Интерполяция формул с кванторами в CSIsat на основе инстанцирования[1]

В. С. Мутилин <mutilin@ispras.ru>,
М. У. Мандрыкин <mandrykin@ispras.ru>

Аннотация. В статье рассказывается о том, как на основе инструмента CSIsat реализована интерполяция Крейга для формул с кванторами, заданных в рамках комбинации теорий вещественной линейной арифметики и неинтерпретируемых функций. Предложен способ реализации интерполирования таких формул с помощью инстанцирования подкванторных выражений с использованием внешнего SMT-решателя CVC3. В статье описываются изменения, которые были внесены в интерполятор и решатель для реализации поддержки кванторной интерполяции. Также приводятся результаты тестирования модифицированного интерполятора как на задачах, полученных из набора SMTLIB, так и на специально сгенерированных для этих целей задачах.

Ключевые слова: интерполяция, интерполянт Крейга, кванторы, инстанцирование, решатель, аксиомы.

1. Введение

В настоящее время в области проверки моделей программ одним из наиболее распространённых и успешных подходов является предикатная абстракция. Это в значительной степени объясняется успехами инструмента статической верификации SLAM2 [1], использующего этот подход, и применяемого в рамках набора инструментов WDDK(Windows Driver Developer's Kit) для верификации драйверов ОС Windows. Предикатная абстракция программы строится на основе логических предикатов, набор возможных значений которых составляет абстрактный домен, разделяя всё множество возможных состояний программы на подмножества с одинаковыми значениями выбранных предикатов[2]. Отличительной чертой этого подхода является то, что выбор предикатов для абстракции обычно либо полностью определяется рассматриваемой программой, либо очень сильно зависит от неё. Поэтому основной проблемой при использовании предикатной абстракции является

[1] Работа поддержана ФЦП "Исследования и разработки по приоритетным направлениям развития научно-технологического комплекса России на 2007-2013 годы (контракт N 11.519.11.4006).

автоматическое определение подходящего набора предикатов, подходящих для верификации данной конкретной программы. Наиболее часто с этой целью используют подход CEGAR [3] (Counter-Example Guided Abstraction Refinement – уточнение абстракции по контрпримеру). Он основывается на том, что абстракция программы строится итеративно, причём, если проверка модели с некоторой абстракцией даёт ложный (реально не существующий) контрпример (это значит, что абстракция оказалась недостаточно точной), то этот контрпример используется для получения новых предикатов и построения уточнённой абстракции программы.

Среди современных инструментов статической верификации программ, использующих предикатную абстракцию и реализующих подход CEGAR, на практике наиболее распространены инструменты SLAM2[1], BLAST[4,5] и CPAchecker[6]. В этих инструментах уточнение абстракции по контрпримеру осуществляется на основе двух различных и в то же время схожих подходов. Во всех этих инструментах, так или иначе, строится некоторая логическая формула, соответствующая рассматриваемому контрпримеру. В SLAM2 для этого используется наислабейшее предусловие для последовательности операторов, составляющих контрпример. В BLAST и CPAchecker используется построенная на основе SSA-представления формула пути. Так как контрпример является ложным, то достаточно точно построенная соответствующая ему логическая формула являться невыполнимой. Этот факт используется инструментами для получения из этой формулы новых предикатов с целью уточнения абстракции. При этом верификаторы прибегают к помощи специальных инструментов. В SLAM2 используется эвристика получения новых предикатов из минимальной невыполнимой подформулы («ядра невыполнимости» или unsatisfiable core), которую может выдавать инструмент проверки выполнимости логических формул (SMT-решатель) Z3[7], а также другие синтаксические методы получения новых предикатов[8]. BLAST и CPAchecker получают новые предикаты локально, то есть отдельно для нескольких выбранных точек программы, из интерполянтов Крейга для частей формулы пути до и после каждой выбранной такой точки. Для поиска этих интерполянтов оба инструмента используют специальные инструменты – интерполирующие решатели. BLAST может использовать FOCI[9] или CSIsat[10], а CPAchecker – MathSAT[11] или тот же CSIsat.

Одним из преимуществ текущей версии инструмента SLAM перед BLAST и CPAchecker является его хорошая точность (определяемая по числу ложных срабатываний) при анализе программ, существенно использующих указатели, в том числе на динамически выделенную память. При этом в инструменте не используется какой-либо специальный вид анализа структур в динамической памяти (вроде Shape-анализа[12]) или специальная логика для представления объектов кучи (например, separation logic [13]). Вместо этого используется некоторая упрощенная модель памяти на основе размещений. Эта модель предложена в статье [14], в которой она используется для построения предикатов с указателями, значения которых эффективно вычисляются

современным SMT-решателем (в данном случае Z3[7]). Для вычисления значений предикатов с указателями авторы этой статьи предлагают использовать специальный набор аксиом, который определяет логическую модель памяти. В статье также предлагается алгоритм представления предикатов с помощью неинтерпретируемых функций от целых чисел для передачи соответствующих задач SMT-решателю.

Возможность применения похожей логической модели памяти было предложено исследовать в верификаторах BLAST[4,5] и CPAchecker[6]. Как уже было сказано, получение новых предикатов для построения уточненной абстракции в них реализовано на основе интерполирования некоторых частей формулы невыполнимого пути [15].

Конъюнкция упорядоченной пары формул, соответствующих двум частям невыполнимого пути, невыполнима. Поэтому для этой пары формул существует интерполянт Крейга, то есть формула, логически следующая из первой формулы пары, невыполнимая в конъюнкции со второй формулой и содержащая лишь общие для обеих формул неинтерпретируемые символы. Предикаты, входящие в этот интерполянт, используются в BLAST и CPAchecker при построении уточненной абстракции программы. Для получения этого интерполянта используется специальный инструмент – интерполирующий решатель (также «интерполирующая процедура» или просто «интерполятор»). В BLAST и CPAchecker используются интерполирующие решатели (CSIsat[10] и MathSAT[11]) для бескванторных формул, заданных в рамках комбинации теорий линейной вещественной арифметики и неинтерпретируемых функций (LA+EUF). Для таких формул эти инструменты выдают интерполянт, также не содержащий кванторов. При применении модели памяти с набором аксиом возникает необходимость интерполировать также формулы, содержащие кванторы. При этом для таких формул становится допустимым получение интерполянта, содержащего кванторы.

Таким образом, если для реализации рассматриваемой модели памяти в инструменте SLAM2 оказалось достаточно SMT-решателя Z3, не поддерживающего интерполяцию, но поддерживающего наборы аксиом с кванторами, то для реализации подобной модели памяти в инструменте BLAST или CPAchecker обязательно требовался именно интерполирующий решатель, поддерживающий кванторы или наборы аксиом. Однако подходящего для этих целей инструмента найти не удалось.

