Orchestrating Decision Engines
CP 2011, Perugia, Italy

Leonardo de Moura (Microsoft Research) and Grant Passmore (University of Cambridge)
A Satisfiability Checker with built-in support for useful theories
$b + 2 = c$ and $f(\text{read}(\text{write}(a, b, 3), c-2)) \neq f(c-b+1)$
\[ b + 2 = c \quad \text{and} \quad f(\text{read}(\text{write}(a, b, 3), c - 2)) \neq f(c - b + 1) \]
b + 2 = c  and  \( f(\text{read}(\text{write}(a,b,3), c-2) \neq f(c-b+1) \)
Satisfiability Modulo Theories (SMT)

\[ b + 2 = c \quad \text{and} \quad f(\text{read}(\text{write}(a,b,3), c-2)) \neq f(c-b+1) \]
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
- ...
Some Applications @ Microsoft

- The Spec# Programming System
- HAVOC
- Terminaotr T-2
- Hyper-V (Microsoft Virtualization)
- VCC
- NModel
- SLAM
- SpecExplorer
- SAGE
- Vigilante
- F7

Microsoft Research
Application Scenarios

“Big” and hard formulas

Thousands of “small” and easy formulas

Short timeout (< 5secs)
Application Scenarios

“Big” and hard formulas

Thousands of “small” and easy formulas

Short timeout (< 5secs)
Verification/Analysis Tool: “Template”

1. Problem
2. Verification/Analysis Tool
3. Logical Formula
4. Theorem Prover/Satisfiability Checker
5. Satisfiable (Counter-example)
6. Unsatisfiable
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
(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)
Verification/Analysis tools need some form of Symbolic Reasoning
Logic is “The Calculus of Computer Science” (Z. Manna).
High computational complexity
Yes, we cannot solve arbitrary problems from the “complexity ladder”.

But,...
We can try to solve the problems we find in real applications
Main challenges

- Scalability (huge formulas)
- Complexity
- Undecidability
- Quantified formulas
SMT@MS: Applications

A Sample
Directed Automated Random Testing (DART)

Run Test and Monitor

Execution Path

Known Paths

Path Condition

Test Inputs

seed

New input

Constraint System

Solve

Z3

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

(x_0 = 2) and
(y_0 = 4) 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)

We want a trace where the loop is executed twice.
White box testing in practice

How to test this code?
(Real code from .NET base class libraries.)

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

    _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();
}
```
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.

```csharp
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.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.
        }
    }
    catch (Exception ex) {
        BCLDebug.Log("RESMGRFILEFORMAT", "ReadResources: Error while reading resmgr header: ", ex);
    }
}
```

...
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.

```java
private void ReadResources()
{
    BCLDebug.Assert(_store != null, "ResourceReader is closed!");
    BinaryFormatter bf = new BinaryFormatter(null, new StreamingContext(StreamingContextStates.File |
    #if !FEATURE_PAL
    _typeLimitingBinder = new TypeLimitingDeserializerBinder();
    bf.Binder = _typeLimitingBinder;
    #endif

    _objFormatter = bf;
    try {
        // Read ResourceManager header
        // Check for magic number
        int magicNum = _store.ReadInt32();
        if (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");
                int
                return mStream.InternalReadInt32();
            }
            else
                FillBuffer(4);
```

Pex - Test Input Generation

Test input, generated by Pex

