Orchestrating Decision Engines

CP 2011, Perugia, Italy

Leonardo de Moura (Microsoft Research) and
Grant Passmore (University of Cambridge)

Satisfiability Modulo Theories (SMT)

A Satisfiability Checker

with built-in support for useful theories

Satisfiability Modulo Theories (SMT)

$$b + 2 = c \;\; and \;\; f(read(write(a,b,3), c-2) \neq f(c-b+1)$$

Microsoft Research

Satisfiability Modulo Theories (SMT)

$$\boxed{b + 2 = c} \text{ and } f(read(write(a,b,\boxed{3}), \boxed{c-2}) \neq f(\boxed{c-b+1})$$

Arithmetic

Satisfiability Modulo Theories (SMT)

$b + 2 = c$ *and* $f(\boxed{read}(\boxed{write}(a,b,3), c\text{-}2) \neq f(c\text{-}b+1)$

Array Theory

Microsoft® Research

Satisfiability Modulo Theories (SMT)

$$b + 2 = c \text{ and } \boxed{f}(read(write(a,b,3), c\text{-}2) \neq \boxed{f}(c\text{-}b+1)$$

Uninterpreted
Functions

Microsoft Research

SMT Solvers & LIB & COMP

Solvers:

AProve, Barcelogic, Boolector, CVC3, CVC4, MathSAT5, OpenSMT, SMTInterpol, SOLONAR, STP2, veriT, Yices, Z3

SMT-LIB: library of benchmarks (> 100k problems)

http://www.smtlib.org

SMT-COMP: annual competition

http://www.smtcomp.org

Applications

- Test case generation
- Verifying Compilers
- Predicate Abstraction
- Invariant Generation
- Type Checking
- Model Based Testing
- Scheduling & Planning
- …

Microsoft Research

Some Applications @ Microsoft

The Spec# Programming System

HAVOC

ForμLa

Hyper-V Microsoft Virtualization

Terminator T-2

VCC

SLAM

Yogi

NModel

Vigilante

SpecExplorer

Pex

F7

SAGE

Microsoft Research

Application Scenarios

"Big" and hard formulas

Thousands of "small" and easy formulas

Short timeout (< 5secs)

Microsoft Research

Application Scenarios

The Spec# Programming System

"Big" and hard formulas

HAVOC **VCC**

Thousands of "small" and easy formulas

Yogi

SLAM

Short timeout (< 5secs) **SAGE**

Microsoft Research

Verification/Analysis Tool: “Template”

Problem

Verification/Analysis Tool

Logical Formula

Theorem Prover/ Satisfiability Checker

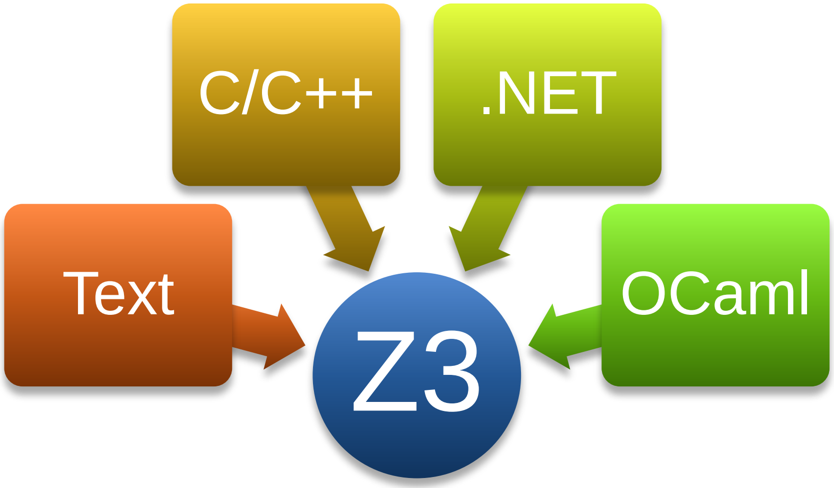
Satisfiable
(Counter-example)

Unsatisfiable

Microsoft Research

SMT@Microsoft: Solver

- Z3 is a solver developed at Microsoft Research.
- Development/Research driven by internal customers.
- Free for non-commercial use.
- Interfaces:



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

Microsoft Research

http://rise4fun.com/Z3

Z3 @ RiSE4fun - Efficient T...

rise4fun.com/z3

RiSE4fun

gave 146,333 answers!

Click on a tool to load a sample then ask!

agl bek boogie code contracts concurrent revisions dafny dkal esm f* formula heapdbg poirot pex rex slayer spec# vcc z3

```
(declare-fun x () Int)
(declare-fun y () Int)
(declare-fun z () Int)
(assert (>= (* 2 x) (+ y z)))
(declare-fun f (Int) Int)
(declare-fun g (Int Int) Int)
(assert (< (f x) (g x x)))
(assert (> (f y) (g x x)))
(check-sat)
(get-model)
(push)
(assert (= x y))
(check-sat)
(pop)
(exit)
```

ask z3 *Is this formula satisfiable? Click 'ask z3'!* tutorial home video

Tweet Like 164

samples tutorials projects live permalink developer about

for Windows Phone 7

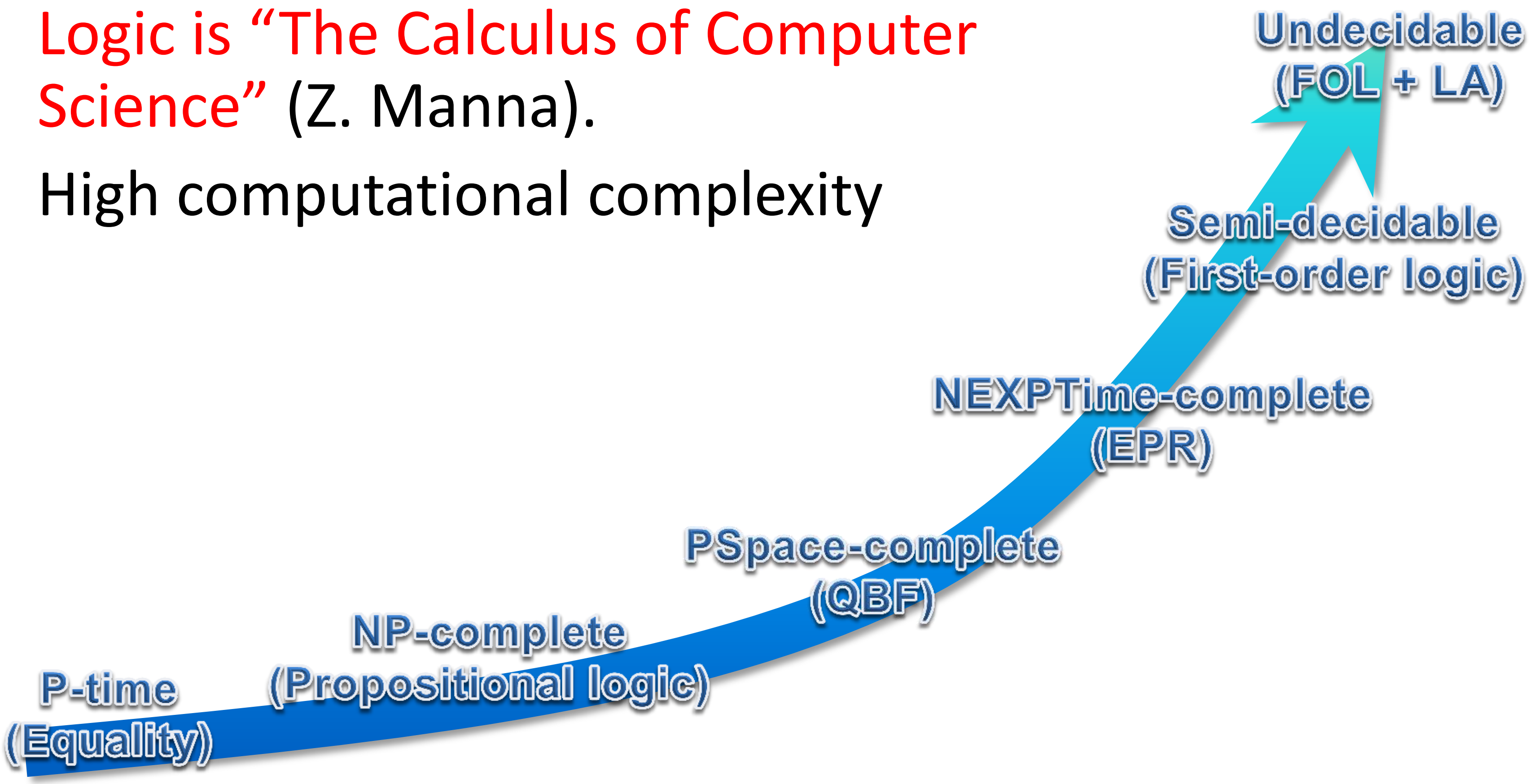
Microsoft Research

Symbolic Reasoning

Verification/Analysis tools need some form of **Symbolic Reasoning**

Symbolic Reasoning

- Logic is "The Calculus of Computer Science" (Z. Manna).
- High computational complexity

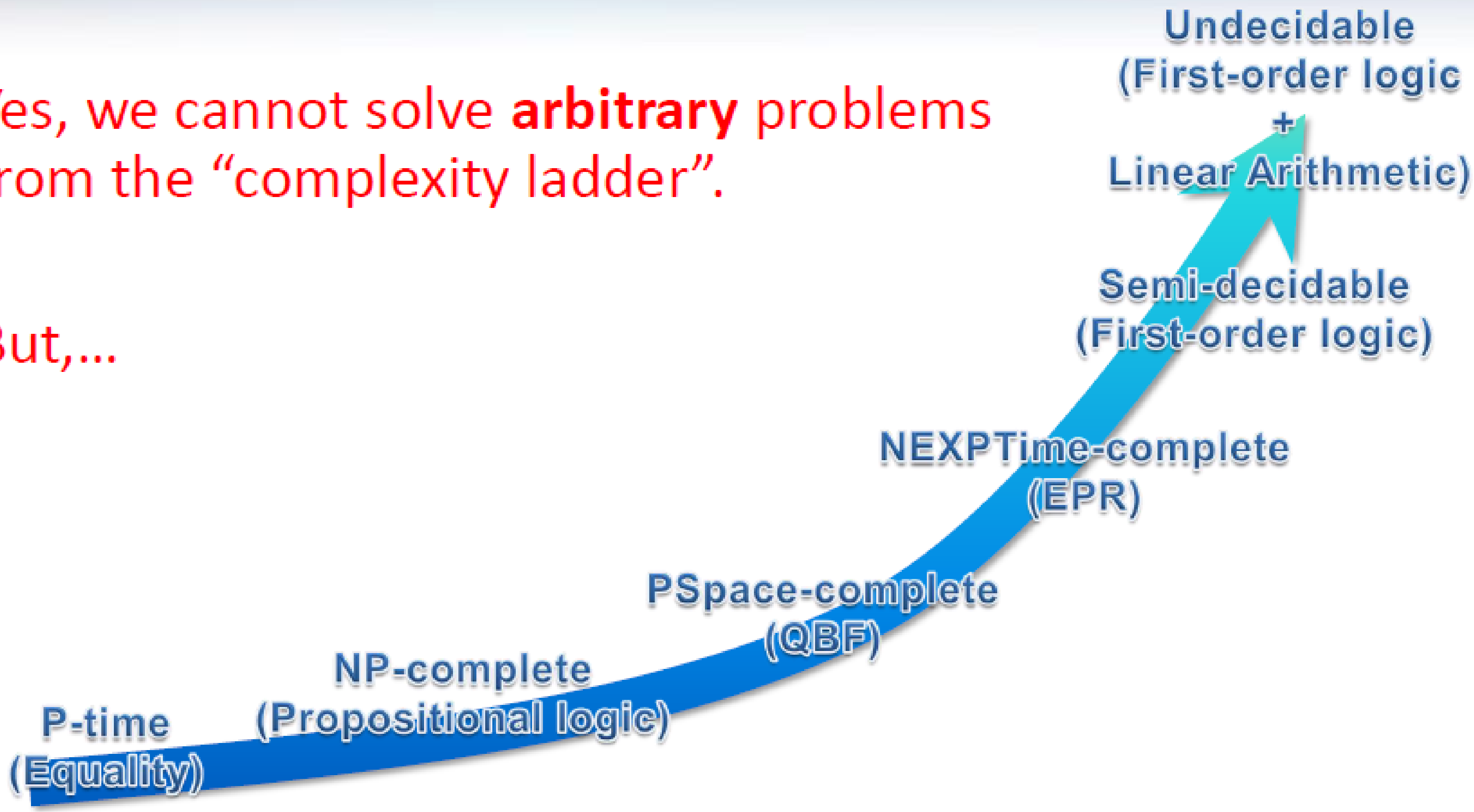


Microsoft Research

Symbolic Reasoning

Yes, we cannot solve **arbitrary** problems from the "complexity ladder".