Вместе с тем, в статье[16] предлагается расширение алгоритма интерполяции Макмиллана[9] для интерполирования формул с кванторами на основе инстанцирования. Эти формулы могут быть заданы в рамках произвольных теорий, для которых применим алгоритм Макмиллана. Расширенный алгоритм Макмиллана позволяет достаточно просто искать интерполянты, возможно, содержащие кванторы, для таких формул, при условии, что заранее известен набор экземпляров (инстанцирований) исходных подкванторных

выражений (для кванторов всеобщности), достаточный для доказательства невыполнимости конъюнкции, заданной в задаче поиска интерполянта.

Идея реализации расширенного алгоритма Макмиллана показалась интересной еще и потому, что при уточнении абстракции рассматриваемые инструменты (BLAST и CPAchecker) еще до применения интерполяции делают один или несколько соответствующих запросов к SMT-решателю. При этом большинство современных SMT-решателей реализуют поддержку формул с кванторами через инстанцирование подкванторных выражений на основе различных эвристик. Поэтому если в таком случае решателю удается доказать невыполнимость заданной формулы с кванторами, это означает, что перед запуском интерполятора у решателя уже имеется набор инстанцированных выражений, достаточный для доказательства невыполнимости конъюнкции в задаче интерполирования.

Поскольку реализация расширенного алгоритма Макмиллана с использованием инстанцирований, полученных из SMT-решателя, оказывается достаточно простой, такую реализацию можно было бы использовать для предварительной оценки эффективности различных возможных реализаций новой модели памяти в инструментах верификации, использующих интерполяцию для уточнения абстракции.

Поэтому было предложено реализовать расширенный алгоритм Макмиллана в таком виде на основе какого-либо существующего интерполятора и SMT-решателя и оценить эффективность работы полученного инструмента на специально сгенерированных задачах, которые бы мог подавать на вход модифицированному интерполятору верификатор при реализации в нем новой модели памяти.

2. Выбранный подход

Для применения расширенного алгоритма Макмиллана в том виде, в котором он описан в статье[17], нужно иметь дерево резолютивного вывода для доказательства невыполнимости исходной конъюнкции формул, в котором бы уже использовались необходимые подстановки подкванторных выражений и были заранее вычислены так называемые частичные интерполянты для листьев этого дерева. Такое дерево можно было получить различными способами. Одним из возможных способов является реализация некоторой эвристики для подстановки термов в подкванторные выражения (например, e-matching[17]) в каком-либо инструменте, реализующем алгоритм Макмиллана. Другой способ – использование для получения нужных подстановок внешнего инструмента, в котором уже хорошо реализованы соответствующие эвристики, например, SMT-решателя. В случае использования внешнего решателя набор подстановок можно извлекать из предъявляемого им доказательства невыполнимости.

Если из этого доказательства извлекать лишь набор необходимых постановок, то невыполнимость входной формулы будет доказываться дважды: сначала в SMT-решателе, поддерживающем кванторы (для получения подстановок), а затем – в интерполирующем решателе, с нужными подстановками и уже без кванторов, – для получения интерполянта. При этом напрямую использовать полученное SMT-решателем доказательство для получения интерполянта в интерполирующем решателе не получится, так как для интерполяции по алгоритму Макмиллана необходимо иметь доказательство невыполнимости в специальной системе правил вывода, которая значительно отличается от используемых современными SMT-решателями. Современные SMT-решатели по своей эффективности значительно превосходят известные нам реализации интерполяторов, работающие по алгоритму Макмиллана (FOCI, CSIsat и несколько экспериментальных реализаций), поэтому при использовании внешнего SMT-решателя накладные расходы на повторное доказательство невыполнимости формулы в интерполяторе оказываются невелики по сравнению со временем его работы после получения от SMT-решателя нужных подстановок. При такой реализации накладными расходами выглядит скорее повторное получение доказательства невыполнимости в самом интерполяторе.

В силу простоты реализации был выбран именно подход с использованием доказательства невыполнимости, выдаваемого внешним SMT-решателем. Для его реализации нужно было выбрать подходящий существующий интерполятор, который бы удовлетворял следующим требованиям:

- Реализовывал бы алгоритм индуктивного построения интерполянта по дереву резолютивного вывода пустого дизъюнкта на основе комбинирования частичных интерполянтов, описанный в статье Макмиллана (McMillan) [9]. Это необходимо, поскольку алгоритм, предложенный в [16] является расширением алгоритма Макмиллана.
- Поддерживал бы интерполяцию для формул в рамках комбинации теорий, которые используются в задачах поиска интерполянта инструментами BLAST и CPAchecker. Это теории линейной целочисленной (LIA) или вещественной (LRA) арифметики и равенства с неинтерпретируемыми функциями (EUF). Их комбинация часто обозначается как LA+EUF.
- Распространялся бы в виде исходного кода под лицензией, разрешающей его свободную модификацию.

Среди существующих интерполирующих решателей только CSIsat [10] удовлетворял всем этим требованиям. Он написан на языке OCaml, при этом части, отвечающие за чтение входной формулы, ее предобработку, и процесс комбинирования частичных интерполянтов в реализации соответствуют различным модулям. Поэтому оказалось достаточно реализовать

соответствующие модифицированные версии этих модулей и использовать их в случае, когда входная задача содержит кванторы.

Для реализации был выбран простой подход, использующий только набор подстановок, и не требующий глубокого анализа доказательства невыполнимости, выданного SMT-решателем. Поэтому основными критериями при выборе этого инструмента стали достаточно высокая эффективность при доказательстве невыполнимости для формул с кванторами и доступность его исходного кода для модификации – с целью упрощения интеграции. Исходя из этих критериев был выбран SMT-решатель CVC3[18]. При этом основным недостатком этого решателя является большая сложность предъявляемых им доказательств, в которых используется большое число правил вывода. На момент выбора SMT-решателя этот недостаток казался несущественным, поскольку для выделения нужных подстановок планировалось использовать достаточно простой анализ доказательства невыполнимости, однако впоследствии оказалось, что в некоторых случаях CVC3 вообще не включает в окончательное доказательство необходимых подстановок, что приводит к аварийному завершению модифицированного интерполятора.

3. Сравнение с другими подходами

Присутствие значительных накладных расходов на повторное доказательство невыполнимости формулы в интерполирующем решателе наталкивает на мысль о другом способе интерполяции. Преобразование дерева доказательства SMT-решателя к системе правил вывода, подходящей для индуктивного построения интерполянта, представляет собой этот другой, принципиально отличный от предложенного в данной статье, подход к интерполяции. Этот подход также кардинально отличается от используемого в CSIsat и описанного в статье [16]. Такой подход описан в статье[19] и, вполне возможно, превосходит выбранный для реализации в рамках данной работы по эффективности. Ведь при его использовании удается избежать существенных накладных расходов на повторное доказательство невыполнимости в интерполяторе CSIsat, на которое обычно уходит большая часть ресурсов, затрачиваемых этим инструментом. В статье[19] этот подход описан для решателя Z3[7]. Его применение для другого наиболее распространенного SMT-решателя, поддерживающего кванторы, CVC3[18] осложняется большим количеством правил вывода, используемым в этом инструменте. Следует отметить, что упомянутая статья была впервые представлена уже после того, как был реализован описываемый в данной статье инструмент.

