Исследования Microsoft Research в области верификации программных систем

Elena Pavlova
Microsoft
v-elpavl@microsoft.com

Microsoft

Microsoft Research

Области исследований:

- Прикладная математика и информатика
- Науки о земле, энергетика и науки об окружающей среде
- Образование
- Здравоохранение и благосостояние
- Естественные пользовательские интерфейсы

MSR Cambridge, UK

MSR Redmond, USA

MSR Bangalore, India

MSR Silicon Valley, USA

MSR New England, USA

MSR Beijing, China

Содержание

- Понятие верификации
- Фундаментальные разработки Microsoft Research в области верификации
- Верификация параллельных систем
- Проверка свойств безопасности
- Верификация драйверов устройств
- Источники дополнительной информации

Понятие верификации

- Формальная верификация – проверка выполнения определённых формальных свойств для программы (модели) при помощи формальных методов
- Области применения – mission-critical applications: программно-аппаратные комплексы для космонавтики, военного дела, авиации, медицины, борьбы с пожарами и т.д.

Фундаментальные разработки Microsoft Research в области верификации

- Z3
- Boogie
- Spec#
- Code Contracts
- Zing

Z3 – SMT Solver

Z3 – низкоуровневый инструмент. Обычно используется в качестве компоненты системы верификации вместе с другими инструментами в качестве решателя формул. Z3 предоставляет ряд API для того чтобы разнородные программы могли использовать Z3. Поэтому не существует отдельного user-centric редактора для взаимодействия с Z3. Синтаксис входного языка более ориентирован на простоту, чем на лингвистическое удобство.

Проверка при помощи Z3:

- Команда **assert** добавляет формулу во внутренний стек Z3. Множество формул в стеке считается выполнимым, если существует интерпретация (для определённых пользователем констант и функций), которая обращает все формулы в истину
- Если формулы в стеке выполнимы, Z3 возвращает **sat**. Если множество формул невыполнимо, возвращается **unsat**. Z3 также может вернуть **unknown**
- Если команда **check-sat** возвращает **sat**, то команда **get-model** может быть использована для получения интерпретации, которая обращает все формулы стека Z3 в истину

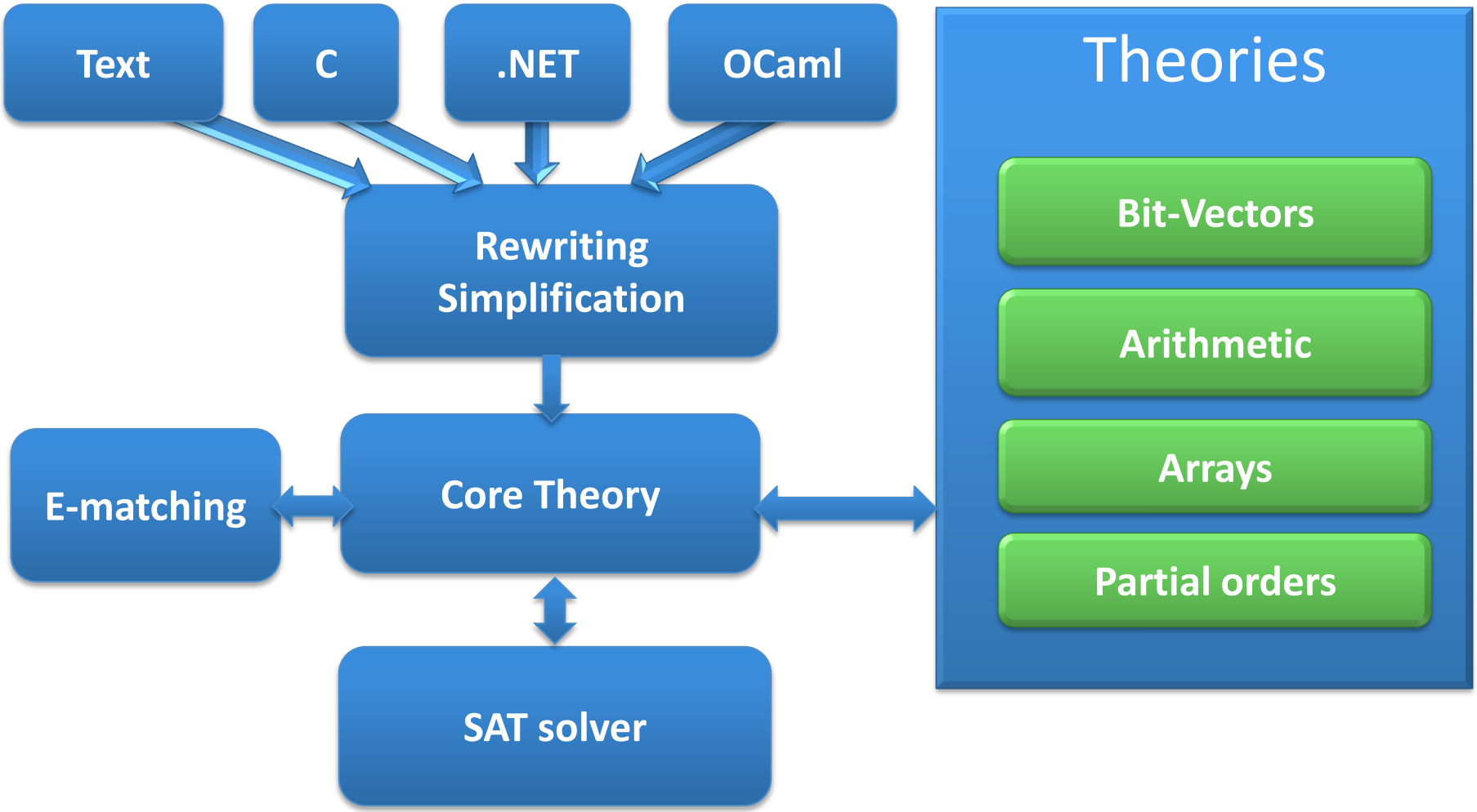
http://research.microsoft.com/projects/z3

