Data Structures in Leo II

Frank Theiß
joint work with C. Benzmüller, L. Paulson, A. Fietzke

March 18, 2008

An Automated Theorem Prover

- resolution based reasoning in HOL
- compact standalone system implemented in OCaml
- using efficient Data Structures
- especially **Term Sharing** and **Term Indexing**

An Automated Theorem Prover

- resolution based reasoning in HOL
- compact standalone system implemented in OCaml
- using efficient Data Structures
- especially **Term Sharing** and **Term Indexing**

An Automated Theorem Prover

- resolution based reasoning in HOL
- compact standalone system implemented in OCaml
- using efficient Data Structures
- especially **Term Sharing** and **Term Indexing**

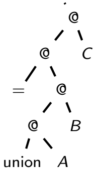
An Automated Theorem Prover

- resolution based reasoning in HOL
- compact standalone system implemented in OCaml
- using efficient Data Structures
- especially **Term Sharing** and **Term Indexing**

The Problem

Typical data in a resolution based Theorem Prover:

$$
\begin{array}{llll}
\mathbf{c_1} & l_{1,1} & \dots & l_{1,m_1} \\
\vdots & & & \\
\mathbf{c_i} & l_{i,1} & \dots l_{i,j} \dots & l_{i,m_i} \\
\vdots & & & \\
\mathbf{c_n} & l_{n,1} & \dots & l_{i,m_n}
\end{array}
$$



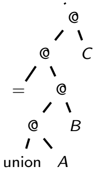
In a naive implementation:

- a list of clauses
- clauses are lists of literals
- symbols are **string values**
- terms are **values** of a recursive data structure (`appl(t1,t2),abstr(type,t1)`)

The Problem

Typical data in a resolution based Theorem Prover:

$\mathbf{c_1}$	$l_{1,1}$	$\ldots$	l_{1,m_1}
$\vdots$			
$\mathbf{c_i}$	$l_{i,1}$	$\ldots l_{i,j} \ldots$	l_{i,m_i}
$\vdots$			
$\mathbf{c_n}$	$l_{n,1}$	$\ldots$	l_{i,m_n}



In a naive implementation:

- a list of clauses
- clauses are lists of literals
- symbols are **string values**
- terms are **values** of a recursive data structure `(appl(t1,t2),abstr(type,t1))`

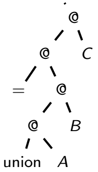
The Problem

Typical data in a resolution based Theorem Prover:

$\mathbf{c_1} \quad l_{1,1} \quad \ldots \quad l_{1,m_1}$

$\vdots$

$\mathbf{c_i} \quad l_{i,1} \quad \ldots\, l_{i,j} \,\ldots \quad l_{i,m_i}$



$\vdots$

$\mathbf{c_n} \quad l_{n,1} \quad \ldots \quad l_{i,m_n}$

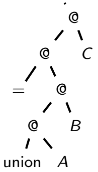
In a naive implementation:

- a list of clauses
- clauses are lists of literals
- symbols are **string values**
- terms are **values** of a recursive data structure (`appl(t1,t2),abstr(type,t1)`)

The Problem

Typical data in a resolution based Theorem Prover:

$\mathbf{c_1}$	$l_{1,1}$	$\dots$	l_{1,m_1}
$\vdots$			
$\mathbf{c_i}$	$l_{i,1}$	$\dots l_{i,j} \dots$	l_{i,m_i}
$\vdots$			
$\mathbf{c_n}$	$l_{n,1}$	$\dots$	l_{i,m_n}



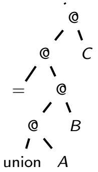
In a naive implementation:

- a list of clauses
- clauses are lists of literals
- symbols are **string values**
- terms are **values** of a recursive data structure `(appl(t1,t2),abstr(type,t1))`

The Problem

Typical data in a resolution based Theorem Prover:

c_1	$l_{1,1}$	$\dots$	l_{1,m_1}
$\vdots$			
c_i	$l_{i,1}$	$\dots l_{i,j} \dots$	l_{i,m_i}
$\vdots$			
c_n	$l_{n,1}$	$\dots$	l_{i,m_n}

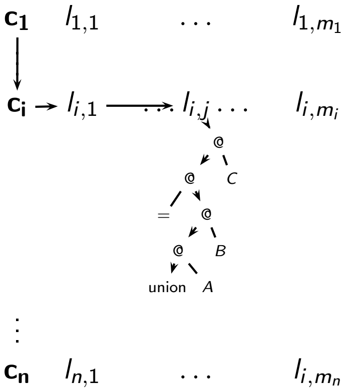


In a naive implementation:

- a list of clauses
- clauses are lists of literals
- symbols are **string values**
- terms are **values** of a recursive data structure (`appl(t1,t2),abstr(type,t1)`)

