Лиа Хоффман (Leah Hoffman)
(Перевод К. Фируна (constantine@firun.ru) статьи Hoffman L. Talking Model-Checking Technology // Communications of the ACM. 2008. Vol. 51. № 07/08, p.110-112).

Разговоры о model checking

Беседа с учёными получившими премию ACM имени Тьюринга в 2007 году.

Эдмунд М. Кларк, Аллен Эмерсон и Джозеф Сифакис (Edmund M, Clarke, E. Allen Emerson, и Joseph Sifakis) были удостоены премии за их роль в разработке Model checking,ставшей эффективной технологией верификации, широко распространённой в программно-аппаратной индустрии.

Давайте поговорим об истории формальной верификации программ.

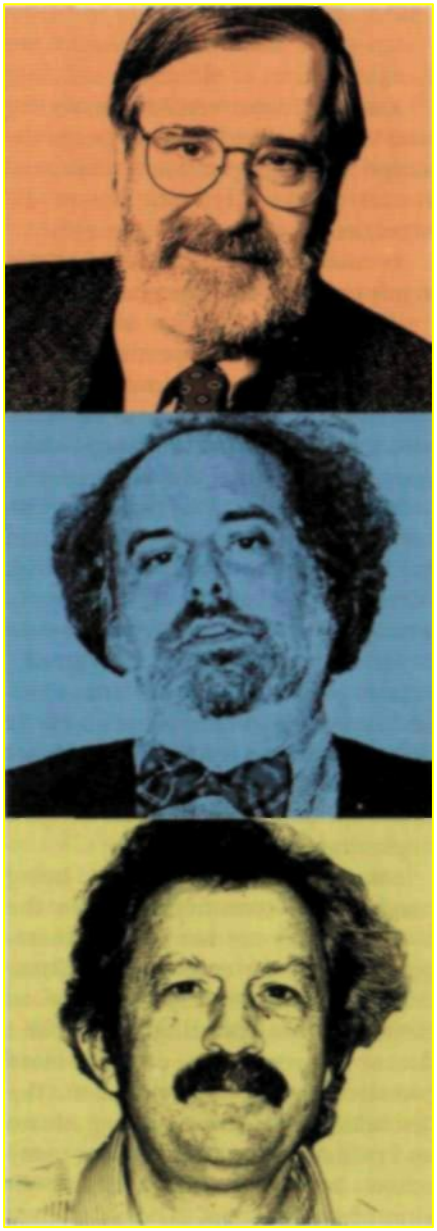
Аллен Эмерсон: К концу 60-ых мы поняли, что программа должна рассматриваться как математический объект. У него есть синтаксис и семантика и поведение этого объекта формально ими описано. Идеей было дать математическое доказательство, что программа была верна при определённых условиях. Таким образом одни рассматривали аксиомы, описывающие работу программы, а другие конструировали формальное доказательство правильности работы программы, как философы в общем-то.

Но казалось, что эту методику не применить к большим и сложным программам. Мы заканчивали с 15 страничным отчётом, о том что программа на полстраницы работала корректно. В общем-то это было отличной идеей, но в реальной жизни это навряд ли бы пригодилось бы.

Что насчёт истории model checking?

Эдмунд М. Кларк: Вообще, рождение *model checking* местами было очень болезненно. Как и в любом деле на стыке теории и практики, теоретики считали, что всё очень просто, а программисты думали что это слишком теоретично. Исследователи в области формальных методов были ещё менее восприимчивы к проблеме. Исследование в области формальных методов в 80-ые обычно состояло в верификации программ на 50 строк кода, а то и меньше, используя только ручку и бумагу. Если кто-нибудь решался спросить как эта программа работала на реальном компьютере, то это воспринималось как оскорбление, или, быть может, как неважное замечание.

АЭ: Идея model checking состояла в том чтобы избавить людей от конструирования доказательств. Выходит, что многие важные программы, например, как операционные системы, имеют



поведение в реальном времени и в идеале работают бесконечно долго. Они не просто запускаются и завершаются. В 1977 году Амир Пнуэли (Amir Pnueli) предложил, что временная логика может хорошо подойти для описания программ. Сейчас, если программа может быть описана при помощи временной логики, то она может быть реализована в виде программы с конечным числом состояний. Это стало толчком для развития идеи *model checking* – проверять является ли конечный граф состояний моделью временной логики. Затем можно разработать эффективный алгоритм, проверяющий описание временной логики на верность, используя поиск определённых шаблонов на графе состояний.

ЭМК: Да, Аллен и я заметили, что многие параллельные программы имели свойство, которое мы назвали «скелет синхронизации с конечным числом состояний (*finite state synchronization skeletons*)» (Джозеф Сифакис и Дж.П. Келль (J.P. Queille) сделали тот же вывод независимо). Например, часть программы взаимного исключения, которая отвечает за синхронизацию, не зависит от данных, которыми обмениваются в критической секции. Многие коммуникационные протоколы обладают тем же свойством. Мы решили попробовать анализировать алгоритмически программы с конечным числом состояний.