But,…



Microsoft Research

Symbolic Reasoning

We can try to solve the problems we find in

real applications

Main challenges

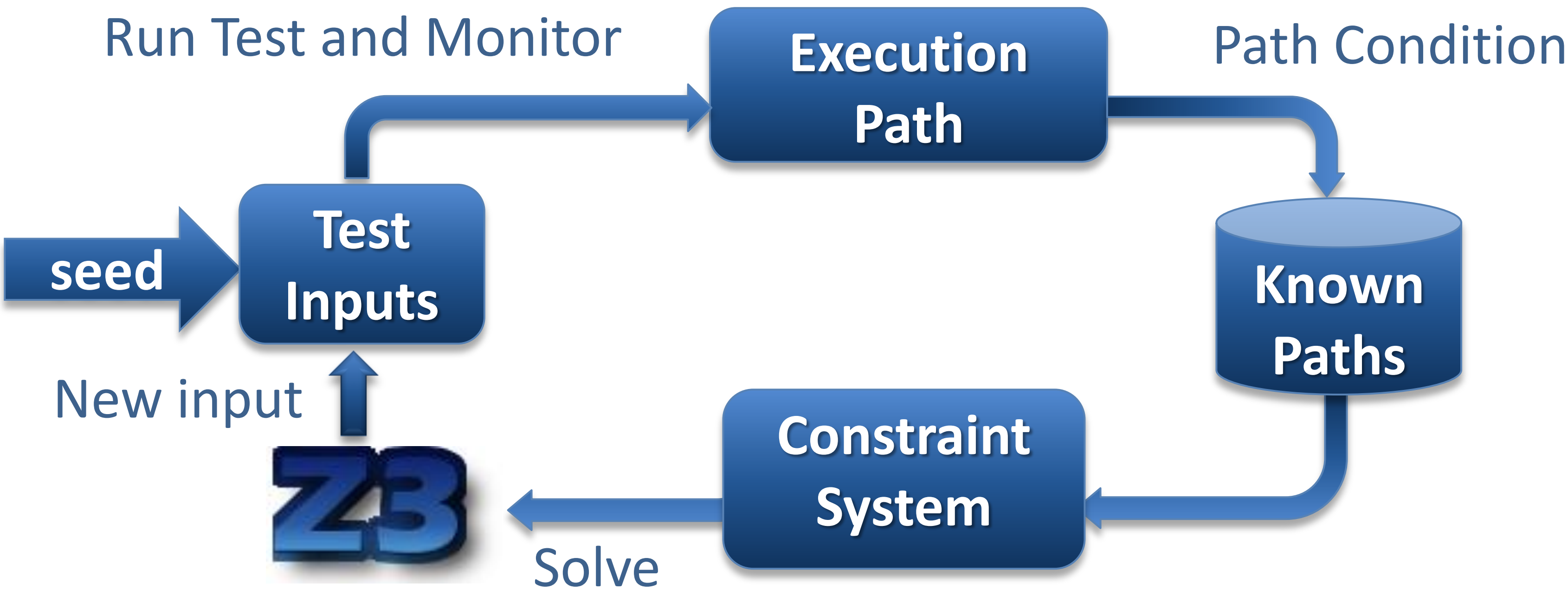
- Scalability (huge formulas)
- Complexity
- Undecidability
- Quantified formulas

Microsoft Research

SMT@MS: Applications

A Sample

Directed Automated Random Testing (DART)



Microsoft Research

Test case generation

```
unsigned GCD(x, y) {
  requires(y > 0);
  while (true) {
    unsigned m = x % y;
    if (m == 0) return y;
    x = y;
    y = m;
  }
}
```

SSA

$(y_0 > 0)$ and

$(m_0 = x_0 \% y_0)$ and

not $(m_0 = 0)$ and

$(x_1 = y_0)$ and

$(y_1 = m_0)$ and

$(m_1 = x_1 \% y_1)$ and

$(m_1 = 0)$

model

$x_0 = 2$

$y_0 = 4$

$m_0 = 2$

$x_1 = 4$

$y_1 = 2$

$m_1 = 0$

We want a trace where the loop is executed twice.

Assignment

Microsoft Research

White box testing in practice

How to test this code?

(Real code from .NET base class libraries.)

```
[SecurityPermissionAttribute(SecurityAction.LinkDemand, Flags=SecurityPermissionFlag.SerializationFormatter)]
public ResourceReader(Stream stream)
{
    if (stream==null)
        throw new ArgumentNullException("stream");
    if (!stream.CanRead)
        throw new ArgumentException(Environment.GetResourceString("Argument_StreamNotReadable"));

    _resCache = new Dictionary<String, ResourceLocator>(FastResourceComparer.Default);
    _store = new BinaryReader(stream, Encoding.UTF8);
    // We have a faster code path for reading resource files from an assembly.
    _ums = stream as UnmanagedMemoryStream;

    BCLDebug.Log("RESMGRFILEFORMAT", "ResourceReader .ctor(Stream).  UnmanagedMemoryStream: "+(_ums!=null));
    ReadResources();
}
```

White box testing in practice

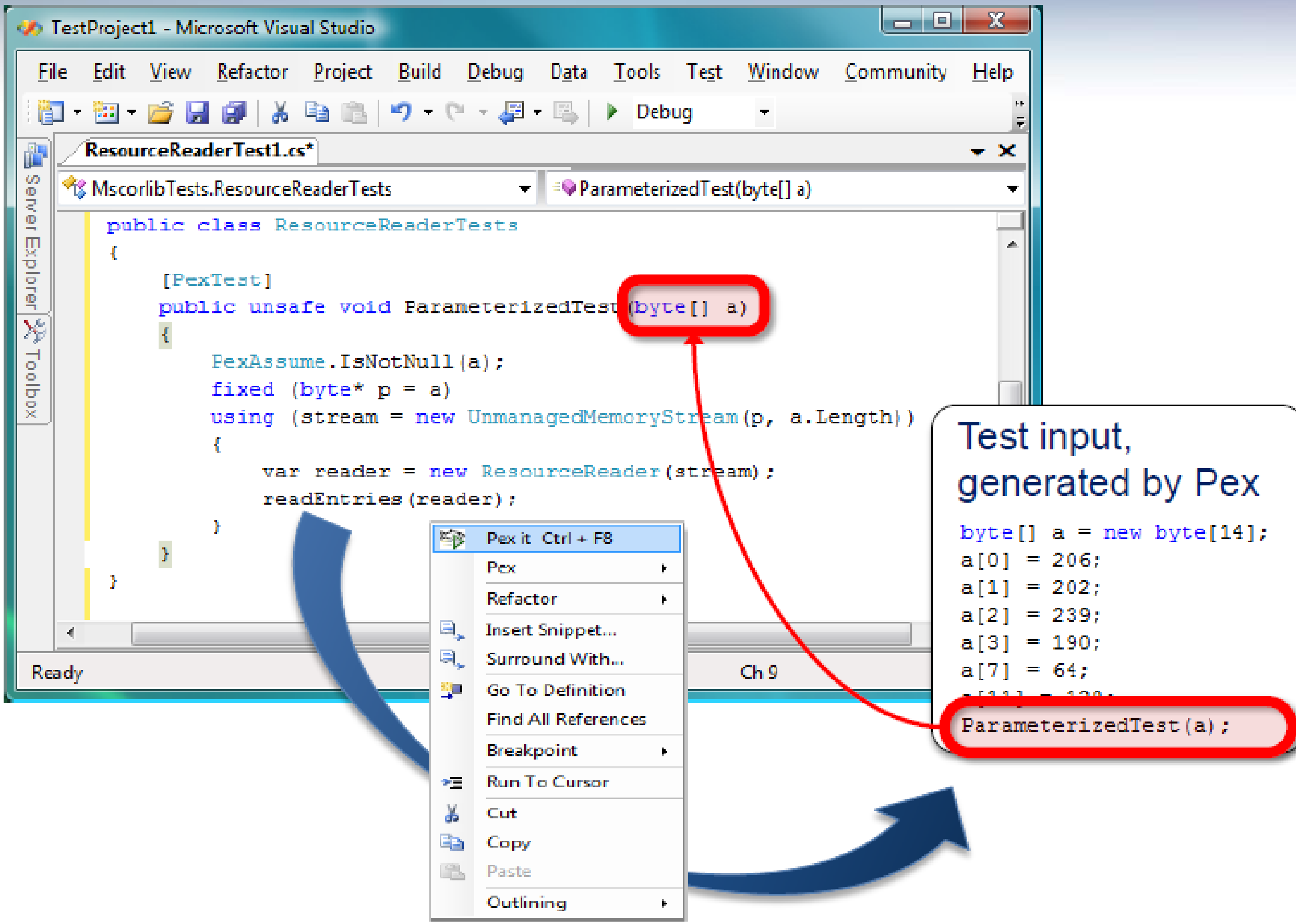
```
        // Reads in the header information for a .resources file.  Verifies some
        // of the assumptions about this resource set, and builds the class table
        // for the default resource file format.
        private void ReadResources() {
            BCLDebug.Assert(_store != null, "ResourceReader is closed!");
            BinaryFormatter bf = new BinaryFormatter(null, new StreamingContext(StreamingContextStates.File |
#if !FEATURE_PAL
            _typeLimitingBinder = new TypeLimitingDeserializationBinder();
            bf.Binder = _typeLimitingBinder;
#endif
            _objFormatter = bf;
            try {
                // Read ResourceManager header
                // Check for magic number
                int magicNum = _store.ReadInt32();
                if (magicNum != ResourceManager.MagicNumber)
                    throw new ArgumentException(Environment.GetResourceString("Resources_StreamNotValid"));
                // Assuming this is ResourceManager header V1 or greater, hopefully
                // after the version number there is a number of bytes to skip
                // to bypass the rest of the ResMgr header.
                int resMgrHeaderVersion = _store.ReadInt32();
                if (resMgrHeaderVersion > 1) {
                    int numBytesToSkip = _store.ReadInt32();
                    BCLDebug.Log("RESMGRFILEFORMAT", LogLevel.Status, "ReadResources: Unexpected ResMgr header
                    BCLDebug.Assert(numBytesToSkip >= 0, "numBytesToSkip in ResMgr header should be positive!"
                    _store.BaseStream.Seek(numBytesToSkip, SeekOrigin.Current);
                } else {
                    BCLDebug.Log("RESMGRFILEFORMAT", "ReadResources: Parsing ResMgr header v1.");
                    SkipInt32();    // We don't care about numBytesToSkip.
                    // Read in type name for a suitable ResourceReader
```