The Problem

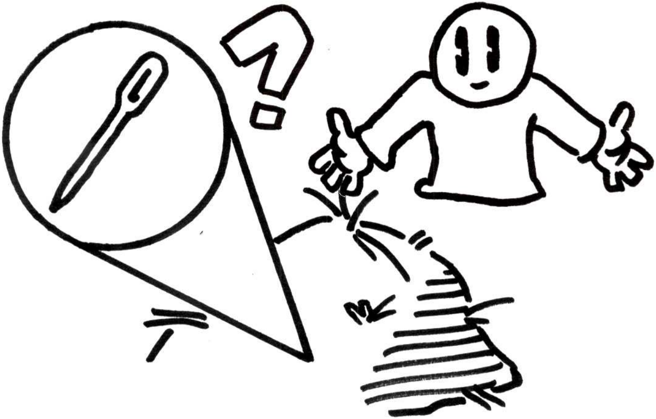
Typical data in a resolution based Theorem Prover:



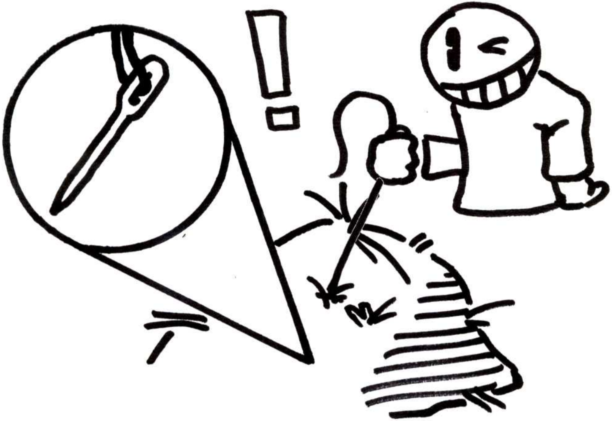
In a naive implementation:

- a list of clauses
- clauses as lists of literals
- symbols as **string values**
- terms as **values** of a recursive data structure (`appl(t1,t2),abstr(type,t1)`)
- long access paths

A Needle and Haystack Problem...

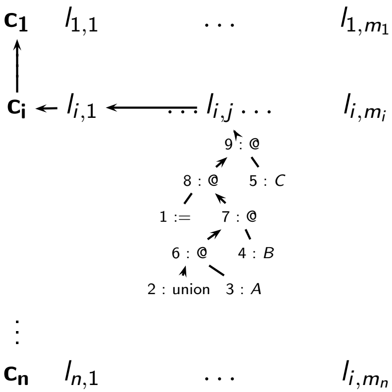


. . . with an efficient Solution



In Leo II

Terms as graphs:



Symbols:

Id	Symbol
1	=
2	union
3	A
4	B
5	C

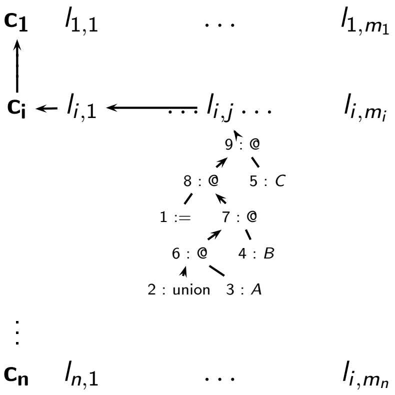
Applications:

Appl	Func	Arg
6	2	3
7	6	4
8	1	7
9	8	5

Abstractions:

Abstr	Body
⋮	⋮

Terms as graphs:



Symbols:

Id	Symbol
1	=
2	union
3	A
4	B
5	C

Applications:

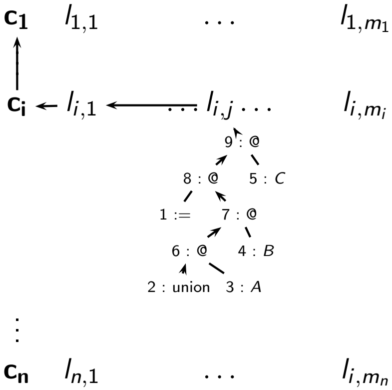
Appl	Func	Arg
6	2	3
7	6	4
8	1	7
9	8	5

Abstractions:

Abstr	Body
⋮	⋮

In Leo II

Terms as graphs:



Symbols:

Id	Symbol
1	=
2	union
3	A
4	B
5	C

Applications:

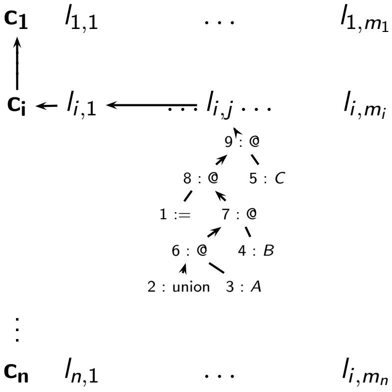
Appl	Func	Arg
6	2	3
7	6	4
8	1	7
9	8	5

Abstractions:

Abstr	Body
⋮	⋮

In Leo II

Terms as graphs:



Symbols:

