Term Indexing Techniques in OCaml

Final Presentation

Student: Simon Legner
Supervisor: Sarah Winkler, MSc.

Computational Logic
Institute of Computer Science
University of Innsbruck

November 16, 2010

Bachelor Project

Goals of the Bachelor Project

- implement a term indexing library in OCaml
  - **Discrimination Trees**
  - **Code Trees**
  - **Substitution Trees**
- run **performance** tests

Bachelor Project

Goals of the Bachelor Project

- implement a term indexing library in OCaml
  - **Discrimination Trees**
  - **Code Trees**
  - **Substitution Trees**
- run **performance** tests

Goals of this Talk

- What is term indexing?
- Why is term indexing necessary?
- How do the implemented techniques work?
- Which technique works best?

Preliminaries

Definition (Terms)

$$\mathbf{t} ::= x \mid \mathsf{c} \mid \mathsf{f}(\mathbf{t_1}, \dots, \mathbf{t_n})$$

Definition (Substitutions)

- substitution σ is given by bindings $\{x_1 \mapsto t_1, \dots, x_n \mapsto t_n\}$
- $t\sigma$ denotes application of σ to term t

Preliminaries

Definition (Terms)

$$\mathbf{t} ::= x \mid \mathsf{c} \mid \mathsf{f}(\mathbf{t_1}, \ldots, \mathbf{t_n})$$

Definition (Substitutions)

- substitution σ is given by bindings $\{x_1 \mapsto t_1, \ldots, x_n \mapsto t_n\}$
- $t\sigma$ denotes application of σ to term t

Definition (Relations between Terms)

- $\mathsf{UNIF}(t, q) \iff \exists\sigma\ t\sigma = q\sigma$
- $\mathsf{INST}(t, q) \iff \exists\sigma\ t = q\sigma$
- $\mathsf{GEN}(t, q) \iff \exists\sigma\ t\sigma = q$
- $\mathsf{VAR}(t, q) \iff \exists\sigma\ t\sigma = q$ and σ is a renaming

Motivation

Why is Term Indexing necessary?

fast retrieval of terms from a huge set is required for:

- selecting candidate clauses in logic programming
- finding applicable rules in Knuth-Bendix completion
- automated reasoning systems

Motivation

Why is Term Indexing necessary?

fast retrieval of terms from a huge set is required for:

- selecting candidate clauses in logic programming
- finding applicable rules in Knuth-Bendix completion
- automated reasoning systems

Program Degeneration

> *"...after a few CPU minutes of use ..., a reasoning program typically made deductions at less than 1% of its ability at the beginning of a run." [Wos, 1992]*

Indexing

Definition (Indexing)

Building a **data structure** on top of a set of data to **speedup retrieval** of data.

Examples (Indexing)

B-trees on relational databases or indexes at the end of textbooks

Term Indexing

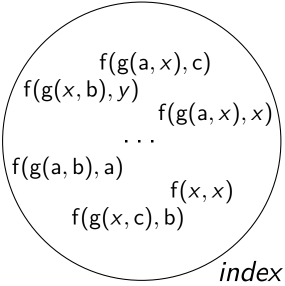
Definition (Term Indexing Problem)

Given a set of terms, a **relation** between two terms and a **query term**, retrieve all **candidate terms** from the set so that the relation holds.

Term Indexing

Definition (Term Indexing Problem)

Given a set of terms, a **relation** between two terms and a **query term**, retrieve all **candidate terms** from the set so that the relation holds.



$f(g(a, x), c)$

$f(g(x, b), y)$

$f(g(a, x), x)$

$\dots$

$f(g(a, b), a)$

$f(x, x)$

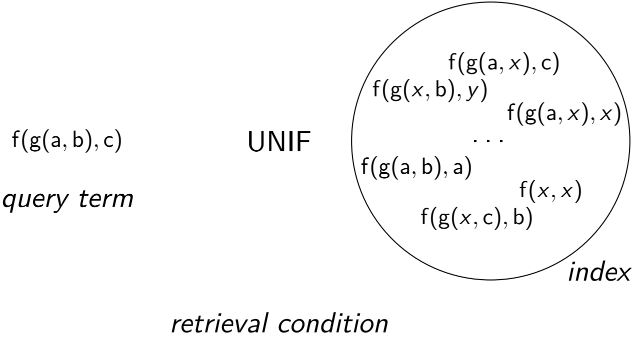
$f(g(x, c), b)$

index

Term Indexing

Definition (Term Indexing Problem)

Given a set of terms, a **relation** between two terms and a **query term**, retrieve all **candidate terms** from the set so that the relation holds.



