истинное значение, или установлении, что таких наборов не существует. В 1971 г. Стивеном Куком и независимо от него Леонидом Левиным была доказана теорема о том, что SAT-задача принадлежит классу NP-полных задач. В дальнейшем эти результаты использовались для доказательства NP-полноты большого количества других задач, приведением за полиномиальное время данной задачи к SAT. Поэтому SAT-задача имеет большое распространение в прикладных задачах, например:

- Метод кластеризации k-means (метод к-средних) может быть решен с помощью SAT.
- Задача оптимального распределения задач по процессорам может быть решена с помощью SAT.
- Многие рекомендательные системы используют алгоритм разложения булевых матриц для рекомендации контента. Оптимальное разложение таких матриц может быть найдено с помощью SAT.
- Большое количество задач, связанных с графами, также может быть решено при помощи SAT.

В связи с большой распространенностью в прикладных задачах, в настоящее время проводятся ежегодные соревнования между SAT-решателями, что дает большой толчок в развитии и улучшении алгоритмов. Таким образом, можно сделать вывод, что SAT-

задача является одной из наиболее актуальных современных проблем ввиду своей большой распространенности как в теоретических исследованиях, так и в прикладных задачах. Поэтому решение SAT-проблемы может оказаться очень важным как с точки зрения ответа на вопрос равенства классов P и NP, так и с точки зрения решения прикладных задач.

Булева формула SAT-задачи представляется в виде КНФ: $K = 1$. Необходимо найти корень уравнения или убедиться, что решения не существует. Под корнем будем понимать некоторую дизъюнкцию ранга r.

Для выполнимости формулы необходимо, чтобы все дизъюнкты принимали значение 1. В дизъюнктах, для которых все литеры, кроме одной, принимают значение 0, оставшаяся литера обязана принимать значение 1. Переменные, соответствующие таким литерам, будем называть переменными с фиксированными значениями.

Для решения задачи используются алгоритмы, основанные на ветвлении и возврате. Алгоритм DPLL (Davis – Putnam – Logemann – Loveland) является одним из первых и высокоэффективных алгоритмов решения SAT-задачи, на основе которого разрабатываются современные SAT-решатели.

DPLL включает в себя 3 основных шага:

1. Выбор переменной ветвления.
2. Распространение переменной.
3. Решение конфликта.

Алгоритм CDCL, который рассматривается в работе, является улучшением DPLL. Основные отличия CDCL от DPLL заключаются в использовании графа импликаций для анализа и разрешения конфликта, запоминании новых дизъюнктов в ходе анализа и нехронологическом возврате.

В работе предложен метод оптимизации алгоритма CDCL, позволяющий уменьшить как ранг получаемого корня, так и время работы.

1. Описание алгоритма CDCL

1.1. Выбор переменной

На этапе выбора переменной необходимо выбрать и зафиксировать значение переменной, у которой оно еще не зафиксировано; если таких нет, то задача решена. При фиксации переменной необходимо убрать из КНФ все дизъюнкты, которые стали принимать истинное значение. При удалении дизъюнктов вместе с ними из КНФ могут исчезнуть некоторые переменные, эти переменные не повлияют на значения оставшихся переменных, следовательно, за счет таких переменных будет расширяться интервал корня уравнения.

1.2. Распространение переменной

Распространение переменной заключается в выявлении всех переменных с фиксированным значением, проверке на наличие конфликта и фиксации значений этих переменных. Конфликтом будем называть ситуацию, при которой для одной переменной необходимо зафиксировать два значения.

1.2.1. Граф импликаций

Граф импликаций показывает связи между переменными, то, как фиксация значения одной переменной влияет на значения других.

Вершины графа обозначают переменные. Переменные без инверсий зафиксированы со значением 1, с инверсией – со значением 0. Связи между вершинами означают, что фиксация значения одной или нескольких переменных однозначно определяют значение другой. При фиксации выбранных переменных в графе формируются новые уровни. Вершины, через которые проходят все пути на текущем уровне от выбранной переменной до конфликтной, называются точками единичной импликации (ТЕИ). Выбранная переменная всегда является точкой единичной импликации.

