Term Indexing Techniques in OCaml

Initial presentation

Simon Legner

Computational Logic
Institute of Computer Science
University of Innsbruck

March 9, 2010

Term + Indexing

Term

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

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

Indexing

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

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

Term + Indexing

Term

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

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

Indexing

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

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

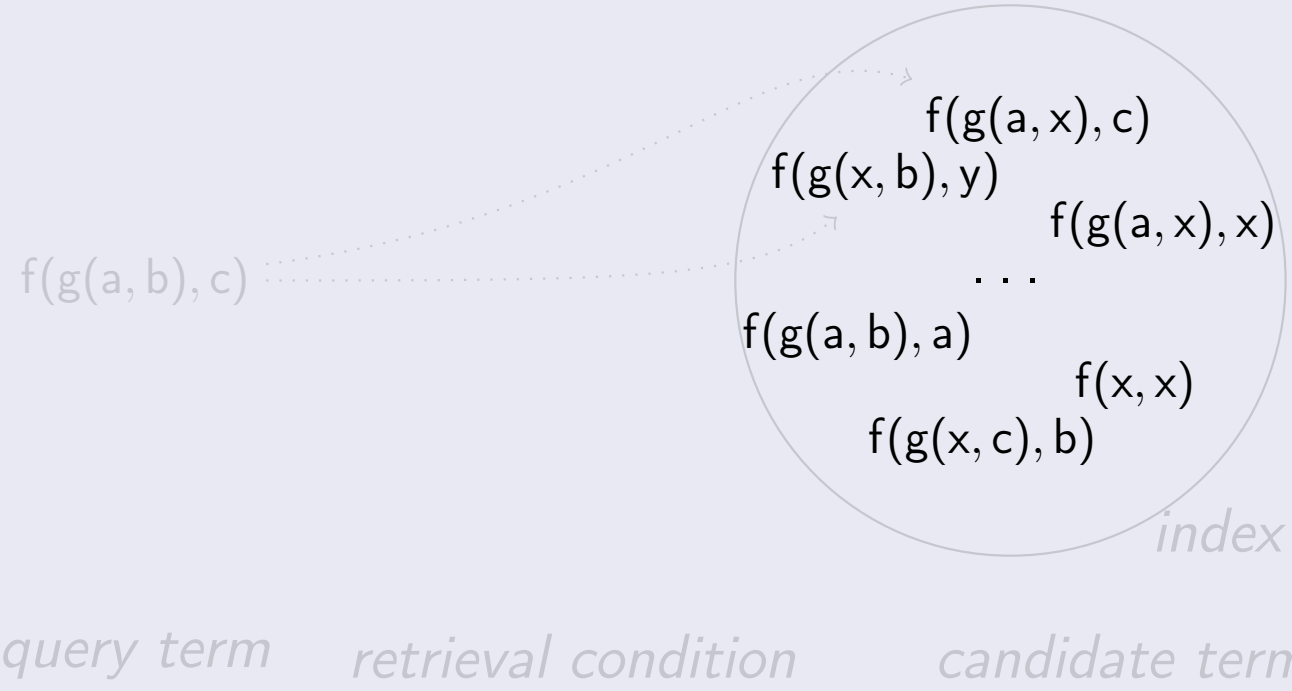
Why is term indexing necessary?

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

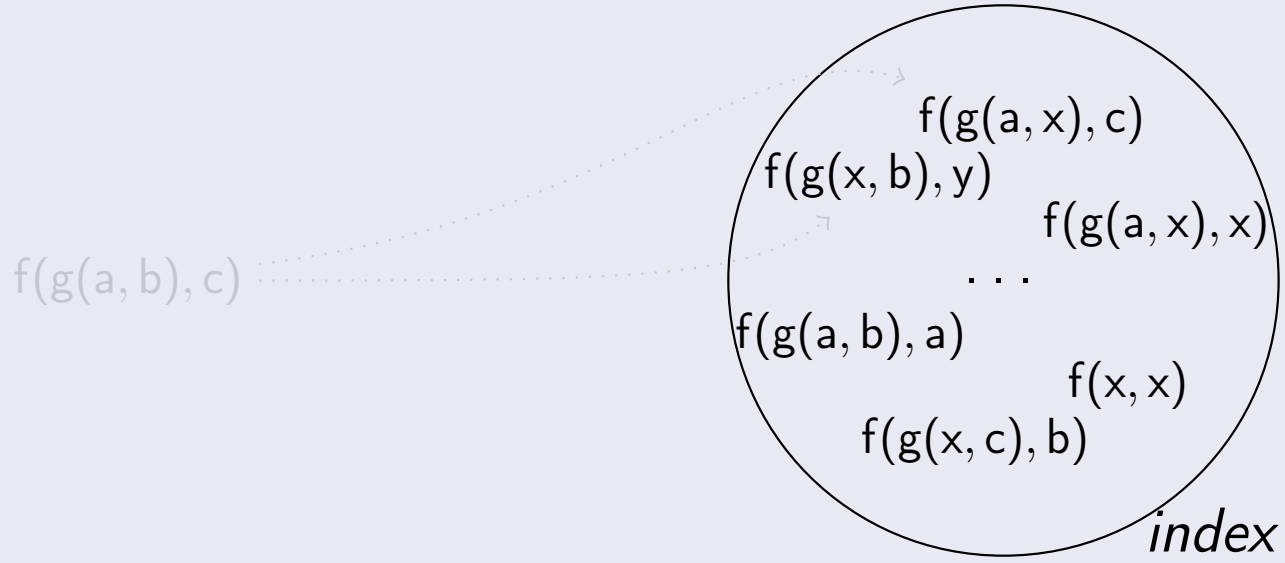
- selecting candidate clauses in logic programming
- finding applicable rules in Knuth-Bendix completion
- automated reasoning systems