$\mathsf{f}(\mathsf{g}(\mathsf{a},\mathsf{b}),\mathsf{c})$

query term

UNIF

$\mathsf{f}(\mathsf{g}(\mathsf{a},x),\mathsf{c})$

$\mathsf{f}(\mathsf{g}(x,\mathsf{b}),y)$

$\mathsf{f}(\mathsf{g}(\mathsf{a},x),x)$

$\cdots$

$\mathsf{f}(\mathsf{g}(\mathsf{a},\mathsf{b}),\mathsf{a})$

$\mathsf{f}(x,x)$

$\mathsf{f}(\mathsf{g}(x,\mathsf{c}),\mathsf{b})$

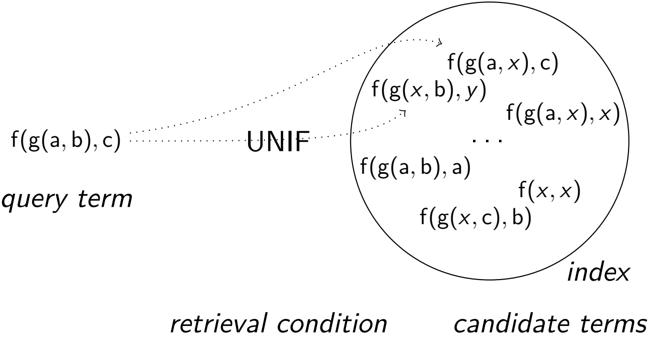
index

retrieval condition

Term Indexing

Definition (Term Indexing Problem)

Given a set of terms, a **relation** between two terms and a **query term**, retrieve all **candidate terms** from the set so that the relation holds.



Term Indexing

Operations on Term Indexes

- index maintenance
  - `val initialize: Term.t list -> t`
  - `val insert: Term.t -> t -> t`
  - `val remove: Term.t -> t -> t`
- retrieval
  - `val retrieve_generalizations: Term.t -> t -> Term.t list`
  - `val retrieve_instances: Term.t -> t -> Term.t list`
  - `val retrieve_unifiable_terms: Term.t -> t -> Term.t list`
  - `val retrieve_variants: Term.t -> t -> Term.t list`
- visualization
  - `val to_dot: t -> string`

Discrimination Tree Indexing

Idea: Index a *string representation* of terms in a *trie*.
Support for retrieval conditions GEN, INST, UNIF and VAR.

Discrimination Tree Indexing

Idea: Index a *string representation* of terms in a *trie*.
Support for retrieval conditions GEN, INST, UNIF and VAR.

Running Example

$\mathrm{f}(x, x)$ $\mathrm{f}(x, y)$

$\mathrm{g}(\mathrm{b}, \mathrm{a})$ $\mathrm{g}(\mathrm{b}, x)$

Discrimination Tree Indexing

Idea: Index a *string representation* of terms in a *trie*.
Support for retrieval conditions GEN, INST, UNIF and VAR.

Running Example

$$\mathsf{f}(x,x) \quad \mathsf{f}(x,y)$$

$$\mathsf{g}(\mathsf{b},\mathsf{a}) \quad \mathsf{g}(\mathsf{b},x)$$

P-Strings

$$\mathsf{f}\,.\,*\,.\,* \quad \mathsf{f}\,.\,*\,.*$$

$$\mathsf{g}\,.\,\mathsf{b}\,.\,\mathsf{a} \quad \mathsf{g}\,.\,\mathsf{b}\,.*$$

Discrimination Tree Indexing

Idea: Index a *string representation* of terms in a *trie*.
Support for retrieval conditions GEN, INST, UNIF and VAR.