Построение графа импликаций:

1. Выбирается переменная, которой будет присвоено значение 1 или 0.
2. Проверяется, не появились ли переменные с фиксированным значением; если таких нет, то построение закончено.
3. Фиксируются значения этих переменных и добавляются в дерево. Их «родителями» будут все переменные, которые влияют на выбор значения.
4. Проверяется, имеется ли в дереве конфликт; если есть, то построение завершено, осуществляется переход к анализу конфликта; если нет, осуществляется переход на п. 2.

1.3. Решение конфликта

Если после этапа распространения переменной конфликт не образовался, возвращаемся к выбору переменной, иначе нужно его разрешить путем возврата на предыдущие уровни и выбора других переменных и их значений; если возвращаться некуда – у задачи нет решения.

1.3.1. Построение среза и нового дизъюнкта

Для решения конфликта используется механизм запоминания новых дизъюнктов, который позволяет предотвратить появление одинаковых конфликтов в будущем.

Построение запоминаемых дизъюнктов происходит путем разреза графа – разделения на 2 части: конфликтную и причинную. Конфликтная сторона – та, которая содержит конфликтную вершину. Все вершины, находящиеся на причинной стороне и имеющие рёбра, проходящие через границу раздела, являются причиной конфликта, поэтому должны быть добавлены в запоминаемый дизъюнкт, причем со знаком инверсии.

Разрезы могут проходить в разных местах (это зависит от конкретного алгоритма) и будут давать разные дизъюнкты для обучения. Было доказано [1], что разрез по первой точке единичной импликации обладает наибольшей эффективностью. При таком подходе построения разреза на конфликтной стороне окажутся все вершины с текущего уровня графа, находящиеся правее точки единичной импликации.

1.4. Возврат

После построения запоминаемого дизъюнкта необходимо вернутся на предыдущий уровень, который выбирается как второй по величине уровень переменных, входящих в дизъюнкт. В ситуации, когда запоминаемый дизъюнкт состоит из одной переменной, возврат происходит на нулевой уровень, т.е. до момента выбора первой переменной ветвления.

После возврата новый дизъюнкт гарантирует, что в процессе распространения переменной на текущем уровне будет зафиксировано противоположное значение единичной точки импликации, по которой строился дизъюнкт.

2. Предлагаемая оптимизация

Представим КНФ в виде троичной матрицы, в которой строки соответствуют дизъюнктам, а столбцы – переменным. «1» или «0» на пересечении строки и столбца означает, что соответствующая переменная находится в дизъюнкте без знака инверсии или со знаком инверсии соответственно, а «–» означает, что соответствующая переменная отсутствует в данном дизъюнкте.

Тогда поиск корня уравнения сводится к поиску покрытия всех строк матрицы, представляющей КНФ.

Введем следующие обозначения:

M^+ – подмножество положительно однородных векторов.

M^- – подмножество отрицательно однородных векторов.

M^* – подмножество неоднородных векторов.

В [2] были предложены правила, дающие достаточные условия существования корня для случаев, когда матрица имеет вид:

- $M = M^*$
- $M = M^- \cup M^*$
- $M = M^+ \cup M^*$

Для матрицы вида $M = M^+ \cup M^- \cup M^*$ нельзя однозначно определить, есть для нее корень или нет. Поэтому предлагается выделить монотонные подмножества из M^* и анализировать подматрицы вида $M = M^+ \cup M^- \cup M^{*+}$ и $M = M^+ \cup M^- \cup M^{*-}$.

Рассмотрим возможные ситуации для этих подмножеств на примере $M = M^+ \cup M^- \cup M^{*+}$, для второго подмножества ситуации аналогичные:

1. В M^{*+} имеются векторы ранга 1.

1.1. Векторы полностью покрывают хотя бы один вектор из M^-.

Корня в подмножестве нет, т.к. существует хотя бы один вектор из M^-, который мы не сможем покрыть.

1.2. Векторы не покрывают полностью ни один вектор из M^-.

Нельзя утверждать об отсутствии или наличии корня.

2. В M^{*+} нет векторов ранга 1.

Ситуация аналогична 1.2.

