CITIUS ALTIUS FORTIUS: Lessons learned from the Theorem Prover WALDMEISTER

Thomas Hillenbrand

Max-Planck-Institut für Informatik
Saarbrücken

Unit Equational Logic

- Example: group axiomatization

  $$\mathcal{E}: \ (x+y)+z = x+(y+z) \qquad x+0=x \qquad x+(-x)=0$$

  Word problem: Does $\mathcal{E} \models x = --x$ hold?

- Tackle word problem with Knuth-Bendix completion
  – idea: equations $l = r$ oriented into rewrite rules $l \to r$
  – aim: $\mathcal{E} \models s = t$ iff $s{\downarrow} \equiv t{\downarrow}$
  – price: saturation of rules necessary

- Perform fully automated proof search
  Return proof log in case of success

WALDMEISTER Searching for a Proof

```
*********************************************************************
************************* COMPLETION - PROOF *************************
*********************************************************************

new rule:                 1  +(x1,0) -> x1
new rule:                 2  +(x1,-(x1)) -> 0
new rule:                 3  +(+(x1,x2),x3) -> +(x1,+(x2,x3))
new rule:                 4  +(x1,+(0,x2)) -> +(x1,x2)
new rule:                 5  +(x1,-(0)) -> x1
new rule:                 6  +(x1,+(-(x1),x2)) -> +(0,x2)
new rule:                 7  +(0,-(-(x1))) -> x1
new rule:                 8  +(x1,-(-(x2))) -> +(x1,x2)
remove rule:              7
new rule:                 9  +(0,x1) -> x1
remove rule:              4
simplify rhs of rule:     6
new rule:                10  -(0) -> 0
remove rule:              5
new rule:                11  -(-(x1)) -> x1
remove rule:              8
joined goal:            1  c ?= -(-(c)) to c
        +---------------------------+
        |   this proves the goal    |
        +---------------------------+

Proved Goals:
No.  1:  c ?= -(-(c)) joined, current: c = c
1 goal was specified, which was proved.

Waldmeister states: Goal proved.
```

WALDMEISTER Presenting a Proof

Consider the following set of axioms:

Axiom 1: $x + 0 = x$
Axiom 2: $x + (-x) = 0$
Axiom 3: $(x + y) + z = x + (y + z)$

This theorem holds true:

Theorem 1: $x = - - x$

Proof:

Lemma 1: $0 + (- - x) = x$

$$\begin{array}{ll} & 0 + (- - x) \\ = & \quad \text{by Axiom 2 RL} \\ & (x + (-x)) + (- - x) \\ = & \quad \text{by Axiom 3 LR} \\ & x + ((-x) + (- - x)) \\ = & \quad \text{by Axiom 2 LR} \\ & x + 0 \\ = & \quad \text{by Axiom 1 LR} \\ & x \end{array}$$

Lemma 2: $x + (- - y) = x + y$

$$\begin{array}{ll} & x + (- - y) \\ = & \quad \text{by Axiom 1 RL} \\ & (x + 0) + (- - y) \\ = & \quad \text{by Axiom 3 LR} \\ & x + (0 + (- - y)) \\ = & \quad \text{by Lemma 1 LR} \\ & x + y \end{array}$$

Lemma 3: $0 + x = x$

$$\begin{array}{ll} & 0 + x \\ = & \quad \text{by Lemma 2 RL} \\ & 0 + (- - x) \\ = & \quad \text{by Lemma 1 LR} \\ & x \end{array}$$

Theorem 1: $x = - - x$

$$\begin{array}{ll} & x \\ = & \quad \text{by Lemma 3 RL} \\ & 0 + x \\ = & \quad \text{by Lemma 2 RL} \\ & 0 + (- - x) \\ = & \quad \text{by Lemma 3 LR} \\ & - - x \end{array}$$

Aim of this Talk

- FTP organizers:

  ... would like to learn more about WALDMEISTER, what makes it so efficient ...

- Some evidence of efficiency from performance in the CADE ATP System Competitions. A.D. 2002 (70 problems attempted):

	WM	E-SETH	E	Gandalf	Vampire	Otter	SCOTT	CiME
Solved	70	40	36	27	25	17	17	15
Av. Time	3.2	24.0	15.6	78.9	76.5	45.1	130.7	36.6

- What are the underlying concepts?

Outline: Lessons Learned...

- CITIUS: on algorithms and data structures
- ALTIUS: on integrating “intelligence”
- FORTIUS: on the proof procedure

CITIUS:

Tailor your Algorithms and Data Structures!

Algorithms and Data Structures

- Design and analysis of A&D core subject of computer science

- Highly valuable in actual construction of theorem prover

- To be achieved:
  efficient algorithm $\rightsquigarrow$ efficient implementation

- Use problem of *term indexing* as an example

Term Indexing

- Provers incrementally construct data base of facts
  inference application involves complex retrieval from data base

- Retrieval conditions: central term-level operations
  constitute major part of system's work

- Observation: *performance degradation*
  retrieval handled 1:1 $\leadsto$ inference rate soon sharply decreases

- Remedy: retrieval in set-based fashion
  Process at a time one query against a compiled data base!

Perfect Discrimination Trees

- Interpret term as string of its symbols
  Index strings in a *trie* data structure
  Gain: Sharing of common prefixes

- Example: Index for term set

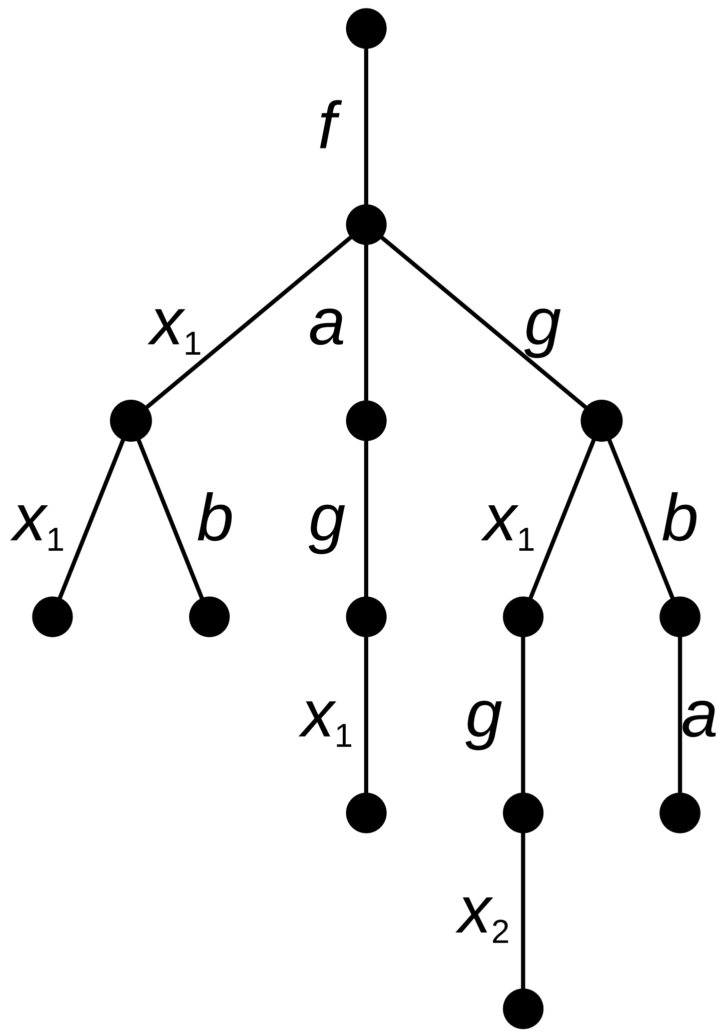
  $f(x_1, x_1)$

  $f(x_1, b)$

  $f(a, g(x_1))$

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

  $f(g(b), a)$

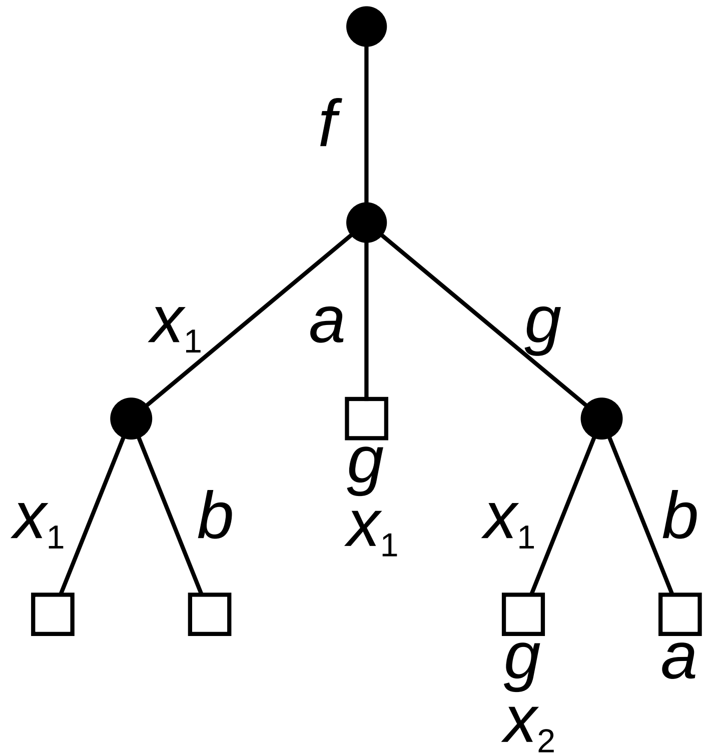


Perfect Discrimination Trees

- Interpret term as string of its symbols
  Index strings in a *trie* data structure
  Gain: Sharing of common prefixes