Running Example

$\mathsf{f}(x, x) \quad \mathsf{f}(x, y)$

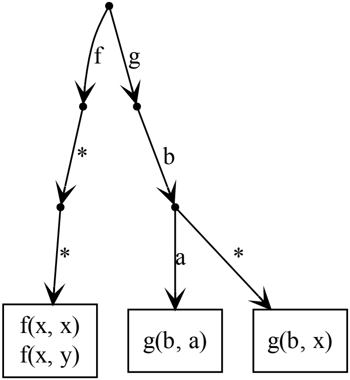
$\mathsf{g}(\mathsf{b}, \mathsf{a}) \quad \mathsf{g}(\mathsf{b}, x)$

P-Strings

$\mathsf{f} . * . * \quad \mathsf{f} . * . *$

$\mathsf{g} . \mathsf{b} . \mathsf{a} \quad \mathsf{g} . \mathsf{b} . *$

Discrimination Tree



Code Tree Indexing

Idea:

- Compile term into *instruction sequence*.
- Integrate sequences into *code tree*.
- *Evaluate* instructions during retrieval.

Support only for retrieval conditions GEN and VAR.

Code Tree Indexing

Idea:

- Compile term into *instruction sequence*.
- Integrate sequences into *code tree*.
- *Evaluate* instructions during retrieval.

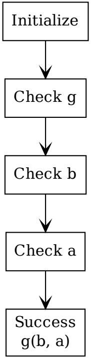
Support only for retrieval conditions GEN and VAR.

Instructions

$$\begin{aligned} \mathbf{i} ::= {} & \mathtt{Initialize}(\mathbf{i}) \mid \mathtt{Success} \mid \mathtt{Failure} \mid \\ & \mathtt{Check}(\mathsf{fs}, \mathbf{i}, \mathbf{i}) \mid \mathtt{Put}(\mathbf{i}, \mathbf{i}) \mid \mathtt{Compare}(v_1, v_2, \mathbf{i}, \mathbf{i}) \end{aligned}$$

Code Tree Indexing

Instruction Sequences



Initialize

Check g

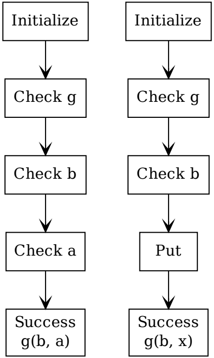
Check b

Check a

Success
g(b, a)

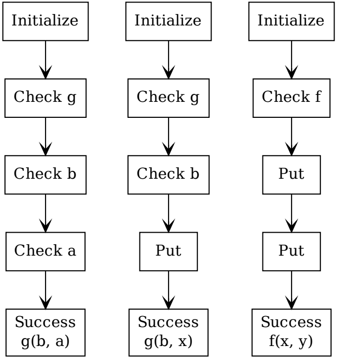
Code Tree Indexing

Instruction Sequences



Code Tree Indexing

Instruction Sequences



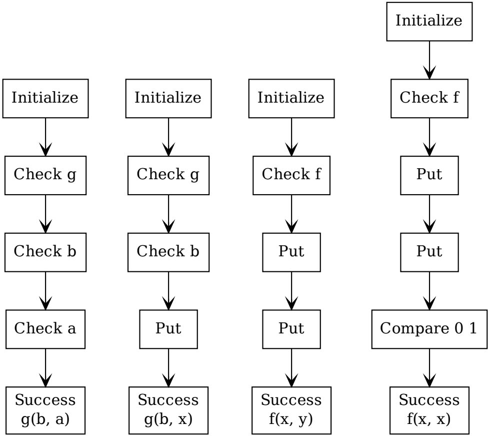
Initialize → Check g → Check b → Check a → Success g(b, a)

Initialize → Check g → Check b → Put → Success g(b, x)

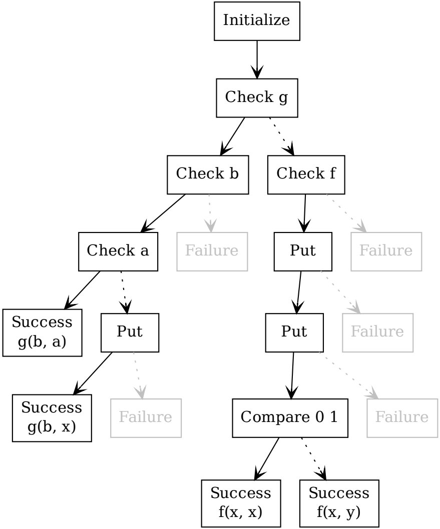
Initialize → Check f → Put → Put → Success f(x, y)

Code Tree Indexing

Instruction Sequences



Code Tree Indexing

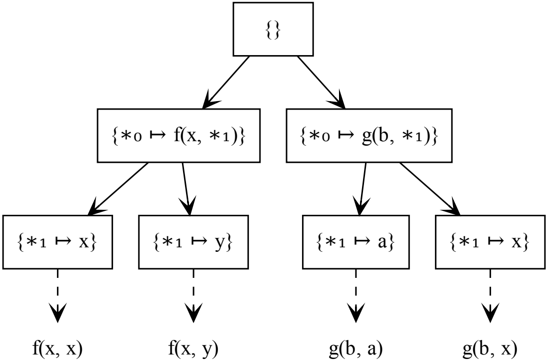


Substitution Tree Indexing

Idea: Represent terms as substitution sequences, collect in tree.
Support for retrieval conditions GEN, INST, UNIF and VAR.