White box testing in practice

```
        // Reads in the header information for a .resources file.  Verifies some
        // of the assumptions about this resource set, and builds the class table
        // for the default resource file format.
        private void ReadResources() {
            BCLDebug.Assert(_store != null, "ResourceReader is closed!");
            BinaryFormatter bf = new BinaryFormatter(null, new StreamingContext(StreamingContextStates.File |
#if !FEATURE_PAL
            _typeLimitingBinder = new TypeLimitingDeserializationBinder();
            bf.Binder = _typeLimitingBinder;
#endif
            _objFormatter = bf;
            try {
                // Read ResourceManager header
                // Check for magic number
                int magicNum = _store.ReadInt32();
                if
                //
                //
                //
                int
                if

                } e
                    // Read in type name for a suitable ResourceReader
```

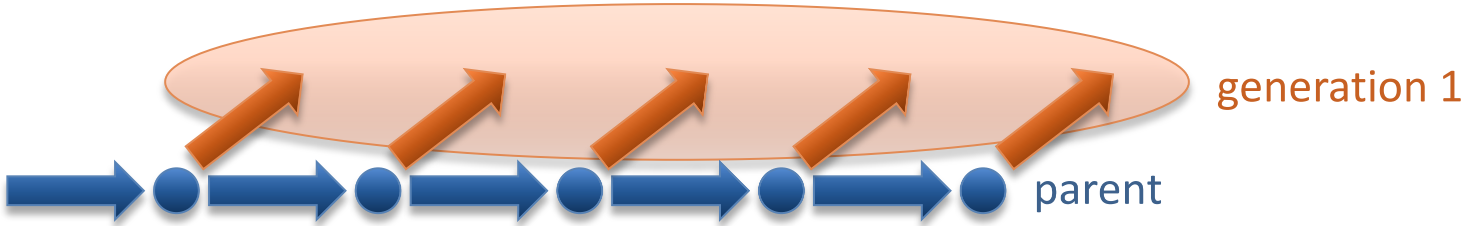
```
public virtual int ReadInt32() {
    if (m_isMemoryStream) {
        // read directly from MemoryStream buffer
        MemoryStream mStream = m_stream as MemoryStream;
        BCLDebug.Assert(mStream != null, "m_stream as MemoryStream != null");

        return mStream.InternalReadInt32();
    }
    else
    {
        FillBuffer(4);
        return (int)(m_buffer[0] | m_buffer[1] << 8 | m_buffer[2] << 16 | m_buffer[3] << 24);
    }
}
```

Pex – Test Input Generation



SAGE

- Apply DART to large applications (not units).
- Start with well-formed input (not random).
- Combine with generational search (not DFS).
  - Negate 1-by-1 each constraint in a path constraint.
  - Generate many children for each parent run.



Microsoft Research

Zero to Crash in 10 Generations

- Starting with 100 zero bytes …
- SAGE generates a crashing test for Media1 parser

```
00000000h: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ; ................
00000010h: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ; ................
00000020h: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ; ................
00000030h: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ; ................
00000040h: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ; ................
00000050h: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ; ................
00000060h: 00 00 00 00                                     ; ....
```

Generation 0 – seed file

Zero to Crash in 10 Generations

- Starting with 100 zero bytes …
- SAGE generates a crashing test for Media1 parser

```
00000000h: 52 49 46 46 00 00 00 00 00 00 00 00 00 00 00 00 ; RIFF............
00000010h: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ; ................
00000020h: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ; ................
00000030h: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ; ................
00000040h: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ; ................
00000050h: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ; ................
00000060h: 00 00 00 00                                     ; ....
```

Generation 1

Zero to Crash in 10 Generations

- Starting with 100 zero bytes …
- SAGE generates a crashing test for Media1 parser

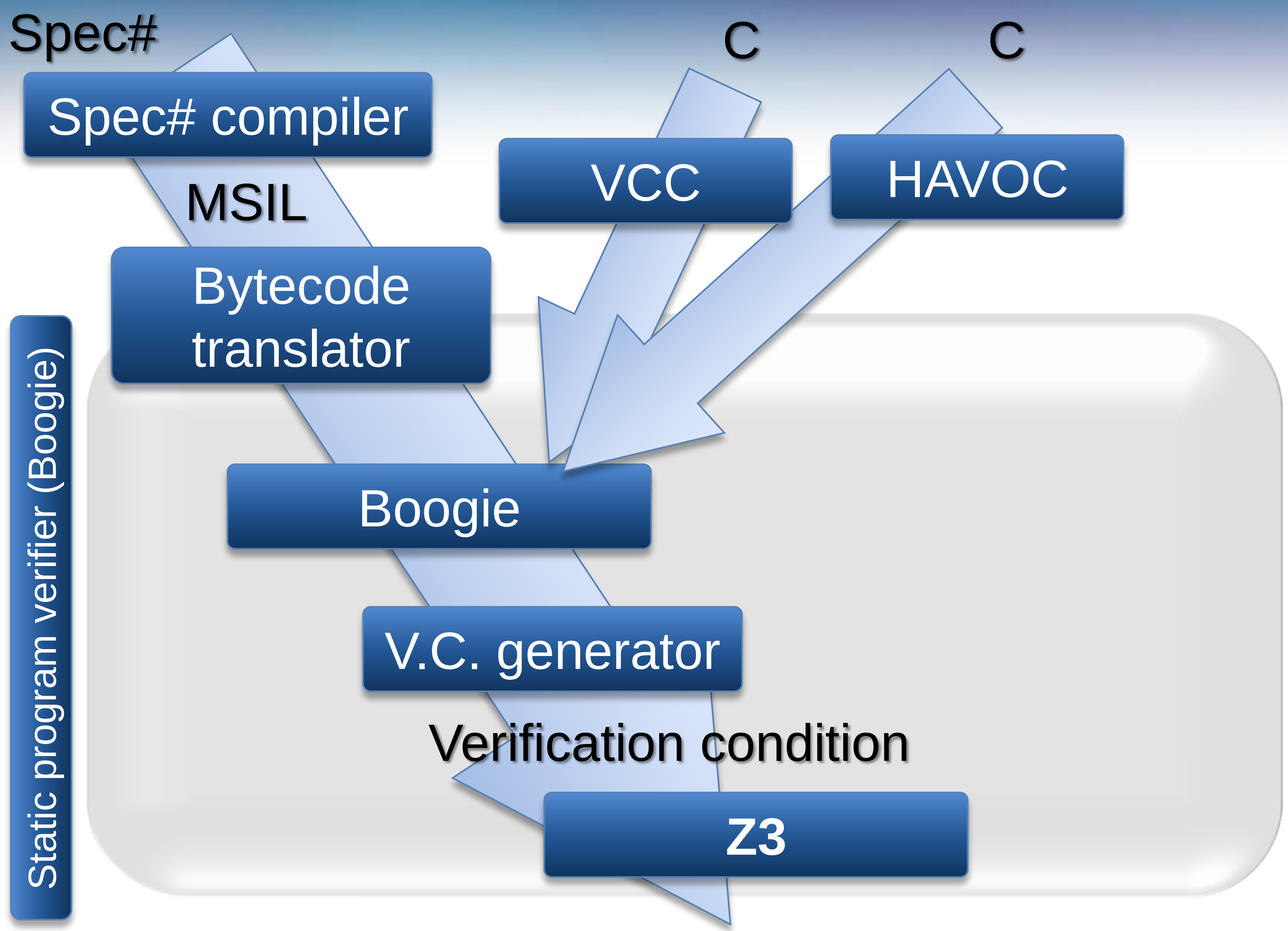
```
00000000h: 52 49 46 46 3D 00 00 00 ** ** ** 20 00 00 00 00 ; RIFF=...*** ....
00000010h: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ; ................
00000020h: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ; ................
00000030h: 00 00 00 00 73 74 72 68 00 00 00 00 76 69 64 73 ; ....strh....vids
00000040h: 00 00 00 00 73 74 72 66 B2 75 76 3A 28 00 00 00 ; ....strf²uv:(...
00000050h: 00 00 00 00 00 00 00 00 00 00 00 00 01 00 00 00 ; ................
00000060h: 00 00 00 00                                     ; ....
```

Generation 10 – CRASH

SAGE $\leftrightarrow$ Z3

- Formulas are usually big conjunctions.
- SAGE uses only the bitvector and array theories.
- Pre-processing step has a huge performance impact.
  - Eliminate variables.
  - Simplify formulas.
- Early unsat detection.

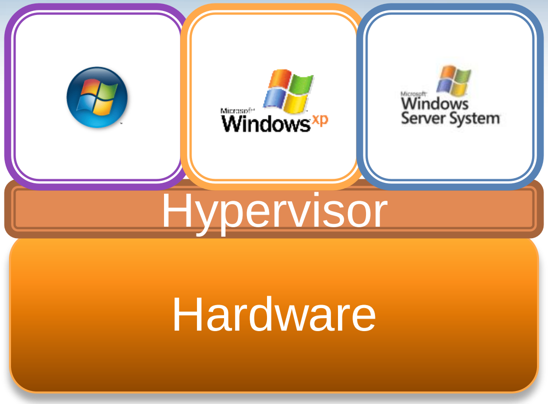
Verification architecture



A Verifying C Compiler

- VCC translates an *annotated C program* into a *Boogie PL* program.
- A C-ish memory model
  - Abstract heaps
  - Bit-level precision
- Microsoft Hypervisor: verification grand challenge.

Hypervisor: A Manhattan Project



- **Meta OS**: small layer of software between hardware and OS
- **Mini**: 60K lines of non-trivial concurrent systems C code
- **Critical:** must provide functional resource abstraction
- **Trusted**: a verification grand challenge

Hypervisor: Some Statistics

- VCs have several Mb
- Thousands of non ground clauses
- Developers are willing to wait at most 5 min per VC

Other Microsoft clients

- Model programs (M. Veanes – MSRR)
- Termination (B. Cook – MSRC)
- Security protocols (A. Gordon and C. Fournet - MSRC)
- Business Application Modeling (E. Jackson - MSRR)
- Cryptography (R. Venki – MSRR)
- Verifying Garbage Collectors (C. Hawblitzel – MSRR)
- Model Based Testing (L. Bruck – SQL)
- Semantic type checking for D models (G. Bierman – MSRC)
- More coming soon…

Microsoft Research

http://rise4fun.com

Pex, Spec#, VCC and many other tools are available online.