http://rise4fun.com/Z3

Z3 - кратко об основных особенностях

- Поддержка следующих теоретических основ:
  - Линейная целочисленная и вещественная арифметика
  - Битовые векторы фиксированного размера
  - Неинтерпретированные функции
  - Массивы
  - Кванторы
- Генерация моделей
- Несколько входных форматов (Z3, Simplify, SMT-LIB, Dimacs)
- Расширенные API (C/C++, .Net, OCaml)

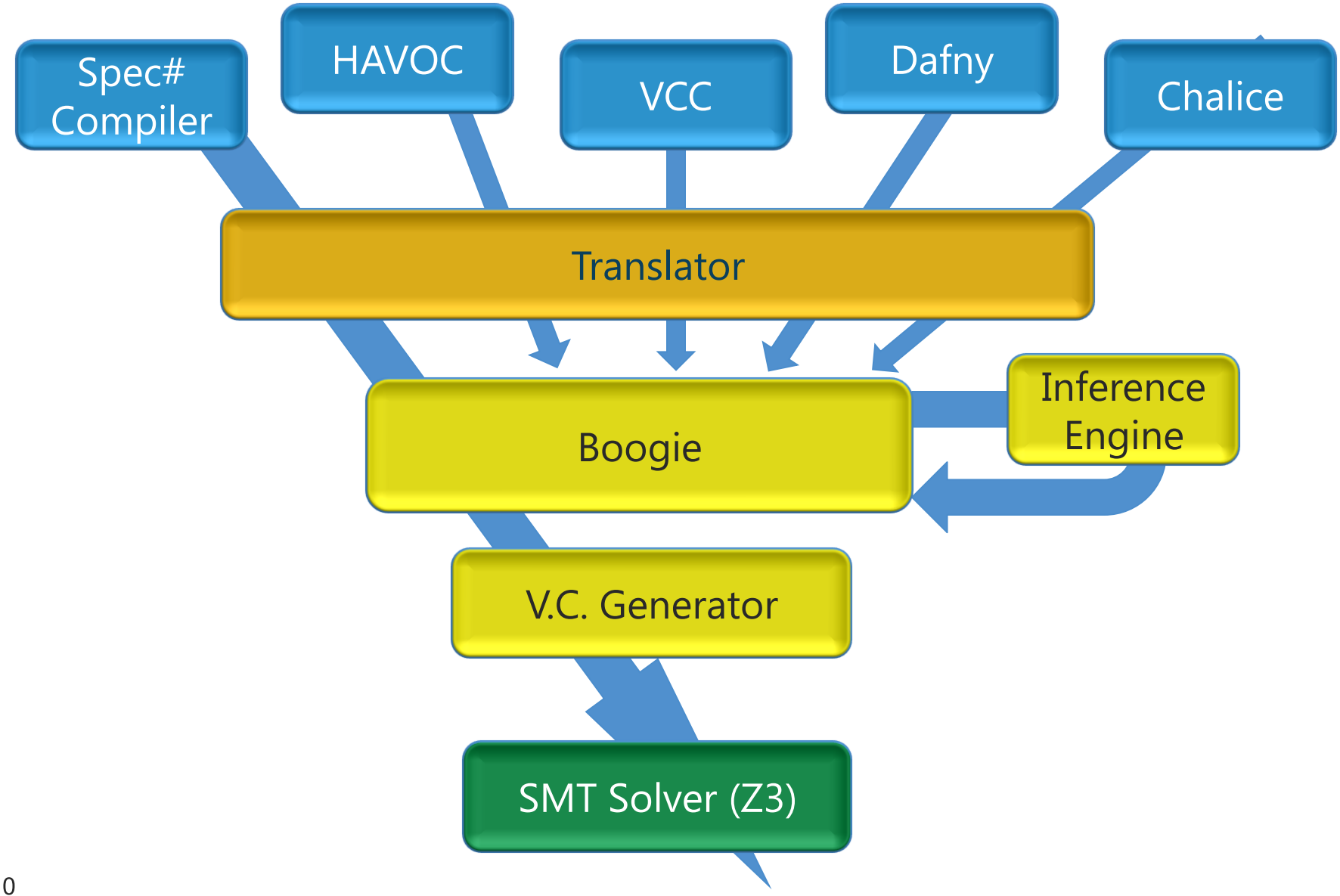
Z3 - основные элементы архитектуры



Boogie

- Boogie – язык промежуточного уровня для верификации, предназначенный для построения верификаторов других языков. На основе Boogie было построено несколько верификаторов, в том числе верификатор языка Spec#, верификаторы C - HAVOC и Vcc, язык и верификатор Dafny, а также язык Chalice. Предыдущая версия Boogie называлась BoogiePL, новая - Boogie 2.
- Boogie – также программная система, принимающая на вход язык Boogie, выводящая при необходимости некоторые инварианты для заданной программы на языке Boogie, и генерирующая затем верифицируемые свойства, которые передаются решателю (SMT solver). По умолчанию в качестве решателя используется Z3.
- Интегрированы в Microsoft Visual Studio
- http://research.microsoft.com/en-us/projects/boogie/
- http://research.microsoft.com/en-us/um/people/leino/papers/krml160.pdf
- http://rise4fun.com/Boogie