- Optimization: collapse subtrees
  that contain one leaf node only
  into a single node

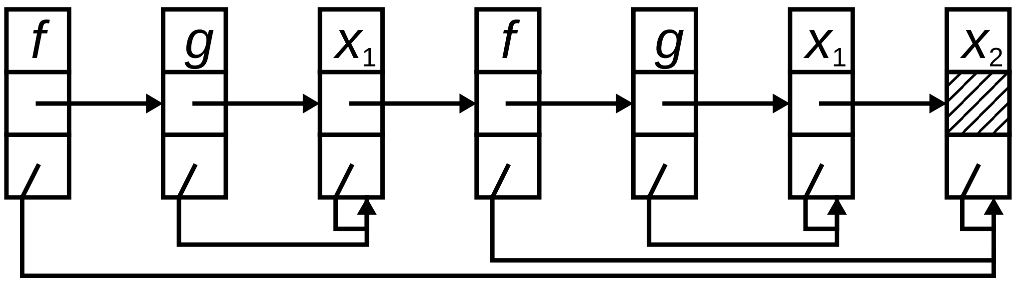
- May cut away more
  than half of the nodes

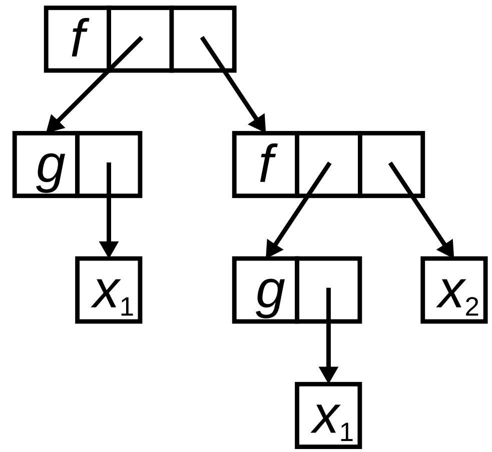


Representation of Terms

- PDTs favour term traversal "from left to right"
  corresponding term structure:

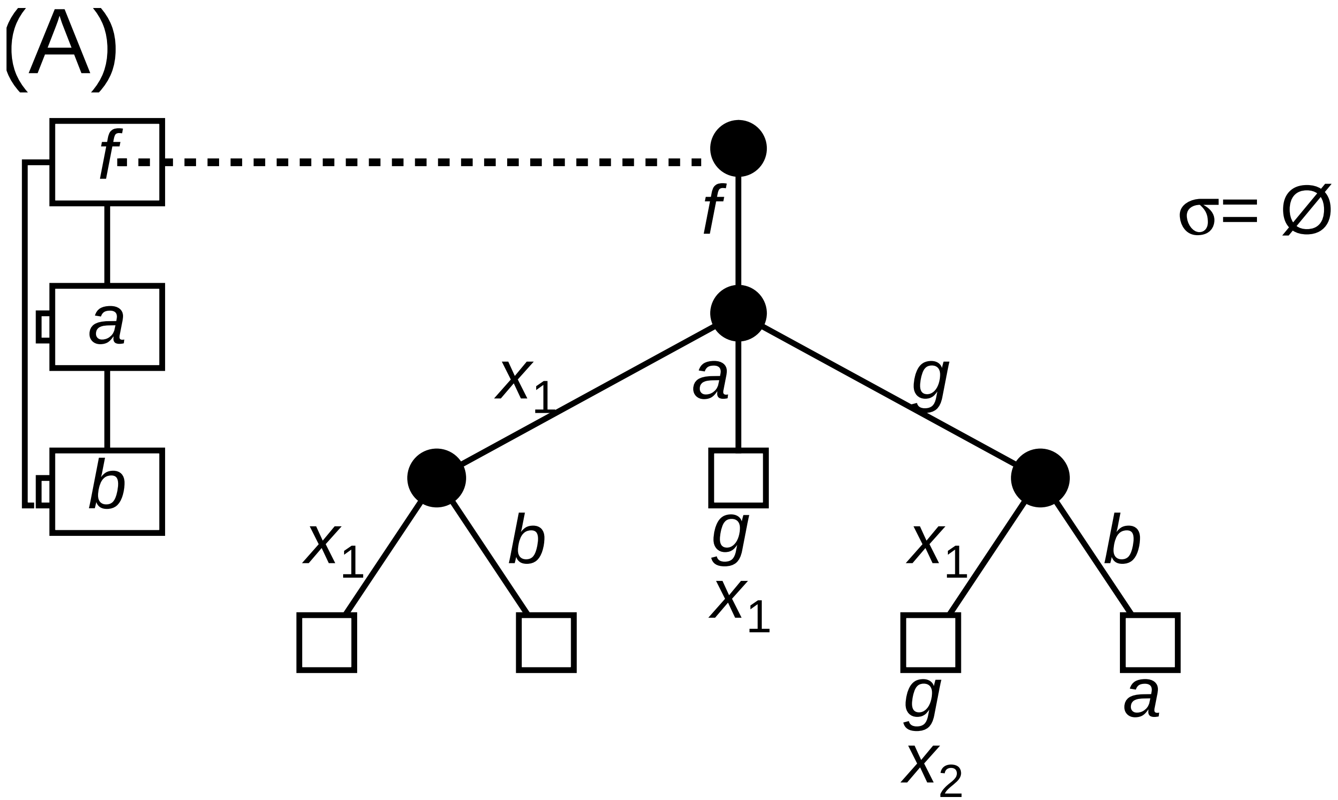
linear instead of tree-like



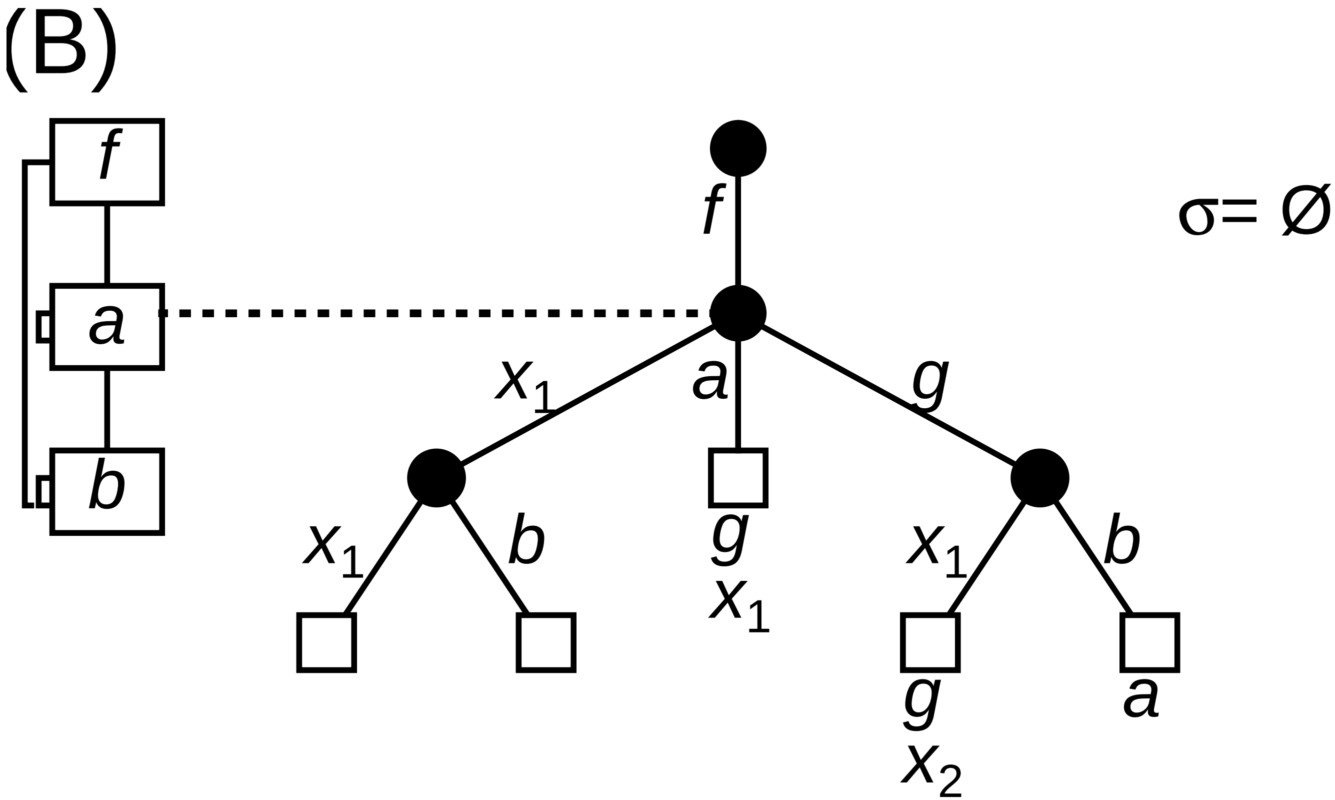


- *flatterms* accelerate preorder term traversal
  more expensive wrt. memory
  but allocation intertwined with free-list memory management

