The Strategy Challenge in SMT Solving (part I)

IWS 2012, Manchester, UK

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

Microsoft Research

Satisfiability Modulo Theories (SMT)

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

Microsoft Research

Satisfiability Modulo Theories (SMT)

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

Arithmetic

Satisfiability Modulo Theories (SMT)

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

Array Theory

Microsoft® Research

Satisfiability Modulo Theories (SMT)

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

Uninterpreted Functions

Microsoft Research

Main challenges

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

SMT → SAT Abstraction/Refinement

Basic Idea

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

Abstract (aka "naming" atoms)

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

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

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

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

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

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

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

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

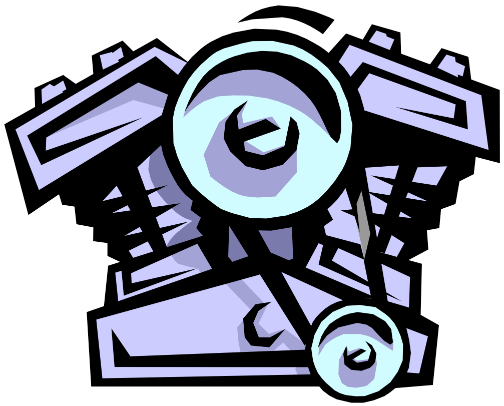
Orchestrating Decision Engines

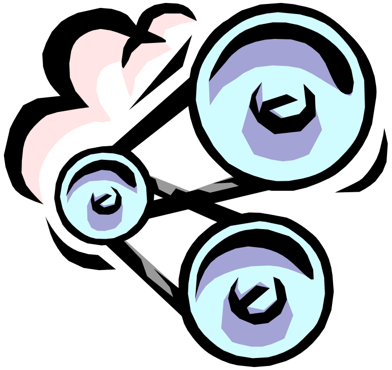
Combining Engines

Current SMT solvers provide

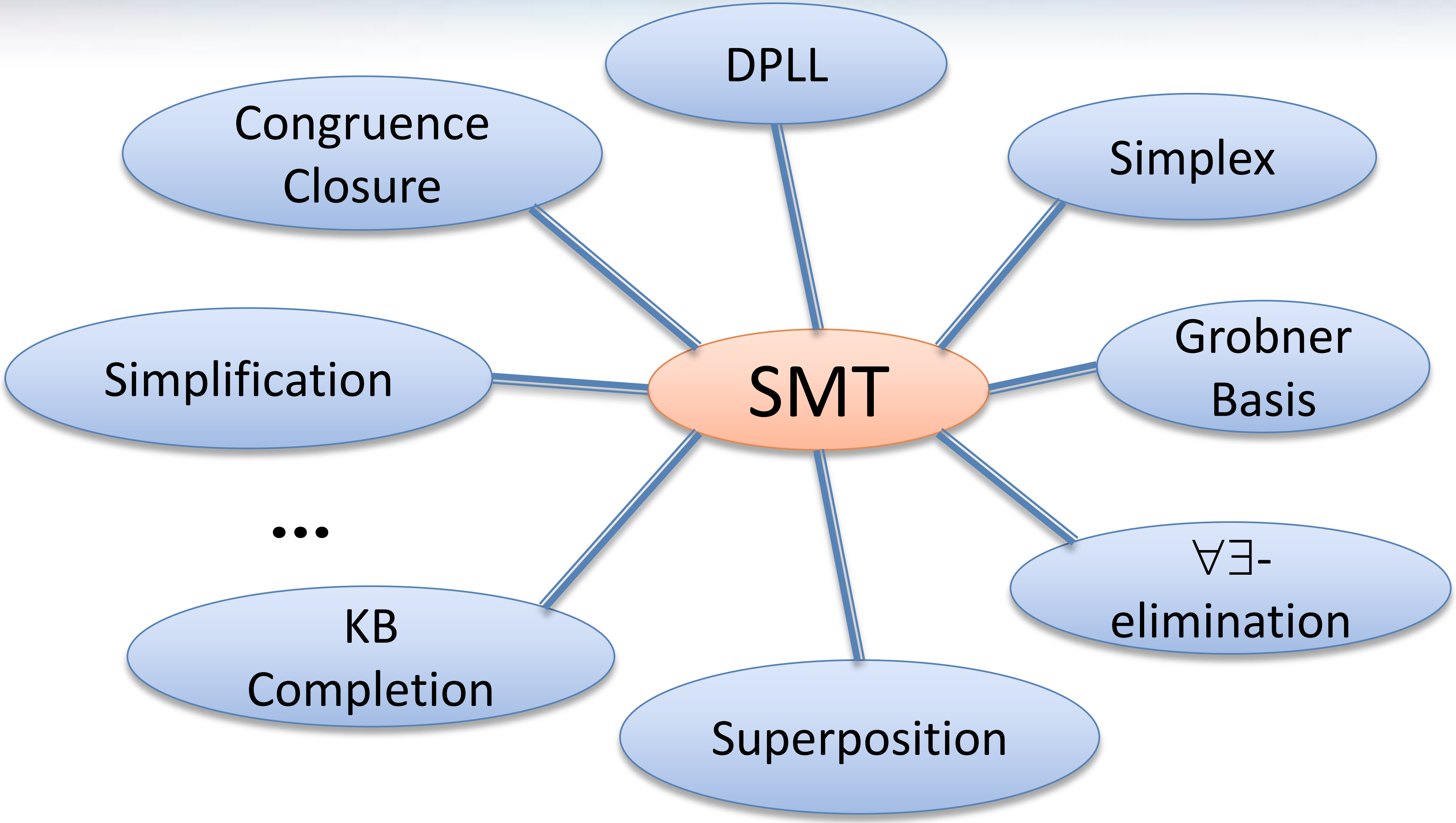
a combination

of different engines

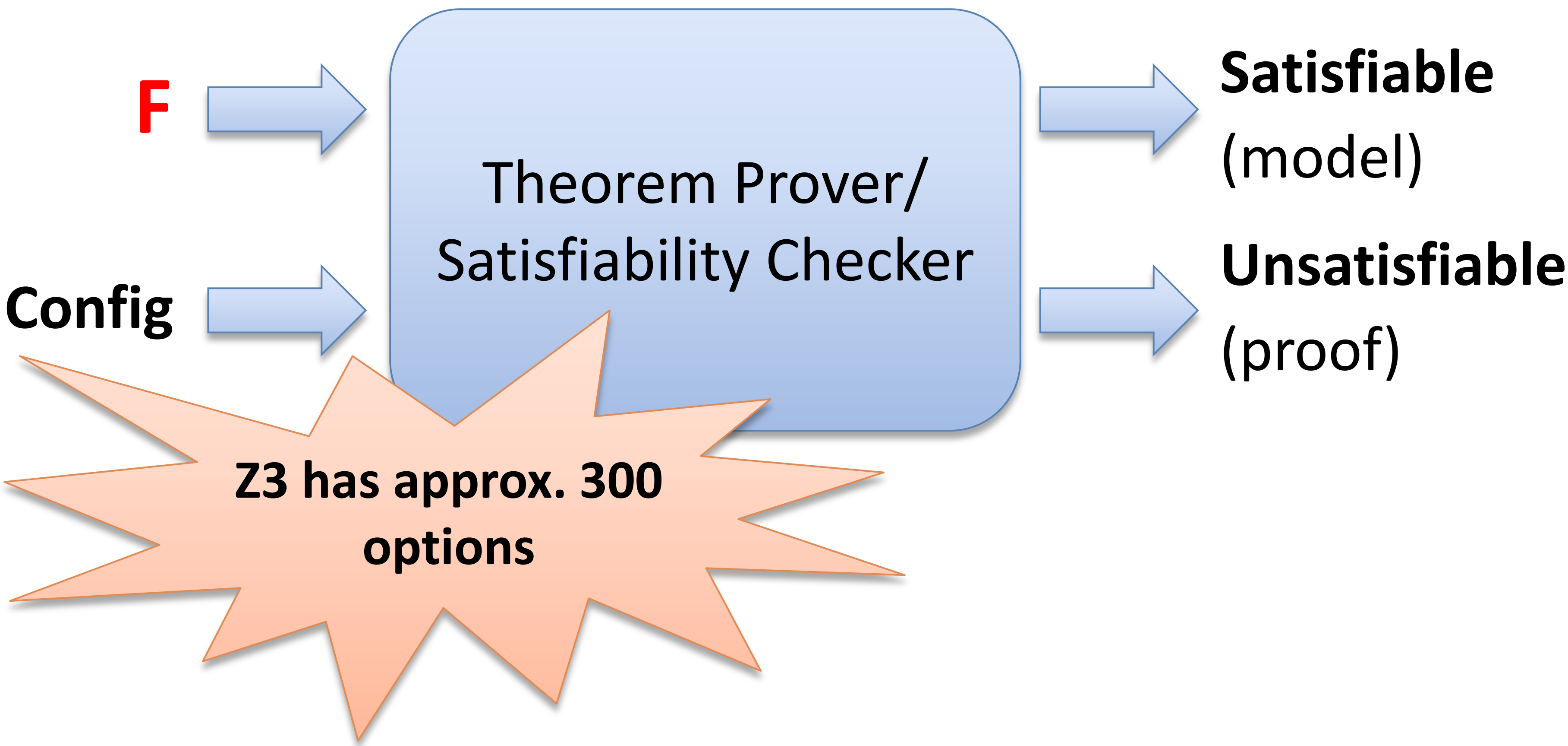




Combining Engines



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

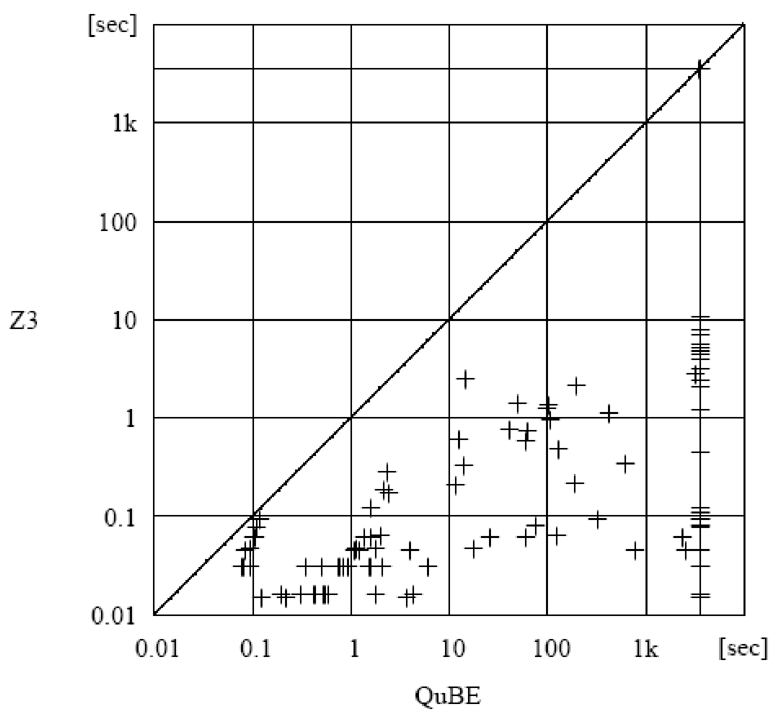
Hardware Fixpoint Checks.

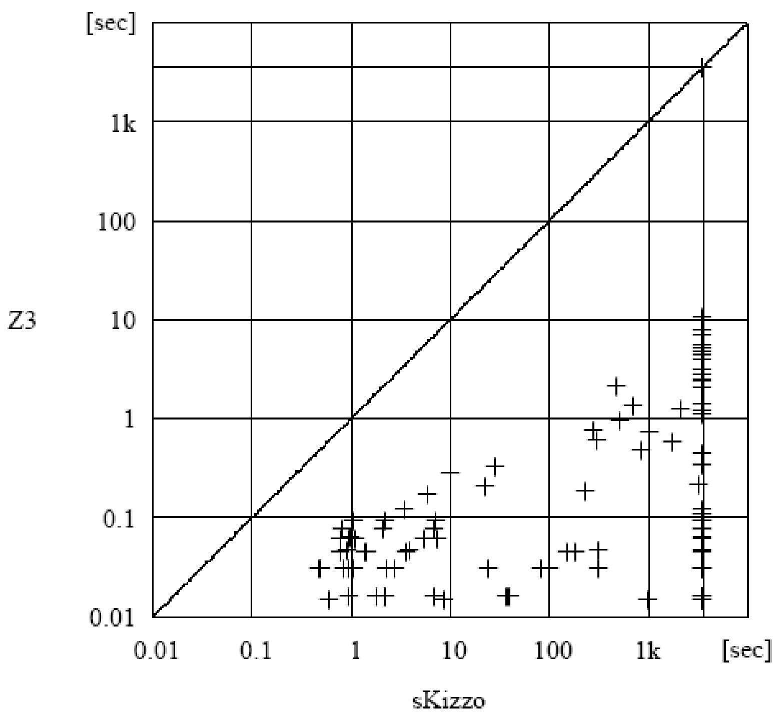
Given: $I[x]$ and $T[x, x']$

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

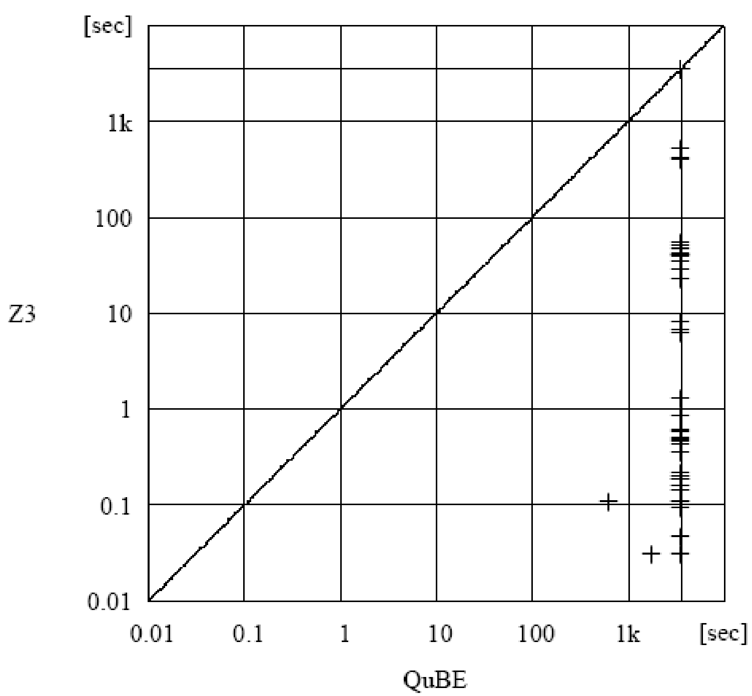
Ranking function synthesis.