Boogie


Spec#
Compiler
HAVOC
VCC
Dafny
Chalice
Translator
Boogie
Inference
Engine
V.C. Generator
SMT Solver (Z3)

Spec#

Spec# - формальный язык для описания контрактов API (разработанный под влиянием JML, AsmL и Eiffel), расширяющий C# конструкциями для ненулевых типов, пред- и постусловий, и различных объектных инвариантов. В основе Spec# лежит полная программная методология, позволяющая создавать спецификации и выполнять формальный вывод для объектных инвариантов, в том числе, с учётом многопоточности и обратных вызовов. Spec# применялся исследователями для непосредственного анализа спецификаций и построения инструментов статического и динамического анализа. Spec# system состоит из:

- Языка программирования Spec#, являющегося расширением C#
- Компилятора Spec#, интегрированного в Microsoft Visual Studio. Компилятор проводит статическую проверку non-null types, вставляет проверки времени выполнения для контрактов методов и инвариантов и записывает контракты в метаданные, чтобы их затем могли использовать другие инструменты анализа
- Статического верификатора Spec# (Boogie), генерирующего логические условия, которые затем должны быть проверены для программы на Spec#. При этом используется средство автоматического доказательства теорем, анализирующее эти условия и доказывающее корректность программы (или нарушение условий)

Особенностью Spec# является гарантированная поддержка инвариантов с учётом callbacks, многопоточности и отношений между объектами

http://research.microsoft.com/en-us/projects/specsharp/

http://specsharp.codeplex.com/

Spec#

- http://rise4fun.com/SpecSharp


```
class SlowpokeAddition {
  public int Add(int x, int y)
    requires 0 <= x && 0 <= y;
    ensures result == 2*x + y;
  {
    int r = x;
    for (int n = 0; n < y; n++)
      invariant r == x+n && n <= y;
    {
      r++;
    }
    return r;
  }
}
```

ask spec# *Is this program correct?* ***Click 'ask spec#'!*** home video

		Description	Line	Column
	1	Method SlowpokeAddition.Add(int x, int y), unsatisfied postcondition: result == 2*x + y	2	14
	2	(trace position)	13	3

```
c.ssc(2,14): warning CS2663: Method SlowpokeAddition.Add(int x, int y), unsatisfied
postcondition: result == 2*x + y
c.ssc(13,3): warning CS2663: (trace position)
```

Code Contracts

Code Contracts предоставляет способ выражения свойств кода для языков .NET. Контракты-свойства могут представлять собой пред- и постусловия, а также инварианты объектов. Контраткы можно рассматривать как документацию или проверяемую спецификацию для внешних и внутренних API. Контракты могут быть использованы для тестирования посредством проверок времени выполнения, статической верификации контрактов и генерации документации.

Code Contracts позволяют воспользоваться преимуществами подхода design-by-contract для всех языков семейства .NET. Основные особенности Code Contracts:

- **Улучшение процесса тестирования**
- Каждый контракт действует как оракул, предоставляя для каждого тестового прогона информацию pass/fail.
- Средства автоматического тестирования, такие как Pex, могут использовать контракты для улучшения генерации unit тестов, исключая тесты, чьи аргументы не удовлетворяют пред-условиям.
- **Static verification** В настощее время разрабатываемые средства статического анализа используют контракты для уменьшения числа ложных положительных срабатываний и генерации более информативных сообщений об ошибках.
- **Contracts** выражаются в виде вызовов статических методов на входе в методы, для которых нужно задать контракт. Методы находятся в пространстве имён **System.Diagnostics.Contracts**.
- http://research.microsoft.com/en-us/projects/contracts/

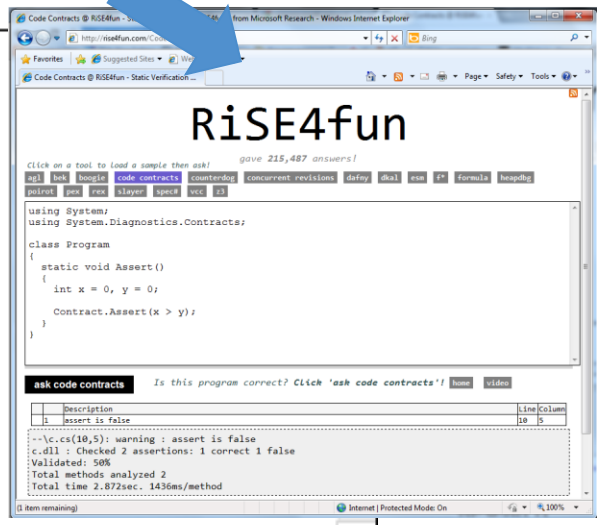
Code Contracts

http://rise4fun.com/CodeContracts

```
using System;
using System.Diagnostics.Contracts;

class IncorrectClass
{
  static void IncorrectFunction()
  {
    int x = 0, y = 0;

    Contract.Assert(x > y);
  }
}
```



ask code contracts *Is this program correct? Click 'ask code contracts'!* home video