slow: linear search
possible solution: term index

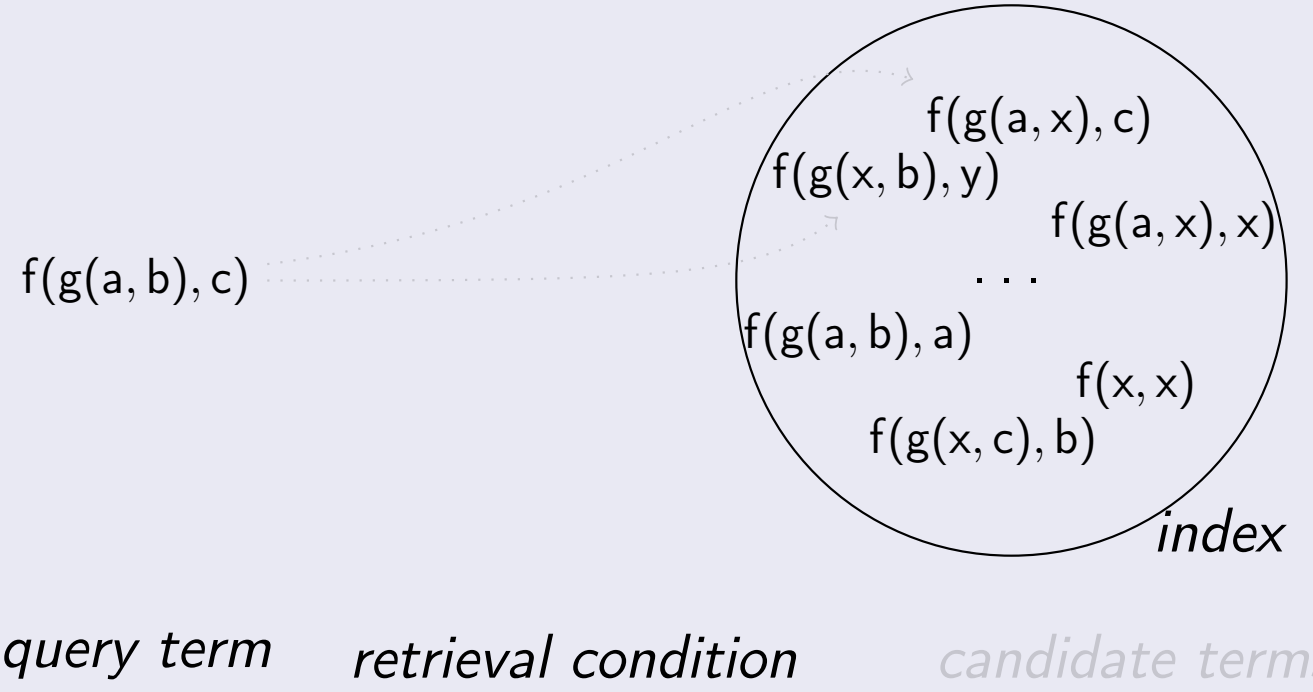
Principle of term indexing



Principle of term indexing

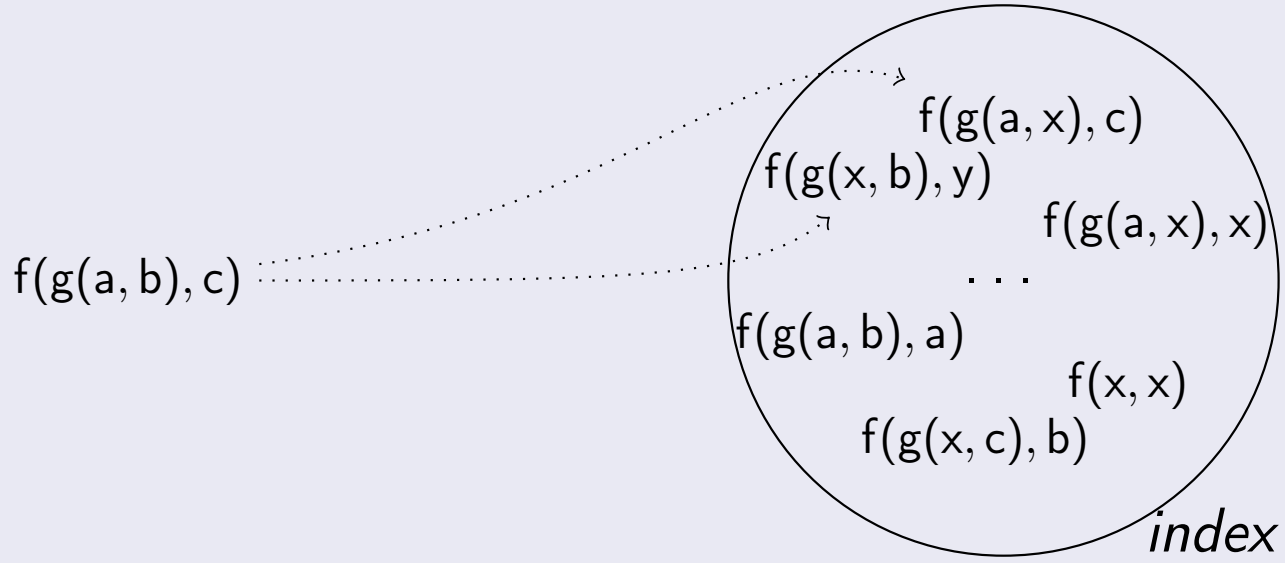


Principle of term indexing



query term *retrieval condition* *candidate terms*

Principle of term indexing



query term *retrieval condition* *candidate terms*

Relations between terms

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

Examples

- $\mathrm{UNIF}\left(\underline{\mathsf{f}(\mathsf{a}, x)}, \underline{\mathsf{f}(y, \mathsf{f}(\mathsf{a}, \mathsf{a}))}\right)$ with $\sigma = \{x \to \mathsf{f}(\mathsf{a}, \mathsf{a}), y \to \mathsf{a}\}$
- $\mathrm{INST}\left(\underline{\mathsf{f}(\mathsf{a}, \mathsf{f}(\mathsf{a}, \mathsf{a}))}, \underline{\mathsf{f}(x, y)}\right)$ with $\sigma = \{x \to \mathsf{a}, y \to \mathsf{f}(\mathsf{a}, \mathsf{a})\}$
- $\mathrm{GEN}\left(\underline{\mathsf{f}(x, y)}, \underline{\mathsf{f}(\mathsf{a}, \mathsf{f}(\mathsf{a}, \mathsf{a}))}\right)$ with $\sigma = \{x \to \mathsf{a}, y \to \mathsf{f}(\mathsf{a}, \mathsf{a})\}$
- $\mathrm{VAR}\left(\underline{\mathsf{f}(\mathsf{a}, x)}, \underline{\mathsf{f}(\mathsf{a}, y)}\right)$ with $\sigma = \{x \to y\}$

Relations between terms

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

Examples