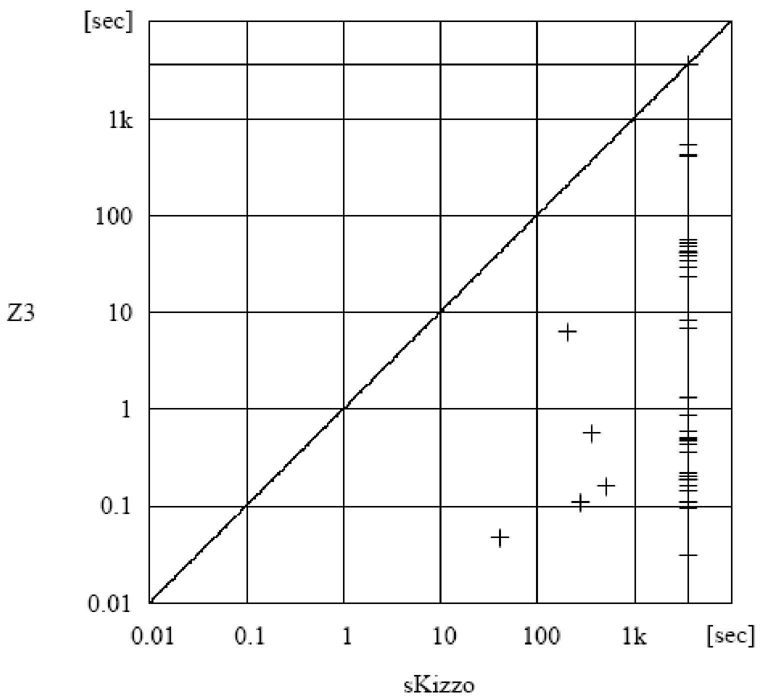
Hardware Fixpoint Checks





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

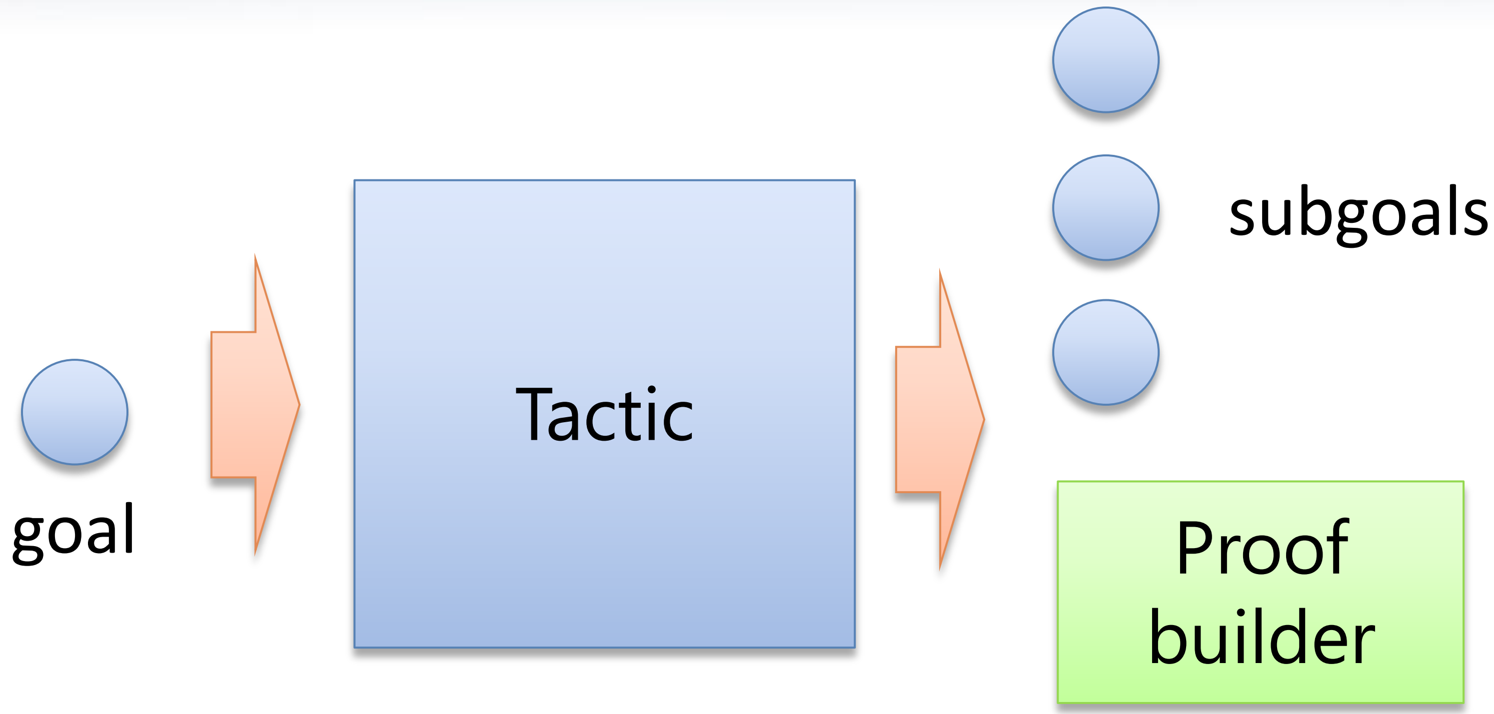
we have access to the source code.