		Description	Line	Column
	1	assert is false	10	5

```
--\c.cs(10,5): warning : assert is false
c.dll : Checked 2 assertions: 1 correct 1 false
Validated: 50%
Total methods analyzed 2
Total time 2.956sec. 1478ms/method
```

Zing

- Инструмент для верификации на модели (Model-checker)
- Целью проекта Zing является построение гибкой и масштабируемой инфраструктуры для систематизированного анализа пространства состояний программной системы.
- В основе инфраструктуры лежат новые алгоритмы редукции частичного порядка. Она может быть использована для обнаружения ошибок в программных системах на уровне описаний протоколов, спецификаций workflow, веб-сервисов, драйверов устройств и протоколов в ядре операционной системы.
- http://research.microsoft.com/en-us/projects/zing/

Верификация параллельных систем

- VCC

  http://research.microsoft.com/en-us/projects/vcc/
- HAVOC

  http://research.microsoft.com/en-us/projects/havoc/
- Traver

  http://research.microsoft.com/en-us/projects/traver

VCC – верификатор параллельных программ на языке C

- VCC – инструмент доказательства корректности аннотированных параллельных программ на языке C и обнаружения в них различных ошибок. VCC расширяет C средствами описания контрактов, в том числе в виде пред-, постусловий и инвариантов типов. Аннотированные таким образом программы транслируются в логические формулы при помощи инструмента Boogie, который затем передаёт их автоматическому решателю Z3 для проверки выполнимости
- VCC доступен для некоммерческого использования на сайте codeplex. Там же есть исходный код

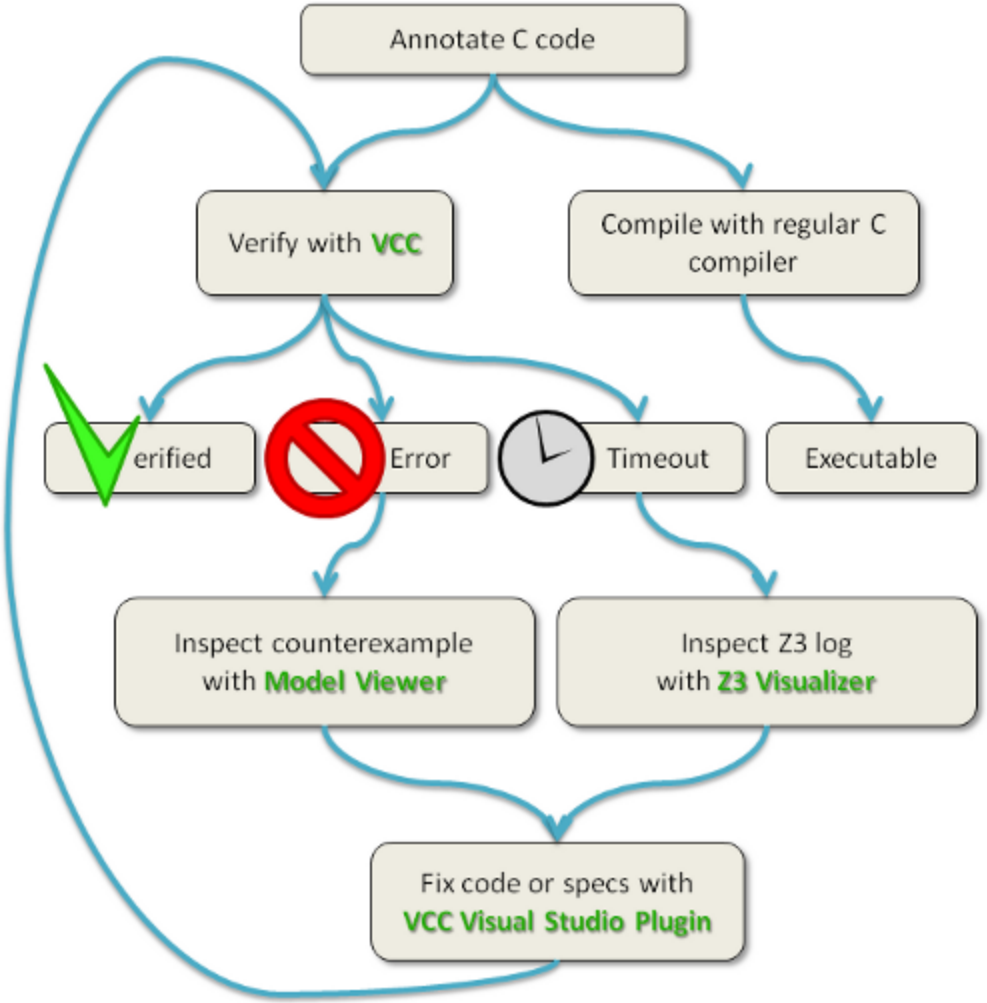
- http://research.microsoft.com/en-us/projects/vcc/
- http://vcc.codeplex.com/

VCC – верификатор параллельных программ на языке C

Аннотация и проверка кода на C при помощи контрактов.

