6 Implementing Saturation Procedures

Problem:

Refutational completeness is nice in theory, but ...

...it guarantees only that proofs will be found eventually, not that they will be found quickly.

Even though orderings and selection functions reduce the number of possible inferences, the search space problem is enormous.

First-order provers "look for a needle in a haystack": It may be necessary to make some millions of inferences to find a proof that is only a few dozens of steps long.

Coping with Large Sets of Formulas

Consequently:

- We must deal with large sets of formulas.
- We must use efficient techniques to find formulas that can be used as partners in an inference.
- We must simplify/eliminate as many formulas as possible.
- We must use efficient techniques to check whether a formula can be simplified/eliminated.

Note:

Often there are several competing implementation techniques.

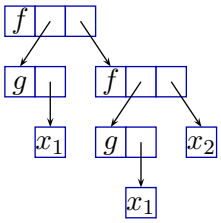
Design decisions are not independent of each other.

Design decisions are not independent of the particular class of problems we want to solve. (FOL without equality/FOL with equality/unit equations, size of the signature, special algebraic properties like AC, etc.)

6.1 Term Representations

The obvious data structure for terms: Trees

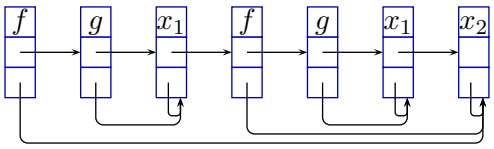
$f(g(x_1), f(g(x_1), x_2))$



optionally: (full) sharing

An alternative: Flatterms

$f(g(x_1), f(g(x_1), x_2))$



need more memory;
but: better suited for preorder term traversal and easier memory management.

6.2 Index Data Structures

Problem:

For a term t, we want to find all terms s such that

- s is an instance of t,
- s is a generalization of t (i. e., t is an instance of s),
- s and t are unifiable,
- s is a generalization of some subterm of t,
- ...

Requirements:

fast insertion,

fast deletion,

fast retrieval,

small memory consumption.

Note: In applications like functional or logic programming, the requirements are different (insertion and deletion are much less important).

Many different approaches:

- Path indexing
- Discrimination trees
- Substitution trees
- Context trees
- Feature vector indexing
- ...

Perfect filtering:

The indexing technique returns exactly those terms satisfying the query.

Imperfect filtering:

The indexing technique returns some superset of the set of all terms satisfying the query.

Retrieval operations must be followed by an additional check, but the index can often be implemented more efficiently.

Frequently: All occurrences of variables are treated as different variables.

Path Indexing

Path indexing:

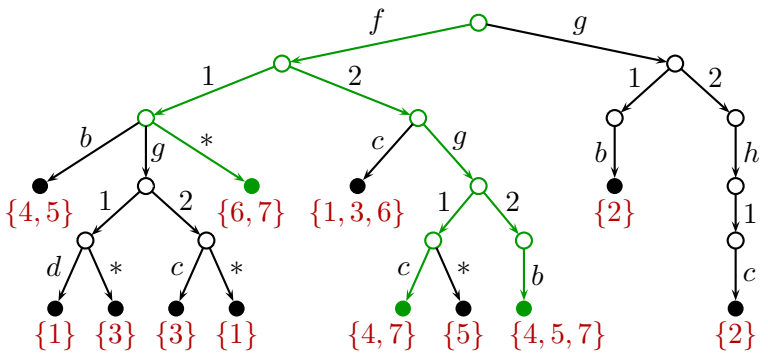
Paths of terms are encoded in a trie ("retrieval tree").

A star $*$ represents arbitrary variables.

Example: Paths of $f(g(*, b), *)$: $f.1.g.1.*$
$f.1.g.2.b$
$f.2.*$

Each leaf of the trie contains the set of (pointers to) all terms that contain the respective path.

Example: Path index for $\{f(g(d, *), c),\ g(b, h(c)),\ f(g(*, c), c),\ f(b, g(c, b)),\ f(b, g(*, b)),\ f(*, c),\ f(*, g(c, b))\}$



Advantages:

Uses little space.
No backtracking for retrieval.
Efficient insertion and deletion.
Good for finding instances, also usable for finding generalizations.

Disadvantages:

Retrieval requires combining intermediate results for all paths.