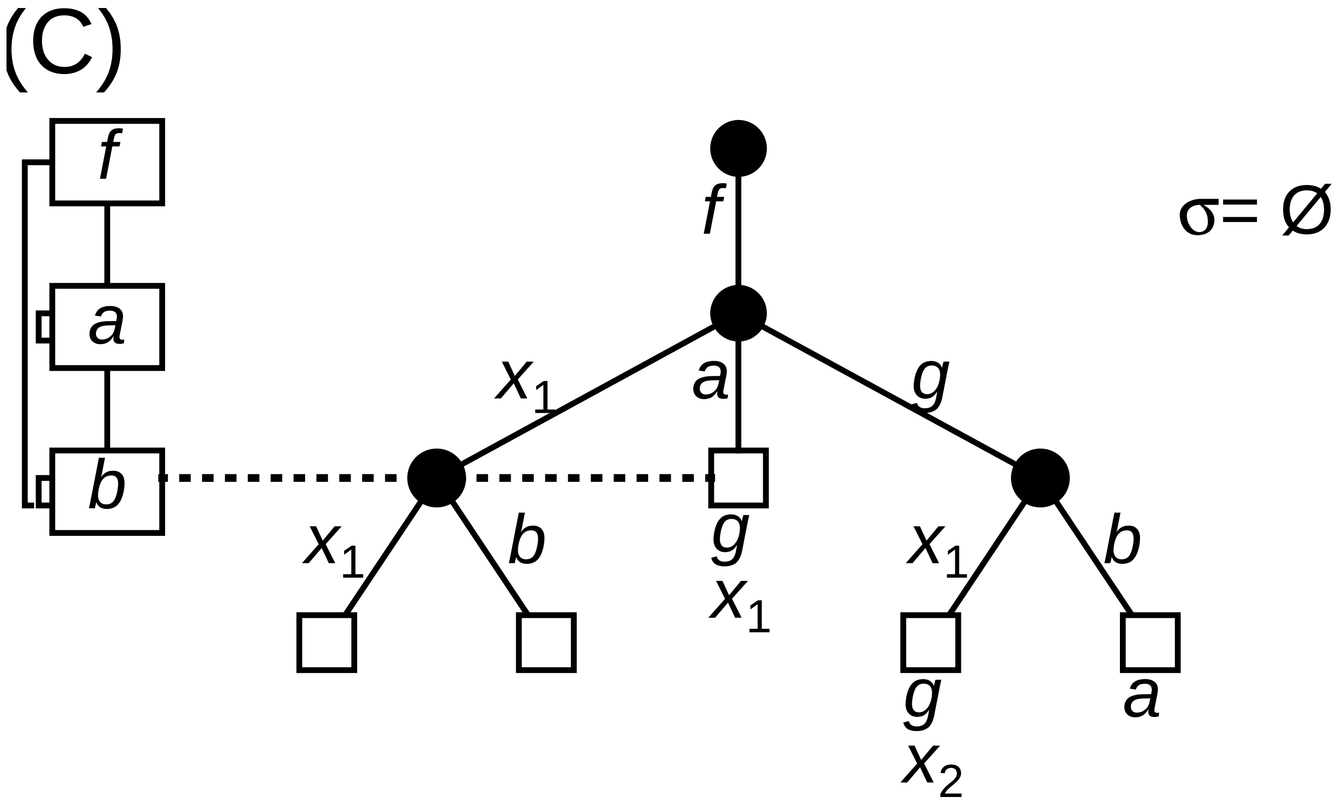
Retrieval of Generalizations



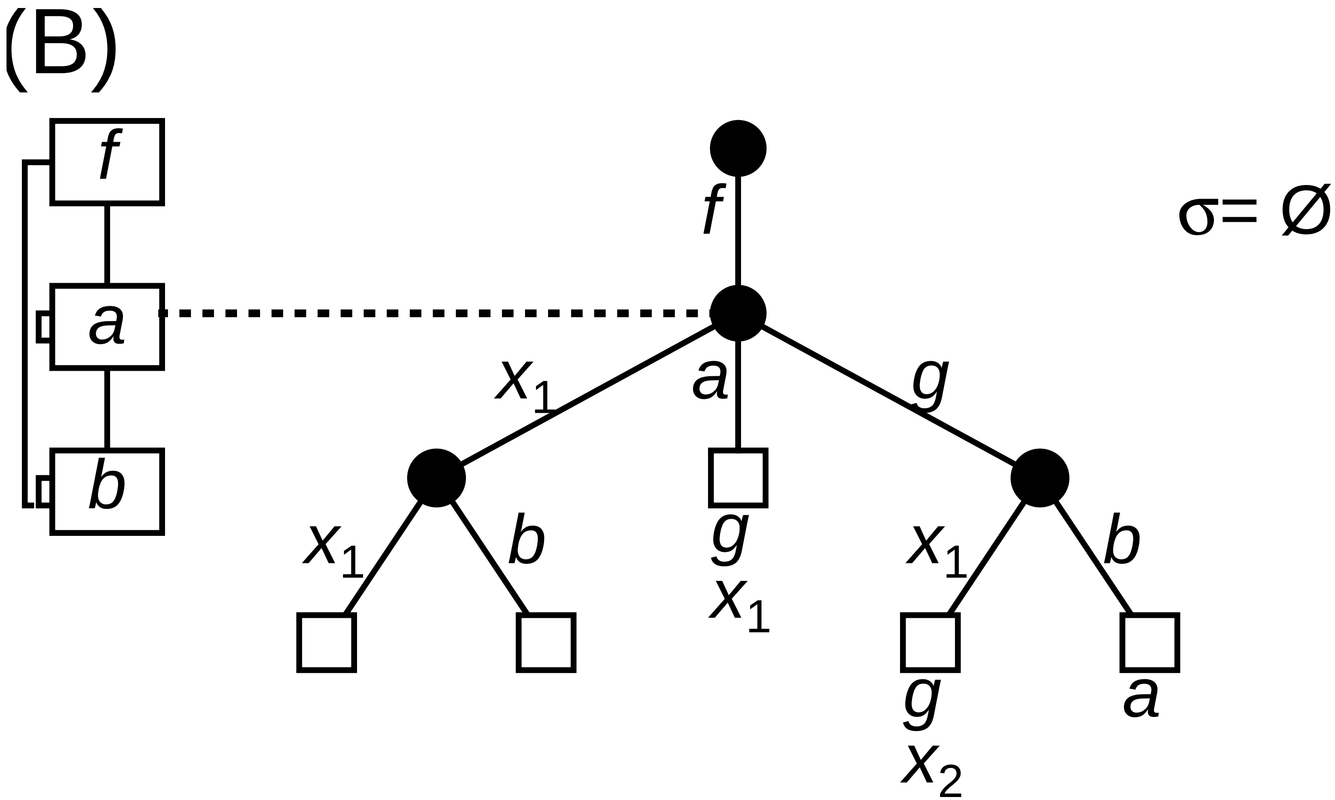
Retrieval of Generalizations



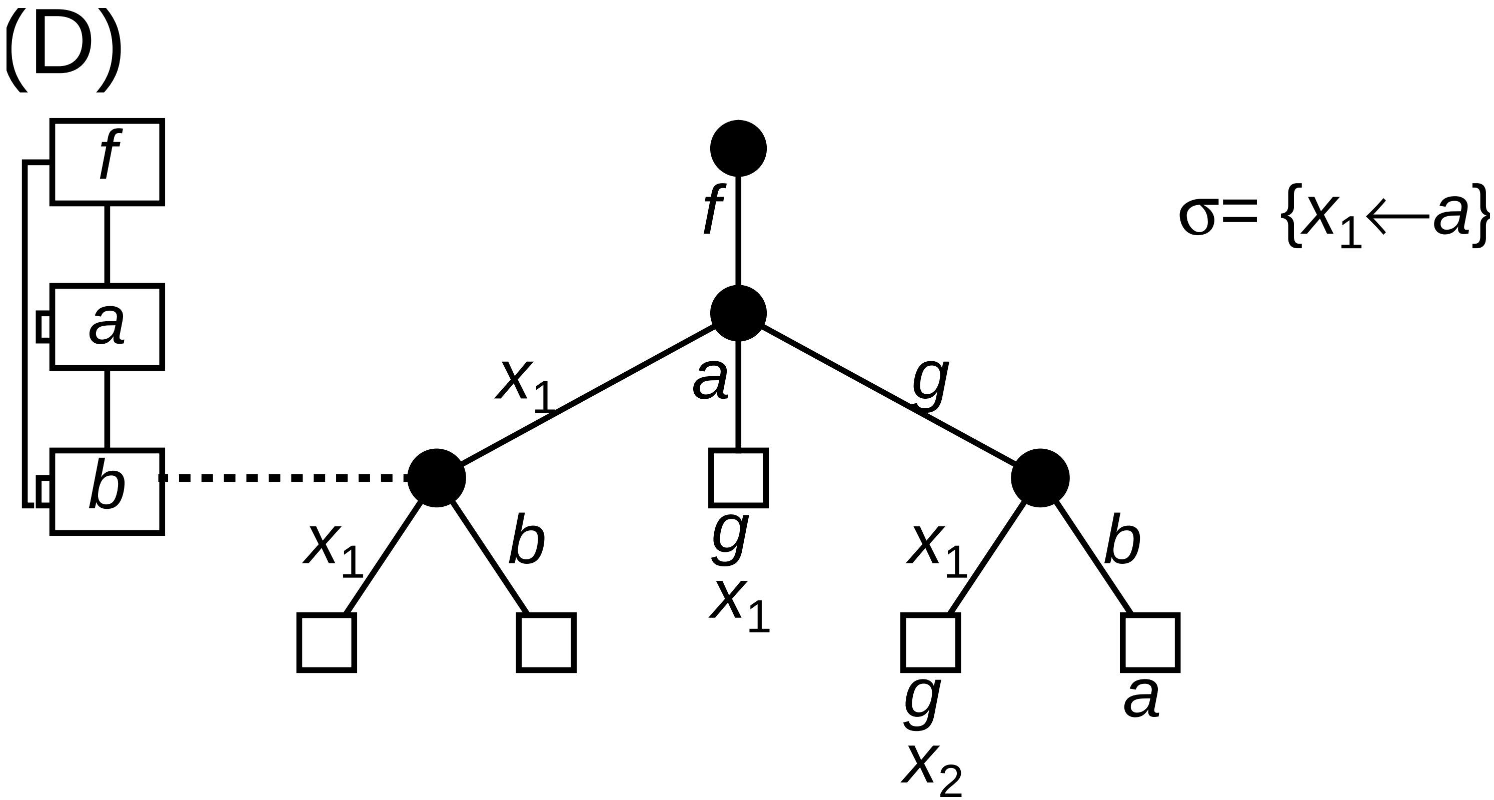
Retrieval of Generalizations



Retrieval of Generalizations

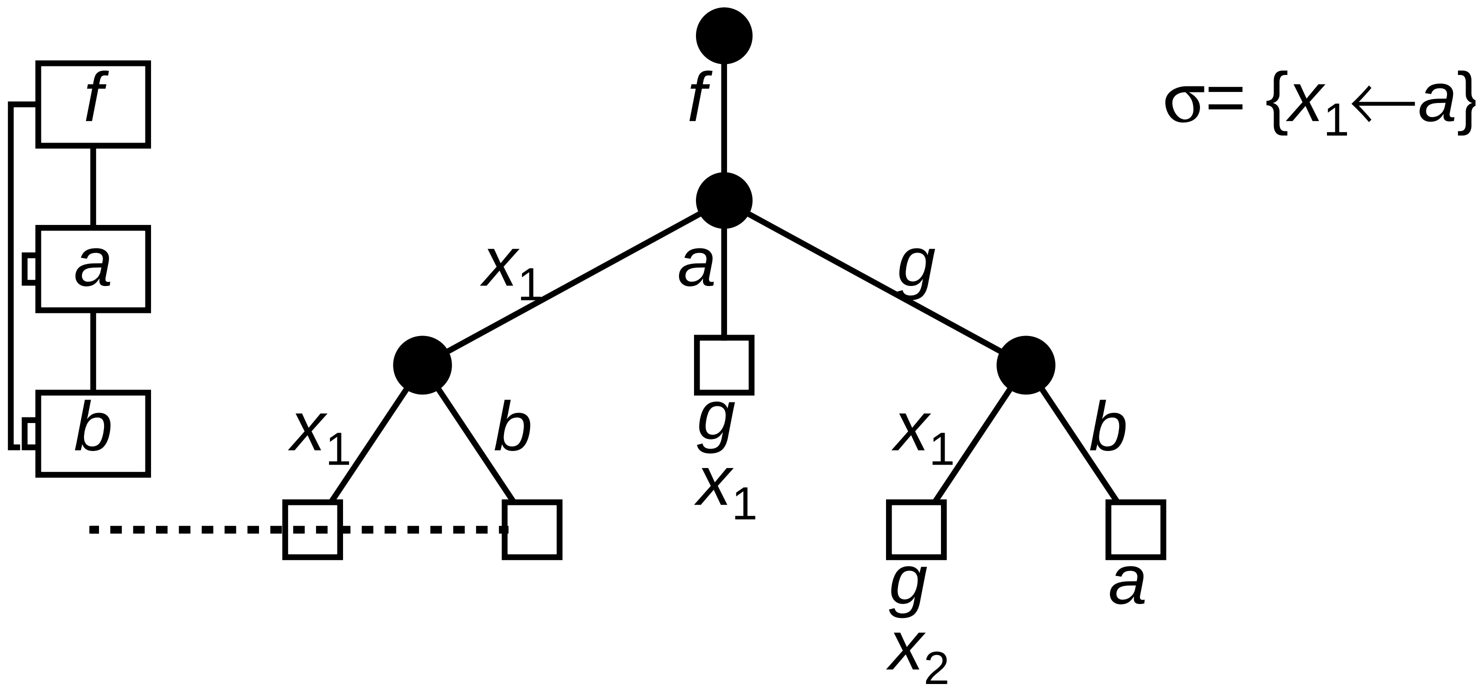


Retrieval of Generalizations



Retrieval of Generalizations

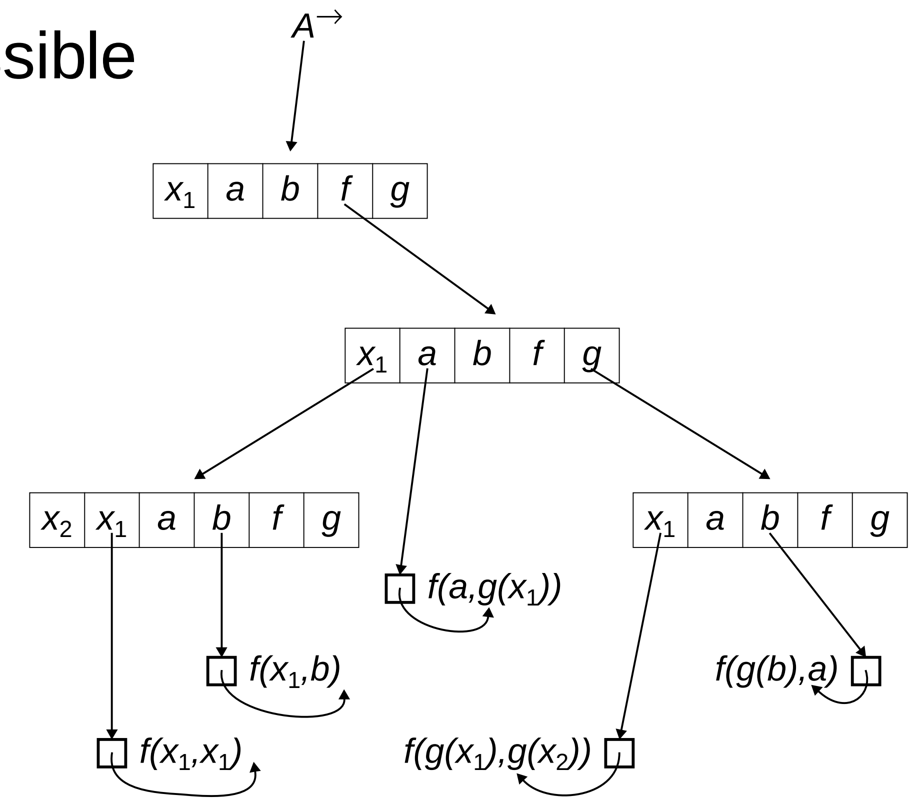
(E)



Careful Coding Counts...

- Reduction of tree size pays off:
  – less memory locations to be inspected during retrieval
  – fewer cache faults, and therefore first-class execution

- Array representation of nodes possible
  hence low-level operations
  on tree very simple!



Further Examples

- Implementation of reduction orderings:
  - arrangement of conditions to evaluate
  - cheap but effective filters
- Computation of normal forms:
  - different strategies of rewriting
  - different combinations of standard and ordered rewrite steps
  - different strategies for recycling of substitutes
- On a larger scale:
  comparison of indexing techniques
  joint work with R. Nieuwenhuis, A. Riazanov, A. Voronkov

COMPIT Web Site

INFORMATIK

COMPIT

COMPIT

COMPIT Home
Download Area
Tutorial
Statistics
People
Useful Links
Contact

Let's COMPIT

A method for COMParing Indexing Techniques for automated deduction

What is Compit ?

If you use automated deduction for your application, you probably use some kind of indexing technique for terms. This technique may even be crucial to the efficiency of your program. But do you know how efficient it really is? Compared to others?