Контракты – это макросы для препроцессора C, так что их легко отключить и перекомпилировать код на C без аннотаций. В случае программы, аннотированной контрактами, VCC проводит анализ программы и выдаёт один из возможных результатов:

- Программа корректна
- Возможно нарушение определённых свойств (assertions) во время выполнения программы. В таком случае можно использовать **Model Viewer** (инструмент, разработанный в рамках VCC проекта) для анализа условий, в которых возможно нарушение свойства
- Задача верификации неразрешима. В таком случае можно проанализировать ход доказательства при помощи **Z3 Axiom Profiler (предыдущее название -** Z3 Visualizer) – ещё один инструмент, созданный в рамках проекта VCC. Результаты анализа могут быть использованы для модификации спецификации



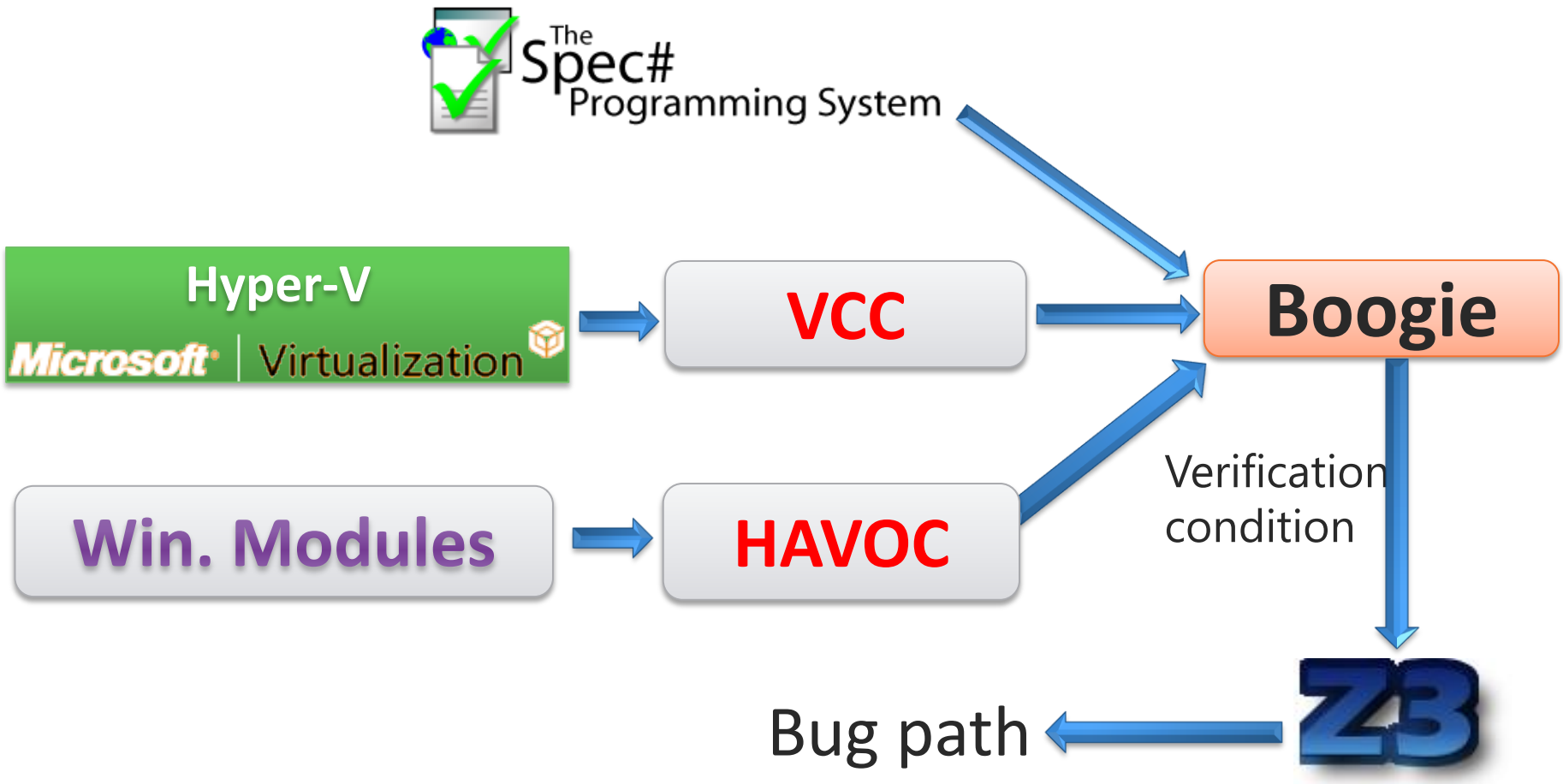
HAVOC

- HAVOC – инструмент для спецификации и проверки свойств программных систем, написанных на языке C, с учётом операций над указателями, небезопасных преобразований типов и динамического выделения памяти. Логика HAVOC позволяет в том числе формулировать свойства для таких конструкций, как связные списки и массивы.
- Основные усилия проекта HAVOC направлены на (1) решение дилеммы выбора между выразительными возможностями языка и эффективностью решателя, (2) разработку алгоритмов вывода, позволяющих обойтись без пользовательских аннотаций, что особенно важно для больших программных систем, состоящих из множества модулей.
- В качестве решателя используется Boogie + Z3
- http://research.microsoft.com/en-us/projects/havoc/
- Tom Ball, Towards Scalable Modular Checking of User-defined Properties - http://research.microsoft.com/pubs/132138/paper.pdf http://fm.csl.sri.com/UV10/slides/BallUV2010.pptx

