The Focus Problem: A Fundamental Issue in Automatic Verification

John Slaney
John.Slaney@nicta.com.au

NATIONAL ICT AUSTRALIA

Outline

Background

- automatic proof search
- the focus problem
- an example

The Soft-SCOTT algorithm

- the SCOTT project
- using soft constraints

Experimental results

- some algebraic problems
- the set theory example

Conclusion and future work

Automated deduction

Automated deduction

Show $\Gamma \vdash B$. Two fundamental techniques.

Automated deduction

Show $\Gamma \vdash B$. Two fundamental techniques.

1. Bottom-up methods:
   $\langle A_1, A_2, \ldots \quad \ldots \quad B\rangle$
   (Γ)

Automated deduction

Show $\Gamma \vdash B$. Two fundamental techniques.

1. Bottom-up methods:
   $\langle A_1, A_2, \ldots \quad \ldots \quad B\rangle$
   (Γ)
   Search in space of formulae to extend proof fragments.

Automated deduction

Show $\Gamma \vdash B$. Two fundamental techniques.

1. Bottom-up methods:

   $\langle A_1, A_2, \ldots \quad \ldots \quad B \rangle$
   (Γ)

   Search in space of formulae to extend proof fragments.

2. Top-down methods:

   $$\frac{\Delta_1 \vdash A_1 \qquad \ldots \qquad \Delta_k \vdash A_k}{\Gamma \vdash B}$$

Automated deduction

Show $\Gamma \vdash B$. Two fundamental techniques.

1. Bottom-up methods:

   $\langle A_1, A_2, \ldots \quad \ldots \quad B \rangle$
   (Γ)

   Search in space of formulae to extend proof fragments.

2. Top-down methods:

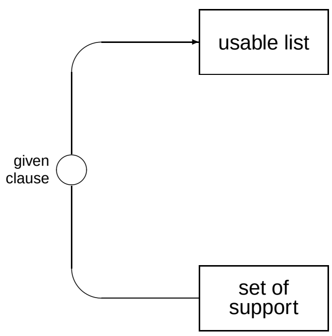
   $$\frac{\Delta_1 \vdash A_1 \qquad \ldots \qquad \Delta_k \vdash A_k}{\Gamma \vdash B}$$

   Search in space of sequents for provable subgoals.

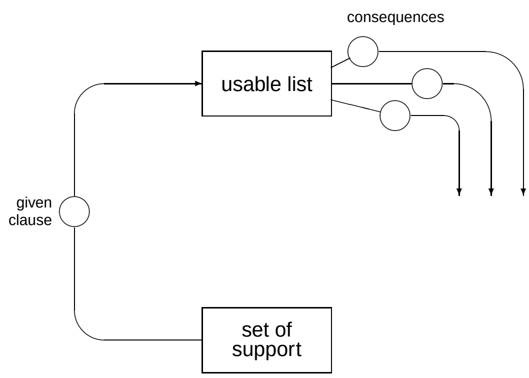
The Given Clause Loop

The Given Clause Loop

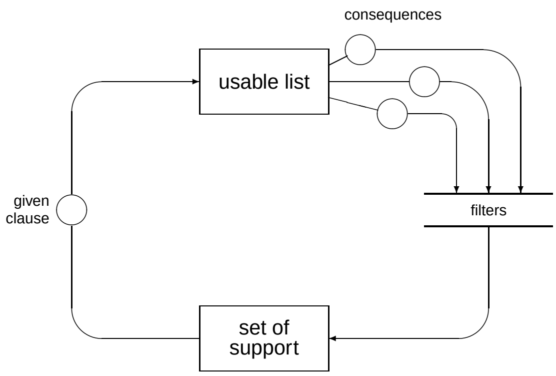
The Given Clause Loop



The Given Clause Loop



The Given Clause Loop



Application of Automatic Deduction

Application of Automatic Deduction

Software certification

First order provers now powerful enough to be used for software certification in industry

- SafeLogic (Sweden)
- Escher Technologies (UK) “Perfect Developer”
- NASA (USA) using SETHEO and other provers

Application of Automatic Deduction

Software certification

First order provers now powerful enough to be used for software certification in industry

- SafeLogic (Sweden)
- Escher Technologies (UK) “Perfect Developer”
- NASA (USA) using SETHEO and other provers

General technique

- Use e.g. Hoare to reduce to small proof obligations
- Prove these without human intervention
- Require extensions e.g. for numbers
- Most are easy, a few are hard

The Focus Problem (Wos)

The Focus Problem (Wos)

A difficulty

Many proof obligations have:

- Short and simple proofs
- Hundreds or thousands of (irrelevant) assumptions

The Focus Problem (Wos)

A difficulty

Many proof obligations have:

- Short and simple proofs
- Hundreds or thousands of (irrelevant) assumptions

How to choose the relevant ones?