```csharp
byte[] a = new byte[14];
a[0] = 206;
a[1] = 202;
a[2] = 239;
a[3] = 190;
a[7] = 64;
a[11] = 100;
ParameterizedTest(a);
```
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.
Starting with 100 zero bytes ...

SAGE generates a crashing test for Media1 parser

Generation 0 – seed file
Starting with 100 zero bytes ...

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

SAGE generates a crashing test for Media1 parser

Generation 10 – CRASH
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

Spec#
Spec# compiler
MSIL
Bytecode translator
Boogie
V.C. generator
Verification condition
Z3

C
VCC
HAVOC

Static program verifier (Boogie)
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
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...
Pex, Spec#, VCC and many other tools are available online.
Current SMT solvers provide a combination of different engines
Combining Engines

SMT

- Congruence Closure
- Simplification
- KB Completion
- Superposition
- DPLL
- Simplex
- Grobner Basis
- $\forall\exists$-elimination
Configuring SAT/SMT Solvers: “state-of-the-art”

F
Config

Theorem Prover/Satisfiability Checker

Satisfiable (model)
Unsatisfiable (proof)

Z3 has approx. 300 options

Config

Z3 has approx. 300 options
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.”
To build theoretical and practical tools allowing users to exert strategic control over core heuristic aspects of high performance SMT solvers.
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.
Different Strategies for Different Domains.
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 \( T[x, x'] \)
\[ \forall x, x' . I[x] \land T^k[x, x'] \rightarrow \exists y, y' . I[y] \land T^{k-1}[y, y'] \]

Ranking function synthesis.
Hardware Fixpoint Checks

![Graph 1: QuBE vs. Z3](image1)

![Graph 2: sKizzo vs. Z3](image2)
Ranking Function Synthesis

Graphs showing the relationship between QuBE and [sec] on the left, and sKizzo and [sec] on the right.
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.
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

- Tactic
- Goal
- Subgoals
- Proof builder
Main inspiration: LCF-approach

- Tactic
- Proof builder
- Proof for goal
- Proofs for subgoals
- goal
- subgoals
Main inspiration: LCF-approach

Tactic → Proof builder → Tactic

Proof builder → Tactic → Proof builder

goal
Main inspiration: LCF-approach
Main inspiration: LCF-approach

Proof Builder

Proof Builder

Proof Builder

$thm$ in LCF terminology

$proof$ in LCF terminology
Tacticals aka Combinators

then( Tactic , Tactic ) = Tactic

orelse( Tactic , Tactic ) = Tactic

repeat( Tactic ) = Tactic
goal = formula sequence × attribute sequence

proofconv = proof sequence → proof
modelconv = model × nat → model
trt = sat model
| unsat proof
| unknown goal sequence × modelconv × proofconv
| fail

tactic = goal → trt
SMT Tactic

\[
\text{goal} \quad = \quad \text{formula sequence} \times \text{attribute sequence}
\]
\[
\text{proofconv} \quad = \quad \text{proof sequence} \rightarrow \text{proof}
\]
\[
\text{modelconv} \quad = \quad \text{model} \times \text{nat} \rightarrow \text{model}
\]
\[
\text{trt} \quad = \quad \text{sat model}
\]
\[
\quad | \quad \text{unsat proof}
\]
\[
\quad | \quad \text{unknown goal sequence} \times \text{modelconv} \times \text{proofconv}
\]
\[
\quad | \quad \text{fail}
\]
\[
\text{tactic} \quad = \quad \text{goal} \rightarrow \text{trt}
\]

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

\[
goal = \text{formula sequence } \times \text{attribute sequence}
\]

\[
\text{proofconv} = \text{proof sequence } \rightarrow \text{proof}
\]

\[
\text{modelconv} = \text{model} \times \text{nat} \rightarrow \text{model}
\]

\[
trt = \text{sat model}
\]

\[
| \text{unsat proof}
\]

\[
| \text{unknown goal sequence } \times \text{modelconv } \times \text{proofconv}
\]

\[
| \text{fail}
\]

\[
tactic = \text{goal } \rightarrow \text{trt}
\]

non-branching tactics:

sb is a singleton in unknown(sb, mc, pc)
Empty goal [ ] is trivially satisfiable

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

basic : tactic
Tactic: elim-vars

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

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

Tactic: elim-vars

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

M, M(a) = M(b) + 1

Proof builder

Model builder

M
SMT Tactic example

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

**Tactic:** split-or

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

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

**Proof builder**

**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
then : \((tactic \times tactic) \rightarrow tactic\)

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

then* : \((tactic \times tactic sequence) \rightarrow tactic\)

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 : \((tactic \times tactic) \rightarrow tactic\)

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 : \((tactic \times tactic) \rightarrow tactic\)

par\((t_1, t_2)\) executes \(t_1\) and \(t_2\) in parallel.
then(skip, t) = then(t, skip) = t

orelse(fail, t) = orelse(t, fail) = t
repeat : tactic \rightarrow tactic