Discrimination Trees

Discrimination trees:

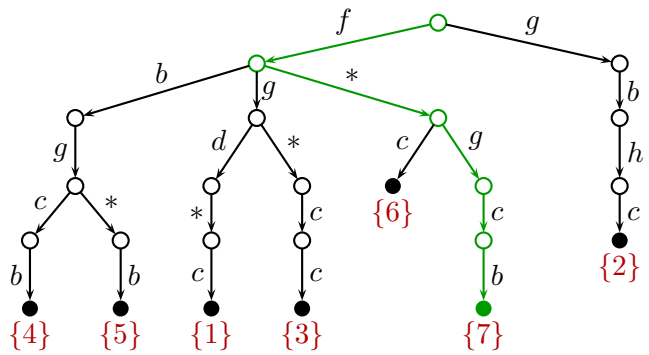
Preorder traversals of terms are encoded in a trie.

A star $*$ represents arbitrary variables.

Example: String of $f(g(*, b), *)$: $f.g.*.b.*$

Each leaf of the trie contains (a pointer to) the term that is represented by the path.

Example: Discrimination tree for $\{f(g(d, *), c)$, $g(b, h(c))$, $f(g(*, c), c)$, $f(b, g(c, b))$, $f(b, g(*, b))$, $f(*, c)$, $f(*, g(c, b))\}$



Advantages:

Each leaf yields one term, hence retrieval does not require intersections of intermediate results for all paths.

Good for finding generalizations, not so good for finding instances.

Disadvantages:

Uses more storage than path indexing (due to less sharing).

Uses still more storage, if jump lists are maintained to speed up the search for instances or unifiable terms.

Feature Vector Indexing

Goal:

C' is subsumed by C if $C' = C\sigma \vee D$.

Find all clauses C' for a given C or vice versa.

If C' is subsumed by C, then

- C' contains at least as many literals as C.
- C' contains at least as many positive literals as C.
- C' contains at least as many negative literals as C.
- C' contains at least as many function symbols as C.
- C' contains at least as many occurrences of f as C.
- C' contains at least as many occurrences of f in negative literals as C.
- the deepest occurrence of f in C' is at least as deep as in C.
- ...

Idea:

Select a list of these "features".

Compute the "feature vector" (a list of natural numbers) for each clause and store it in a trie.

When searching for a subsuming clause: Traverse the trie, check all clauses for which all features are smaller or equal. (Stop if a subsuming clause is found.)

When searching for subsumed clauses: Traverse the trie, check all clauses for which all features are larger or equal.

Advantages:

Works on the clause level, rather than on the term level.

Specialized for subsumption testing.

Disadvantages:

Needs to be complemented by other index structure for other operations.

Literature

R. Sekar, I. V. Ramakrishnan, and Andrei Voronkov: Term Indexing, Ch. 26 in Robinson and Voronkov (eds.), *Handbook of Automated Reasoning, Vol. II*, Elsevier, 2001.

Stephan Schulz: Simple and Efficient Clause Subsumption with Feature Vector Indexing, in Bonacina and Stickel (eds.), *Automated Reasoning and Mathematics*, LNCS 7788, Springer, 2013.

Christoph Weidenbach: Combining Superposition, Sorts and Splitting, Ch. 27 in Robinson and Voronkov (eds.), *Handbook of Automated Reasoning, Vol. II*, Elsevier, 2001.

7 Outlook

7.1 Satisfiability Modulo Theories (SMT)

CDCL checks satisfiability of propositional formulas.

CDCL can also be used for ground first-order formulas without equality:

Ground first-order atoms are treated like propositional variables.

Truth values of $P(a), Q(a), Q(f(a))$ are independent.

For ground formulas with equality, independence is lost:

If $b \approx c$ is true, then $f(b) \approx f(c)$ must also be true.

Similarly for other theories, e. g. linear arithmetic: $b > 5$ implies $b > 3$.

We can still use CDCL, but we must combine it with a decision procedure for the theory part T:

$M \models_T C$: M and the theory axioms T entail C.

New CDCL rules:

T-Propagate:

$M \parallel N \;\Rightarrow_{\text{CDCL(T)}}\; M\ L \parallel N$

if $M \models_T L$ where L is undefined in M and L or $\overline{L}$ occurs in N.