Fundamental open problem in theorem proving

The Focus Problem (Wos)

A difficulty

Many proof obligations have:

- Short and simple proofs
- Hundreds or thousands of (irrelevant) assumptions

How to choose the relevant ones?

Fundamental open problem in theorem proving

Sources:

- John Harrison (INTEL)
- David Crocker (Escher)
- Bernd Fischer (NASA)

Example (not from verification)

Example (not from verification)

Virtual set theory

Example (not from verification)

Virtual set theory

- Simple language (4 predicates, 7 function symbols)
- 33 axioms
- Formulated without equality

Example (not from verification)

Virtual set theory

- Simple language (4 predicates, 7 function symbols)
- 33 axioms
- Formulated without equality
- Require many trivial theorems
  - $\cap$ and $\cup$ idempotent, commutative, associative
  - set equality is transitive
  - $\emptyset \cup x = x$
  - etc.

Example (not from verification)

Virtual set theory

- Simple language (4 predicates, 7 function symbols)
- 33 axioms
- Formulated without equality
- Require many trivial theorems
  - $\cap$ and $\cup$ idempotent, commutative, associative
  - set equality is transitive
  - $\emptyset \cup x = x$
  - etc.

Exhibits focus problem

Simple examples e.g. $x \cap y = y \cap x$ too hard for OTTER

Results

plain OTTER without any guidance

topic focus OTTER with term weighting to make it prefer clauses about $\cap$ to clauses about $\cup$ or $\emptyset$ etc

formula focus OTTER with topic focus plus a weighting scheme to make it prefer clauses containing actual subterms of the goal

	$x \cap y = y \cap x$			$x \cap y \subseteq y \cap x$		
	plain	**topic**	**formula**	**plain**	**topic**	**formula**
iterations	—	—	128	766	350	66
clauses generated	—	—	1729	12742	6593	1018
time (seconds)	—	—	0.2	4.4	1.3	0.1

False Preference Strategy

False Preference Strategy

1. Suppose S is a set of clauses all true in a model M.

False Preference Strategy

1. Suppose S is a set of clauses all true in a model M.
2. Suppose c is a clause inconsistent with S.

False Preference Strategy

1. Suppose S is a set of clauses all true in a model M.
2. Suppose c is a clause inconsistent with S.
3. Then there are proofs of a contradiction from S and c together, and c occurs in all of them.

False Preference Strategy

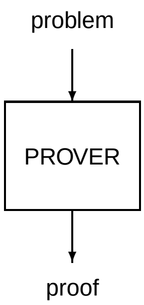
1. Suppose S is a set of clauses all true in a model M.
2. Suppose c is a clause inconsistent with S.
3. Then there are proofs of a contradiction from S and c together, and c occurs in all of them.
4. So if M makes most of the usable list true, and c is in the set of support, it is good to take c as the next given clause.

False Preference Strategy

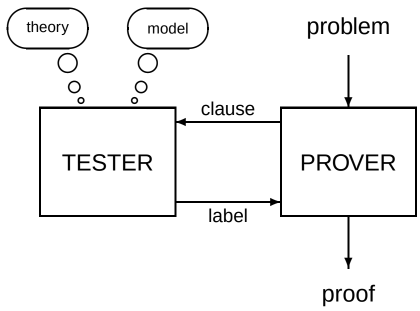
1. Suppose S is a set of clauses all true in a model M.
2. Suppose c is a clause inconsistent with S.
3. Then there are proofs of a contradiction from S and c together, and c occurs in all of them.
4. So if M makes most of the usable list true, and c is in the set of support, it is good to take c as the next given clause.

In fact we don't know whether c is inconsistent with S, but if we choose a clause that is false in M we have a better chance than if we choose arbitrarily.