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

repeatupto : tactic \times \text{nat} \rightarrow tactic

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

tryfor : tactic \times \text{seconds} \rightarrow tactic

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.
Probing structural features of formulas.
Feature / Measures: Yices Strategy

diff logic?  

no  

yes  

no  

atom/dim < k

no  yes  

simplex  

simplex  floyd warshall
orelse(then(failif(diff ∧ \frac{atom}{dim} > k), simplex), floydwarshall)

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

\[
\text{if}(c, \ t_1, \ t_2) = \text{orelse}(\text{then}(\text{failif}(\neg c), \ t_1), \ t_2) \\
\text{when}(c, \ t) = \text{if}(c, \ t, \ \text{skip})
\]
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
Combining under and over is bad!

sat and unsat answers cannot be trusted.
In principle, proof and model converters can check if the resultant models and proofs are valid.
In principle, proof and model converters can check if the resultant models and proofs are valid.

Problem: if it fails what do we do?
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.
Solution

Associate an precision attribute to each goal.
Store extra logical information

Examples:

- precision markers
- goal depth
- polynomial factorizations
Basic Idea

\[ x \geq 0, \ y = x + 1, \ (y > 2 \lor y < 1) \]

Abstract (aka “naming” atoms)

\[ p_1, \ p_2, (p_3 \lor p_4) \]

\[ p_1 \equiv (x \geq 0), \ p_2 \equiv (y = x + 1), \]
\[ p_3 \equiv (y > 2), \ p_4 \equiv (y < 1) \]
Basic Idea

\[ x \geq 0, \ y = x + 1, \ (y > 2 \lor y < 1) \]

Abstract (aka “naming” atoms)

\[ p_1, \ p_2, \ (p_3 \lor p_4) \]

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

SAT Solver
Basic Idea

\[ \begin{align*}
x & \geq 0, \ y = x + 1, \ (y > 2 \lor y < 1) \\
\end{align*} \]

Abstract (aka “naming” atoms)

\[ \begin{align*}
p_1 & \equiv (x \geq 0), \ p_2 \equiv (y = x + 1), \\
p_3 & \equiv (y > 2), \ p_4 \equiv (y < 1) \\
\end{align*} \]

Assignment

\[ \begin{align*}
p_1, \ p_2, \ \neg p_3, \ p_4 \\
\end{align*} \]
Basic Idea

\[ x \geq 0, \ y = x + 1, \ (y > 2 \lor y < 1) \]

Abstract (aka “naming” atoms)

\[ p_1, \ p_2, (p_3 \lor p_4) \]

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

Assignment

\[ p_1, \ p_2, \neg p_3, \ p_4 \]

\[ x \geq 0, \ y = x + 1, \]
\[ \neg(y > 2), \ y < 1 \]
Basic Idea

\[ x \geq 0, y = x + 1, (y > 2 \lor y < 1) \]

Abstract (aka “naming” atoms)

\[ p_1, p_2, (p_3 \lor p_4) \]

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

Assignment

\[ p_1, p_2, \neg p_3, p_4 \]

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

Unsatisfiable

\[ x \geq 0, y = x + 1, y < 1 \]
Basic Idea

\[ x \geq 0, \ y = x + 1, \ (y > 2 \lor y < 1) \]

Abstract (aka “naming” atoms)

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

SAT Solver

Assignment

\[ p_1, \ p_2, \ \neg p_3, \ p_4 \]

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

New Lemma

\[ \neg p_1 \lor \neg p_2 \lor \neg p_4 \]

Unsatisfiable

\[ x \geq 0, \ y = x + 1, \ y < 1 \]

Theory Solver
New Lemma
\neg p_1 \lor \neg p_2 \lor \neg p_4

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

AKA Theory conflict

Theory Solver
then(preprocess, smt(finalcheck))

Apply “cheap” propagation/pruning steps; and then apply complete “expensive” procedure
AP-CAD ( tactic ) = tactic
then(then(simplify, gaussian), orelse(modelfinder, smt(apcad(icp)))))
### RAHD Calculemus Strategy

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

**Simplification**
- Constant propagation
- Interval propagation
- Contextual simplification
- If-then-else elimination
- Gaussian elimination
- Unconstrained terms
proof procedure as a transition system

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

UnitPropagate:

\[ M \parallel F, C \land l \rightarrow M \parallel F, C \land l \quad \text{if} \quad \begin{cases} M \models \neg C \\ l \text{ is undefined in } M \end{cases} \]

PureLiteral:

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

Decide:

\[ M \parallel F \rightarrow M \, \text{true} \parallel F \quad \text{if} \quad \begin{cases} l \text{ or } \neg l \text{ occurs in a clause of } F \\ l \text{ is undefined in } M \end{cases} \]

Fail:

\[ M \parallel F, C \rightarrow \text{FailState} \quad \text{if} \quad \begin{cases} M \models \neg C \\ M \text{ contains no decision literals} \end{cases} \]

Backtrack:

\[ M \, \text{true} \parallel N, F, C \rightarrow M \, \text{true} \parallel \neg l, F, C \quad \text{if} \quad \begin{cases} M \, \text{true} \parallel N \models \neg C \\ N \text{ contains no decision literals} \end{cases} \]
proof procedure as a transition system

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

**Challenge:**
Efficient strategic control

```
M ⊨ F → M ⊨ d σ F

Decide:

M ⊨ F, C → M ⊨ d σ F if \{ l or \neg l occurs in a clause of F, l is undefined in M \}

Fail:

M ⊨ F, C → FailState if \{ M ⊨ \neg C, M contains no decision literals \}

Backtrack:

M ⊨ d N ⊨ F, C → M ⊨ d N ⊨ \neg l | F, C if \{ M ⊨ d N ⊨ \neg C, N contains no decision literals \}
```
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.