T-Learn:

$M \parallel N \;\Rightarrow_{\text{CDCL(T)}}\; M \parallel N \cup \{C\}$

if $N \models_T C$ and each atom of C occurs in N or M.

T-Backjump:

$M\ L^{\mathrm{d}}\ M' \parallel N \cup \{C\} \;\Rightarrow_{\text{CDCL(T)}}\; M\ L' \parallel N \cup \{C\}$

if $M\ L^{\mathrm{d}}\ M' \models \neg C$
and there is some "backjump clause" $C' \vee L'$ such that
$N \cup \{C\} \models_T C' \vee L'$ and $M \models \neg C'$,
L' is undefined under M, and
L' or $\overline{L'}$ occurs in N or in $M\ L^{\mathrm{d}}\ M'$.

7.2 Sorted Logics

So far, we have considered only unsorted first-order logic.

In practice, one often considers many-sorted logics:

$read/2$ becomes $read : array \times nat \to data.$

$write/3$ becomes $write : array \times nat \times data \to array.$

Variables: $x : data$

Only one declaration per function/predicate/variable symbol.

All terms, atoms, substitutions must be well-sorted.

Algebras:

Instead of universe $U_{\mathcal{A}}$, one set per sort: $array_{\mathcal{A}}$, $nat_{\mathcal{A}}$.

Interpretations of function and predicate symbols correspond to their declarations:

$read_{\mathcal{A}} : array_{\mathcal{A}} \times nat_{\mathcal{A}} \to data_{\mathcal{A}}$

Proof theory, calculi, etc.:

Essentially as in the unsorted case.

More difficult:

Subsorts

Overloading

7.3 Splitting

Tableau-like rule within resolution to eliminate variable-disjoint (positive) disjunctions:

$$\frac{N \cup \{C_1 \vee C_2\}}{N \cup \{C_1\} \quad | \quad N \cup \{C_2\}}$$

if $\mathrm{var}(C_1) \cap \mathrm{var}(C_2) = \emptyset$.

Split clauses are smaller and more likely to be usable for simplification.

Splitting tree is explored using intelligent backtracking.

Improvement:

Use a CDCL solver to manage the selection of split clauses.

$\Rightarrow$ AVATAR.

7.4 Integrating Theories into Resolution

Certain kinds of axioms are

important in practice,

but difficult for theorem provers.

Most important case: equality

but also: orderings, (associativity and) commutativity, ...

Idea: Combine ordered resolution and critical pair computation.

Superposition (ground case):

$$\frac{D' \vee t \approx t' \qquad C' \vee s[t] \approx s'}{D' \vee C' \vee s[t'] \approx s'}$$

Superposition (non-ground case):

$$\frac{D' \vee t \approx t' \qquad C' \vee s[u] \approx s'}{(D' \vee C' \vee s[t'] \approx s')\sigma}$$

where $\sigma = \mathrm{mgu}(t, u)$ and u is not a variable.

Advantages:

No variable overlaps (as in KB-completion).

Stronger ordering restrictions:
Only overlaps of (strictly) maximal sides of (strictly) maximal literals are required.

Stronger redundancy criteria.

Similarly for orderings:

Ordered chaining:

$$\frac{D' \vee t' < t \qquad C' \vee s < s'}{(D' \vee C' \vee t' < s')\sigma}$$

where σ is a most general unifier of t and s.

Integrating other theories:

Black box:

Use external decision procedure.

Easy, but works only under certain restrictions.

White box:

Integrate using specialized inference rules and theory unification.

Hard work.

Often: integrating more theory axioms is better.

7.5 Higher-Order Logics

What's new if we switch to higher-order logics?

Applied variables: $x\ b$.

Partially applied functions: $\mathit{times}\ z$.

Lambda-expressions with $\alpha\beta\eta$-conversion: $(\lambda x.\ f\ (x\ b)\ c)\ (\lambda y.\ d) = f\ d\ c$.

Embedded booleans: $(\lambda x.\,\text{if } x \text{ then } b \text{ else } c)\ (p \vee q)$

Problems:

Orderings cannot have all desired compatibility properties.
$\Rightarrow$ additional inferences.

Most general unifiers need not exist anymore.
$\Rightarrow$ interleave enumeration of unifiers and computation of inferences.

CNF transformation by preprocessing is no longer sufficient.
$\Rightarrow$ need calculus with embedded clausification.

Contents