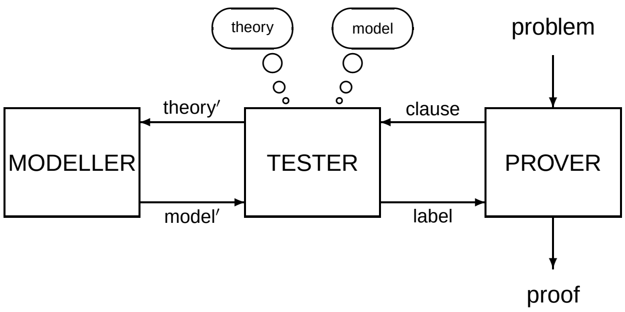
SCOTT Architecture



SCOTT Architecture



SCOTT Architecture



History of SCOTT

History of SCOTT

First phase 1991–3

Single model used to constrain the logical inferences

- Incomplete: many proofs missed
- Fragile: sensitive to the order of clauses

History of SCOTT

First phase 1991–3

Single model used to constrain the logical inferences

- Incomplete: many proofs missed
- Fragile: sensitive to the order of clauses

Second phase 1997–2001

Multiple models used for false preference strategy

- Complete and relatively robust
- Very slow: often minutes for a few clauses

History of SCOTT

First phase 1991–3

Single model used to constrain the logical inferences

- Incomplete: many proofs missed
- Fragile: sensitive to the order of clauses

Second phase 1997–2001

Multiple models used for false preference strategy

- Complete and relatively robust
- Very slow: often minutes for a few clauses

Third phase 2003–2004

Single approximate model instead of many exact ones.

Soft Constraints

Soft Constraints

- Hard constraints must hold: define what counts as a solution

Soft Constraints

- Hard constraints must hold: define what counts as a solution
- Soft constraints may fail: define what is a *good* solution

Soft Constraints

- Hard constraints must hold: define what counts as a solution
- Soft constraints may fail: define what is a *good* solution
- MAX-CSP: maximise the number of soft constraints satisfied

Soft Constraints

- Hard constraints must hold: define what counts as a solution
- Soft constraints may fail: define what is a *good* solution
- MAX-CSP: maximise the number of soft constraints satisfied
- For SCOTT, treat any initially usable clauses as hard and all later activated clauses as soft

Soft Constraints

- Hard constraints must hold: define what counts as a solution
- Soft constraints may fail: define what is a *good* solution
- MAX-CSP: maximise the number of soft constraints satisfied
- For SCOTT, treat any initially usable clauses as hard and all later activated clauses as soft
- Gives an *approximate* model of *all* of the usable list rather than an *exact* model of just *part* of it

Soft Constraints

- Hard constraints must hold: define what counts as a solution
- Soft constraints may fail: define what is a *good* solution
- MAX-CSP: maximise the number of soft constraints satisfied
- For SCOTT, treat any initially usable clauses as hard and all later activated clauses as soft
- Gives an *approximate* model of *all* of the usable list rather than an *exact* model of just *part* of it
- Gains speed because only one model, and robustness because all usable clauses modelled together regardless of activation order

Implementation

Implementation

Underlying theorem prover OTTER (McCune)

Implementation

Underlying theorem prover OTTER (McCune)

- Existing high-performance prover

Implementation

Underlying theorem prover OTTER (McCune)

- Existing high-performance prover
- Changed as little as possible

Implementation

Underlying theorem prover OTTER (McCune)

- Existing high-performance prover
- Changed as little as possible

Constraint solver FINDER

Implementation

Underlying theorem prover OTTER (McCune)

- Existing high-performance prover
- Changed as little as possible

Constraint solver FINDER

- Reasonably fast for finding small models

Implementation

Underlying theorem prover OTTER (McCune)

- Existing high-performance prover
- Changed as little as possible

Constraint solver FINDER

- Reasonably fast for finding small models
- Depth-first branch and bound for soft constraints

Implementation

Underlying theorem prover OTTER (McCune)

- Existing high-performance prover
- Changed as little as possible

Constraint solver FINDER

- Reasonably fast for finding small models
- Depth-first branch and bound for soft constraints
- Cutoff to force termination (and speed)

Implementation

Underlying theorem prover OTTER (McCune)

- Existing high-performance prover
- Changed as little as possible

Constraint solver FINDER

- Reasonably fast for finding small models
- Depth-first branch and bound for soft constraints
- Cutoff to force termination (and speed)

Big issue: tradeoff

Implementation

Underlying theorem prover OTTER (McCune)

- Existing high-performance prover
- Changed as little as possible

Constraint solver FINDER

- Reasonably fast for finding small models
- Depth-first branch and bound for soft constraints
- Cutoff to force termination (and speed)

Big issue: tradeoff

- Model search versus proof search

Implementation

Underlying theorem prover OTTER (McCune)

- Existing high-performance prover
- Changed as little as possible

Constraint solver FINDER

- Reasonably fast for finding small models
- Depth-first branch and bound for soft constraints
- Cutoff to force termination (and speed)

Big issue: tradeoff

- Model search versus proof search
- Time in model generator versus quality of guidance