Microsoft Research

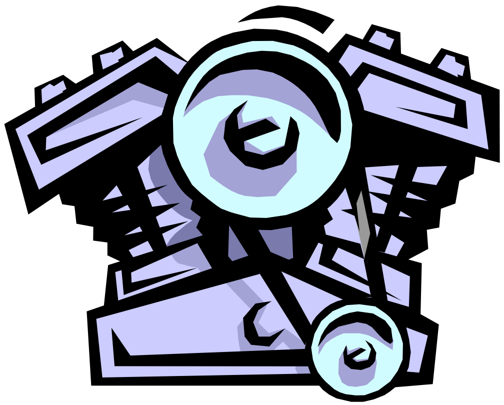
Orchestrating Decision Engines

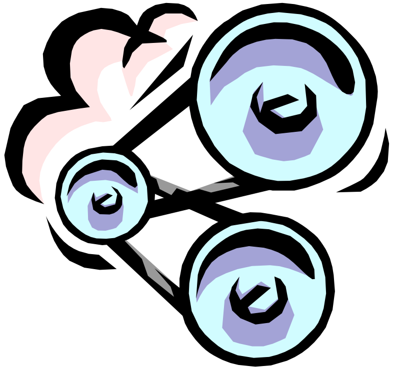
Combining Engines

Current SMT solvers provide

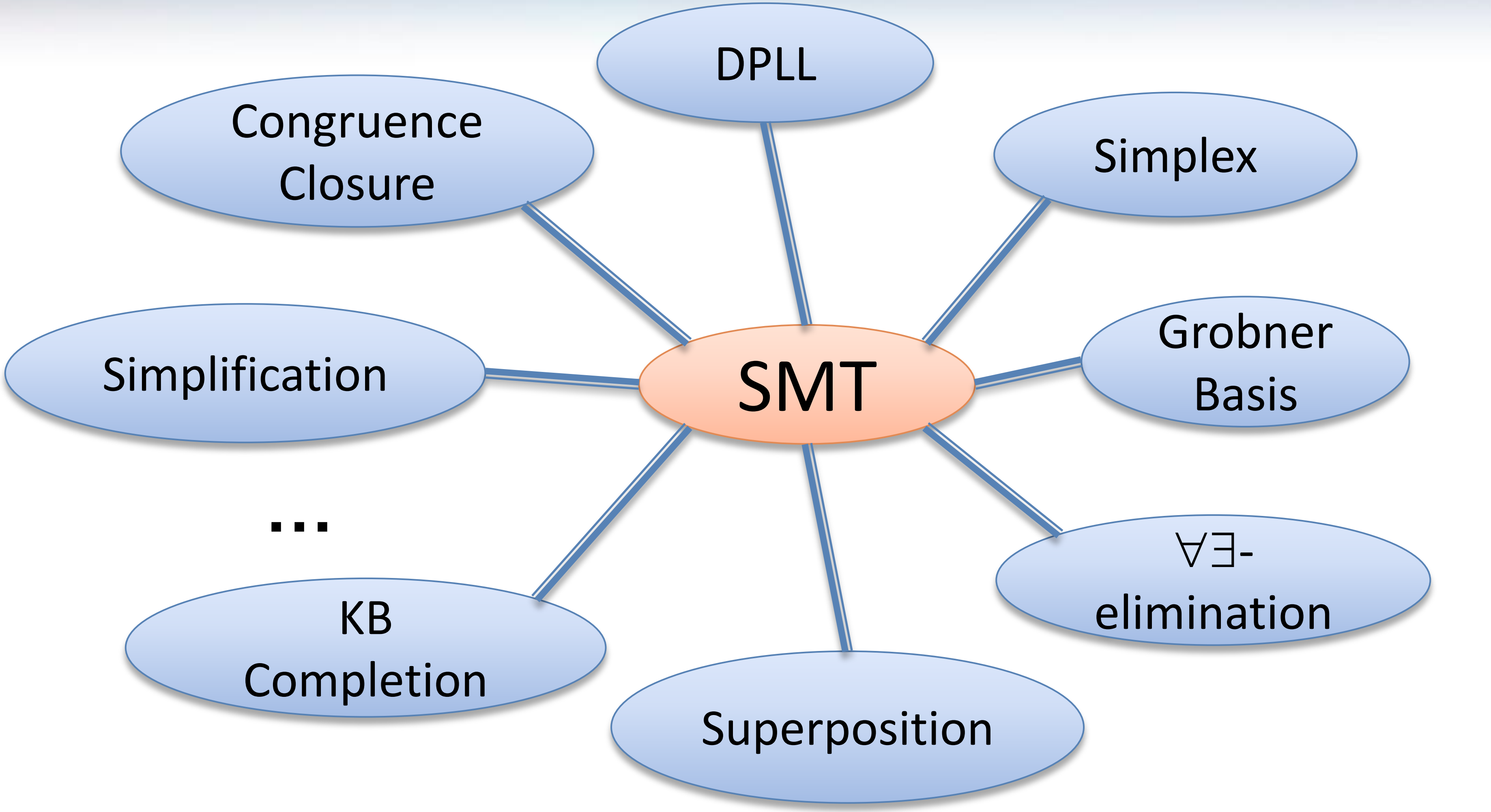
a combination

of different engines

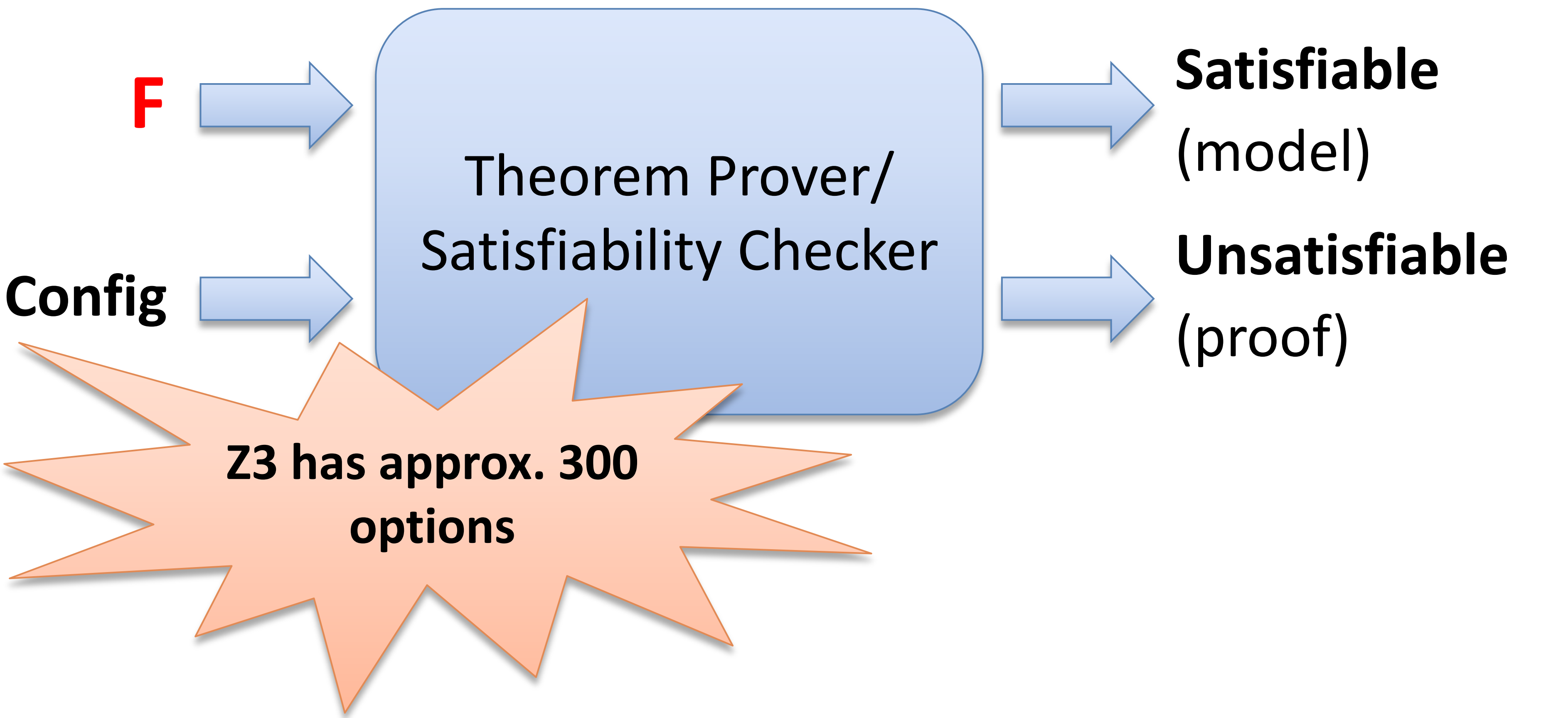




Combining Engines



Configuring SAT/SMT Solvers: “state-of-the-art”



Opening the "Black Box"

Actual feedback provided by Z3 users:

"Could you send me your CNF converter?"

"I want to implement my own search strategy."

"I want to include these rewriting rules in Z3."

"I want to apply a substitution to term *t*."

"I want to compute the set of implied equalities."

The Strategy Challenge

To build theoretical and practical tools allowing users to exert strategic control over core heuristic aspects of high performance SMT solvers.

What is a strategy?

Theorem proving as an exercise of combinatorial search

Strategies are adaptations of general search mechanisms which reduce the search space by tailoring its exploration to a particular class of formulas.

The Need for “Strategies”

Different Strategies for Different Domains.

The Need for “Strategies”

Different Strategies for Different Domains.

From timeout to 0.05 secs…

Example in Quantified Bit-Vector Logic (QBVF)

Join work with C. Wintersteiger and Y. Hamadi
FMCAD 2010

QBVF = Quantifiers + Bit-vectors + uninterpreted functions

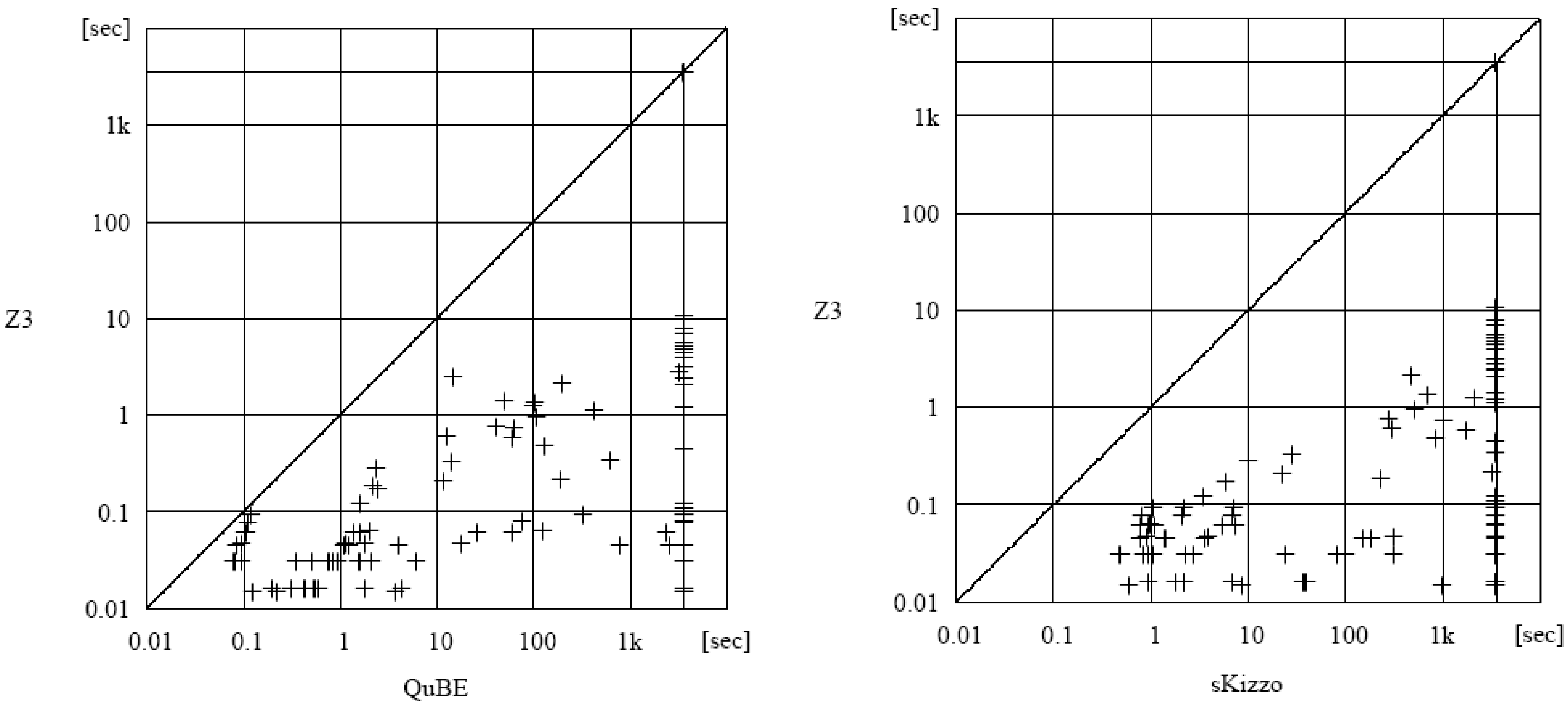
Hardware Fixpoint Checks.