Для ситуаций 1.1 и 2 возможно наличие корня, поэтому мы попытаемся его отыскать. Поиск корня в таких подмножествах проще за счет того, что происходит только фиксация значений переменных без обработки встречаемого конфликта, и, следовательно, нет необходимости строить граф импликаций. Причем в ранее проводившихся исследованиях [3,4] было установлено, что процесс распространения переменной занимает до 90% времени работы алгоритма.

Также для увеличения вероятности нахождения корня следует фиксировать значения переменных, находящихся в векторах меньшего веса, т.к. такие переменные с большей вероятностью попадут в решение ввиду малой степени свободы для выбора переменной, покрывающей соответствующий вектор.

Если для обоих подмножеств нет корня, это говорит о том, что корень находится где-то на их пересечении и необходимо дополнительно упростить матрицу, проведя очередную итерацию основного алгоритма.

3. Экспериментальные результаты

Экспериментальные результаты (табл. 1) были получены с использованием тестовых примеров [5], задаваемых в формате DIMACS. Данный формат формализует задание КНФ для SAT-решателей. Каждая строка начинается с буквы, обозначающей тип строки:

- Комментарии к тестовому примеру начинаются с буквы “c”.
- Буквой “p” обозначается строка, содержащая проблему, т.е. формат задания КНФ, количество переменных и дизъюнктов. Такая строка для каждого примера единственная. Сразу после проблемной строки идет перечисление дизъюнктов, по одному в каждой строке. Дизъюнкт задается перечислением всех переменных, входящих в него, причем отрицательные числа обозначают инверсию переменной, а ноль – конец дизъюнкта.

Для всех тестовых примеров действуют ограничения:

- не допускается наличие дизъюнкции ранга 1;
- не допускается наличие в одной дизъюнкции двух одинаковых литер;
- не допускается наличие в одной дизъюнкции переменной с инверсией и без нее.

Описание тестовых примеров:

- UF n.k.m (Uniform Random – 3SAT) – каждый дизъюнкт генерируется путем выбора трех случайных литер, причем вероятность выбора литер одинакова и равна $1/2n$.

• JNH (John Hooker) – дизъюнкты генерируются случайным образом, причем вероятность появления литеры в дизъюнкте всегда фиксирована.
• RTI – дизъюнкты ранга 3, генерируются случайным образом. Каждому примеру из RTI соответствует пример из BMS.
• BMS (Backbone Minimal Size) – примеры генерируются на основе примеров из RTI таким образом, что в конечной формуле в каждом дизъюнкте существует литера, фиксирование противоположного значения которой ведет к конфликту, причем если удалить этот дизъюнкт из формулы, то переменная потеряет свой эффект.
• CBS n.k.m. (Controlled Backbone Size) – генерируются случайные дизъюнкты ранга 3, причем для каждого примера имеется множество литер фиксированного размера m таких, что фиксирование противоположного значения приводит к конфликту.
• II – интерпретация задачи уменьшения помех в электронных схемах.

В табл. 1 Nt – количество тестовых примеров, Nc – количество дизъюнктов, Nv – количество переменных, T – среднее время работы алгоритма CDCL в мс, R – средний ранг корня найденного алгоритмом CDCL, Topt – среднее время работы алгоритма CDCL с применением предложенной оптимизации в мс, Ropt – средний ранг найденного корня алгоритмом CDCL с применением предложенной оптимизации

Таблица 1

Экспериментальные результаты

Тестовый пример	Nt	Nc	Nv	T	R	Topt	Ropt
UF100.430.100	100	430	100	340	93	276	92
JNH	16	800	100	267	94	272	92
RTI100.429	100	429	100	377	93	341	93
BMS100.429	100	290	100	1723	89	1532	89
CBS.100.403.10	100	403	100	128	92	125	92
CBS.100.403.50	100	403	100	462	93	426	93
CBS.100.403.90	100	403	100	697	94	664	94
ii8a1	1	186	66	4	59	16	60
ii8a2	1	800	180	85	151	203	147
ii8a3	1	1552	264	916	226	964	214
ii8a4	1	2798	396	324	298	435	288
ii8b3	1	6108	816	14095	519	1394	509
ii8b4	1	8214	1068	156094	716	5101	667
ii8d1	1	3207	530	9098	343	2703	357
ii8d2	1	6547	930	33257	540	1249	540
ii8e1	1	3136	520	4434	361	453	343

В ходе экспериментов было получено, что для примеров со случайной генерацией алгоритм с использованием предложенной оптимизации и без неё показывают сравнительно равные результаты, что обусловлено большой разреженностью примеров и наличием малого количества монотонных дизъюнктов. Также была выявлена задача уменьшения помех в электронных схемах, для которой предложенная оптимизация дает наибольший прирост производительности.