Substitution Tree

$$*_0\,\{\}\,\{*_0 \mapsto \mathsf{f}(x, *_1)\}\,\{*_1 \mapsto y\} = \mathsf{f}(x, *_1)\,\{*_1 \mapsto y\} = \mathsf{f}(x, y)$$



Demonstration

Demonstration

```
(* functor allows arbitrary entries to be indexed *)
# module DT = DiscriminationTree.Make(struct
    type t = Term.t * string ;;
    let term = fst ;;
    let to_string = snd ;;
    let compare (t1, _) (t2, _) = Term.compare t1 t2 ;;
    end) ;;
module DT : sig (* ... *) end
```

Demonstration

```
(* functor allows arbitrary entries to be indexed *)
# module DT = DiscriminationTree.Make(struct
    type t = Term.t * string ;;
    let term = fst ;;
    let to_string = snd ;;
    let compare (t1, _) (t2, _) = Term.compare t1 t2 ;;
    end) ;;
module DT : sig (* ... *) end

# let tl = [x, "variable x"; f(x, a), "complex term"] ;;
val tl : (Term.t * string) list =
    [(x, "variable x"); (f(x, a), "complex term")]
```

Demonstration

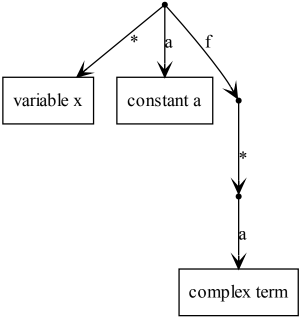
```
(* functor allows arbitrary entries to be indexed *)
# module DT = DiscriminationTree.Make(struct
    type t = Term.t * string ;;
    let term = fst ;;
    let to_string = snd ;;
    let compare (t1, _) (t2, _) = Term.compare t1 t2 ;;
    end) ;;
module DT : sig (* ... *) end

# let tl = [x, "variable x"; f(x, a), "complex term"] ;;
val tl : (Term.t * string) list =
    [(x, "variable x"); (f(x, a), "complex term")]

# let i = DT.init tl ;;
val i : DT.t = <abstr>
```

Demonstration

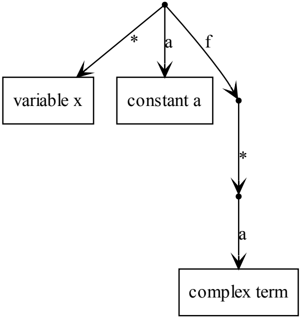
```
# let tl = [x, "variable x"; f(x, a), "complex term"] ;;
val tl : (Term.t * string) list =
    [(x, "variable x"); (f(x, a), "complex term")]

# let i = DT.init tl ;;
val i : DT.t = <abstr>

# let i' = DT.insert (a, "constant a") i ;;
val i' : DT.t = <abstr>
```

Demonstration

```
# print_string (DT.to_dot i') ;;
digraph {
    ...
    1 -> 2[label="*"];
    2[label="variable x"];
    ...
}
- : unit = ()
```