Далее в статье приводятся детали реализации и результаты оценки эффективности только для модифицированного интерполятора CSIsat, реализующего расширенный алгоритм Макмиллана с использованием внешнего SMT-решателя CVC3.

4. Детали реализации

Модифицированный инструмент CSIsat поддерживает интерполяцию формул с кванторами, заданных в рамках комбинации теорий вещественной линейной арифметики и неинтерпретируемых функций (LA+EUF). Предложенная реализация основана на последней версии CSIsat 1.2 (июль 2008 года) от оригинальных разработчиков. Инструмент использует специально модифицированный инструмент CVC3, который позволяет ему получать необходимые для интерполяции экземпляры подкванторных выражений.

Перечислим основные изменения, внесенные в инструменты CSIsat и CVC3 при реализации выбранного подхода.

4.1. Изменения в интерполяторе CSIsat

- В модифицированной версии инструмента был расширен входной формат для задачи интерполяции, который совместим с использованным в интерполяторе FOCI (этот интерполятор реализует подход, описанный в [9]). В синтаксис описания входной формулы были добавлены обозначения для кванторов всеобщности и существования:

<квантор всеобщности> ::= ‘!’ <список переменных> ‘.’ <формула>
<квантор существования> ::= ‘?’ <список переменных> ‘.’ <формула>
<список переменных> ::= <идентификатор> <список переменных> | <пусто>

При этом список связываемых квантором переменных (<список переменных>) должен быть не пуст.

- Модифицированный инструмент поддерживает интерполяцию только для пар формул. При указании опции -useLemmata в исходной задаче задается три формулы *A, B* и *C*. Формула *C* при этом должна быть конъюнкцией выражений под кванторами всеобщности. При этом интерполянт ищется для пары формул (*A, B∧C)*, но считается, что кванторы из формулы *C* являются аксиомами теории и свободные символы в них – общие для формул *A* и *B*.
- В инструмент были добавлены некоторые предварительные преобразования исходной задачи перед приведением ее в конъюнктивную нормальную форму для дальнейшего решения. Каждая формула в исходной задаче вначале подвергается некоторым преобразованиям. В частности, в ней выделяются подформулы с кванторами. Эти подформулы переводятся в предваренную нормальную форму и сколемизуются[2]. Если после приведения в предваренную нормальную форму и сколемизации хотя бы в одной формуле все еще присутствуют кванторы всеобщности, то применяется модифицированный алгоритм интерполяции (расширенный алгоритм Макмиллана). Для формул без кванторов используется не модифицированный алгоритм.
- Была добавлена поддержка вывода формул в формате SMTLIB[20] и вызов внешнего SMT-решателя для получения необходимых подстановок. В случае интерполирования задачи с кванторами преобразованная конъюнкция *A∧B* передается модифицированному инструменту CVC3 с опцией +dump-insts. При успешном доказательстве невыполнимости CVC3 помимо вердикта unsat выдает также и набор подстановок, достаточный для доказательства невыполнимости конъюнкции. Если же конъюнкция оказывается выполнимой, CSIsat вместо интерполянта выдает вердикт Satisfiable. Так как решатель CVC3 не является полным, возможен также и вердикт Unknown.
- Был реализован алгоритм очищения (*purification*) полученных подстановок от смешанных термов (*mixed terms*) в соответствии с алгоритмом, описанным в[16]. Если конъюнкция оказывается невыполнимой, CSIsat очищает полученные от CVC3 подстановки, при этом строятся вспомогательные хеш-таблицы, содержащие информацию о новых общих символах.
- Был реализован расширенный алгоритм интерполяции Макмиллана для формул с кванторами. К полученной на предыдущих шагах работы конъюнкции вместе с очищенными подстановками применяется этот алгоритм, который в общем случае дает кванторный интерполянт.

[2] Сколемизация – это удаление кванторов существования при помощи введения новых неинтерпретируемых функций и констант.

4.2. Изменения в решателе CVC3

Рассматриваемая реализация требует от решателя список подстановок подкванторных выражений, достаточных для доказательства невыполнимости исходной формулы. Для того, чтобы получить такой набор подстановок, можно использовать генерируемое решателем доказательство невыполнимости, в котором в явном виде присутствуют правила вывода, соответствующие инстанцированию подкванторных выражений.

В формате доказательств CVC3 для этого используются четыре правила вывода:

$$\frac{T \vdash \forall \bar{x}.\, e(\bar{x})}{T \vdash e(\bar{t})}\, universal\_elimination1$$

$$\frac{T \vdash \forall \bar{x}.\, e(\bar{x})}{T \vdash \psi \Rightarrow e(\bar{t})}\, universal\_elimination2, universal\_elimination3$$

$$\frac{T \vdash \forall \overline{x_1 x_2}.\, e(\overline{x_1}, \overline{x_2})}{T \vdash \psi \Rightarrow \forall \overline{x_2}.\, e(\bar{t}, \overline{x_2})}\, partial\_universal\_instantiation$$

Здесь $\bar{x} = (x_1, x_2, \ldots, x_n)$ — вектор связанных переменных, e – выражение, в которое переменные $x_1, \ldots, x_n$ входят свободно, $\bar{t} = (t_1, \ldots, t_n)$ – вектор термов для подстановки, все переменные в которых свободно входят в e. Для правила *universal_elimination1* x_i и t_i должны иметь одинаковые типы для $i = \overline{1,n}$, а для остальных правил – соответственно одинаковые базовые типы, в таких случаях ψ – предикат, накладывающий ограничения подтипа на подставляемые термы.

Однако, как оказалось на практике, CVC3 не всегда включает данные правила вывода в доказательство невыполнимости для формул с кванторами. Иногда инстанцирование кванторов в нем заменяется упрощенным правилом вывода вида:

$$\frac{}{T \vdash e(\bar{t})}\, assump$$

Поэтому для извлечения необходимых подстановок была реализована эвристика, рассматривающая некоторые части еще не окончательно сформированного доказательства на предмет присутствия в них инстанцирований. Также была реализована эвристика упрощения выражений, получившихся в результате подстановки. Эти эвристики включаются опциями +extr-insts и +simpl-insts соответственно. Для включения выдачи экземпляров аксиом вместе с вердиктом unsat используется опция +dump-insts.

5. Результаты

5.1. Результаты на наборе из архива SMT-LIB

Реализованный инструмент был протестирован на двух различных наборах задач.