Id	Symbol
1	=
2	union
3	A
4	B
5	C

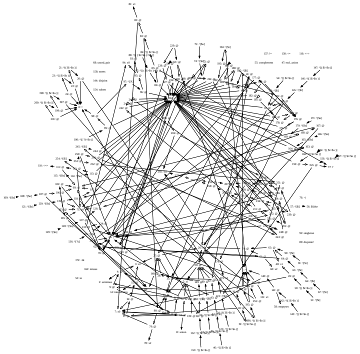
Applications:

Appl	Func	Arg
6	2	3
7	6	4
8	1	7
9	8	5

Abstractions:

Abstr	Body
⋮	⋮

Termsharing



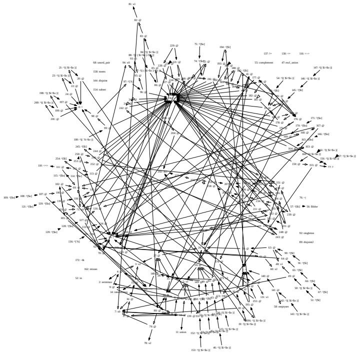
In Leo II:

- Terms as unique instances
- Perfect Term Sharing
- Shallow data structures

Adaption to HOL:

- β-η-normalization
- DeBruijn indices
- local contexts for polymorphic type variables

Termsharing



In Leo II:

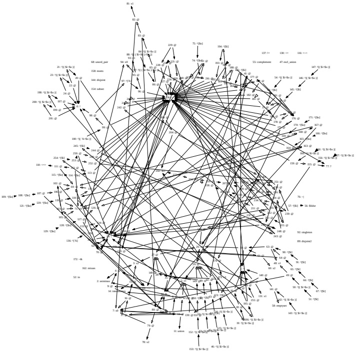
- Terms as unique instances
- Perfect Term Sharing
- Shallow data structures

Adaption to HOL:

- β-η-normalization
- DeBruijn indices
- local contexts for polymorphic type variables

OMEGA GROUP

Termsharing



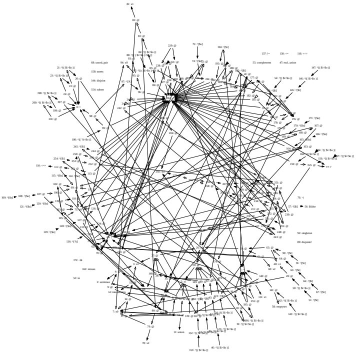
In Leo II:

- Terms as unique instances
- Perfect Term Sharing
- Shallow data structures

Adaption to HOL:

- β-η-normalization
- DeBruijn indices
- local contexts for polymorphic type variables

Termsharing



In Leo II:

- Terms as unique instances
- Perfect Term Sharing
- Shallow data structures

Adaption to HOL:

- β-η-normalization
- DeBruijn indices
- local contexts for polymorphic type variables

Results

Problem	Vamp. 9.0	LEO+Vamp.	LEO-II+E
014+4	114.5	2.60	0.300
017+1	1.0	5.05	0.059
066+1	–	3.73	0.029
067+1	4.6	0.10	0.040
076+1	51.3	0.97	0.031
086+1	0.1	0.01	0.028
096+1	5.9	7.29	0.033
143+3	0.1	0.31	0.034
171+3	68.6	0.38	0.030
580+3	0.0	0.23	0.078
601+3	1.6	1.18	0.089
606+3	0.1	0.27	0.033
607+3	1.2	0.26	0.036
609+3	145.2	0.49	0.039
611+3	0.3	4.00	0.125
612+3	111.9	0.46	0.030
614+3	3.7	0.41	0.060
615+3	103.9	0.47	0.035
623+3	–	2.27	0.282
624+3	3.8	3.29	0.047
630+3	0.1	0.05	0.025
640+3	1.1	0.01	0.033
646+3	84.4	0.01	0.032
647+3	98.2	0.12	0.037

Problem	Vamp. 9.0	LEO+Vamp.	LEO-II+E
648+3	98.2	0.12	0.037
649+3	117.5	0.25	0.037
651+3	117.5	0.09	0.029
657+3	146.6	0.01	0.028
669+3	83.1	0.20	0.041
670+3	–	0.14	0.067
671+3	214.9	0.47	0.038
672+3	–	0.23	0.034
673+3	217.1	0.47	0.042
680+3	146.3	2.38	0.035
683+3	0.3	0.27	0.053
684+3	–	3.39	0.039
716+4	–	0.40	0.033
724+4	–	1.91	0.032
741+4	–	3.70	0.042
747+4	–	1.18	0.028
752+4	–	516.00	0.086
753+4	–	1.64	0.037
764+4	0.1	0.01	0.032

Vamp. 9.0: 2.80GHz, 1GB memory, 600s time limit
LEO+Vamp.: 2.40GHz, 4GB memory, 120s time limit
LEO-II+E: 1.60GHz, 1GB memory, 60s time limit

Springbreak

2008: Automated Theorem Proving enters the Dancefloor

Is Automated Reasoning sexy?