The "Message"

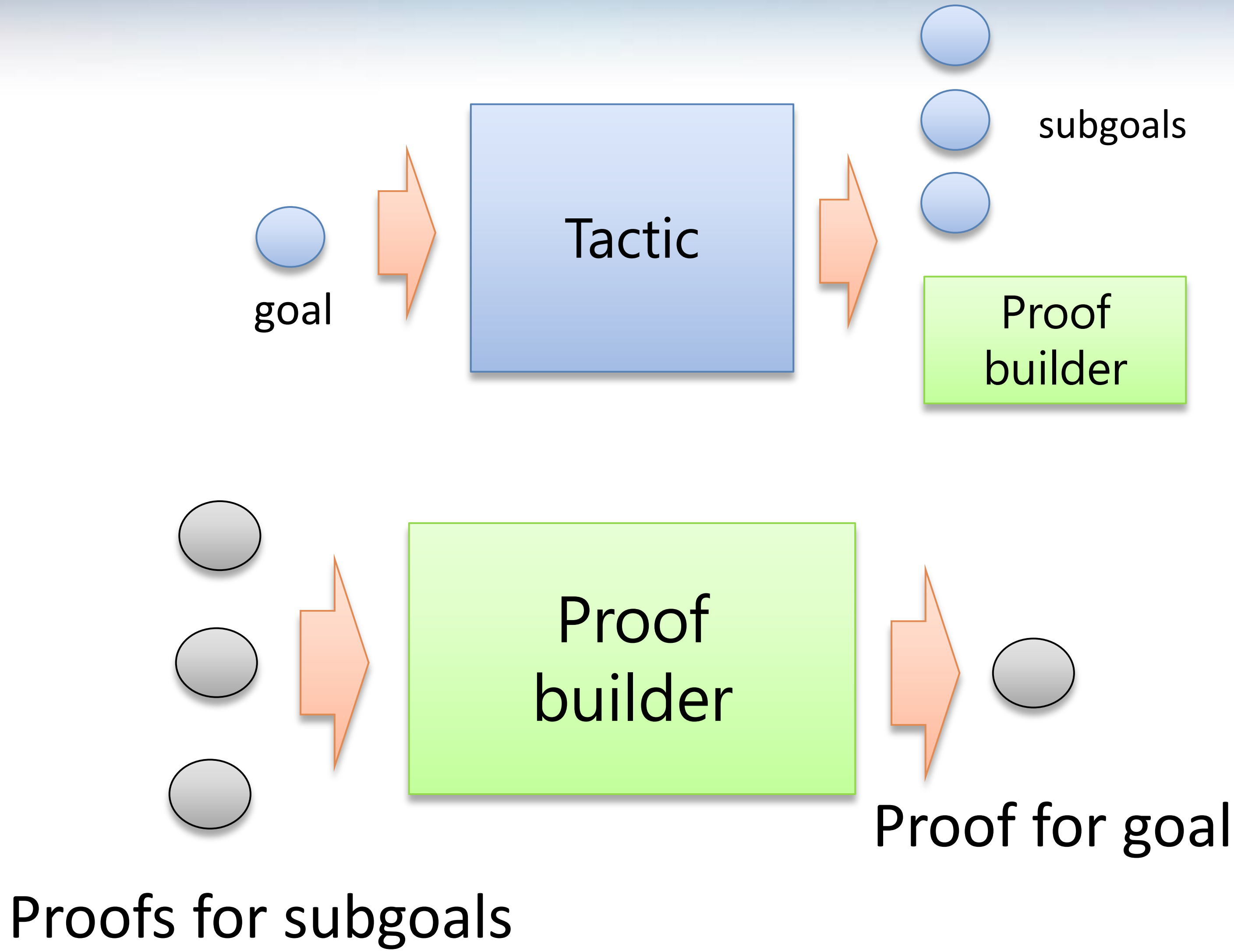
SMT solvers are collections of little engines.

They should provide access to these engines.
Users should be able to define their own strategies.