Первый набор задач был получен из архива SMT-LIB[21] путем случайного разделения невыполнимых формул, заданных в рамках логик AUFLIA и AUFLIRA на две части (AUFLIA – Linear Integer Arithmetic with Uninterpreted Functions and Arrays – линейная целочисленная арифметика с неинтерпретируемыми функциями и массивами, AUFLIRA – Arrays, Uninterpreted Functions, and Linear Arithmetic – массивы, неинтерпретируемые функции и линейная арифметика, в том числе целочисленная и вещественная). Полученные задачи интерполяции были переведены во входной формат нового инструмента. При этом целочисленные типы переменных заменялись вещественными, а операции с массивами представлялись неинтерпретируемыми функциями с соответствующим набором аксиом. Некоторые задачи при этом могли переставать быть корректными, а другие могли давать вырожденные интерполянты true и false.

Тесты запускались на бинарной оптимизированной версии инструмента с лимитом 5 секунд по времени и 1 Гб по памяти. В таблице 1 приведены результаты для всех формул из категорий AUFLIA и AUFLIRA.

	Результат	Число тестов	%
успешно	Интерполянт с кванторами	167	0,63%
	Интерполянт без кванторов	551	2,09%
	Вырожденный интерполянт true	8259	31,38%
	Вырожденный интерполянт false	8342	31,70%
	Входная конъюнкция выполнима	1	0,00%
не успешно	CVC3 выдал результат Unknown	536	2,04%
	Превышение лимита времени (5 с)	7846	29,81%
	Превышение лимита памяти	71	0,27%
	Выполнимая формула (неверный набор подстановок)	515	1,96%
	Другие ошибки	31	0,12%
	Всего	26319	100,00%

Таблица 1. Результаты тестирования набора задач из архива SMTLIB без эвристик

5.2. Результаты на специально сгенерированном наборе

Второй набор задач был специально сгенерирован как набор возможных запросов верификатора к интерполирующему решателю для получения новых предикатов при реализации в нем модели памяти, похожей на используемую в инструменте SLAM2. В этой модели с размещениями для построения предикатов использовались пять неинтерпретируемых функций:

- *A(l)* – возвращает адрес размещения *l,*
- *L(a)* – возвращает размещение, соответствующее адресу *a,*
- *S(x, f)* – возвращает размещение для элемента составного типа (например, массива или структуры), здесь *x* – размещение первого элемента составного типа, *f* – номер нужного элемента, считая с *0,*
- *B(l)* – возвращает размещение первого элемента составного типа по размещению другого его элемента,
- *O(l)* – возвращает номер элемента составного типа по его размещению

с соответствующим набором аксиом:

1. $\forall x.(x > 0 \Rightarrow A(x) > 0)$
2. $\forall l.L\big(A(l)\big) = l$
3. $\forall a.A\big(L(a)\big) = a$
4. $\forall x.\forall f.S(x,f) > N$

5. $\forall x.\forall f.B\big(S(x,f)\big) = x$
6. $\forall x.\forall f.O\big(S(x,f)\big) = f$

Для обозначения размещений в этой модели использовались строго положительные целые числа. Первая аксиома утверждает, что адрес любого корректного размещения строго положителен. Вторая и третья аксиомы вместе задают функции *A(l)* и *L(a)* как взаимно обратные для всех размещений и их адресов. Четвертая аксиома утверждает, что размещение для элемента составного типа не может совпадать ни с каким базовым размещением (то есть размещением простого типа или размещением составного типа целиком). Базовые размещения соответствуют явно размещаемым объектам в программе, таким как переменные, структуры и массивы. Им всем присваиваются размещения с номерами от 1 до некоторого N. Элементы же составных типов должны иметь размещения с номерами строго больше N, о чем и говорит четвертая аксиома. Пятая и шестая аксиомы задают функции *B(l)* и *O(l)* и одновременно утверждают, что элементам разных составных типов, а также различным элементам одного составного типа соответствуют разные размещения. В противном случае для двух элементов составных типов было бы выполнено:

$$S(x_1,f_1) = S(x_2f_2) \Longrightarrow B\big(S(x_1,f_1)\big) = B\big(S(x_2,f_2)\big), O\big(S(x_1,f_1)\big) = O\big(S(x_2,f_2)\big)$$
$$\Longrightarrow (5,6) \Longrightarrow x_1 = x_2,\ f_1 = f_2$$

Для получения значений размещений использовался специальный набор неинтерпретируемых функции $V_i(l)$. Каждая такая функция соответ-ствовала своей версии памяти между двумя последовательными ее обновлениями. При этом при генерации тестового набора не делалось никаких оптимизаций обновления памяти, каждому обновлению памяти было поставлено в соответствие выражение вида $V_{i+1}\big(l_{upd}\big) = v \wedge \forall l.(l \neq l_{upd} \Rightarrow V_{i+1}(l) = V_i(l))$, где l_{upd} – размещение, значение которого обновляется на *v*.

В тестах с существенным использованием массивов и структур использовался измененный набор аксиом для учета свойства равенства адресов структур или массивов с адресами их первых элементов. В этом наборе аксиомы (4), (5) и (6) заменены следующими аксиомами:

7. $\forall l.A(l) = A(S(l,0))$
8. $\forall x.\forall f.(f \neq 0 \Rightarrow S(x,f) > N)$
9. $\forall x.\forall f.(f \neq 0 \Rightarrow B\big(S(x,f)\big) = x)$
10. $\forall x.\forall f.(f \neq 0 \Rightarrow O\big(S(x,f)\big) = f)$

Здесь размещение для всего составного объекта совпадает с размещением для его первого элемента, а аксиомы (4), (5) и (6) верны только для всех элементов, кроме первого.

Приведем пример преобразования задачи построения новых предикатов в заданной точке неосуществимого пути выполнения программы в задачу

поиска интерполянта для формулы с кванторами. Для примера используем следующий тест из подготовленного набора:

```
s_1->f_1 = 1;
s_1 = s_1->next;
s_1->f_1 = 2;
s_1 = s_1->next;
...
s_1->f_1 = n - 1;
s_1 = s_1->next;
s_1->f_1 = n;
s_2 = s_1;
s_1 = s_1->next;
----------------------
check (s_2->f_1 != n);
ERROR:
```