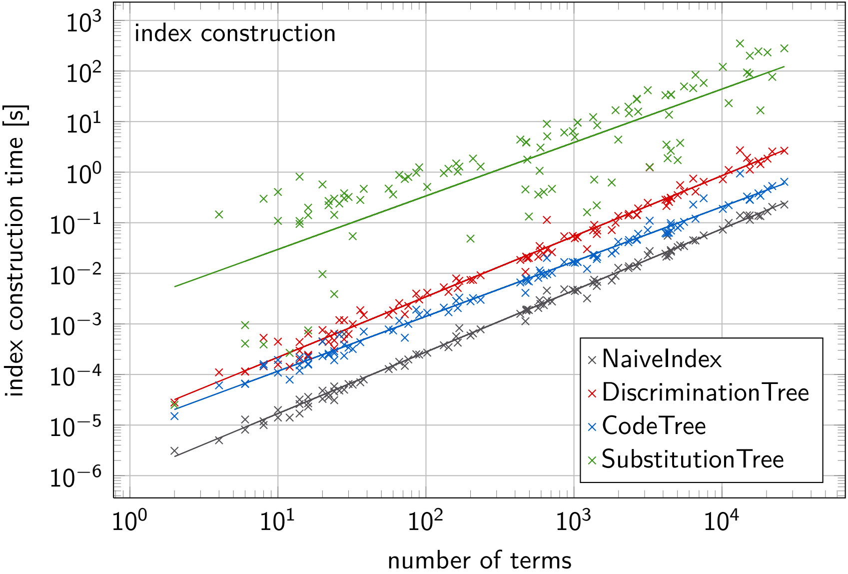
Demonstration



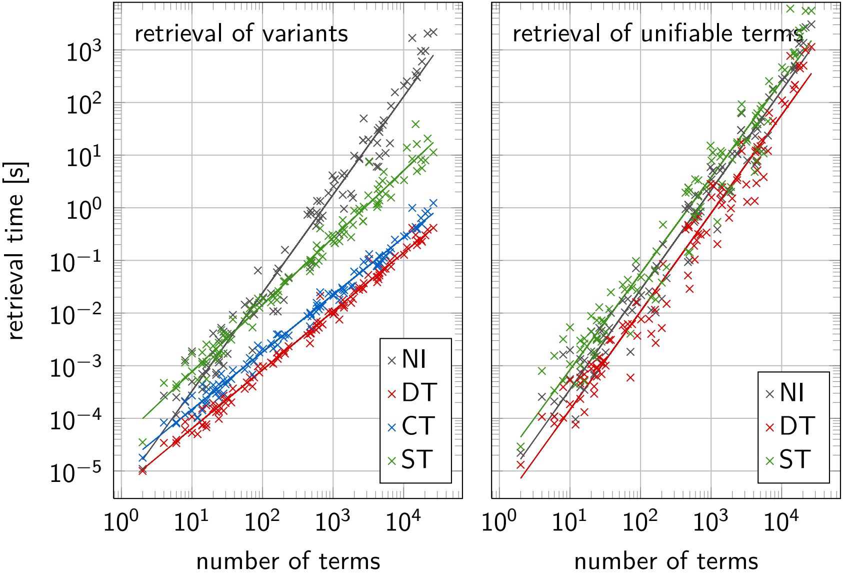
```
# print_string (DT.to_dot i') ;;
digraph {
    ...
    1 -> 2[label="*"];
    2[label="variable x"];
    ...
}
- : unit = ()
```

```
# DT.retrieve_generalizations a i' ;;
- : DT.entry list =
    [(a, "constant a"); (x, "variable x")]
```

Evaluation


index construction
index construction time [s]
10^{3}
10^{2}
10^{1}
10^{0}
10^{-1}
10^{-2}
10^{-3}
10^{-4}
10^{-5}
10^{-6}
10^{0}
10^{1}
10^{2}
10^{3}
10^{4}
number of terms
× NaiveIndex
× DiscriminationTree
× CodeTree
× SubstitutionTree

Evaluation


retrieval of variants
retrieval of unifiable terms
retrieval time [s]
number of terms
number of terms
10^3
10^2
10^1
10^0
10^{-1}
10^{-2}
10^{-3}
10^{-4}
10^{-5}
10^0
10^1
10^2
10^3
10^4
NI
DT
CT
ST
NI
DT
ST

Conclusion

Conclusion

employing term indexing results in performance gain

- Discrimination Trees
  - pros: easy to implement, fast retrieval
  - cons: nonlinearity lost
- Code Trees
  - pros: substitution factoring
  - cons: supports only GEN and VAR
- Substitution Trees
  - pros: compact structure
  - cons: slow, hard to implement

Conclusion

Conclusion

employing term indexing results in performance gain

- Discrimination Trees
  - pros: easy to implement, fast retrieval
  - cons: nonlinearity lost
- Code Trees
  - pros: substitution factoring
  - cons: supports only GEN and VAR
- Substitution Trees
  - pros: compact structure
  - cons: slow, hard to implement

Thank you for your attention!
Questions?