If you are an implementor and you need to know which indexing technique is likely to behave best for your application, or if you are a developer of new indexing techniques, you need to be able to compare techniques in order to get intuition about where to search for improvements, and in order to provide scientific evidence of the superiority of new techniques over other previous ones.

Are you sure your implementation works error free?
If you are an implementor, you need to debug your implementation. Why not use COMPIT for this and solve several problems in one step?
See our tutorial page for a use case.

Compit is a test-framework which allows you to compare indexing techniques for automated deduction. More on Compit you can find in the IJCAR paper "On the Evaluation of Indexing Techniques for Theorem Proving".
How to use Compit to compare your own indexing technique with the ones already included in the test-framework you can see on our tutorial page.

Indexing techniques already included in the testing framework

COMPIT tackles the following indexing problems: retrieval of generalization (forward

COMPIT Joint Initiative

- Observation: complexity analysis of indexing techniques difficult
- Therefore: compare implementations of different techniques on benchmarks corresponding to real runs of real provers
- Speed in 2000:

code trees	:	discr. trees	:	context trees
1.91	:	1.37	:	1.00

- Participants have improved their implementations since e. g. PDTs nearly twice as fast just by more compact node format
- Do not advocate that e. g. PDTs are the best but advocate: that you...

tailor your algorithms and data structures!

FORTIUS:

Design your Proof Procedure Carefully!