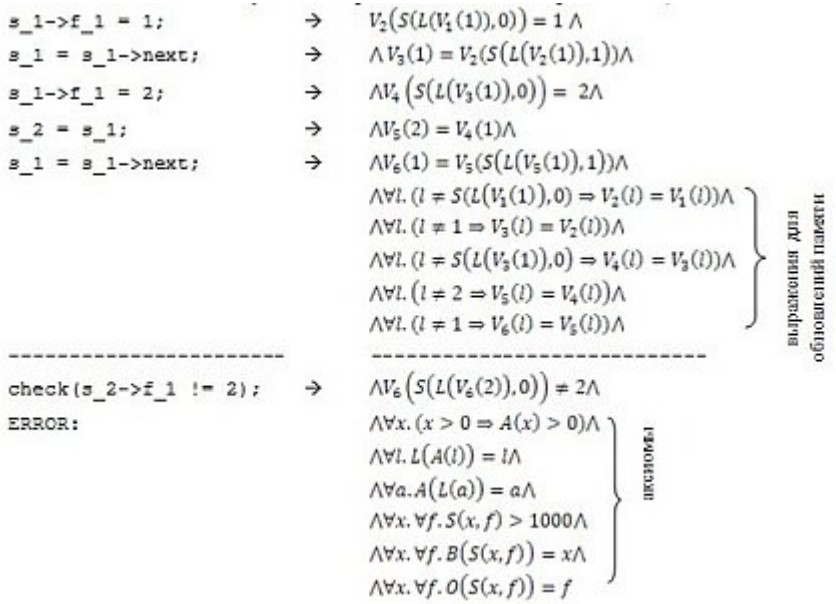
Здесь *n* – статический параметр, задающий размер генерируемого теста. При *n=2* этот тест будет выглядеть следующим образом:

```
s_1->f_1 = 1;
s_1 = s_1->next;
s_1->f_1 = 2;
s_2 = s_1;
s_1 = s_1->next;
----------------------
check(s_2->f_1 != 2);
ERROR:
```

В этом тесте указатель s_1 перемещается по связному списку длиной не менее двух элементов и последовательно присваивает его элементам значения 1 и 2 соответственно. Указателю s_2 присваивается адрес второго элемента списка. Оператор check условно обозначает переход по ветви, в которой выполнено заданное в нем условие. Таким образом, условие перехода на ошибочную метку ERROR – неравенство значения второго элемента списка двум. Оно не может быть выполнено, если верны предположения о корректности работы с памятью в исходной программе на данном пути выполнения. Пунктиром показана точка пути, для которой необходимо построить новый предикат, доказывающий его неосуществимость.

Как верхней (до пунктира), так и нижней части пути соответствует некоторая логическая формула. Для того чтобы верификатор мог исключить из абстракции программы данный невыполнимый путь, истинность построенного предиката должна следовать из формулы для верхней части этого пути, а конъюнкция этого предиката с его нижней частью должна быть невыполнимой. В предикате также должны использоваться только те значения переменных программы, которые они принимают именно в заданной точке (не до и не после нее). Тогда такой предикат можно будет использовать для

построения абстракции верхней части пути программы в указанной точке, в том числе, возможно, и при выполнении по какому-либо другому пути, проходящему через нее. Если в таком случае полученный предикат вновь окажется выполнен, он позволит вновь доказать недостижимость ошибочной метки уже вдоль другого пути. Такие требования к получаемому предикату как раз соответствуют ограничениям, накладываемым на интерполянт Крейга для пары логических формул[3]. Рассмотрим формулы и интерполянт, получаемый модифицированным инструментом CSIsat в данном примере (пусть переменной s_1 соответствует номер размещения 1, а s_2 – 2, в структуре два поля – f_1 и next с номерами 0 и 1 соответственно, константу N из четвёртой аксиомы возьмём равной 1000):



Интерполянт, выдаваемый CSIsat:

$V_6\left(S\left(L\left(V_6(2)\right),0\right)\right) = 2 \vee S\left(L\left(V_6(2)\right),0\right) \leq 1000$ → (что соответствует предикату) V(S(L(V(2)), 0)) = 2 → (то есть) s_2->f_1 == 2. Это условие действительно выполнено в указанной точке программы и доказывает невыполнимость вдоль заданного пути. Оно же позволяет доказать невыполнимость, например, вдоль следующего пути, также проходящего через указанную точку:

s_1->f_1 = 1;

[3] Интерполянт Крейга должен логически следовать из первой формулы, быть невыполним в конъюнкции со второй и содержать лишь общие для обеих формул неинтерпретируемые символы.

s_1 = s_1->next;
s_1->f_1 = 2;
s_2 = s_1;
s_1 = s_1->next;
---
check(s_2->f_1 == 2); // Переход по другой ветви if, например
s_1->f_1 = s_2->f_1 – 2;
check(s_1->f_1 != 0);
ERROR:

На Рис. 1-7 приведены результаты замеров времени работы нового инструмента на сгенерированных тестовых наборах. В легенде для каждого теста условно показан фрагмент кода программы на языке C, примерно соответствующий формуле пути, использованной для получения интерполянтов в этом тесте. В приведенных фрагментах используется параметр генератора тестов *n*, задающий размер теста таким образом, что количество строк кода в нем линейно зависит от этого параметра примерно с коэффициентом 2. Место разреза, разделяющее получаемую формулу пути перед интерполяцией на две части, обозначено пунктиром. Проверки условий в ветвлениях на соответствующем пути выполнения указаны в виде вызовов функции check. В каждом тесте ошибочная метка ERROR недостижима, а соответствующая формула пути до нее невыполнима. Она генерируется с использованием описанных неинтерпретируемых функций и аксиом и передается модифицированному инструменту CSIsat с опциями +extrInsts и +simplInsts. Во всех тестах CSIsat запускался с лимитом по памяти 1 GB или лимитом по времени 600 секунд. Тестирование прекращалось после пяти повторных неуспешных завершений работы инструмента (в том числе превышений лимита по памяти или времени, а также аварийных завершений).

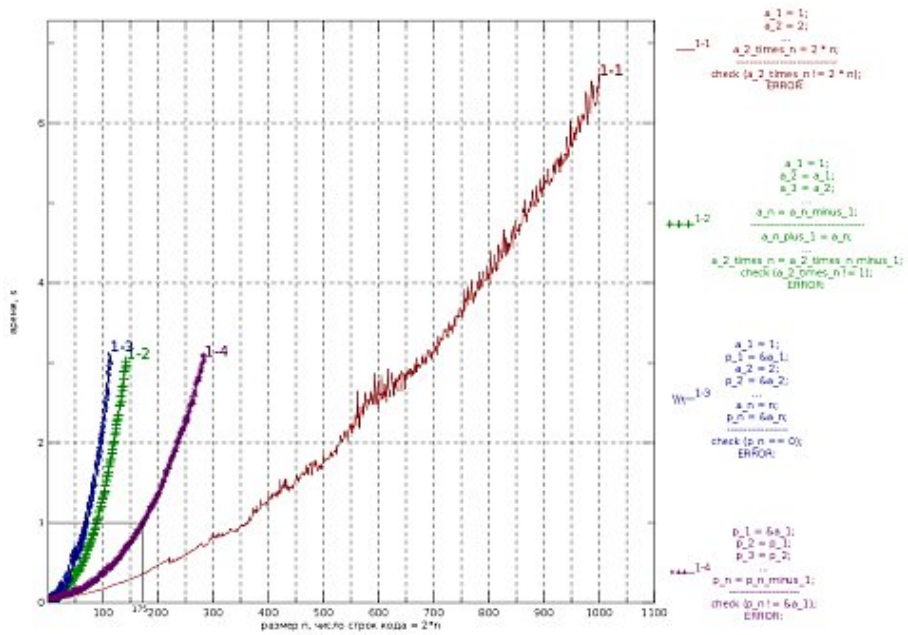


Рис. 1. Результаты тестирования на сгенерированном наборе.