Given: $I[x]$ and $T[x, x']$

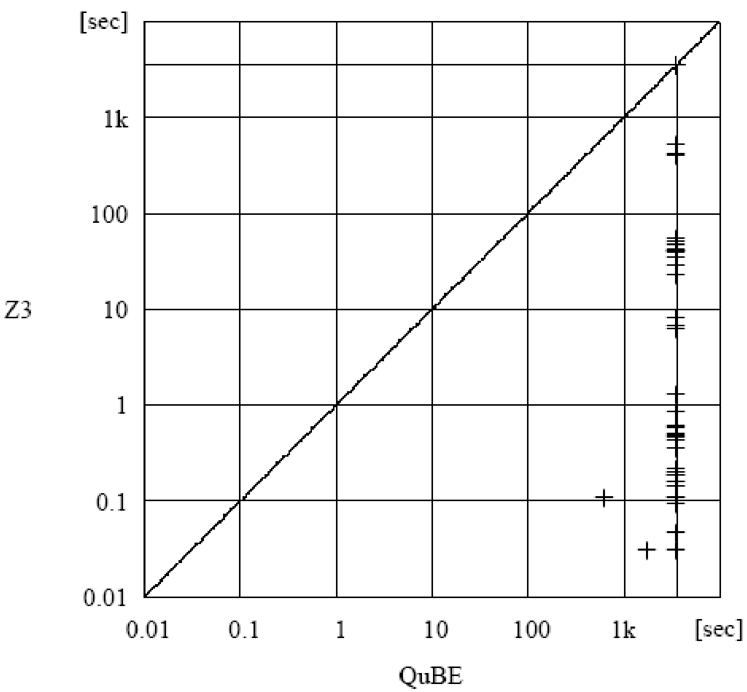
$$\forall x, x' \ . \ I[x] \wedge T^k[x, x'] \rightarrow \exists y, y' \ . I[y] \wedge T^{k-1}[y, y']$$

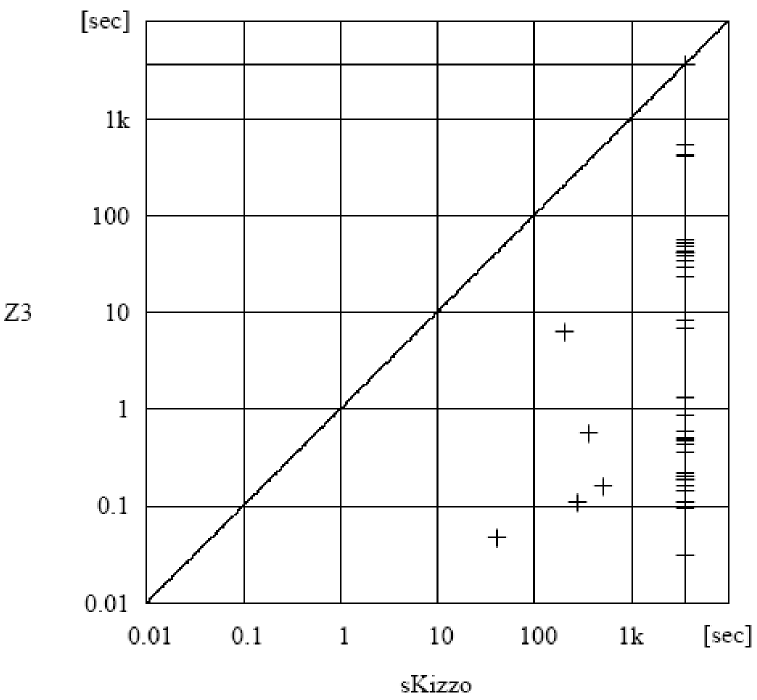
Ranking function synthesis.

Hardware Fixpoint Checks



Ranking Function Synthesis





Why is Z3 so fast in these benchmarks?

Z3 is using different engines:

rewriting, simplification, model checking, SAT, …

Z3 is using a customized **strategy**.

We could do it because

we have access to the source code.

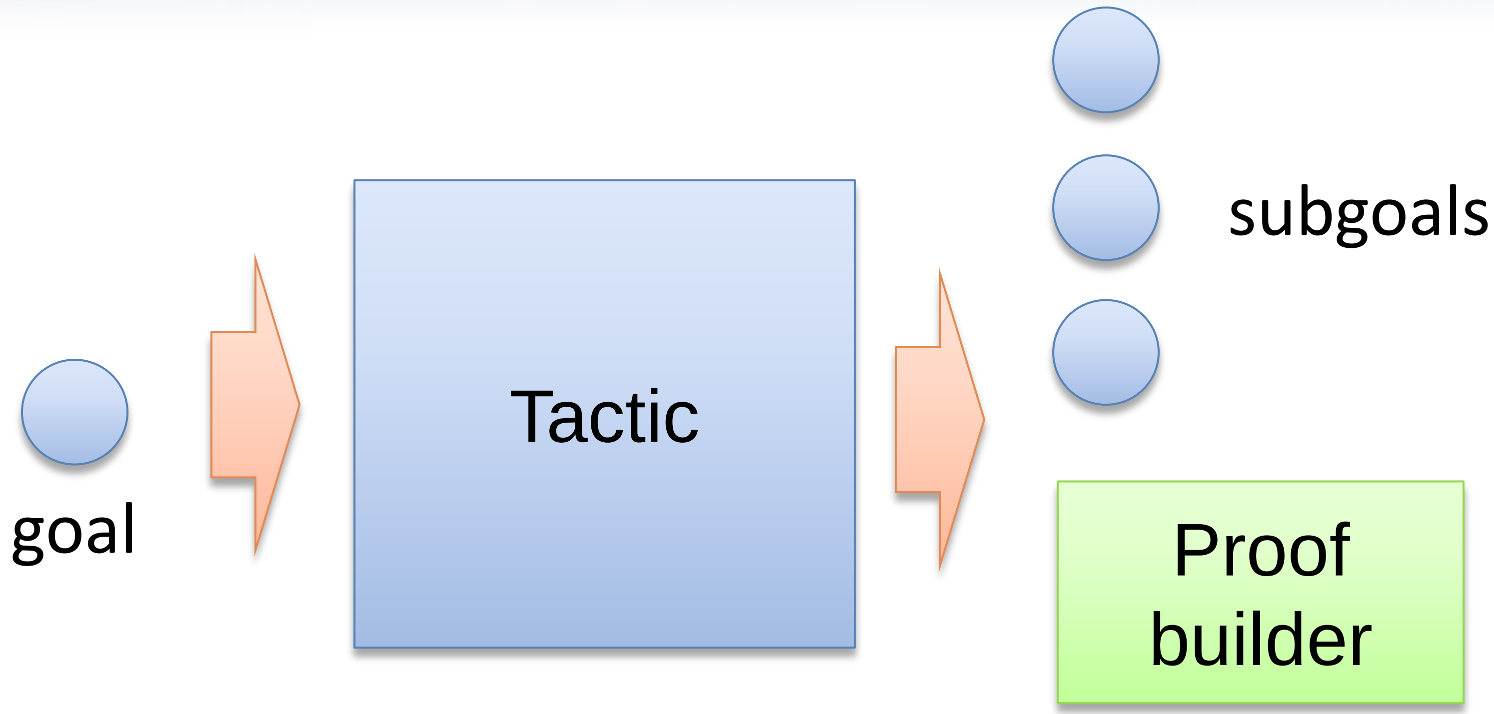
The "Message"

SMT solvers are collections of little engines.

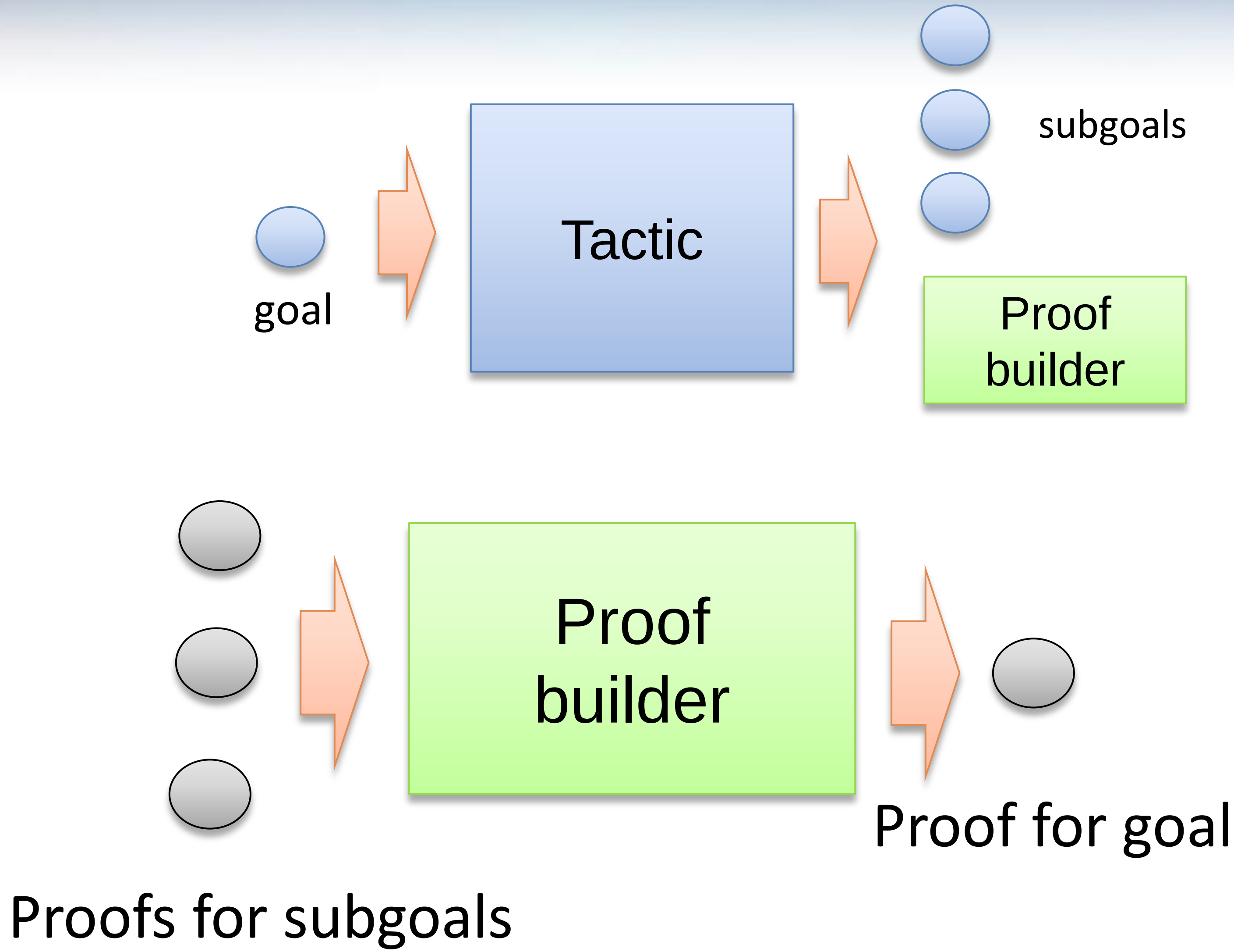
They should provide access to these engines.

Users should be able to define their own strategies.