Traver

- Traver (аббревиатура от Transformation Verification). Traver ориентирован на разработчиков прикладных программ и компиляторов. Traver – инструмент поддержки анализа возможных результатов трансформаций параллельных программ, в том числе
- Анализ влияния различных моделей памяти (особенно relaxed memory models) на корректность трансформаций
- Анализ проблемы обеспечения корректности оптимизированной программы (оптимизация не должна влиять на определённое поведение программы) и корректного преобразования синхронизирующих конструкций (блокировки) в конструкции, поддерживаемые целевой аппаратной платформой
- Traver предоставляет строгое теоретическое обоснование корректности трансформаций программы
- Traver представляет собой автоматизированное средство поддержки доказательства корректности
- http://research.microsoft.com/en-us/projects/traver

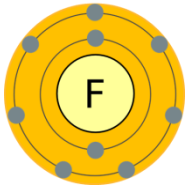
Комплексное применение инструментов



Проверка свойств безопасности - Fern

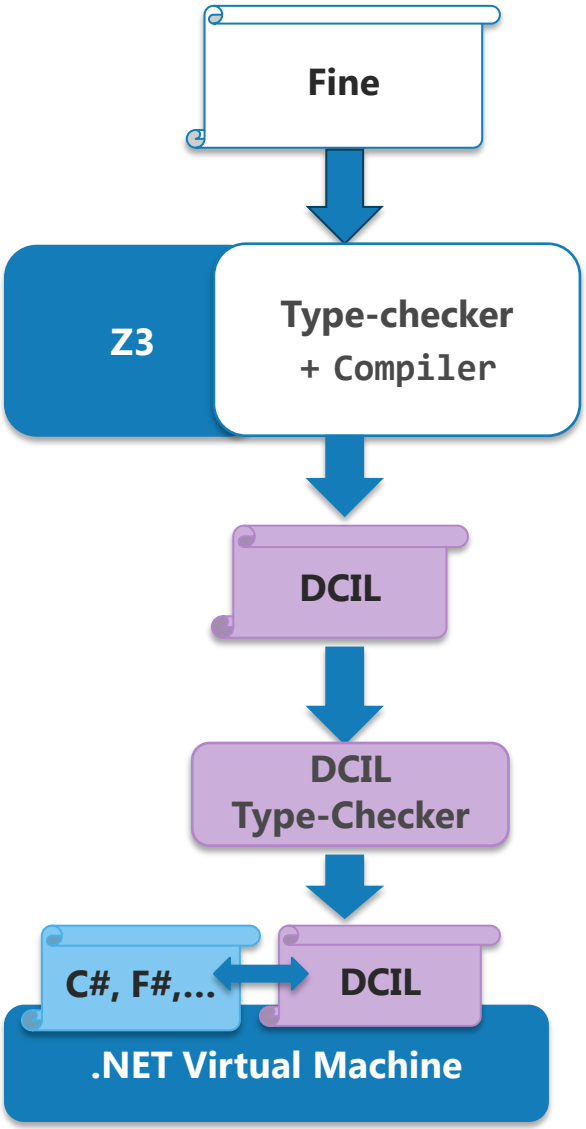
- Проверка свойств безопасности является одной из важнейших задач в области обеспечения качества программных систем. Целью проекта Fine является разработка методов и программных средств поддержки построения, верификации и развёртывания систем с гарантированными свойствами (которые можно доказать)
- Для достижения этой цели была спроектирована и реализована система типов, основанная на идее применения уточнения типов (refinement) и аффинных типов (use-at-most-once) для верификации программ, написанных на подмножестве ядра F#. На основе системы типов были реализованы средства верификации