Заключение

В работе был рассмотрен алгоритм CDCL решения SAT-задачи, а также исследован метод оптимизации алгоритма, основанный на представлении КНФ в виде троичной матрицы, разбиении этой матрицы на подмножества и анализе этих подмножеств. Было проведено экспериментальное сравнение скорости и качества решения для классического алгоритма CDCL и алгоритма CDCL с применением предложенной оптими-

зации, в ходе которого было установлено, что предложенная модификация позволяет добиться уменьшения ранга корня, а также времени работы алгоритма.

ЛИТЕРАТУРА

1. *Zhang L., Madigan C.F., Moskewicz M.H., Malik Sh.* Efficient Conflict Driven Learning in a Boolean Satisfiability Solver: IEEE/ACM International Conference on Computer Aided Design. ICCAD 2001. IEEE/ACM Digest of Technical Papers URL: https: // ieeexplore . ieee . org / document / 968634

2. *Андреева В.В., Тарновская Т.П.* Сокращение ранга конъюнкции, представляющей корень логического уравнения // Вестник Томского государственного университетата. Управление, вычислительная техника и информатика. – 2015. – № 4 (33). – С. 62–68.

3. *Moskewicz M.W., Madigan C.F., Zhao Y., Zhang L., Malik Sh.* Chaff: Engineering an Efficient SAT Solver // DAC '01: Proceedings of the 38th annual Design Automation Conference. – P. 530–535.

4. *Abraham L.* SAT solving techniques: a bibliography // URL: https:/arxiv.org/abs/1802.05159v2

5. SATLIB – benchmark problems: URL: https://www.cs.ubc.ca/~hoos/SATLIB/benchm.html

DOI: 10.17223/978-5-907572-27-0-2022-15

ПРИМЕНЕНИЕ БИНАРНОГО ГРАФА ДЛЯ ВЕРИФИКАЦИИ ПРОГРАММНОГО КОДА

Картавых Г.В., Андреева В.В.
Томский государственный университет
goshantiy@gmail.com

Введение

В процессе разработки программного продукта возможно возникновение ошибок. Ошибки возникают ввиду человеческого фактора и могут повлиять на работу продукта, что может привести к существенным временным или материальным потерям. В связи с этим, для выявления ошибок в процессе разработки становится необходимым осуществление верификации программного обеспечения.

Верификация – это процесс определения, выполняют ли программные средства и их компоненты требования, наложенные на них. Этот процесс включает в себя создание тестовых данных, подаваемых на вход программе, сравнивание ожидаемых и фактических результатов на выходах программы. Зачастую срок разработки строго регламентирован по времени, ввиду чего верификация в ручном режиме на крупных программных продуктах может потребовать больших временных затрат, а также повышает риск упущения ошибки в программе.

Решением этой проблемы является автоматизация некоторых этапов процесса верификации, в том числе генерации тестовых данных, что позволяет значительно сократить время, затрачиваемое на проведение тестов, и снизить влияние человеческого фактора.

1. Постановка задачи

В данной работе с целью верификации программного обеспечения рассматривается методология представления процедур программного кода, возвращающих истинное и ложное значение, в виде модели двоичного дерева, предложенной в работе Efficient Program Verification Using Binary Trees and Program Slicing Masakazu Takahashi, Noriyoshi Mizukoshi, and Kazuhiko Tsuda [1].

Данная модель представления программного кода позволяет представить архитектуру управляющей структуры программы, включая интервалы областей значений участвующих в управляющей структуре переменных. Такой подход позволяет генерировать входные и выходные области данных для каждого элемента управляющей структуры, а также для элементов, оказывающих влияние на области данных, включенных в тело управляющей структуры. Области данных, в свою очередь, будут использоваться для верификации программы.