Calculus and Proof Procedure

- Unfailing completion: given as set of inference rules

  expanding: $$\frac{s[l'] = t \quad l = r}{(s[r] = t)\sigma}$$ critical pair

  contracting: rewrite-based simplification rules

- Parameter: reduction ordering
  additional control constraint: fairness

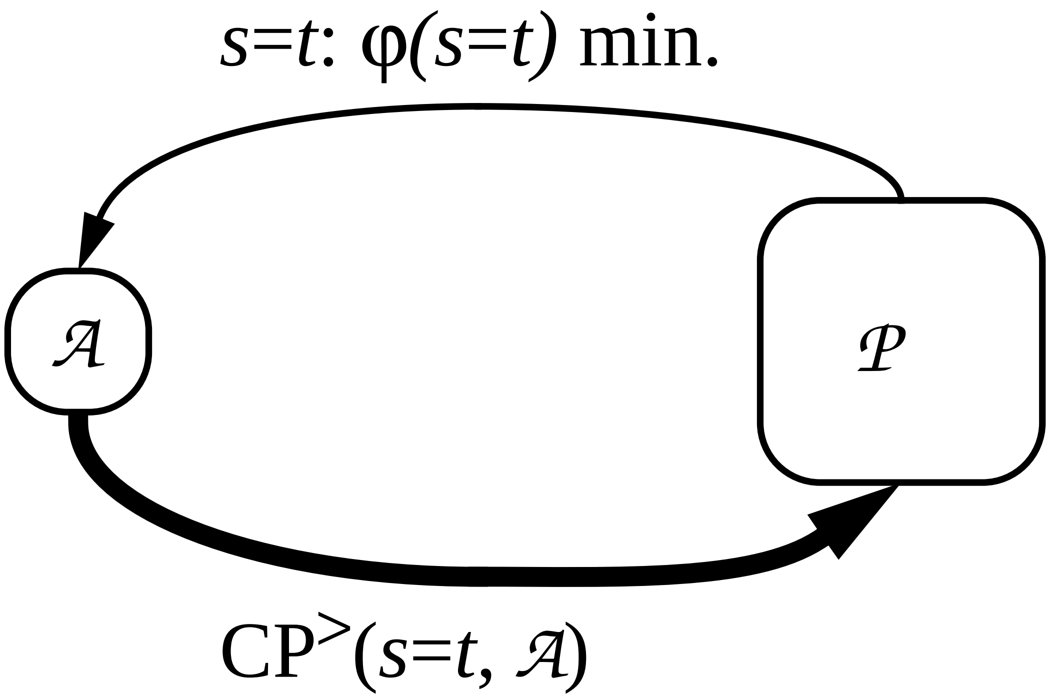
- Non-deterministic algorithm!
  how to resolve non-determinism?

- Common solution: given-clause algorithm (Overbeek 1971)

Given-clause Algorithm

- Approach: incrementally precompute *all* expansion steps
  assess candidate equations heuristically by weighting function φ

- Active facts $\mathcal{A}$ for rewriting and superposition
  passive facts $\mathcal{P}$: critical pairs descending from $\mathcal{A}$



Proof Procedure

FUNCTION $\textsc{Waldmeister}(\mathcal{E}, \mathcal{C}, >, \varphi) : \mathrm{BOOL}$

$(\mathcal{A}, \mathcal{P}) := (\emptyset, \mathcal{E})$
WHILE $\neg\mathrm{trivial}(\mathcal{C}) \wedge \mathcal{P} \neq \emptyset$ **DO**
  $e := \min_{\varphi}(\mathcal{P});\ \mathcal{P} := \mathcal{P} \setminus \{e\}$
  $e := \mathrm{Normalize}_{\mathcal{A}}^{>}(e)$
  IF $\neg\mathrm{redundant}(e)$ **THEN**
    $(\mathcal{A}, P_1) := \mathrm{Interred}^{>}(\mathcal{A}, e)$
    $\mathcal{A} := \mathcal{A} \cup \{\mathrm{Orient}^{>}(e)\}$
    $P_2 := \mathrm{CP}^{>}(e, \mathcal{A})$
    $\mathcal{P} := \mathrm{Update}(\mathcal{P} \cup P_1 \cup P_2)$ Normalize...
    $\mathcal{C} := \mathrm{Normalize}_{\mathcal{A}}^{>}(\mathcal{C})$
  END
END
RETURN $\mathrm{trivial}(\mathcal{C})$

Proof Procedure

FUNCTION $\textsc{Waldmeister}(\mathcal{E}, \mathcal{C}, >, \varphi)$: BOOL

$(\mathcal{A}, \mathcal{P}) := (\emptyset, \mathcal{E})$

WHILE $\neg\mathrm{trivial}(\mathcal{C}) \wedge \mathcal{P} \neq \emptyset$ **DO**

  $e := \min_{\varphi}(\mathcal{P})$; $\mathcal{P} := \mathcal{P} \setminus \{e\}$

  $e := \mathrm{Normalize}^{>}_{\mathcal{A}}(e)$

  IF $\neg\mathrm{redundant}(e)$ **THEN**

    $(\mathcal{A}, P_1) := \mathrm{Interred}^{>}(\mathcal{A}, e)$

    $\mathcal{A} := \mathcal{A} \cup \{\mathrm{Orient}^{>}(e)\}$

    $P_2 := \mathrm{CP}^{>}(e, \mathcal{A})$

    $\mathcal{P} := \mathrm{Normalize}^{>}_{\mathcal{A}}(\mathcal{P} \cup P_1 \cup P_2)$ OTTER loop – eager

    $\mathcal{C} := \mathrm{Normalize}^{>}_{\mathcal{A}}(\mathcal{C})$

  END

END

RETURN $\mathrm{trivial}(\mathcal{C})$

Proof Procedure

FUNCTION $\textsc{Waldmeister}(\mathcal{E}, \mathcal{C}, >, \varphi) : \mathrm{BOOL}$

$(\mathcal{A}, \mathcal{P}) := (\emptyset, \mathcal{E})$
WHILE $\neg\mathsf{trivial}(\mathcal{C}) \wedge \mathcal{P} \neq \emptyset$ **DO**
    $e := \mathsf{min}_\varphi(\mathcal{P});\ \mathcal{P} := \mathcal{P} \setminus \{e\}$
    $e := \mathsf{Normalize}^{>}_{\mathcal{A}}(e)$
    IF $\neg\mathsf{redundant}(e)$ **THEN**
        $(\mathcal{A}, P_1) := \mathsf{Interred}^{>}(\mathcal{A}, e)$
        $\mathcal{A} := \mathcal{A} \cup \{\mathsf{Orient}^{>}(e)\}$
        $P_2 := \mathsf{CP}^{>}(e, \mathcal{A})$
        $\mathcal{P} := \mathcal{P} \cup \mathsf{Normalize}^{>}_{\mathcal{A}}(P_1 \cup P_2)$ DISCOUNT loop – lazy
        $\mathcal{C} := \mathsf{Normalize}^{>}_{\mathcal{A}}(\mathcal{C})$
    END
END
RETURN $\mathsf{trivial}(\mathcal{C})$

Representation of $\mathcal{P}$

- $\mathcal{P}$ ordered under $\varphi \rightsquigarrow$ priority queue
- Typically $|\mathcal{P}|$ exceeding $|\mathcal{A}|$ by three orders of magnitude so space can become a problem
- Representations for elements of $\mathcal{P}$:

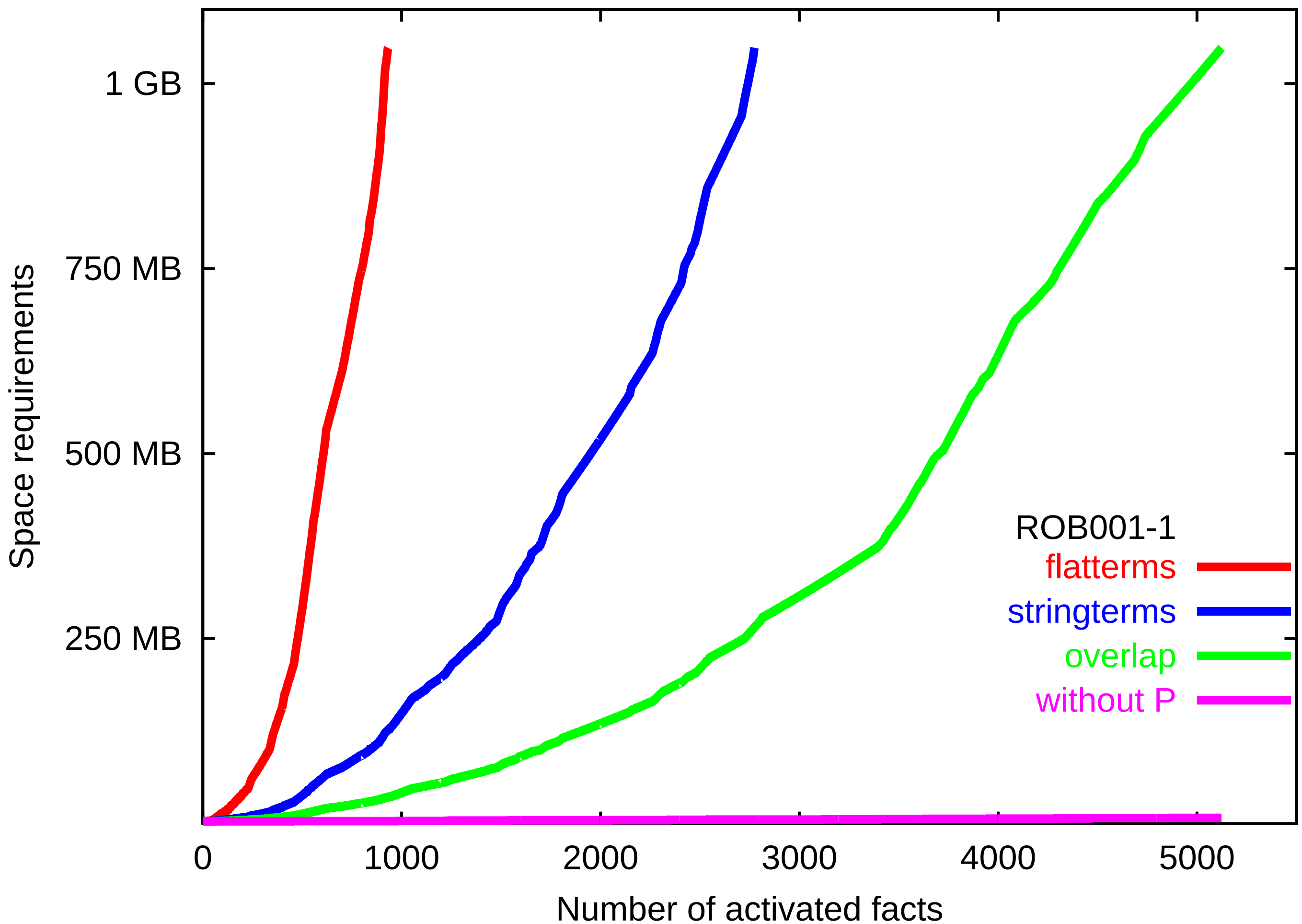
Flatterms $f - \underline{x_1} - f - \underline{a} - \underline{x_2} —— f - \underline{x_1} - \underline{x_2}$