На Рис. 1 приведены результаты замеров времени на интерполяцию в простых случаях, когда в пути отсутствуют операции с массивами или структурами. Кроме этого, для доказательства невыполнимости соответствующих формул не нужно инстанцировать выражения с кванторами для обновлений памяти (значение каждой переменной может использоваться только сразу после его обновления).

На Рис. 2 приведены результаты для аналогичных тестов с той разницей, что для доказательства невыполнимости в этих тестах требовалось линейное по количеству строк фрагмента программы число инстанцирований подкванторных выражений. Как видно из графиков, несмотря на то, что в обоих случаях с ростом размера теста время работы интерполятора возрастало примерно экспоненциально, в случае с линейным по размеру теста количеством инстанцирований время работы на некоторых тестах оказалось в 45 и более раз больше, чем в аналогичных случаях без инстанцирований (см., например, графики 1-4 и 2-4 для размера $n \approx 176$).

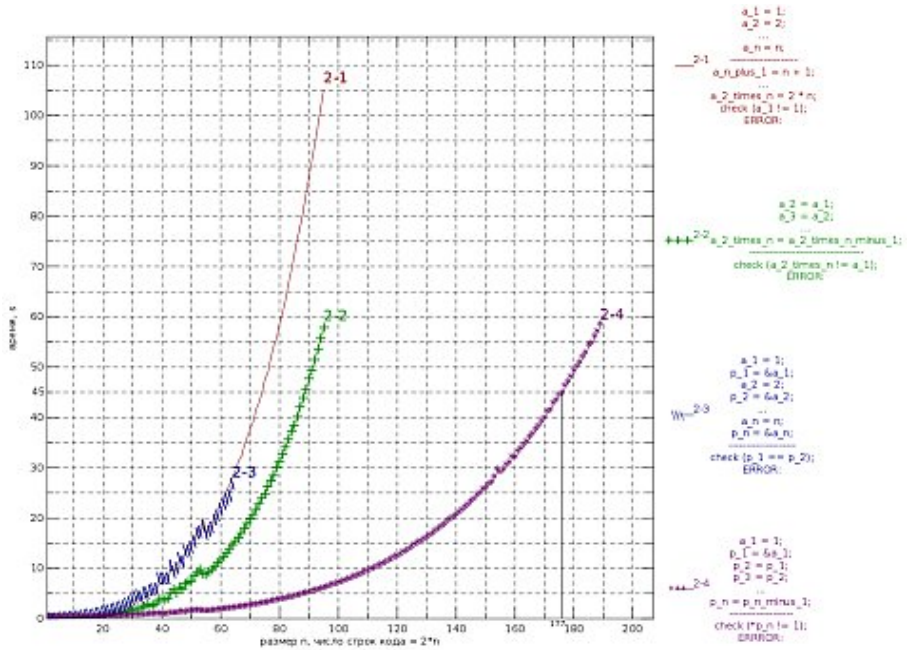


Рис.2. Результаты тестирования на сгенерированном наборе.

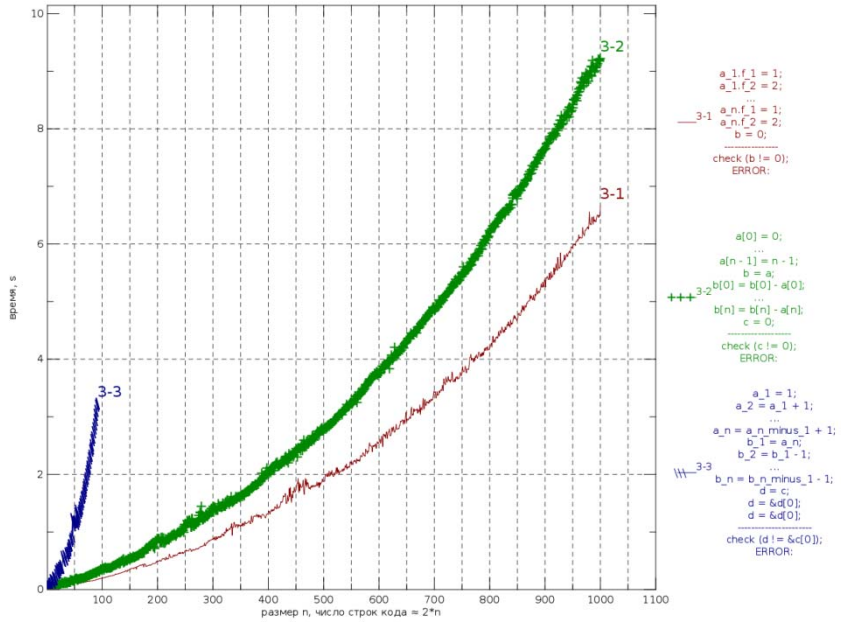


Рис. 3. Результаты тестирования на сгенерированном наборе.

На Рис. 3 приведены три графика: 3-1 и 3-2 для простых тестов с несущественным использованием массивов или структур (соответствующая часть формулы не требуется для доказательства невыполнимости), а также график 3-3 для теста, существенно использующего аксиому равенства адреса массива и его первого элемента.

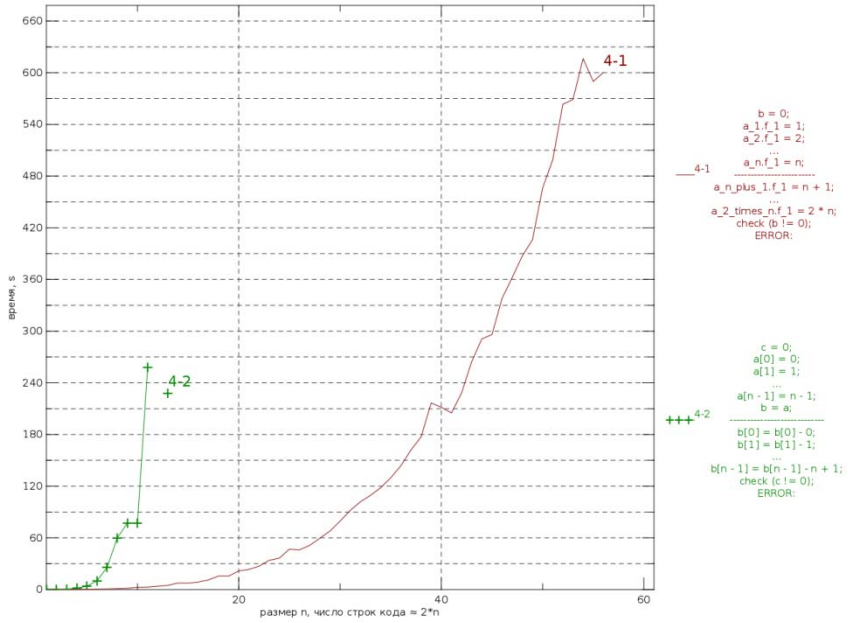


Рис. 4. Результаты тестирования на сгенерированном наборе.