- http://research.microsoft.com/en-us/projects/fine/



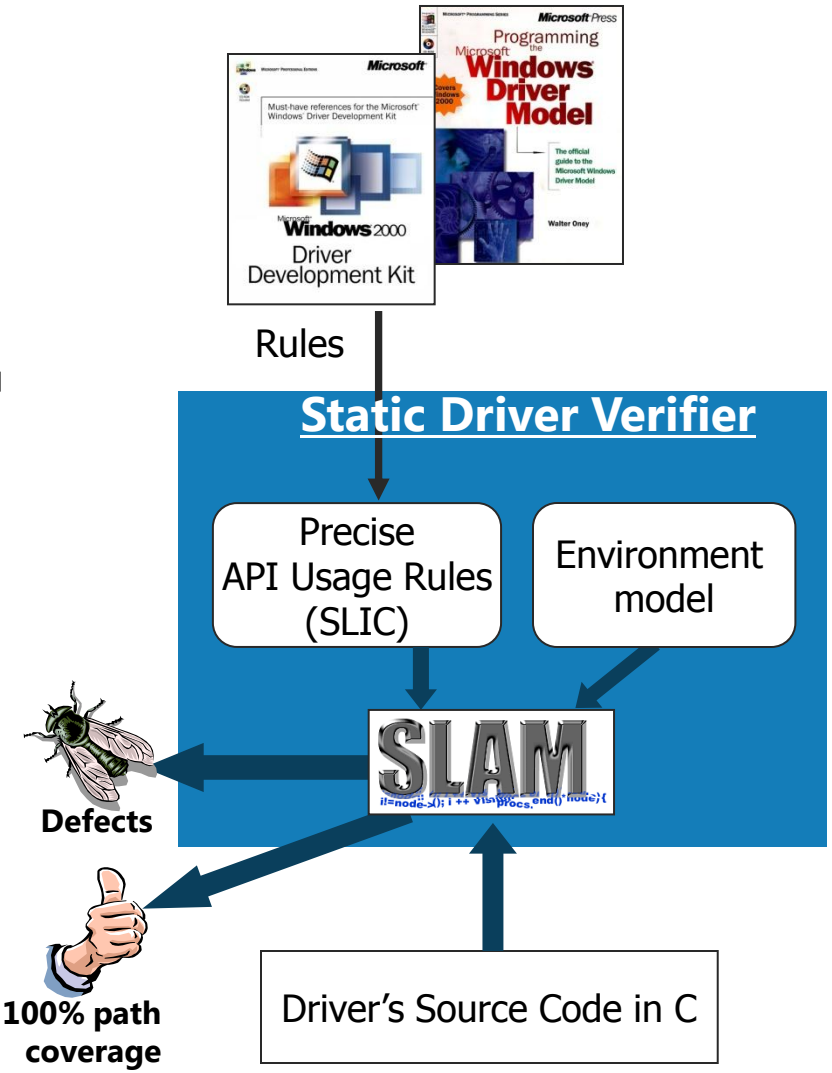

Fine - Fern – F7 - F*

- Разработанная система может быть использована для проверки того, что некоторая программная система выполняет определённые политики безопасности, в том числе политики авторизации, использующие элементы, сохраняющие состояние.
- Инструменты, основанные на этой системе типов, применялись для верификации облачных приложений, схем с нулевым разглашением, многосторонних крипто-протоколов, облачных приложений – итого, 50 000 строк кода
- http://research.microsoft.com/en-us/projects/fstar/
- http://rise4fun.com/fine



Верификация драйверов устройств

- SLAM позволяет проверить, удовлетворяет ли программная система критически важным поведенческим свойствам используемых интерфейсов, и поддерживает разработку интерфейсов и программ с гарантированным надёжным и корректным поведением
- Static Driver Verifier – инструмент набора Windows Driver Development Kit , использующий верификатор SLAM
- Static Driver Verifier Research Platform (SDVRP) - версия SDV/SLAM для академических исследований. Позволяет создавать и выполнять проверку API-специфичных правил и программ (не обязательно имеющих отношение к API драйверов устройств или самими драйверам). Содержит репозиторий программ и результатов проверки
- В основе верификации – проверка на основе модели (model checking)
- http://research.microsoft.com/en-us/projects/slam/



Информация

- Microsoft Research - http://research.microsoft.com
- Microsoft Research Connections EMEA - http://research.microsoft.com/en-us/collaboration/global/europe/default.aspx
- MSR Redmond, Research In Software Engineering Group - http://research.microsoft.com/en-us/groups/rise/default.aspx
- RiSE Online Tools - http://rise4fun.com
- MSR Cambridge, UK, Programming Principles and Tools Group - http://research.microsoft.com/en-us/groups/ppt/#research
- Rustan Leino, Microsoft Research - Program Verification: Yesterday, Today, Tomorrow - http://www.youtube.com/watch?v=5t4WntcsZZo
- How We Test Software at Microsoft - http://www.youtube.com/watch?v=PussKcpJrik&feature=related