Stringterms

f	x_1	f	a	x_2	f	x_1	x_2

implizit $<s[l']_p = t,\ l = r>$

Space Behaviour over Time



Engineering $\mathcal{P}$ and $\mathcal{A}$

- Problem with the overlap representation:

$$((s = t)\downarrow_i) \downarrow_j \qquad \text{vs.} \qquad (s = t) \downarrow_j$$

- Rewrite relation changes over time
  reproduction not exact (confluence not given!)
  negative effects on proof search

- Requirement for $\mathcal{P}$: behave neutral wrt. proof search!
  Requirement for $\mathcal{A}$: remember history!

- History of $\mathcal{A}$ turns simple compression scheme complete...

Iterated Compression

$$\langle w_1, s_1 = t_1, i, j_1, p_1\rangle \ \langle w_2, s_2 = t_2, i, j_2, p_2\rangle \quad \ldots \quad \langle w_n, s_n = t_n, i, j_n, p_n\rangle$$

$$\downarrow \qquad\qquad \downarrow \qquad\qquad \downarrow \quad \downarrow \qquad\qquad \downarrow$$

$$\langle w_1, i, j_1, p_1\rangle \qquad \langle w_2, i, j_2, p_2\rangle \qquad \ldots \qquad \langle w_n, i, j_n, p_n\rangle$$

$$\searrow \quad \swarrow \qquad \swarrow \qquad\qquad \searrow \quad \downarrow \quad \swarrow$$

$$\langle w'_1, i, j_1, *\rangle \qquad \ldots \qquad \langle w'_k, i, j_k, *\rangle$$

$$\searrow \quad \downarrow \quad \swarrow$$

$$\langle w, i, *, *\rangle$$

On Remembering History

- Saving for each $k \leq j$ the indexing structure for $\mathcal{A}_k$ introduces new space problem

- Employ *one* index for all $\rightarrow_k$, $k \leq j$
  – use age constraints
  – match ordering problem
  – additional benefit: detailed proof objects for free

- Most rewrite steps ($>$ 90%) performed with $\rightarrow_j$
  use two indexes: one for $\rightarrow_j$, one for all $\rightarrow_k$, $k \leq j$

- Elegant solution with perfect discrimination trees...

The Match Ordering Problem

- Query term may be matched by *several* indexed terms:

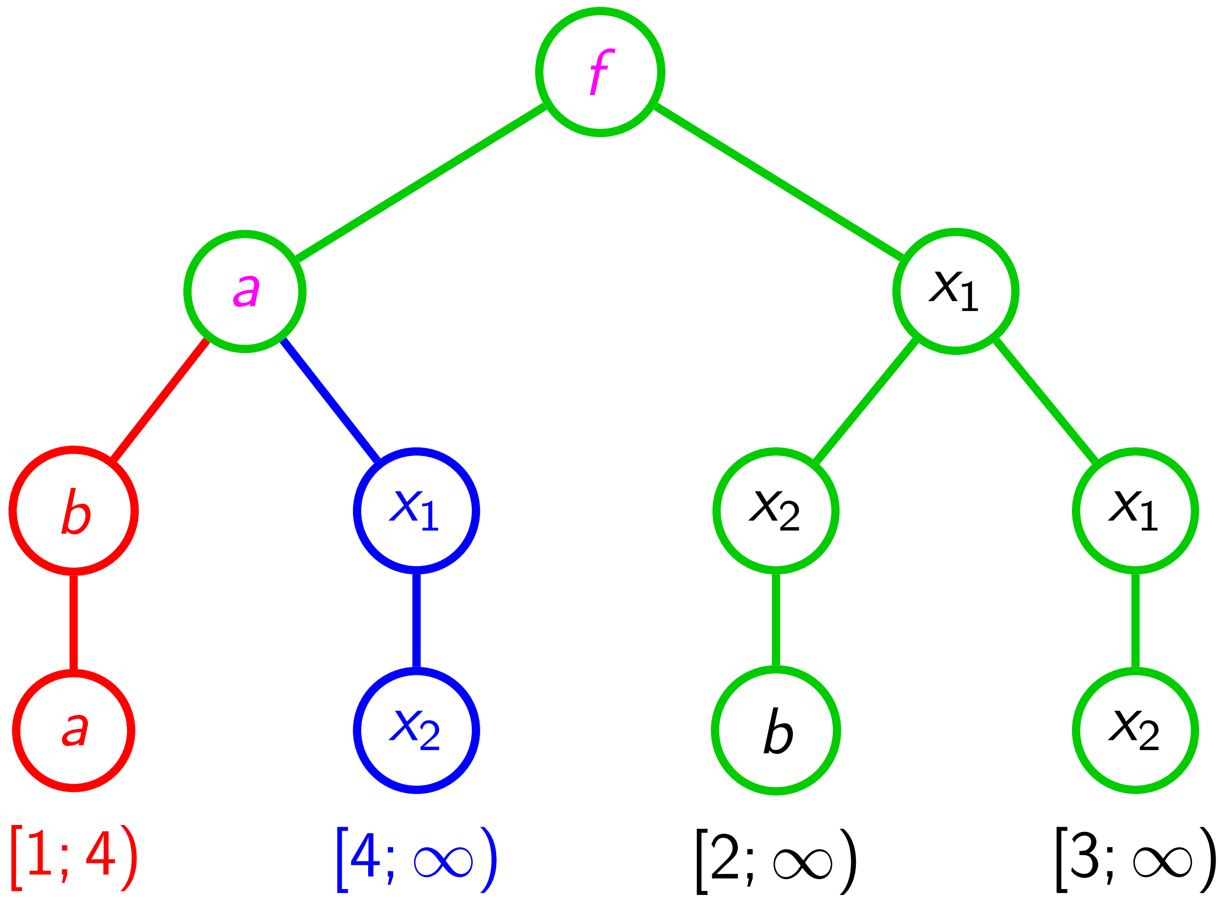
$$f(a,a,b) \qquad \rhd^{?} \qquad \begin{cases} f(a,b,a) & \\ f(x_1,x_2,b) & \longrightarrow \sigma_1 \\ f(x_1,x_1,x_2) & \longrightarrow \sigma_2 \end{cases}$$

- Ordering between matches determined by indexing structure in practice: use first match

- For exact reproduction: store not only all equations, but the ordering relation between the matching terms as well! problem when one index is used for all $\rightarrow_k$, $k \leq j$

- PDTs: match ordering solely determined by traversal strategy

Discrimination Trees for $\rightarrow_k$, $k \leq j$



Match $f(a, a, b)$ at $t = 3$: ✘ ✔ ✔

A Simple Strategy for Compression

- Employ *two buffers*:
  – constant-size cache for individuals up to some weight limit
  – rest buffer: per $\mathcal{A}$-element an entry $\langle w, i, *, * \rangle$ for the rest
  – hence $|\mathcal{P}| = O(|\mathcal{A}|)$

- Cache full $\rightsquigarrow$ move heaver half into second buffer and adapt weight limit

- Minimum selection: from cache individually, from rest buffer by recomputation

- Trade-off between space and time but only a small fraction of $\mathcal{P}$ ever selected

Benefits of Refined Proof Procedure

- Laziness works!

- Huge reduction of space consumption
  at the price of modest run-time overhead
  no discarding – completeness can be retained

- System ready for more demanding problems
  – e. g. $\text{Winker}_2 \Rightarrow$ Boolean: overnight problem the standard way
  – starting point for easily implemented parallelization

- Prover substantially strengthened if you...

design your proof procedure carefully!

ALTIUS:

Integrate more Intelligence!