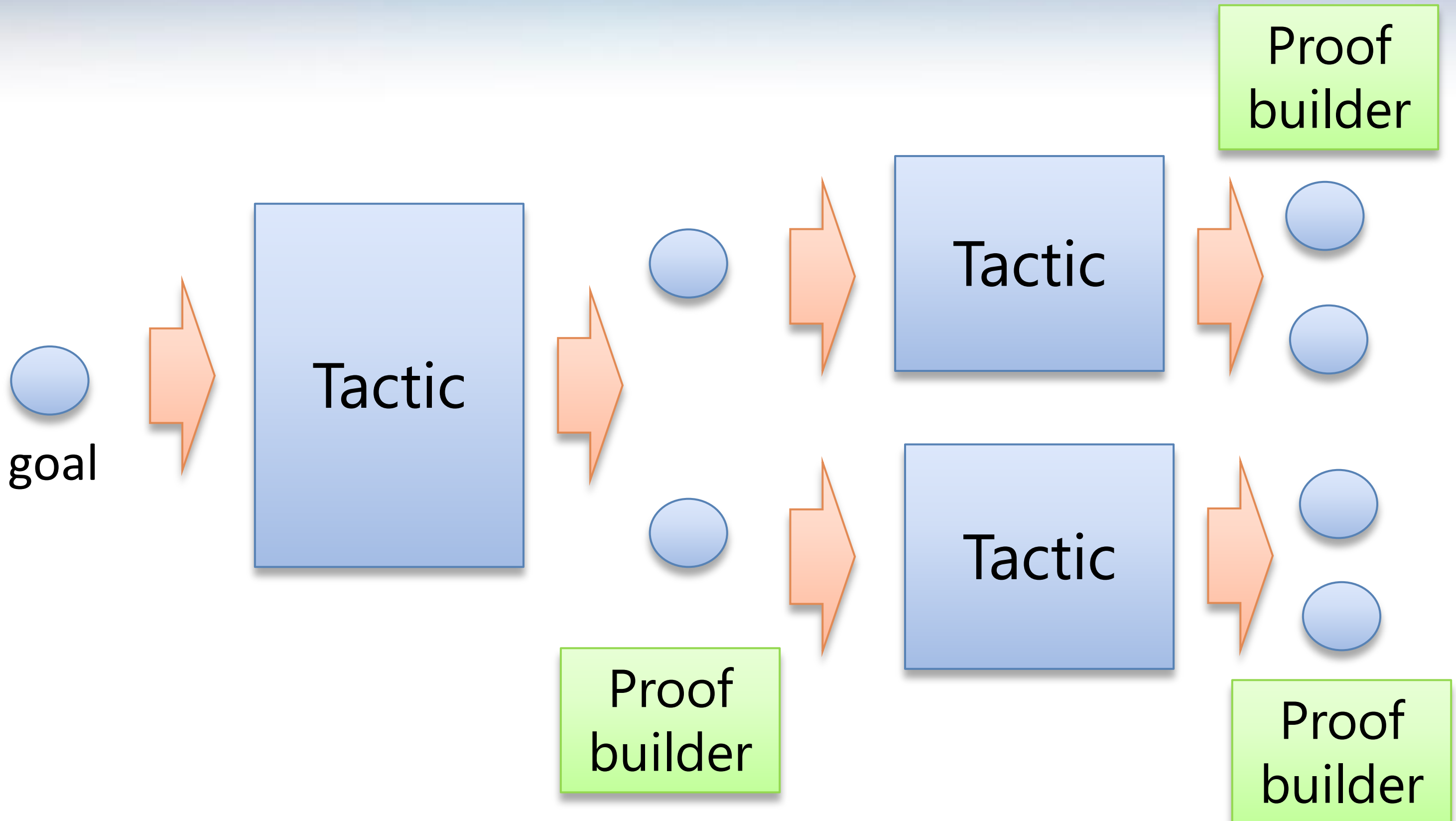
Main inspiration: LCF-approach



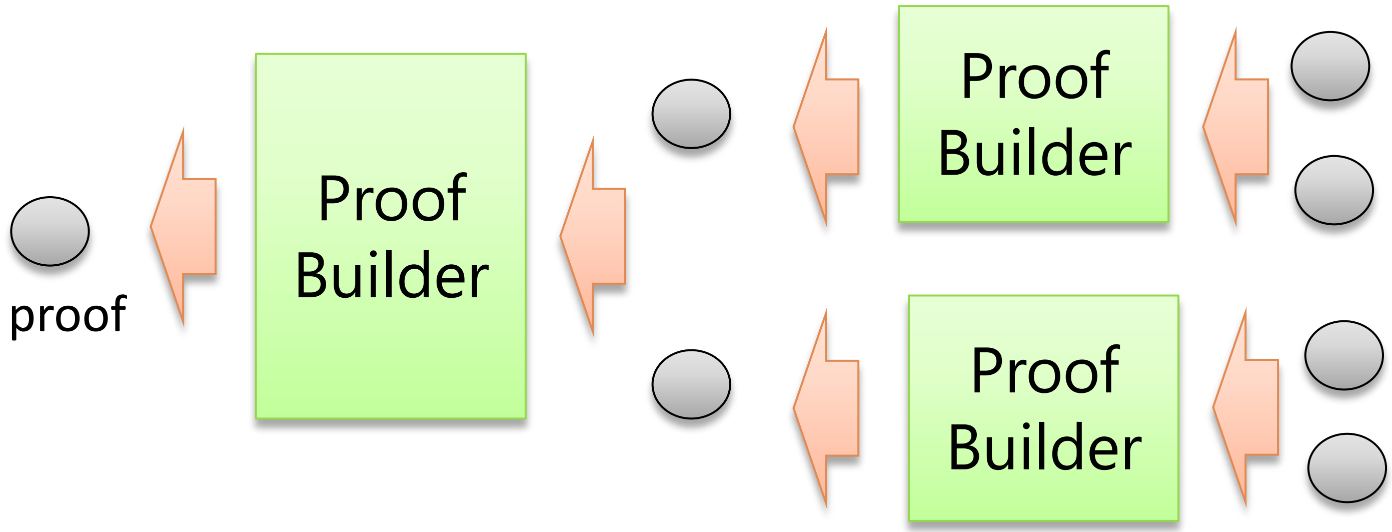
Main inspiration: LCF-approach



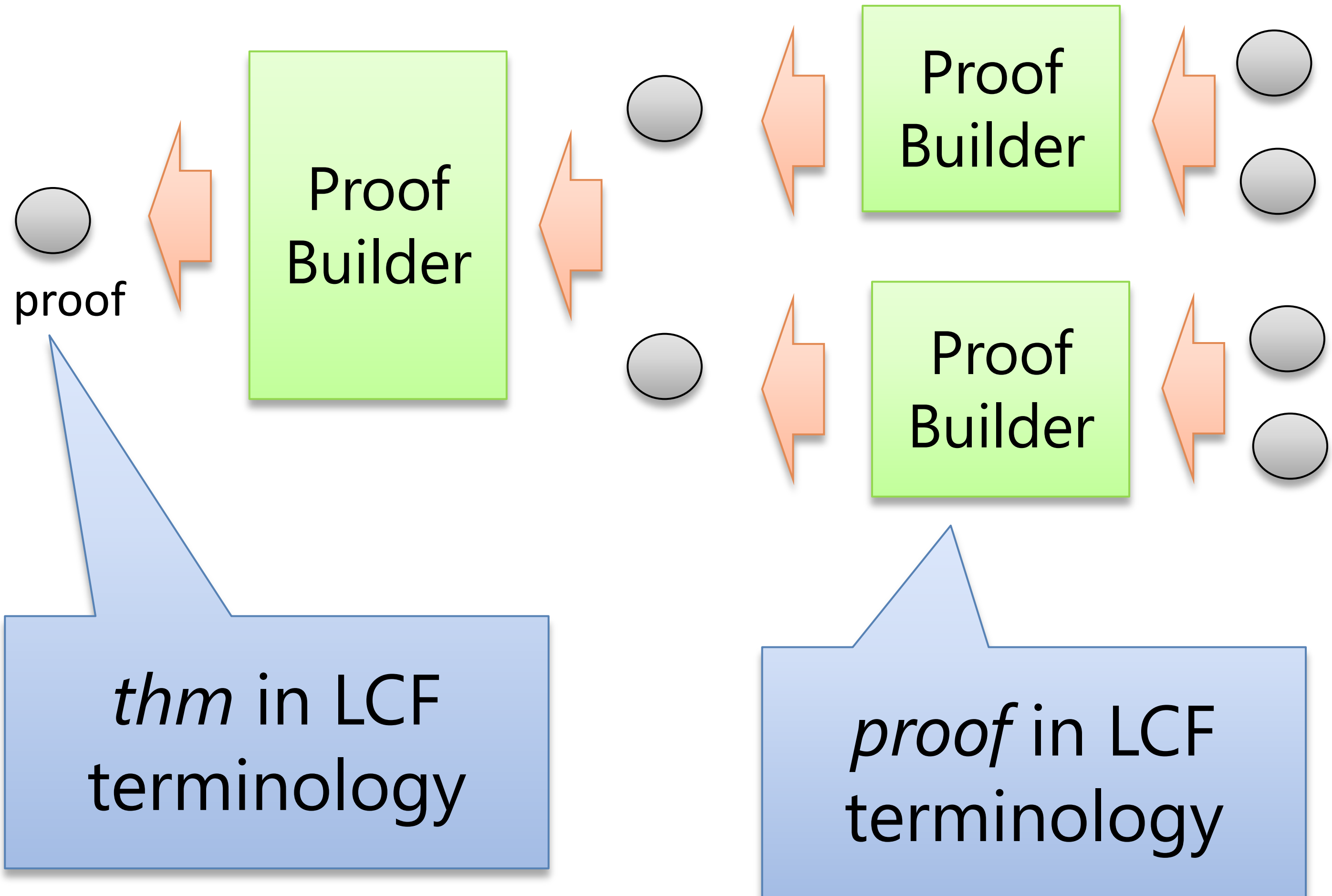
Main inspiration: LCF-approach



Main inspiration: LCF-approach



Main inspiration: LCF-approach



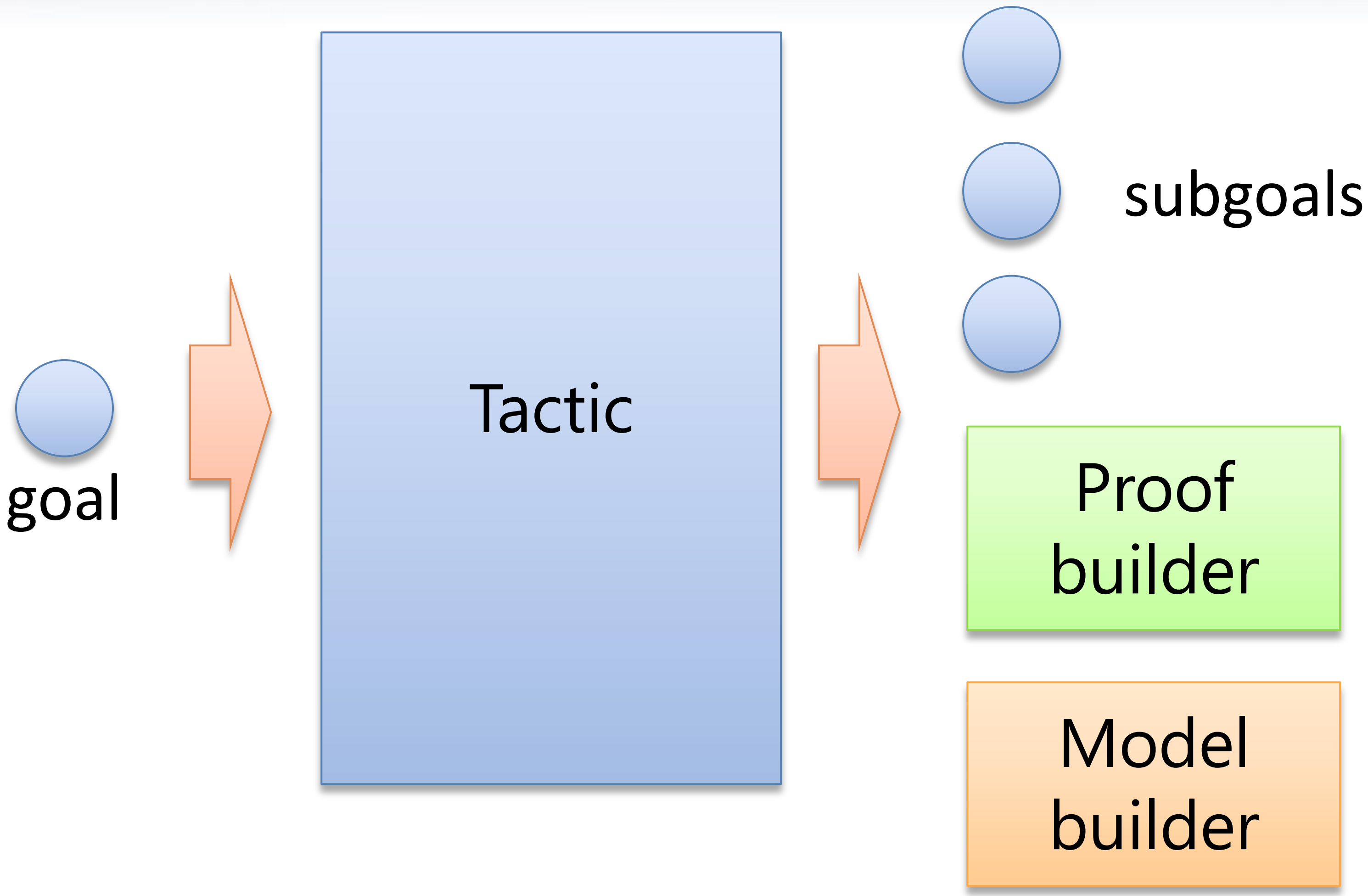
Tacticals aka Combinators

then(Tactic , Tactic) = Tactic

orelse(Tactic , Tactic) = Tactic

repeat(Tactic) = Tactic

SMT Tactic



SMT Tactic

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

SMT Tactic

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

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

SMT Tactic

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

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

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

Tactic:
elim-vars

Proof builder

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