На Рис. 4 приведены два графика для аналогичных тестов с линейным по размеру теста количеством инстанцирований. Разрыв графика 4-2 соответствуют отдельному аварийному завершению работы CSIsat (превышение лимита по памяти). Здесь разница в производительности интерполятора еще заметнее, чем на Рис. 1 и 2.

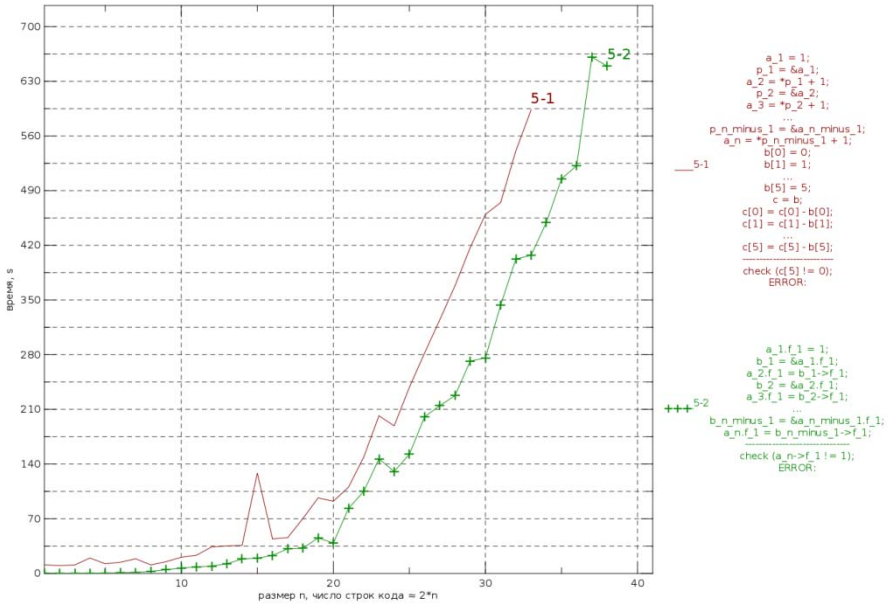


Рис. 5. Результаты тестирования на сгенерированном наборе.

На Рис. 5 в обоих тестах массивы или структуры использованы существенно. В тесте, соответствующем графику 5-1, для доказательства требовалось постоянное число инстанцирований, а в тесте для графика 5-2 – в два раза меньшее числа строк в пути. Однако на тестах небольших размеров эти величины оказались близкими (6 для первого теста и от 1 до 18 для второго).

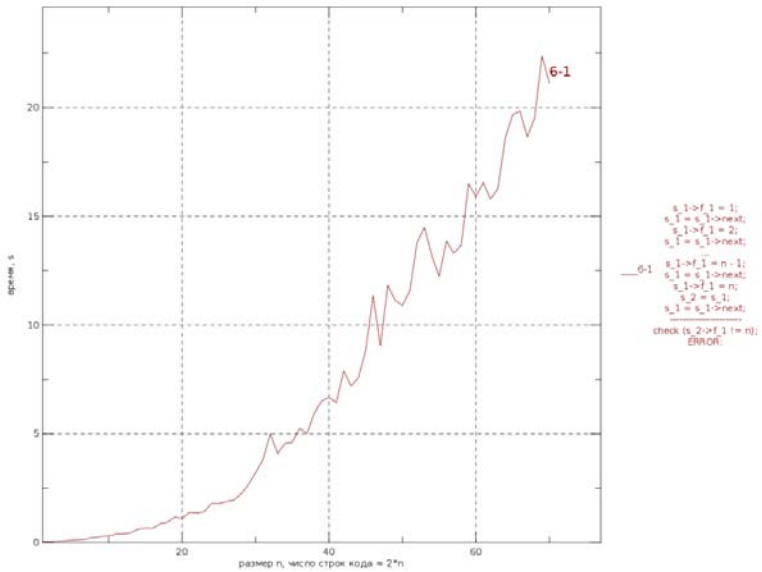


Рис. 6. Результаты тестирования на сгенерированном наборе.

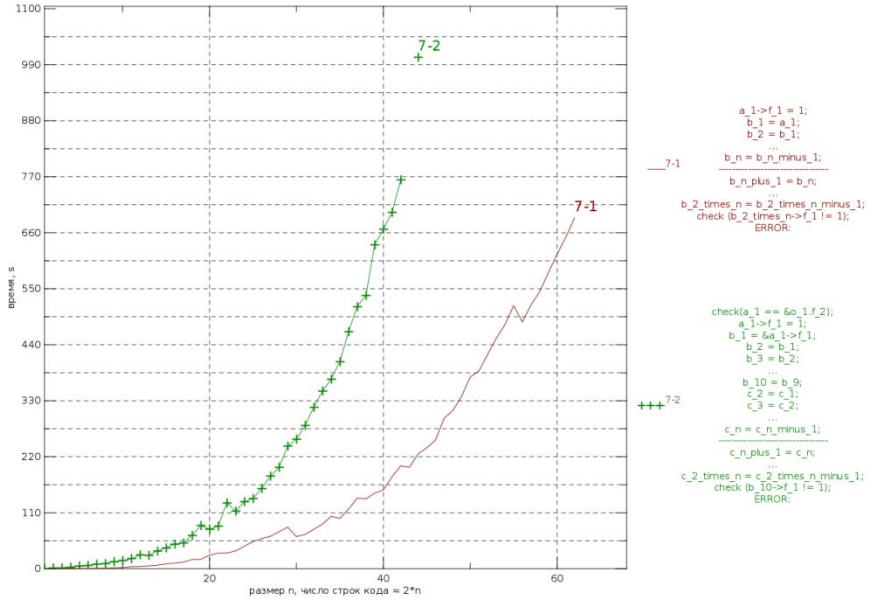


Рис. 7. Результаты тестирования на сгенерированном наборе.

На рисунках 6 и 7 приведены результаты для наиболее сложных тестов с существенным использованием структур. Разрыв графика 7-2 соответствуют отдельному аварийному завершению работы CSIsat (превышение лимита по памяти).