Tackling Redundancy

- Observation: Performance of unfailing completion not satisfactory when AC axioms are involved

- *Ordered* completion: equation $s = t$ redundant if every ground instance has a smaller proof

- Instances employed in WALDMEISTER: if s and t
  – AC-equal (subsystem ACC’ ground convergent), or
  – joinable under all variable ordering constraints
  – finding: then keep redundant equations *for simplification*

- Full confluence trees: turned out to be computationally too expensive here

Experimental Evaluation

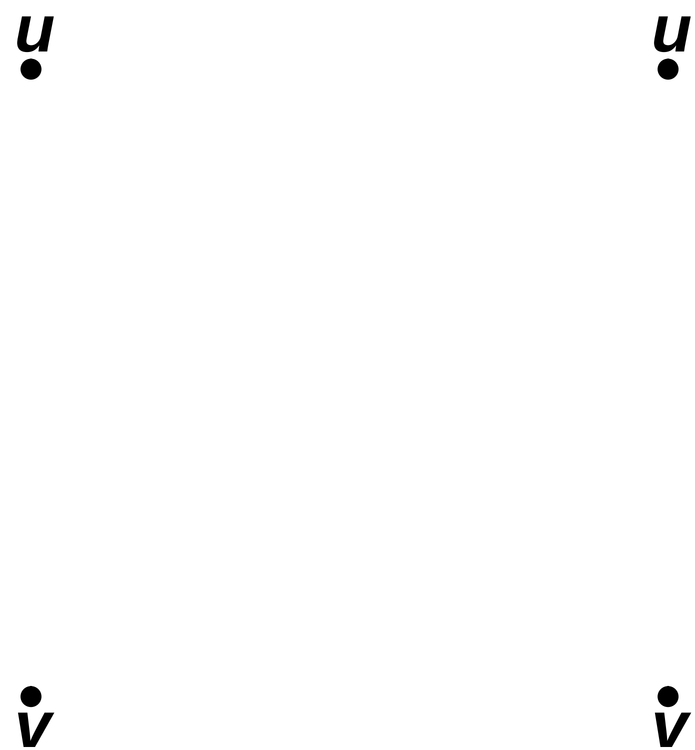
- Number of CPs for representative examples:

Problem	WM	WM-AC	WM-GJ
ROB005-1	305 000	33 000	18 000
RNG027-5	418 000	49 000	54 000
LAT023-1	130 000	66 000	54 000
RNG035-7	237 000	161 000	148 000
GRP180-1	83 000	88 000	65 000

- All in all, cheap alternative to completion modulo AC

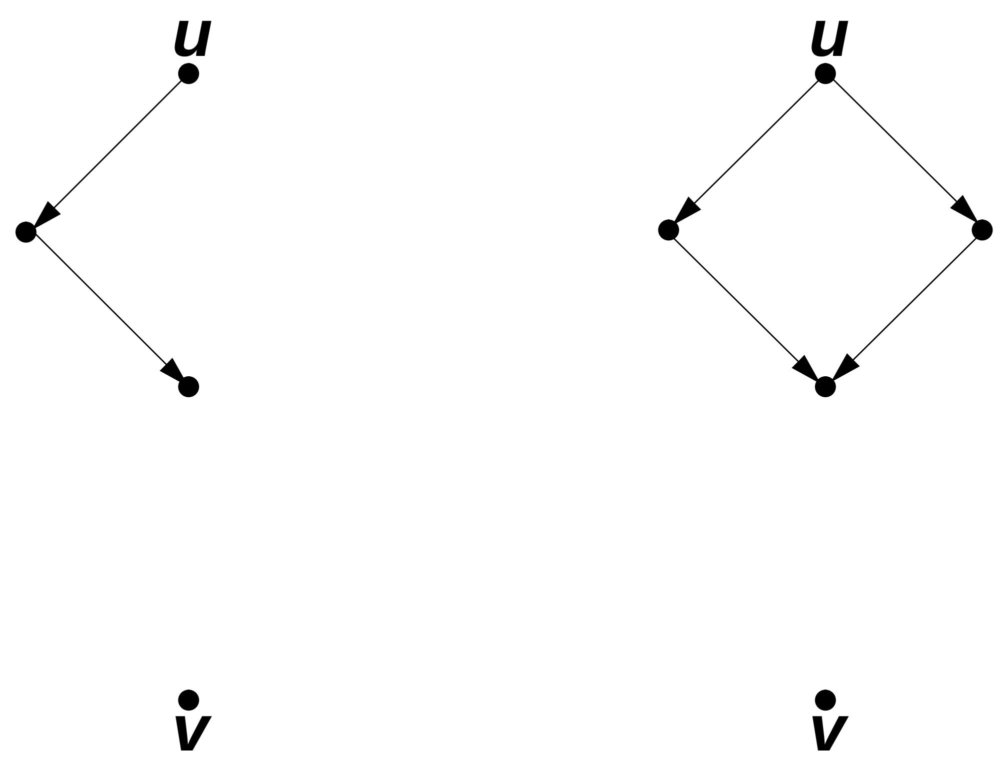
Representing the Conjecture

- Idea: instead of termpairs, consider sets of rewrite successors in order to join left- and right-hand side earlier
- Example: GRP141-1 when 0 rewrite rules derived



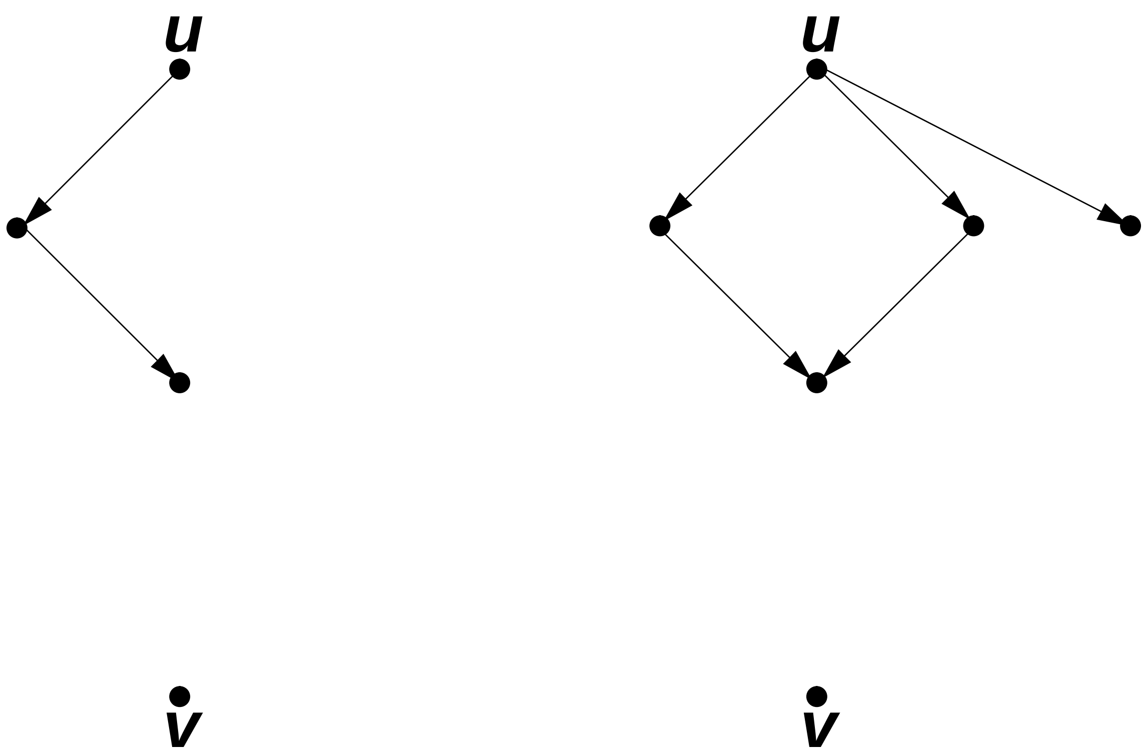
Representing the Conjecture

- Idea: instead of termpairs, consider sets of rewrite successors in order to join left- and right-hand side earlier
- Example: GRP141-1 when 2 rewrite rules derived



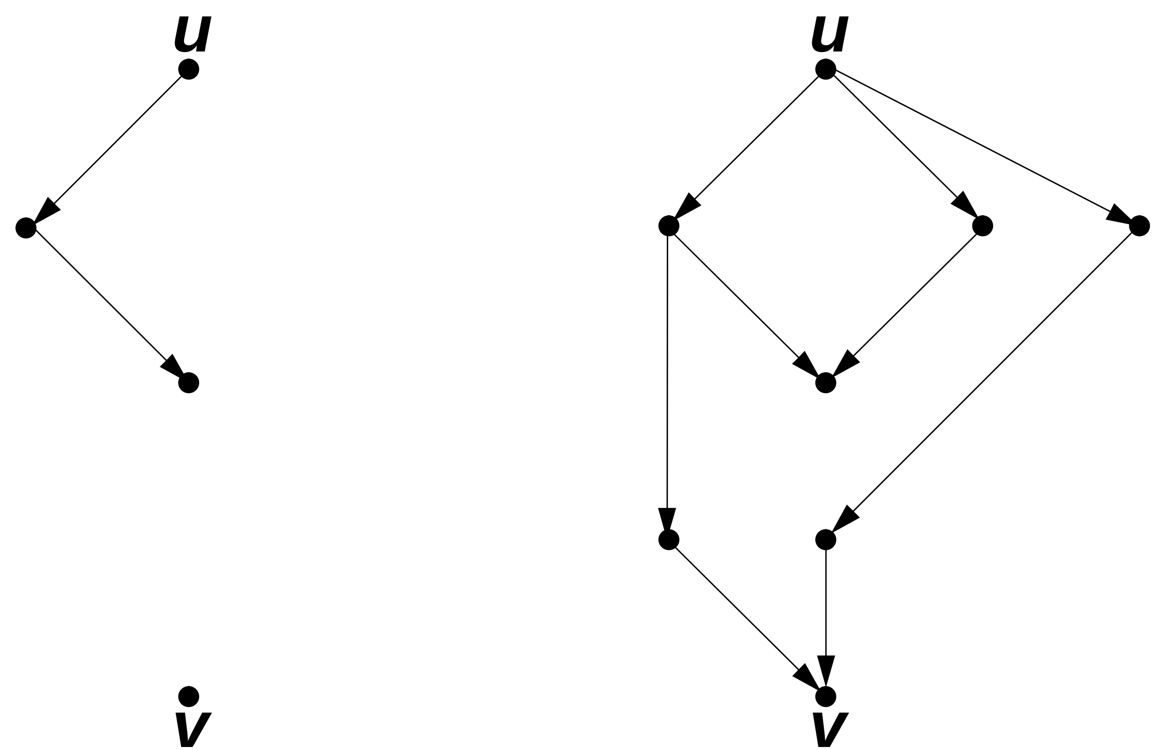
Representing the Conjecture

- Idea: instead of termpairs, consider sets of rewrite successors in order to join left- and right-hand side earlier
- Example: GRP141-1 when 13 rewrite rules derived