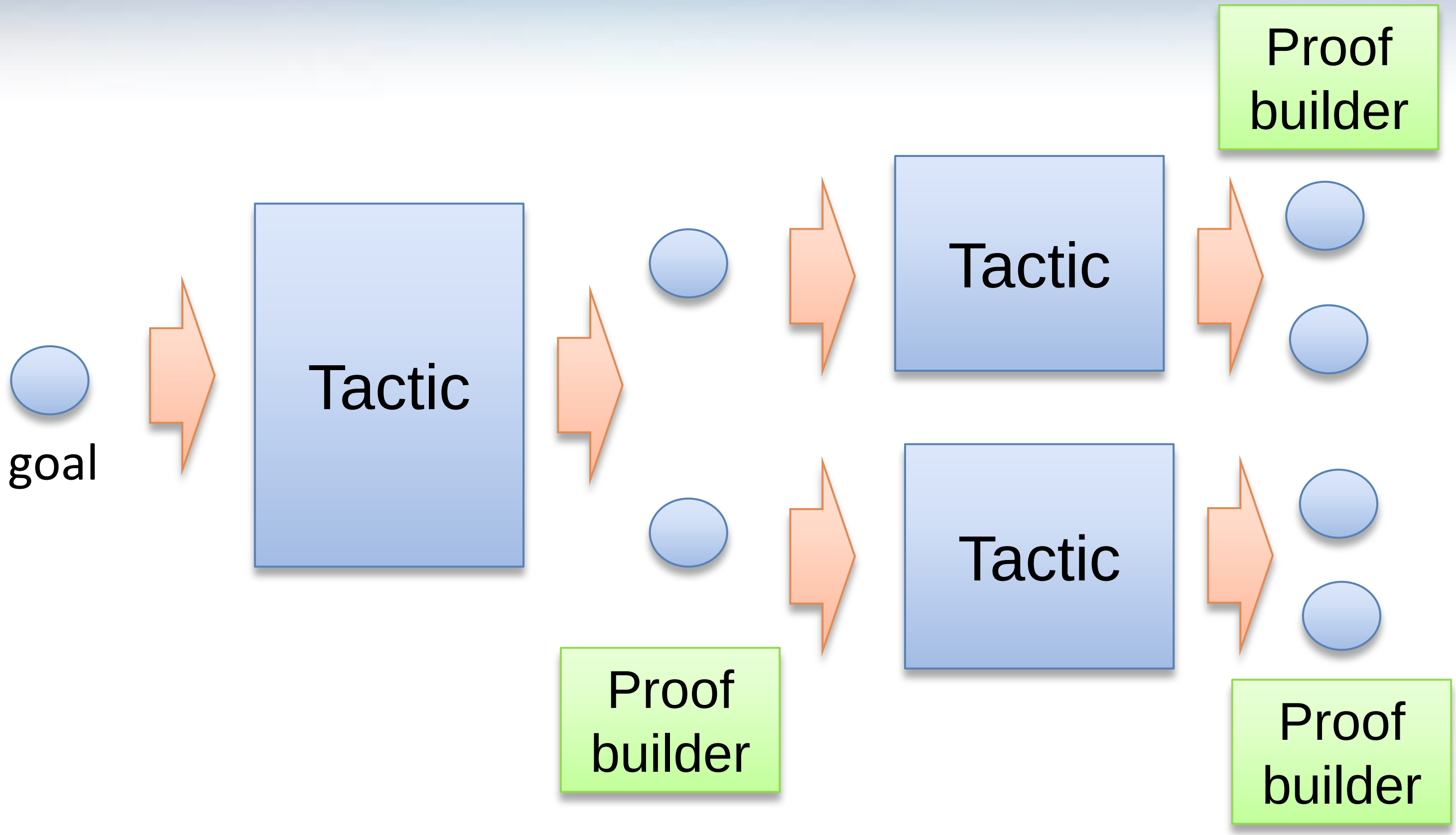
Main inspiration: LCF-approach



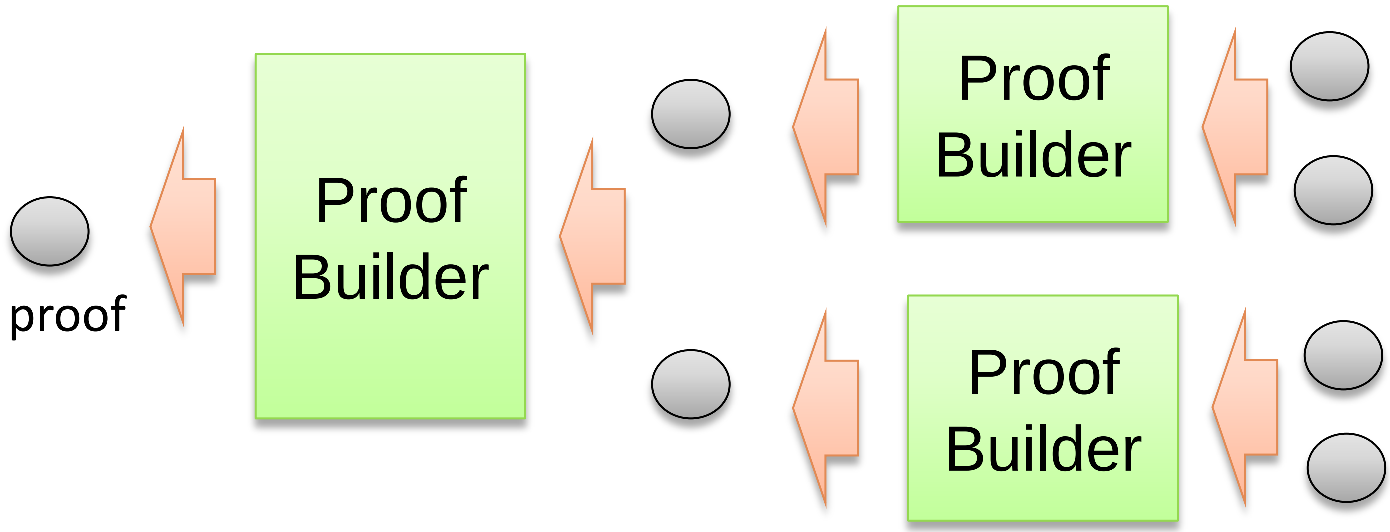
Main inspiration: LCF-approach



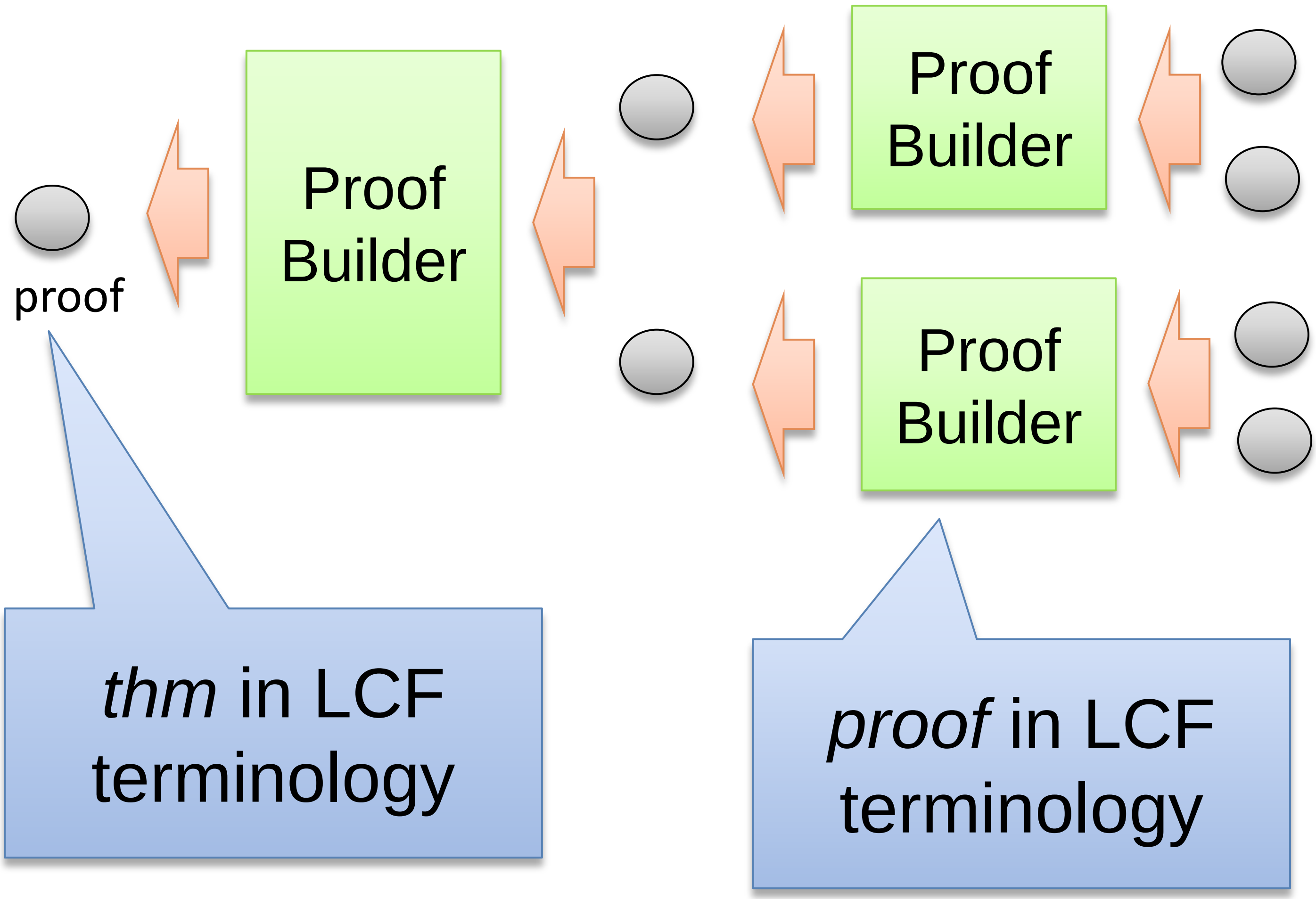
Main inspiration: LCF-approach



Main inspiration: LCF-approach



Main inspiration: LCF-approach



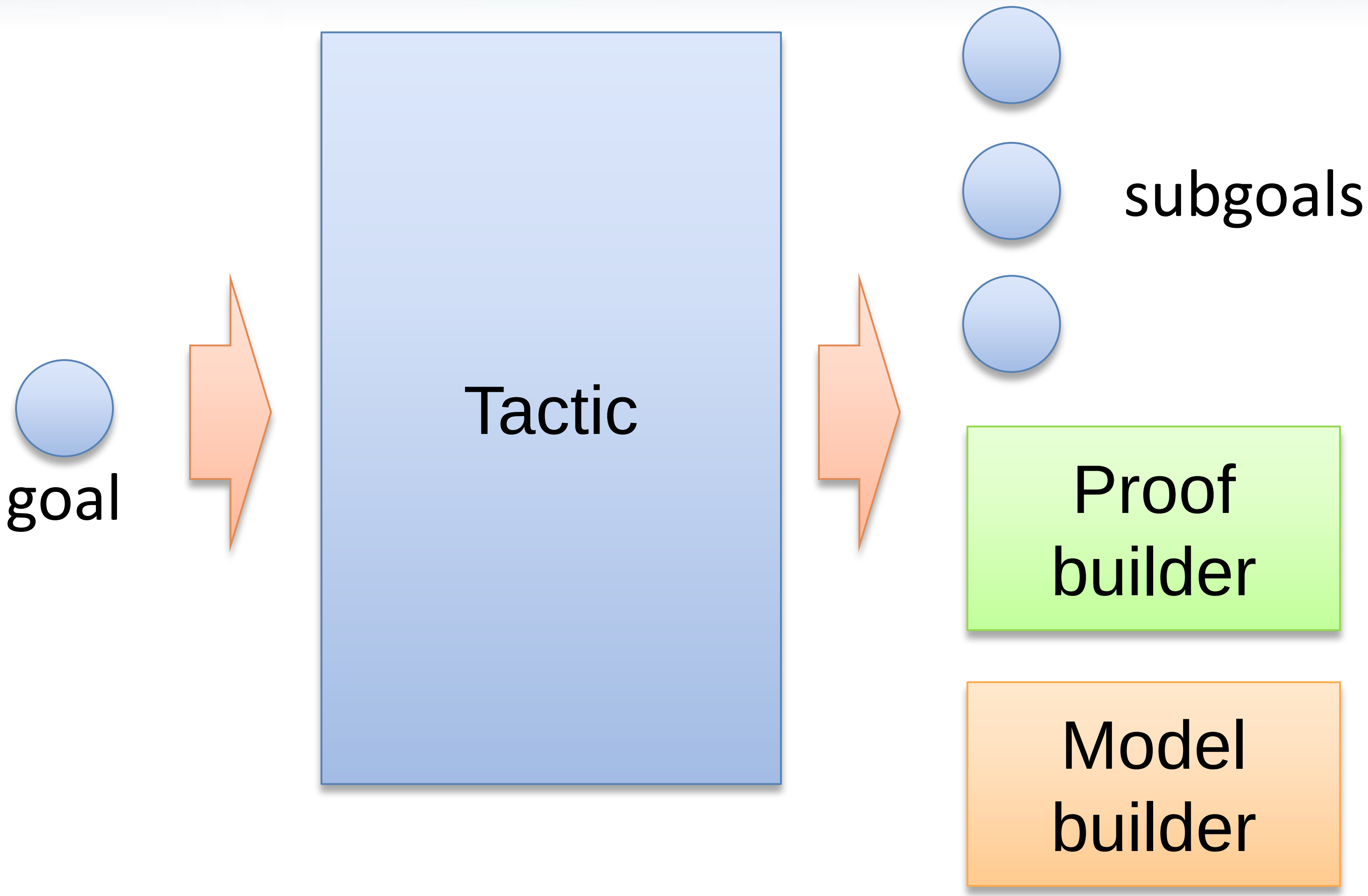
Tacticals aka Combinators

then(Tactic , Tactic) = Tactic

orelse(Tactic , Tactic) = Tactic

repeat(Tactic) = Tactic

SMT Tactic



SMT Tactic

$$
\begin{array}{lcl}
goal & = & formula\ sequence \times attribute\ sequence \\
\\
proofconv & = & proof\ sequence \rightarrow proof \\
modelconv & = & model \times nat \rightarrow model \\
trt & = & \textsf{sat}\ model \\
 & | & \textsf{unsat}\ proof \\
 & | & \textsf{unknown}\ goal\ sequence \times modelconv \times proofconv \\
 & | & \textsf{fail} \\
tactic & = & goal \rightarrow trt
\end{array}
$$