- UNIF $\left(\underline{\mathsf{f}(\mathsf{a}, x)}, \underline{\mathsf{f}(y, \mathsf{f}(\mathsf{a}, \mathsf{a}))}\right)$ with $\sigma = \{x \to \mathsf{f}(\mathsf{a}, \mathsf{a}), y \to \mathsf{a}\}$
- INST $\left(\underline{\mathsf{f}(\mathsf{a}, \mathsf{f}(\mathsf{a}, \mathsf{a}))}, \underline{\mathsf{f}(x, y)}\right)$ with $\sigma = \{x \to \mathsf{a}, y \to \mathsf{f}(\mathsf{a}, \mathsf{a})\}$
- GEN $\left(\underline{\mathsf{f}(x, y)}, \underline{\mathsf{f}(\mathsf{a}, \mathsf{f}(\mathsf{a}, \mathsf{a}))}\right)$ with $\sigma = \{x \to \mathsf{a}, y \to \mathsf{f}(\mathsf{a}, \mathsf{a})\}$
- VAR $\left(\underline{\mathsf{f}(\mathsf{a}, x)}, \underline{\mathsf{f}(\mathsf{a}, y)}\right)$ with $\sigma = \{x \to y\}$

Index functions

- `val initialize : Term.t list -> t`
- `val insert : Term.t -> t -> t`
- `val remove: Term.t -> t -> t`
- `val retrieve_unifiable_terms: Term.t -> t -> Term.t list`
- `val retrieve_instances : Term.t -> t -> Term.t list`
- `val retrieve_generalizations : Term.t -> t -> Term.t list`
- `val retrieve_variants : Term.t -> t -> Term.t list`

Bachelor project

- Supervisor: Sarah Winkler
- February 2010 – Summer 2010
- Programming language: OCaml

Tasks of bachelor project

- implement index based on **Discrimination trees**
- implement index based on **Code trees**
- implement any other indexing technique
- run **performance** tests