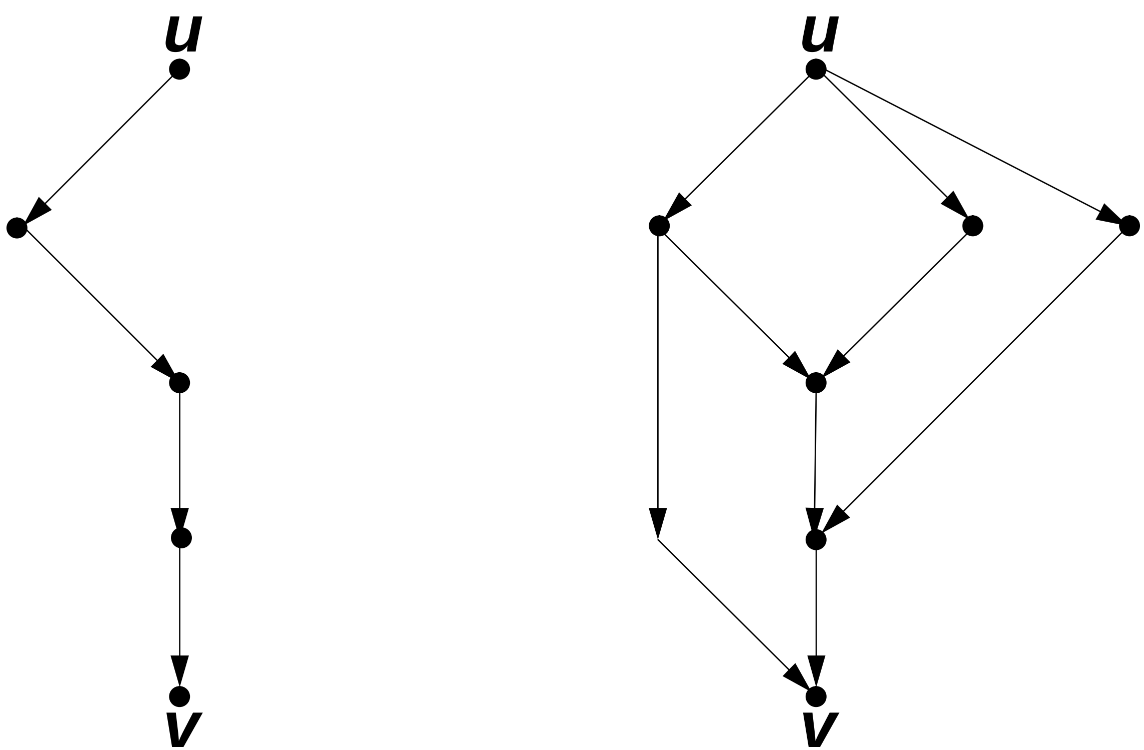
Representing the Conjecture

- Idea: instead of termpairs, consider sets of rewrite successors in order to join left- and right-hand side earlier
- Example: GRP141-1 when 19 rewrite rules derived



Representing the Conjecture

- Idea: instead of termpairs, consider sets of rewrite successors in order to join left- and right-hand side earlier
- Example: GRP141-1 when 30 rewrite rules derived



Benefit Derived from Successor Sets

- Proofs are found
  – in many cases with less steps of saturating the axiomatization
  – at least with no more steps

- Some proofs only found with enlarging

- Focus of completion-based proving slightly shifts
  from axioms to conjecture

- Extension: consider (some) rewrite predecessors as well

- Danger of combinatorical explosion $\rightsquigarrow$ escalation strategy

Automating Control: Weighting Function

- Comparison of different weighting functions in various domains

	addweight	gtweight
BOO003-2	>300	0.1
BOO007-2	>300	81.8
BOO008-4	61.1	7.0
LCL153-1	2.1	>300
LCL154-1	2.0	>300
LCL155-1	1.2	>300
Σ Boolean	22 / 29	29 / 29
	25.4	4.5
Σ Wajsberg	21 / 25	17 / 25
	0.9	0.9

- *Must* employ different classifications on different structures!

Automating Control: Reduction Ordering

- Lexicographic path ordering: lifts operator precedence to terms
  Knuth-Bendix ordering: orders terms according to their length

	LPO	KBO
COL063-4	223.0	0.0
COL063-6	>300	0.0
COL064-6	>300	0.0
Σ BT fragment	21 / 27	25 / 27
	16.6	0.5
Σ non-associa-tive rings	21 / 38	11 / 38
	3.0	1.4
	$A > C > * > - > + > 0$	
Σ lattice-ordered groups	98 / 102	90 / 102
	12.7	23.8
	$+ > \wedge > - > \vee > 0$	

- *Must* employ different orderings on different structures!

Control Component

- Concept: match known axiomatizations on input specification $\mathcal{E}$

- Stage 1: extract known axioms

$\mathcal{E}$:	Table 1:
$+(x,+(y,z))=+(+(x,y),z)$	$F(x,F(y,z))=F(F(x,y),z) \Longrightarrow \mathsf{Ass}(F)$
$+(x,0)=x$	$F(x,E)=x \Longrightarrow \mathsf{Neut_r}(F,E)$
$+(x,-(x))=0$	$F(x,I(x))=E \Longrightarrow \mathsf{Inv_r}(F,I,E)$

- Stage 2: match known structures on extracted axiom set

extracted axioms:	Table 2:
$\{\mathsf{Ass}(+),\mathsf{Neut_r}(+,0),\mathsf{Inv_r}(+,-,0)\}$	$\{\mathsf{Neut_r}(F,E),\mathsf{Ass}(F),\mathsf{Inv_r}(F,I,E)\}$
	$\Longrightarrow \mathsf{Group}(F,I,E)$

Control Component

- Stage 2: match known structures on extracted axiom set

  extracted axioms:

  $\{\mathrm{Ass}(+), \mathrm{Neut_r}(+, 0), \mathrm{Inv_r}(+, -, 0)\}$

  Table 2:

  $\{\mathrm{Neut_r}(F, E), \mathrm{Ass}(F), \mathrm{Inv_r}(F, I, E)\}$
  $\Longrightarrow \mathrm{Group}(F, I, E)$

- Stage 3: instantiate strategy

  detected axiomatization:

  $\mathrm{Group}(+, -, 0)$

  Table 3:

  $\mathrm{Group}(F, I, E) \Longrightarrow$
  $> := \mathrm{LPO}(I > F > E), \varphi := \mathrm{gtweight}$

- Start proof search with reduction ordering $\mathrm{LPO}(- > + > 0)$ and weighting function $\mathrm{gtweight}$

Lesson Learned from these Achievements

- Refinements on inference level a major means of advancing
- There is no general-purpose control strategy!
  therefore: specialization according to the algebraic structure
- Useful control knowledge should be integrated
  a step towards "push button technology"
- System finds more and more proofs if you...

integrate more intelligence!

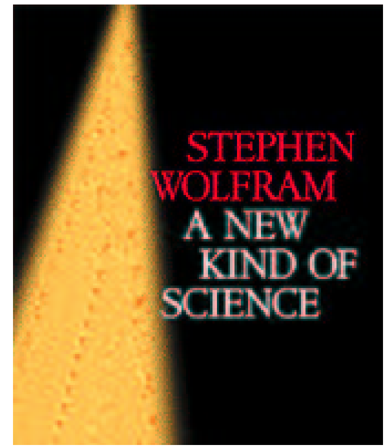
Conclusion

- Many routine problems solved instantly
- Challenging ones may call for expertise
- Expressiveness of logics limited
  but: lessons should carry over to more general calculi
- Conviction: continuous specialization and refinement of deductive techniques is a prerequisite to future progress

Ongoing Work

- Stronger, constraint-based redundancy criteria for specific cases of interest

- Singleton axiomatizations for Sheffer stroke: OTTER and WALDMEISTER applied in writing of *A New Kind of Science*



- Intentions of integrating WALDMEISTER into MATHEMATICA

A Last Secret Finally Revealed

ABOUT WALDMEISTER | ALL TIME NEWS

As you might already know, Waldmeister (asperula odorata, woodruff) is an ingredient for a very popular potable. It is also liked as aroma for sodas. But no, the Waldmeister we are talking about here, you cannot use for your potion. Well, maybe you try and make such a potion. If you are a logic-wizzard after drinking from it, please contact us... Because the Waldmeister we are talking about here is a highly efficient theorem prover.

So be welcomed in the world of Waldmeister, which is the world of logical theorems.

Waldmeister is a theorem prover for unit equational logic. Its proof procedure is unfailing Knuth-Bendix completion [BDP89]. Waldmeister's main advantage is that efficiency has been reached in terms of time as well as of space. Within that scope, a complete proof object is constructed at run-time. Read more about the implementation.

ALL TIME NEWS

For his book

A New Kind Of Science

which is hot from the press, Stephan Wolfram has employed our system to carry out investigations in the area of singleton axiom systems for Boolean algebra. Pages