SMT Tactic

$$
\begin{aligned}
goal &= formula\ sequence \times attribute\ sequence \\[1em]
proofconv &= proof\ sequence \rightarrow proof \\
modelconv &= model \times nat \rightarrow model \\
trt &= \mathsf{sat}\ model \\
&\mid\ \ \mathsf{unsat}\ proof \\
&\mid\ \ \mathsf{unknown}\ goal\ sequence \times modelconv \times proofconv \\
&\mid\ \ \mathsf{fail} \\
tactic &= goal \rightarrow trt
\end{aligned}
$$

end-game tactics:
never return unknown(sb, mc, pc)

SMT Tactic

$$
\begin{array}{lcl}
goal & = & formula\ sequence \times attribute\ sequence \\
\\
proofconv & = & proof\ sequence \rightarrow proof \\
modelconv & = & model \times nat \rightarrow model \\
trt & = & \mathsf{sat}\ model \\
 & \mid & \mathsf{unsat}\ proof \\
 & \mid & \mathsf{unknown}\ goal\ sequence \times modelconv \times proofconv \\
 & \mid & \mathsf{fail} \\
tactic & = & goal \rightarrow trt
\end{array}
$$

non-branching tactics:
sb is a sigleton in
unknown(sb, mc, pc)

Trivial goals

Empty goal [ ] is trivially satisfiable

False goal […, false, …] is trivially unsatisfiable

basic : tactic

SMT Tactic example

$$[\ a = b + 1,\ (a < 0 \lor a > 0),\ b > 3\ ]$$

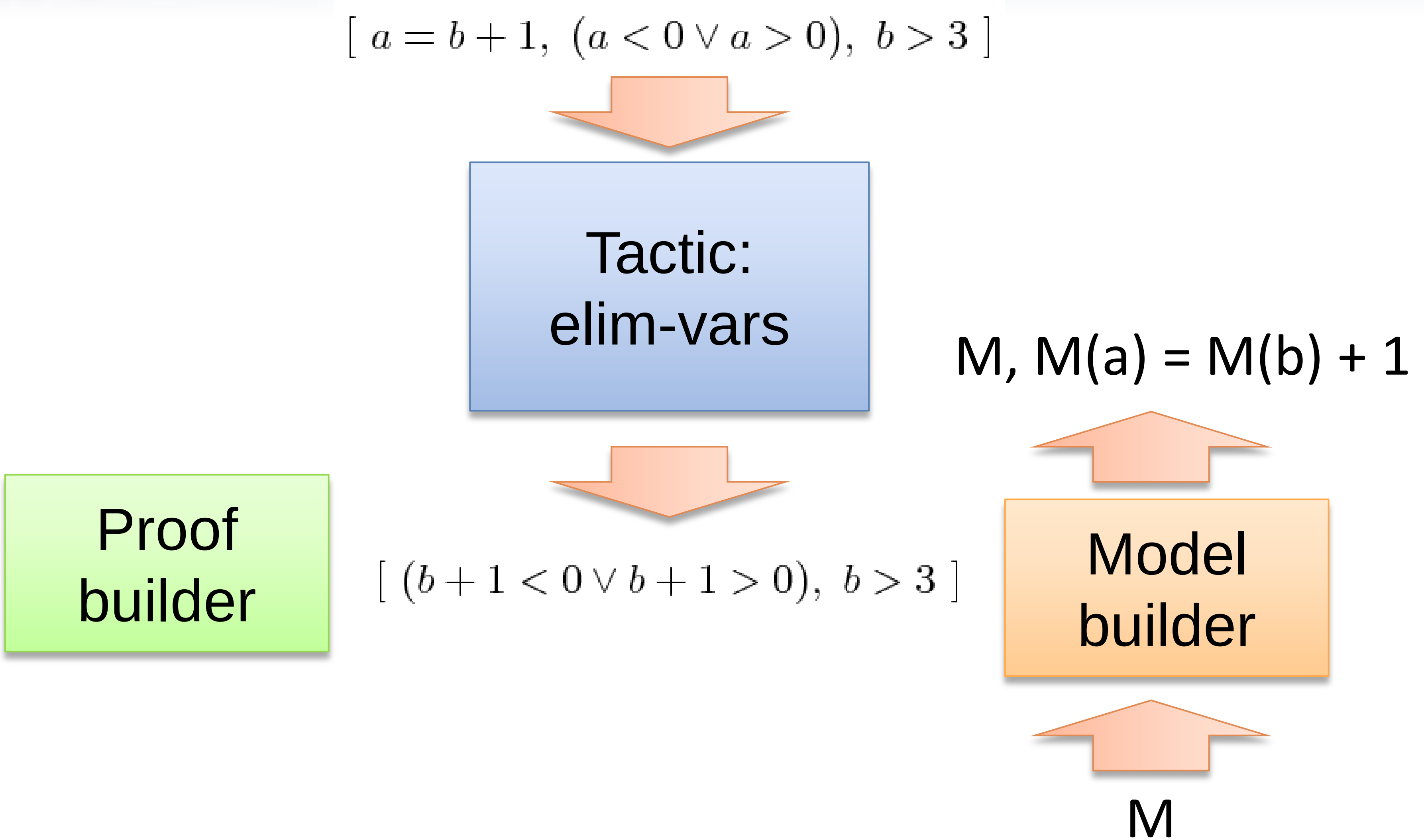
Tactic:
elim-vars

Proof
builder

$$[\ (b + 1 < 0 \lor b + 1 > 0),\ b > 3\ ]$$

Model
builder

SMT Tactic example



SMT Tactic example

$$[\ a = b + 1,\ (a < 0 \vee a > 0),\ b > 3\ ]$$

Tactic: split-or

Proof builder

$$[\ a = b + 1,\ a < 0,\ b > 3\ ]$$

$$[\ a = b + 1,\ a > 0,\ b > 3\ ]$$

Model builder

SMT Tactics

simplify
nnf
cnf
tseitin
lift-if
bitblast
gb
vts

propagate-bounds
propagate-values
split-ineqs
split-eqs
rewrite
p-cad
sat
solve-eqs

SMT Tacticals

then : $(tactic \times tactic) \longrightarrow tactic$

then(t_1, t_2) applies t_1 to the given goal and t_2 to every subgoal produced by t_1.

then$*$: $(tactic \times tactic\ sequence) \longrightarrow tactic$

then$*(t_1, [t_{2_1}, ..., t_{2_n}])$ applies t_1 to the given goal, producing subgoals $g_1, ..., g_m$. If $n \neq m$, the tactic fails. Otherwise, it applies t_{2_i} to every goal g_i.

orelse : $(tactic \times tactic) \longrightarrow tactic$

orelse(t_1, t_2) first applies t_1 to the given goal, if it fails then returns the result of t_2 applied to the given goal.

par : $(tactic \times tactic) \longrightarrow tactic$

par(t_1, t_2) excutes t_1 and t_2 in parallel.

SMT Tacticals

$$\mathsf{then}(\mathsf{skip}, t) = \mathsf{then}(t, \mathsf{skip}) = t$$

$$\mathsf{orelse}(\mathsf{fail}, t) = \mathsf{orelse}(t, \mathsf{fail}) = t$$

SMT Tacticals

$\mathsf{repeat} : \mathit{tactic} \longrightarrow \mathit{tactic}$

Keep applying the given tactic until no subgoal is modified by it.

$\mathsf{repeatupto} : \mathit{tactic} \times \mathit{nat} \longrightarrow \mathit{tactic}$

Keep applying the given tactic until no subgoal is modified by it, or the maximum number of iterations is reached.

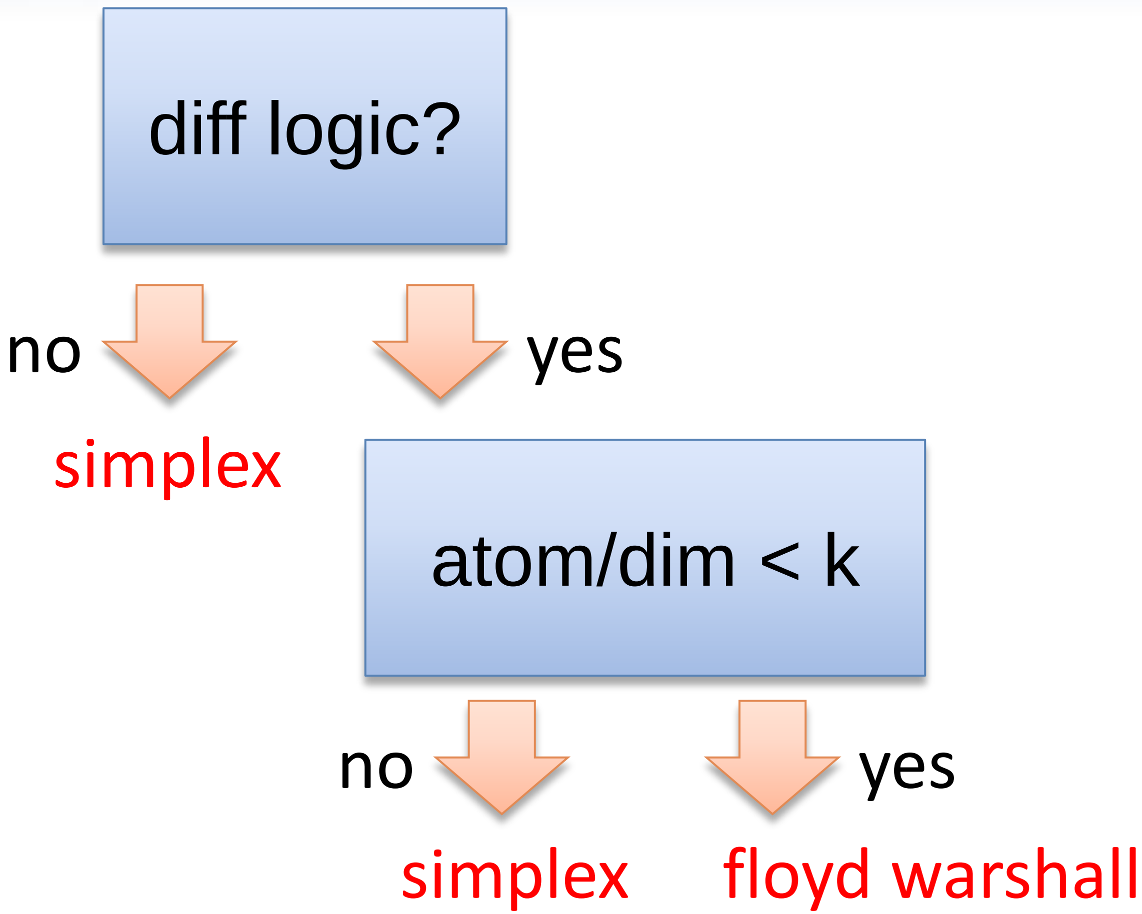
$\mathsf{tryfor} : \mathit{tactic} \times \mathit{seconds} \longrightarrow \mathit{tactic}$

$\mathsf{tryfor}(t, k)$ returns the value computed by tactic t applied to the given goal if this value is computed within k seconds, otherwise it fails.