Как это работает?

АЭ: У вас есть программа, описанная своим исходным кодом, и её спецификация, описанная на языке логики. Либо верно, либо нет, что программа отвечает спецификации, и нужно проверить это.

Джозеф Сифакис: Верно. Вы строите математическую модель программы, и по этой модели вы проверяете свойства, которые также математически определены. Чтобы проверить свойство вам необходим алгоритм *model checking,* который берёт на вход построенную вами математическую модель и даёт на выходе ответ: «да», «нет» или «не знаю». Если свойство не верифицировано, то вы получаете диагностические сведения.

И для формализации этих спецификаций, этих свойств…

АЭ: Люди на самом деле хотят программы, которые по сути ничем не отличаются от неформального представления. У них есть некоторые неясные идеи о том, что за программу они хотят, или, быть может, у них есть на руках техническое задание, написанное филологами в форме прозы, но это не математическая проблема.

Так что, одно из преимуществ model checking заключается в том, что вам приходится описывать программу более формально и точно.

ЭМК: Да. Но для многих людей, наиболее важным преимуществом является возможность представить контр-пример, если не удалось достичь требований спецификации. Другими словами, это позволяет вам добраться до того места, где вы совершили ошибку,

которая рушит требования спецификации, и очень часто такой метод помогает найти едва различимые ошибки в проектировании.

Как развивались алгоритмы model checking за эти годы?

ЭМК: Алгоритмы *model checking* серьёзно развивались за последние 27 лет. Первый алгоритм, созданный Алленом и мною, и независимо Келлем и Сифакисом, был *fixpoint* алгоритмом и время его работы росло с квадратичным увеличением числа состояний. Я сомневаюсь, что этот алгоритм был способен обработать систему с 1000 состояний. Первой реализацией был *EMC Model Checker* (EMC расшифровывает как "*Extended Model Checker*"), который базировался на эффективных алгоритмах на графах, разработанных совместно с Алленом и Прасадом Систлой (Prasad Sistla). EMC работал за линейное время и мы могли верифицировать конструкции с40000 состояний. Из-за проблемы комбинаторного взрыва, это было не очень выгодно – и мы всё равно не могли верифицировать конструкции из реальной жизни. Мой студент Кен МакМиллан (KenMcMillan) предложил куда более мощную технику, называемую *symbolic model checking*. Мы смогли проверять некоторые примеры с 10^{100}состояний. Это было существенным прорывом, но всё равно мы не могли угнаться за комбинаторным взрывом. В конце 90-ых группе под моим руководством удалось разработать методику под названием bounded model checking, которая позволила нам находить ошибки в конструкциях с 10^{10000} состояний.

АЭ: Эти прорывы являются основами *model checking*. Впервые индустриальные проекты верифицируются и в этом нет ничего необычного. Такие организации как IBM, Intel, Microsoft и NASA обладают такими областями, в которых model checking полезен и необходим. Кроме того, сейчас есть большое количество сторонников *model checking*, включая пользователей технологии и исследователей, вносящих вклад в развитие.

Какие ограничения у model checking?

ДС: Есть две проблемы: как построить математическую модель системы, и как проверить свойство, требование на такой математической модели.

Начнём с того, что сконструировать модель сложной системы может быть очень сложно. Для аппаратных средств построить модель довольно просто и у нас получилось продвинуться достаточно далеко в этой области. Для программного обеспечения задача несколько более сложная. Конечно от того как написана программа многое зависит, но мы уже можем верифицировать много сложных программных систем. Но для систем в которых играет роль и аппаратная часть, и программная мы ещё не знаем как создать точную модель для верификации.

Другое ограничение заключается в сложности верифицирующего алгоритма. И здесь у нас возникла проблема комбинаторного взрыва.

ЭМК: Верификация программного обеспечения является сложной задачей. Комбинируя model checking с приёмами статичного анализа, стало возможно находить ошибки, но не давать правильного доказательства. Что касается комбинаторного взрыва, всё зависит от логики и модели вычислений, и вы можете доказать что этот процесс неизбежен. Но мы разработали пару приёмов для борьбы с этим.

Например?

ЭМК: Наиболее важный приём – это абстракция. Идея заключается в том, что часть программы или протокола, которую вы верифицируете, не имеет реального эффекта на те свойства что вы проверяете. Так что вы можете просто удалить эти части из конструкции.

Так же вы можете комбинировать model checking с композиционным заключением (*compositional reasoning*), когда вы берёте сложную конструкцию и разделяете её на более маленькие кусочки. Затем вы проверяете получившиеся компоненты для заключения о корректности всей системы.

Насколько велики сейчас программы, которые мы можем верифицировать?

ЭМК: Ну, вообще-то, не всегда есть прямая связь между сложностью программы и её размером. Но могу сказать, что мы часто проверяем схемы с 10^{100} состояний.