Elena Pavlova v-elpavl@microsoft.com

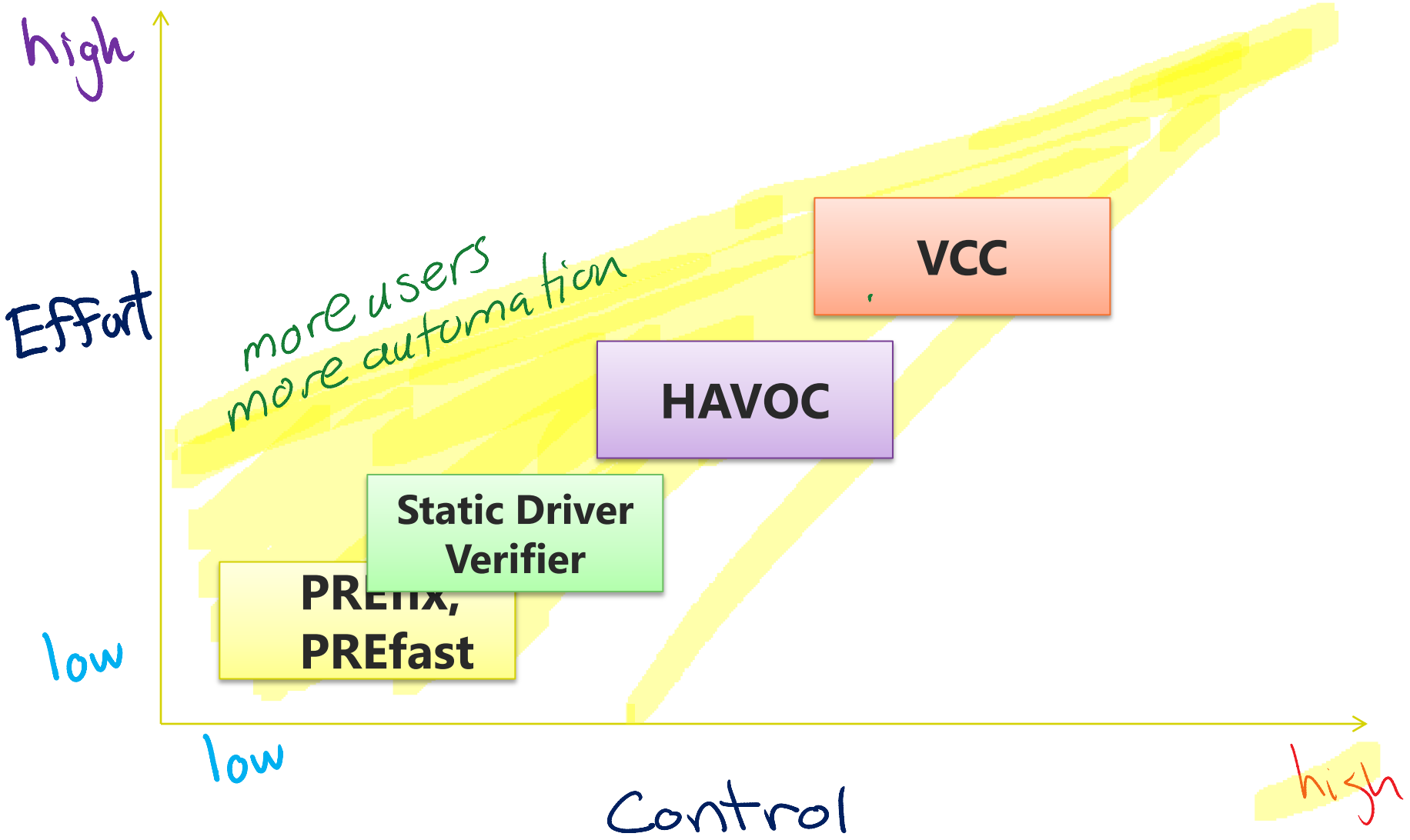
Microsoft®

Информация

- Microsoft Research - http://research.microsoft.com
- Microsoft Research Connections EMEA - http://research.microsoft.com/en-us/collaboration/global/europe/default.aspx
- MSR Redmond, Research In Software Engineering Group - http://research.microsoft.com/en-us/groups/rise/default.aspx
- RiSE Online Tools - http://rise4fun.com
- MSR Cambridge, UK, Programming Principles and Tools Group - http://research.microsoft.com/en-us/groups/ppt/#research
- Rustan Leino, Microsoft Research - Program Verification: Yesterday, Today, Tomorrow - http://www.youtube.com/watch?v=5t4WntcsZZo
- How We Test Software at Microsoft - http://www.youtube.com/watch?v=PussKcpJrik&feature=related

Elena Pavlova v-elpavl@microsoft.com

User Effort and Control



Why Another C Verifier?

	Static Driver Verifier	HAVOC	VCC
Expressiveness	+ (control-oriented)	++ (system-specific)	+++ (functional)
Precision	+ (abstract memory)	++ (precise)	++ (precise)
Scalability	+ (whole program)	++ (modular)	++ (modular)
Automation	++ (push button)	+ (inference)	-- (manual)
Contracts	--	++	++
Users	**Developers**	**Auditors**	**Verif. Experts**
Problem	**Correct API usage**	**Security audit**	**Fully correct TCB**

Specifying and enforcing XACML poli

XACML Policy

```
<Resource>
<ResourceMatch MatchId="
<ResourceAttributeDesigna
<AttributeValue DataType=
</ResourceMatch>
</Resource>
<Action>
<ActionMatch MatchId="ur
<AttributeValue DataType="
</Action>
<Subject>
<SubjectMatch MatchId="u
<AttributeValue DataType="
</SubjectMatch>
</Subject>
<Rule RuleId="1" Effect="
<Rule RuleId="2" Effect="
```

Resources: patient-records

Actions: getData, …

Subjects: Doctors, …

Rules: Grant, Deny, …

Does this code properly enforce the policy? query with right parameters?

```
public Data GetData (string subjectName, string recId) {
  Record rec = DB.GetRecord (subjectName, recId);
  List queryParams = new {subjectName, rec.datastreamId};
  if (XACML.GetAuthCtxt(subjectName).query("CanGetData", queryParams)) {
      return rec.data;
  }
  return null;
}
```

C# code that mediates access to patient records

Fine: Specify and enforce security policies

A reference monitor written in Fine

Policy specified using logical formulas in a header file

```
assume Rule_docCanRead: forall p:prin, r:record.
   (InRole<p,Doctor> && Treating<p, r.patient>) =>
    HasPerm<p, CanRead r>
assume …
```

Types mention security constraints from policy

```
val readRecord: p:prin -> cred<p> ->
                {r:record | HasPerm<p, CanRead r>} ->
                string
```

Type checker ensures authenticity, complete mediation, information hiding

The type of a function to read data from a protected record

Microsoft Research Connections in Russia

- Совместные исследовательские проекты
  - Исследования в области компьютерного зрения совместно с Московским Государственным Университетом, МГУ и MSR Cambridge Computer Vision Group - http://graphics.cs.msu.ru/ru
  - Исследования по распределённому языку авторизации знаний (Distributed Knowledge Authorization Language) совместно между Математическим институтом им. В.А. Стеклова РАН и MSR RiSE Group - http://dkal.codeplex.com
  - Исследования в области соответствия браузеров стандарту W3C (DOM API Testing) совместно между Институтом Системного Программирования РАН (ИСП РАН) и MSR RiSE Group – http://domapi.codeplex.com
  - Исследования в области вычислительной математики и биологии (Computational Mathematics and Biology Research) в ИТМО
- Поддержка научно-исследовательских мероприятий
- Стажировки (Internships) - http://research.microsoft.com/en-us/jobs/intern/
- Поддержка представителей MSR в России