Feature / Measures

Probing structural features of formulas.

Feature / Measures: Yices Strategy



Feature / Measures: Yices Strategy

$$\text{orelse}(\text{then}(\text{failif}(\text{diff} \wedge \frac{\text{atom}}{\text{dim}} > \text{k}),\ \text{simplex}),\ \text{floydwarshall})$$

Fail if condition is not satisfied.
Otherwise, do nothing.

Feature / Measures: Examples

bw: Sum total bit-width of all rational coefficients of polynomials in case.
diff: True if the formula is in the difference logic fragment.
linear: True if all polynomials are linear.
dim: Number of arithmetic constants.
atoms: Number of atoms.
degree: Maximal total multivariate degree of polynomials.
size: Total formula size.

Tacticals: syntax sugar

$$\mathsf{if}(c,\ t_1,\ t_2) = \mathsf{orelse}(\mathsf{then}(\mathsf{failif}(\neg c), t_1), t_2)$$

$$\mathsf{when}(c,\ t) = \mathsf{if}(c,\ t,\ \mathsf{skip})$$

Under/Over-Approximations

Under-approximation

unsat answers cannot be trusted

Over-approximation

sat answers cannot be trusted

Under/Over-Approximations

Under-approximation

model finders

Over-approximation

proof finders

Under/Over-Approximations

Under-approximation

$S \rightarrow S \cup S'$

Over-approximation

$S \rightarrow S \setminus S'$

Under/Over-Approximations

Under-approximation

Example: QF_NIA model finders

add bounds to unbounded variables (and blast)

Over-approximation

Example: Boolean abstraction

Under/Over-Approximations

Combining under and over is bad!

sat and unsat answers cannot be trusted.

Tracking: under/over-approximations

In principle, proof and model converters can check if the resultant models and proofs are valid.

Tracking: under/over-approximations

In principle, proof and model converters can check if the resultant models and proofs are valid.

Problem: if it fails what do we do?

Tracking: under/over-approximations

In principle, proof and model converters can check if the resultant models and proofs are valid.

Problem: if it fails what do we do?

We want to write tactics that can check whether a goal is the result of an abstraction or not.

Tracking: under/over-approximations

Solution

Associate an precision attribute to each goal.

Goal Attributes

Store extra logical information

Examples:

precision markers

goal depth

polynomial factorizations

SMT → SAT Abstraction/Refinement

Basic Idea

$x \geq 0,\ y = x + 1,\ (y > 2 \vee y < 1)$

Abstract (aka "naming" atoms)

$p_1,\ \ p_2,\ (p_3 \vee p_4)$ $\qquad$ $p_1 \equiv (x \geq 0),\ p_2 \equiv (y = x + 1),$

$p_3 \equiv (y > 2),\ p_4 \equiv (y < 1)$

SMT → SAT Abstraction/Refinement

Basic Idea

$x \geq 0,\ y = x + 1,\ (y > 2 \vee y < 1)$

Abstract (aka "naming" atoms)

$p_1,\ \ p_2,\ (p_3 \vee p_4)$

$p_1 \equiv (x \geq 0),\ p_2 \equiv (y = x + 1),$
$p_3 \equiv (y > 2),\ p_4 \equiv (y < 1)$

SAT
Solver

SMT → SAT Abstraction/Refinement

Basic Idea

$x \geq 0,\ y = x + 1,\ (y > 2 \vee y < 1)$

Abstract (aka "naming" atoms)

$p_1,\ p_2,\ (p_3 \vee p_4)$

$p_1 \equiv (x \geq 0),\ p_2 \equiv (y = x + 1),$

$p_3 \equiv (y > 2),\ p_4 \equiv (y < 1)$

SAT Solver

Assignment

$p_1,\ p_2,\ \neg p_3,\ p_4$

SMT → SAT Abstraction/Refinement

Basic Idea

$x \geq 0,\ y = x + 1,\ (y > 2 \vee y < 1)$

Abstract (aka "naming" atoms)

$p_1,\ p_2,\ (p_3 \vee p_4)$

$p_1 \equiv (x \geq 0),\ p_2 \equiv (y = x + 1),$
$p_3 \equiv (y > 2),\ p_4 \equiv (y < 1)$

SAT Solver

Assignment

$p_1,\ p_2,\ \neg p_3,\ p_4$

$x \geq 0,\ y = x + 1,$
$\neg(y > 2),\ y < 1$

SMT → SAT Abstraction/Refinement

Basic Idea

$x \geq 0,\ y = x + 1,\ (y > 2 \vee y < 1)$

Abstract (aka "naming" atoms)

$p_1,\ p_2,\ (p_3 \vee p_4)$

$p_1 \equiv (x \geq 0),\ p_2 \equiv (y = x + 1),$
$p_3 \equiv (y > 2),\ p_4 \equiv (y < 1)$

SAT Solver

Assignment
$p_1,\ p_2,\ \neg p_3,\ p_4$

$x \geq 0,\ y = x + 1,$
$\neg(y > 2),\ y < 1$

Theory Solver

Unsatisfiable
$x \geq 0,\ y = x + 1,\ y < 1$

SMT → SAT Abstraction/Refinement

Basic Idea

$x \geq 0,\ y = x + 1,\ (y > 2 \vee y < 1)$

Abstract (aka "naming" atoms)

$p_1,\ p_2,\ (p_3 \vee p_4)$

$p_1 \equiv (x \geq 0),\ p_2 \equiv (y = x + 1),$
$p_3 \equiv (y > 2),\ p_4 \equiv (y < 1)$

SAT Solver

Assignment
$p_1,\ p_2,\ \neg p_3,\ p_4$

$x \geq 0,\ y = x + 1,$
$\neg(y > 2),\ y < 1$

Theory Solver

Unsatisfiable
$x \geq 0,\ y = x + 1,\ y < 1$

New Lemma
$\neg p_1 \vee \neg p_2 \vee \neg p_4$

SMT → SAT Abstraction/Refinement

New Lemma

$\neg p_1 \vee \neg p_2 \vee \neg p_4$

Unsatisfiable

$x \geq 0,\ y = x + 1,\ y < 1$

Theory Solver

AKA
Theory conflict

Decision Engines as Tacticals

then(preprocess, smt(finalcheck))

Apply “cheap” propagation/pruning steps;
and then apply complete “expensive” procedure

Decision Engines as Tacticals

AP-CAD (tactic) = tactic

Strategy: Example

$$\mathsf{then}(\mathsf{then}(\mathsf{simplify},\ \mathsf{gaussian}),\ \mathsf{orelse}(\mathsf{modelfinder},\ \mathsf{smt}(\mathsf{apcad}(\mathsf{icp}))))$$

RAHD Calculemus Strategy

	dim	deg	div	calc-0	calc-1	calc-2	qepcad-b	redlog/rlqe	redlog/rlcad
P0	5	4	N	.91	1.59	1.7	416.45*	40.4	-
P1	6	4	N	1.69	3.08	3.42	-*	-	-
P2	5	4	N	1.34	2.41	2.62	-*	-	-
P3	5	4	N	1.52	2.56	2.75	-*	-	-
P4	5	4	N	1.14	2.02	2.16	-*	-	-
P5	14	2	N	.25	.26	.27	-*	97.4	-
P6	11	5	N	147.4	.07	.06	-*	<.01	<.01
P7	8	2	N	.05	<.01	<.01	.08	<.01	<.01
P8	7	32	N	4.5	.1	<.01	8.38	<.01	-
P9	7	16	N	4.51	.15	<.01	.29	.01	6.7
P10	7	12	N	100.74	20.76	8.85	-*	-	-
P11	6	2	Y	1.6	.5	.53	.01	.01	.05
P12	5	3	N	.78	.3	.36	.02	.01	.07
P13	4	10	N	3.83	3.95	4.02	-*	-	-
P14	2	2	N	4.55	1.67	.07	.01	-	-
P15	4	3	Y	.177	.2	.12	.01	<.01	<.01
P16	4	2	N	9.99	2.17	2.1	.02	<.01	<.01
P17	4	2	N	.62	.59	.65	.28	.02	.61
P18	4	2	N	1.25	1.28	1.27	.01	<.01	<.01
P19	3	6	Y	3.34	1.72	2.08	.02	.01	.7
P20	3	4	N	1.18	.65	.65	.01	<.01	.3
P21	3	2	N	.02	.03	<.01	.02	.01	.1
P22	2	4	N	<.01	<.01	<.01	.01	<.01	<.01
P23	2	2	Y	<.01	<.01	<.01	<.01	<.01	<.01

Z3 QF_LIA Strategy

then(preamble, orelse(mf, pb, bounded, smt)

Simplification
Constant propagation
Interval propagation
Contextual simplification
If-then-else elimination
Gaussian elimination
Unconstrained terms

Challenge: small step configuration

proof procedure as a transition system

Abstract DPLL, DPLL(T), Abstract GB, cutsat, …

UnitPropagate :

$$M \parallel F, C \vee l \implies M\ l \parallel F, C \vee l \quad \text{if } \begin{cases} M \models \neg C \\ l \text{ is undefined in } M \end{cases}$$

PureLiteral :

$$M \parallel F \implies M\ l \parallel F \quad \text{if } \begin{cases} l \text{ occurs in some clause of } F \\ \neg l \text{ occurs in no clause of } F \\ l \text{ is undefined in } M \end{cases}$$

Decide :

$$M \parallel F \implies M\ l^{\mathsf{d}} \parallel F \quad \text{if } \begin{cases} l \text{ or } \neg l \text{ occurs in a clause of } F \\ l \text{ is undefined in } M \end{cases}$$

Fail :

$$M \parallel F, C \implies \mathit{FailState} \quad \text{if } \begin{cases} M \models \neg C \\ M \text{ contains no decision literals} \end{cases}$$

Backtrack :

$$M\ l^{\mathsf{d}}\ N \parallel F, C \implies M\ \neg l \parallel F, C \quad \text{if } \begin{cases} M\ l^{\mathsf{d}}\ N \models \neg C \\ N \text{ contains no decision literals} \end{cases}$$

Challenge: small step configuration

proof procedure as a transition system

Abstract DPLL, DPLL(T), Abstract GB, cutsat, …

Challenge:
Efficient strategic control

Decide :

$$M \parallel F \quad \Longrightarrow \quad M\ l^{\mathsf{d}} \parallel F \qquad \text{if } \begin{cases} l \text{ or } \neg l \text{ occurs in a clause of } F \\ l \text{ is undefined in } M \end{cases}$$

Fail :

$$M \parallel F, C \quad \Longrightarrow \quad \mathit{FailState} \qquad \text{if } \begin{cases} M \models \neg C \\ M \text{ contains no decision literals} \end{cases}$$

Backtrack :

$$M\ l^{\mathsf{d}}\ N \parallel F, C \quad \Longrightarrow \quad M\ \neg l \parallel F, C \qquad \text{if } \begin{cases} M\ l^{\mathsf{d}}\ N \models \neg C \\ N \text{ contains no decision literals} \end{cases}$$

Conclusion

Different domains need different strategies.

We must expose the little engines in SMT solvers.

Interaction between different engines is a must.

Tactic and Tacticals: big step approach.

More transparency.