ДС: Верно. Сегодня мы знаем как верифицировать системы средней сложности – сложно дать оценку, но это примерно программы на 10000 строк кода. Но мы не знаем, как верифицировать очень сложные системы.

ЭМК: Мы всегда играем в догонялки, и мы всегда догоняем. Мы разработали сложные и мощные приёмы, но всё равно нам сложно удержаться в погоне за ростом технологий и сложности новых систем.

Можно ли использовать model checking для проверки параллельных программ?

АЭ: Вероятно, *model checking* очень хорошо подходит для параллельного программирования. Типично, мы рассматриваем параллельные вычисления недетерминистскими, или, говоря более понятно, вычисления со случайным выбором, так что параллельная программа более сложна, чем последовательная. *Model checking* очень хорошо подходит для описания параллельных программ.

ЭМК: Параллельные программы более сложны в отладке, так как людям сложно отследить за столькими вещами, происходящими в один момент времени. *Model checking* идеально подходит для этого.

ДС: Но для программ, которые взаимодействуют с окружающей действительностью, время становится очень важным. Для таких систем верификация намного более сложна.

Есть ли алгоритмы, которые работают прямо с исполняемым (*implementable*) кодом?

ЭМК: Верифицировать процесс транслирования конструкции в код, или верифицировать сам код намного сложнее. Хотя многие успешные реализации *model checking* используют этот подход. Реализация Java Pathfinder, разработанная NASA, генерирует байт-код для программ на Java и имитирует проблемы для нахождения ошибок в коде.

ДС: Лучшие технологии на сегодняшний день - это те, которые были разработаны американскими компаниями. Но большинство реализаций *model checking* для проверки кода используются для верификации последовательных программ. Если вы хотите верифицировать параллельное программное обеспечение, то вам следует быть очень осторожным.

ЭМК: Средство SLAM от Microsoft, предназначенное для нахождения ошибок в драйверах для Windows, вероятно, самая успешная реализация model checking. SLAM свободно распространяется и доступен для тех, кто хочет писать драйверы устройств для Windows. Однако, это очень узконаправленная реализация model checking.

АЭ: В верификации аппаратных средств широко используются языки Verilog и VHDL. Многие реализации *model checking*, применяющиеся на практике, распознают эти языки.

Преподаётся ли model checking студентам?

ДС: Формальная верификация преподаётся в Европе. Европа изначально была сильна в формальных методах и семантических языках.

ЭМК: Да, в Европе всегда был больший интерес к верификации чем в США. Большинство главных университетов в США - CMU, Stanford,

UC Berkeley, U. Texas, и т.д. – предлагают курсы по *model checking* для студентов и аспирантов, но во многих высших школах по прежнему ничего не знают о *model checking*. Отчасти это связано с отсутствием хороших книг на эту тему.

АЭ: В США формальные методы преподаются с некоторой регулярностью, но они не столь широко внедрены в университетские программы, как курсы по операционным системам, структурам данных. Возможно, они более распространены среди выпускников. Но различие между выпускниками и студентами не столь явно выражено. Во многих высших школах программы продвинутых выпускников и студентов-первокурсников пересекаются.

Что ждёт model checking и верификацию вообще?

ЭМК: Я продолжаю поиски путей для увеличения мощности model checking. До сих пор комбинаторный взрыв – очень сложная задача. Я работаю над этой задачей уже 27 лет и, пожалуй, продолжу работу над ней. Другая тема, на которой я хочу остановиться - это встраиваемые системы в автомобильной и авиационной промышленности. Эти приложения зачастую очень критичны в области безопасности. Например, в течении нескольких лет большинство новых машин будут построены по схеме «*drive-by-wire*», т.е. в них не будет механических соединений между рулём и колёсами. Используемое в таких системах программное обеспечение необходимо верифицировать. К счастью, встраиваемые системы более просты, чем остальное программное обеспечение. Я считаю, что подобные системы более подходящими для верификации и для *model checking* в частности, чем остальные программные разработки.

ДС: Лично я считаю, что нам необходимо углубиться в изучение приёмов композиционного заключения, с помощью которых, мы могли бы делать вывод, полагаясь на локальные свойства, а не на глобальные. Я работаю над этим, равно как и над теориями того, как строить системы из компонентов – системы, основанные на использовании компонентных объектов.

АЭ: *Model checking* изменило наш взгляд о нахождении корректности программы от теорий, связанных с доказательством верности (дедуктивное доказательство) до теорий связанных с моделями (поиск на графе). Я считаю, что мы продолжим прогрессивно двигаться в этом направлении – медленно или быстро, но в независимости от этого скорость разработки программно-аппаратных средств возрастёт. Сможем ли мы догнать прогресс – я не знаю. Разрабатываемые системы становятся всё больше и сложнее. Спонтанный подход к разработке систем не будет эффективен в будущем. Нам придётся улучшить свои способности в модульной разработке систем, и нам понадобятся лучшие абстракции.