Example 1: GRP200-4

Example 1: GRP200-4

Problem

In a loop, $\forall xyz[(x(yz))x = (xy)(zx)]$ implies $((ab)c)b = a(b(cb))$

Example 1: GRP200-4

Problem

In a loop, $\forall xyz[(x(yz))x = (xy)(zx)]$ implies $((ab)c)b = a(b(cb))$

Statistics

	with models	without
Input clauses	20	20
Clauses generated	3149	397803
Clauses kept	1649	30179
Clauses given	57	587
Clauses in proof	36	—
Models generated	13	0
Time	4.28 sec	600.65 sec

Example 2: FLD049-4

Example 2: FLD049-4

Problem

In a field, for nonzero b and d, if $ab^{-1} = cd^{-1}$ then $ad = bc$

Example 2: FLD049-4

Problem

In a field, for nonzero b and d, if $ab^{-1} = cd^{-1}$ then $ad = bc$

Statistics

	with models	without models
Input clauses	38 (61)	38 (61)
Clauses generated	56831	129125
Clauses kept	27071	21709
Clauses given	184	249
Clauses in proof	25	25
Models generated	142	0
Time	417.44 sec	3.01 sec

Results on set theory problem

	$x \cap y = y \cap x$			$x \cap y \subseteq y \cap x$		
without guidance	**plain**	**topic**	**formula**	**plain**	**topic**	**formula**
iterations	—	—	128	766	350	66
clauses generated	—	—	1729	12742	6593	1018
time (seconds)	—	—	0.2	4.4	1.3	0.1

	$x \cap y = y \cap x$			$x \cap y \subseteq y \cap x$		
with guidance	**plain**	**topic**	**formula**	**plain**	**topic**	**formula**
iterations	—	3009	169	496	241	85
clauses generated	—	80239	2430	9576	3520	1426
time (seconds)	—	90.0	3.2	6.7	2.4	0.6

Conclusions and Future Work

Conclusions and Future Work

Achieved:

Conclusions and Future Work

Achieved:

- New theorem prover guided by soft models

Conclusions and Future Work

Achieved:

- New theorem prover guided by soft models
  - More robust than SCOTT-1
  - Faster than SCOTT-2 – SCOTT-5

Conclusions and Future Work

Achieved:

- New theorem prover guided by soft models
  - More robust than SCOTT-1
  - Faster than SCOTT-2 – SCOTT-5
- Reasonable performance in CASC 2004

Conclusions and Future Work

Achieved:

- New theorem prover guided by soft models
  - More robust than SCOTT-1
  - Faster than SCOTT-2 – SCOTT-5
- Reasonable performance in CASC 2004

But:

Conclusions and Future Work

Achieved:

- New theorem prover guided by soft models
  - More robust than SCOTT-1
  - Faster than SCOTT-2 – SCOTT-5
- Reasonable performance in CASC 2004

But:

- Syntax/semantics tradeoff still a big issue

Conclusions and Future Work

Achieved:

- New theorem prover guided by soft models
  - More robust than SCOTT-1
  - Faster than SCOTT-2 – SCOTT-5
- Reasonable performance in CASC 2004

But:

- Syntax/semantics tradeoff still a big issue
- Improvement not yet dramatic

Conclusions and Future Work

Achieved:

- New theorem prover guided by soft models
  - More robust than SCOTT-1
  - Faster than SCOTT-2 – SCOTT-5
- Reasonable performance in CASC 2004

But:

- Syntax/semantics tradeoff still a big issue
- Improvement not yet dramatic

To Do:

Conclusions and Future Work

Achieved:

- New theorem prover guided by soft models
  - More robust than SCOTT-1
  - Faster than SCOTT-2 – SCOTT-5
- Reasonable performance in CASC 2004

But:

- Syntax/semantics tradeoff still a big issue
- Improvement not yet dramatic

To Do:

- Large cardinality soft constraints (local search?)

Conclusions and Future Work

Achieved:

- New theorem prover guided by soft models
  - More robust than SCOTT-1
  - Faster than SCOTT-2 – SCOTT-5
- Reasonable performance in CASC 2004

But:

- Syntax/semantics tradeoff still a big issue
- Improvement not yet dramatic

To Do:

- Large cardinality soft constraints (local search?)
- Better underlying prover (Vampire?)

Conclusions and Future Work

Achieved:

- New theorem prover guided by soft models
  - More robust than SCOTT-1
  - Faster than SCOTT-2 – SCOTT-5
- Reasonable performance in CASC 2004

But:

- Syntax/semantics tradeoff still a big issue
- Improvement not yet dramatic

To Do:

- Large cardinality soft constraints (local search?)
- Better underlying prover (Vampire?)
- Applications (software certification?)