Результаты тестирования модифицированного инструмента показали, что он способен строить интерполянты для формул с кванторами лишь при условии, что для доказательства невыполнимости входной конъюнкции использовалось относительно небольшое число подстановок. При верификации, к примеру, драйверов ОС Linux средний размер трассы ошибки (то есть реальных контрпримеров) составляет порядка 1000-10000 операторов, а число вызовов интерполирующего решателя на один драйвер – примерно от 10 до 30[22]. Если считать разумным время верификации одного драйвера в пределах 15 минут, то для практического использования интерполятор должен в среднем справляться с формулами пути для 5000 строк за 45 секунд. Как видно, к примеру, из Рис. 2, даже для тестов без использования массивов или структур за 45 секунд модифицированный инструмент в лучшем случае может справиться с формулами для тестов с размером $n \approx 175$, то есть примерно с 350 строчками кода. Модифицированный инструмент, таким образом, оказался недостаточно эффективен для практического использования в качестве интерполятора при реализации в верификаторе рассмотренной логической модели памяти. Такая низкая производительность обусловлена как относительно низкой эффективностью решателя, используемого в CSIsat, так и использованием эвристик при извлечении необходимых подстановок из доказательства невыполнимости, полученного от CVC3. При использовании эвристик в получаемую задачу для интерполяции часто попадают несущественные для доказательства подстановки и, кроме этого, могут не попасть существенные, в результате чего инструмент завершит работу с ошибкой.

6. Выводы

Модифицированный инструмент CSIsat реализует интерполяцию Крейга для формул с кванторами, заданных в рамках комбинации теорий вещественной линейной арифметики и неинтерпретируемых функций. В нём используется предложенный в данной работе способ реализации интерполирования таких формул с помощью инстанцирования подкванторных выражений с использованием внешнего SMT-решателя CVC3.

Результаты тестирования модифицированного интерполятора как на задачах, полученных из набора SMTLIB, так и на специально сгенерированных для этих целей задачах показали общую работоспособность предложенного подхода и одновременно с эти его недостаточную эффективность.

Решатель, используемый при интерполяции для получения дерева резолютивного вывода в CSIsat, существенно уступает в эффективности CVC3. Поэтому одним из интересных направлений для дальнейшего

исследования является построение дерева резолютивного вывода по дереву доказательства невыполнимости, получаемому из CVC3. В таком случае удастся избежать существенных накладных расходов на повторное доказательство невыполнимости в самом CSIsat, которое, как оказалось по результатам проведенного тестирования, и являлось главным ограничением для инструмента.

Литература

[1] Ball, Thomas, Bounimova, Ella, Kumar, Rahul, and Levin, Vladimir. SLAM2: Static Driver Verification with Under 4% False Alarms. In *Formal Methods in Computer Aided Design* (2010).

[2] Graf, Susanne and Saidi, Hassen. Construction of Abstract State Graphs with PVS. In *Computer Aided Verification (CAV), 9th International Conference* (Haifa, Israel 1997), Springer, 72-83.

[3] Clarke, Edmund, Grumberg, Orna, Jha, Somesh, Lu, Yuan, and Veith, Helmut. Counterexample-Guided Abstraction Refinement. *Computer Aided Verification*, 1855 (2000), 154—169.

[4] Beyer, Dirk, Henzinger, Thomas A., Jhala, Ranjit, and Majumdar, Rupak. The software model checker BLAST: Applications to software engineering. *Int. J. Softw. Tools Technol. Transf.*, 9, 5 (2007), 505—525.

[5] Shved, Pavel, Mutilin, Vadim, and Mandrykin, Mikhail. Static Verfication `Under The Hood': Implementation Details and Improvements of BLAST. In *Proceedings of SYRCoSE* (Yekaterinburg 2011), 54-60.

[6] Beyer, Dirk and Keremoglu, M. Erkan. *CPAchecker: A Tool for Configurable Software Verification*. SFU-CS-2009-02, School of Computing Science, Simon Fraser University, 2009.

[7] De Moura, Leonardo and Bjorner, Nikolaj. Z3: an efficient SMT solver. In *Proceedings of the Theory and practice of software, 14th international conference on Tools and algorithms for the construction and analysis of systems* (Budapest, Hungary 2008), Springer-Verlag, 337—340. TACAS'08/ETAPS'08.

[8] Jhala, Ranjit and Majumdar, Rupak. Software model checking. *ACM Computing Surveys*, 41 (October 2009), 21:1—21:54.

[9] McMillan, Kenneth L. An Interpolating Theorem Prover. In *TACAS* (2004), 16-30.

[10] Beyer, Dirk, Zufferey, Damien, and Majumdar, Rupak. CSIsat: Interpolation for LA+EUF. In *CAV* (2008), 304-308.

[11] Cimatti, Alessandro, Griggio, Alberto, and Sebastiani, Roberto. Efficient generation of craig interpolants in satisfiability modulo theories. *ACM Transactions on Computational Logic (TOCL)*, 12, 1 (October 2010).

[12] Beyer, Dirk, Henzinger, Thomas A., and Théoduloz, Grégory. Lazy Shape Analysis. In *CAV'2006: Computer aided verification, LNCS 4144* (2005), Springer, 532—546.

[13] Reynolds, John. Separation Logic: A Logic for Shared Mutable Data Structures. (2002), IEEE Computer Society, 55—74.

[14] Ball, Thomas, Bounimova, Ella, Levin, Vladimir, and De Moura, Leonardo. *Efficient evaluation of pointer predicates with Z3 SMT Solver in SLAM2*. 2010.

[15] Henzinger, Thomas A., Jhala, Ranjit, Majumdar, Rupak, and McMillan, Kenneth L. Abstractions from proofs. *SIGPLAN Not.*, 39, 1 (2004), 232—244.

[16] Christ, Juergen and Hoenicke, Jochen. Instantiation-Based Interpolation for Quantified Formulae. In *Decision Procedures in Software, Hardware and Bioware* (Dagstuhl, Germany 2010), Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, Germany. Dagstuhl Seminar Proceedings.

[17] Moura, Leonardo and Bjorner, Nikolaj. Efficient E-Matching for SMT Solvers. In *Proceedings of the 21st international conference on Automated Deduction: Automated Deduction* (Bremen, Germany 2007), Springer-Verlag, 183—198. CADE-21.

[18] Barrett, Clark and Tinelli, Cesare. CVC3. In *Proceedings of the 19th International Conference on Computer Aided Verification (CAV '07)* (Berlin, Germany 2007), Springer-Verlag, 298—302. Lecture Notes in Computer Science.

[19] McMillan, Kenneth L. Interpolants from Z3 proofs. In *Formal Methods in Computer Aided Design (FMCAD 2011)* (Austin, TX, USA 2011).

[20] Barrett, Clark, Stump, Aaron, and Tinelli, Cesare. *The SMT-LIB Standard Version 2.0*. 2010.

[21] Library, SMT-LIB The Satisfiability Modulo Theories. *www.smtlib.org*. 2011.

[22] Khoroshilov, Alexey, Mutilin, Vadim, Novikov, Eugene, Shved, Pavel, and Strakh, Alexander. Towards An Open Framework for C Verification Tools Benchmarking. In *Proceedings of PSI* (Akademgorodok, Novosibirsk, Russia 2011), 82—91.