Model builder

SMT Tactic example

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

Tactic:
elim-vars

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

Proof
builder

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

Model
builder

M

SMT Tactic example

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

Tactic:
split-or

Proof builder

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

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

Model builder

SMT Tactics

simplify	propagate-bounds
nnf	propagate-values
cnf	split-ineqs
tseitin	split-eqs
lift-if	rewrite
bitblast	p-cad
gb	sat
vts	solve-eqs

SMT Tactics

$\mathsf{then} : (\mathit{tactic} \times \mathit{tactic}) \longrightarrow \mathit{tactic}$

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

$\mathsf{then*} : (\mathit{tactic} \times \mathit{tactic\ sequence}) \longrightarrow \mathit{tactic}$

$\mathsf{then*}(t_1, [t_{2_1}, ..., t_{2_n}])$ 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.

$\mathsf{orelse} : (\mathit{tactic} \times \mathit{tactic}) \longrightarrow \mathit{tactic}$

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

$\mathsf{par} : (\mathit{tactic} \times \mathit{tactic}) \longrightarrow \mathit{tactic}$

$\mathsf{par}(t_1, t_2)$ 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

repeat : $tactic \longrightarrow tactic$

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

repeatupto : $tactic \times nat \longrightarrow tactic$

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

tryfor : $tactic \times seconds \longrightarrow tactic$

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.

Strategies online

http://rise4fun.com/z3/tutorial/strategies (SMT 2.0)

http://rise4fun.com/z3py/tutorial/strategies (Python)