Meta-Complexity Theorems for Bottom-up Logic Programs

Harald Ganzinger
Max-Planck-Institut für Informatik

David McAllester
ATT Bell-Labs Research
Introduction

- **logic programming of efficient algorithms**
- complexity analysis through general meta-complexity theorems
- guaranteed execution time
- logical aspects of fundamental algorithmic paradigms (dynamic programming, union-find, congruence closure, priority queues)
- application to program analysis: type inference system = algorithm
- recent papers: McAllester [SAS99], Ganzinger/McAllester [IJCAR01]
- related work: efficient fixpoint iteration by Nielson/Seidl [2001]
Introduction

- logic programming of efficient algorithms
- complexity analysis through general meta-complexity theorems
- guaranteed execution time
- logical aspects of fundamental algorithmic paradigms (dynamic programming, union-find, congruence closure, priority queues)
- application to program analysis: type inference system = algorithm
- recent papers: McAllester [SAS99], Ganzinger/McAllester [IJCAR01]
- related work: efficient fixpoint iteration by Nielson/Seidl [2001]
Introduction

• logic programming of efficient algorithms
• complexity analysis through general meta-complexity theorems
• guaranteed execution time
• logical aspects of fundamental algorithmic paradigms (dynamic programming, union-find, congruence closure, priority queues)
• application to program analysis: type inference system = algorithm
• recent papers: McAllester [SAS99], Ganzinger/McAllester [IJCAR01]
• related work: efficient fixpoint iteration by Nielson/Seidl [2001]
Introduction

- logic programming of efficient algorithms
- complexity analysis through general meta-complexity theorems
- guaranteed execution time
- logical aspects of fundamental algorithmic paradigms (dynamic programming, union-find, congruence closure, priority queues)
- application to program analysis:
  type inference system = algorithm
- recent papers:
  McAllester [SAS99], Ganzinger/McAllester [IJCAR01]
- related work: efficient fixpoint iteration by Nielson/Seidl [2001]
Introduction

- logic programming of efficient algorithms
- complexity analysis through general meta-complexity theorems
- guaranteed execution time
- logical aspects of fundamental algorithmic paradigms (dynamic programming, union-find, congruence closure, priority queues)
- application to program analysis:
  type inference system = algorithm
- recent papers:
  McAllester [SAS99], Ganzinger/McAllester [IJCAR01]
- related work: efficient fixpoint iteration by Nielson/Seidl [2001]
Introduction

• logic programming of efficient algorithms
• complexity analysis through general meta-complexity theorems
• guaranteed execution time
• logical aspects of fundamental algorithmic paradigms (dynamic programming, union-find, congruence closure, priority queues)
• application to program analysis:
  type inference system = algorithm

• recent papers:
  McAllester [SAS99], Ganzinger/McAllester [IJCAR01]

• related work: efficient fixpoint iteration by Nielson/Seidl [2001]
Introduction

- logic programming of efficient algorithms
- complexity analysis through general meta-complexity theorems
- guaranteed execution time
- logical aspects of fundamental algorithmic paradigms (dynamic programming, union-find, congruence closure, priority queues)
- application to program analysis: type inference system = algorithm
- recent papers: McAllester [SAS99], Ganzinger/McAllester [IJCAR01]
- related work: efficient fixpoint iteration by Nielson/Seidl [2001]
Contents

1st meta-complexity theorem
Language: bottom-up logic programs
Algorithmic ingredients: dynamic programming, indexing
Examples: (interprocedural) reachability

2nd meta-complexity theorem
Language: logic programs with deletion and priorities
Logical basis: saturation up to redundancy
Examples: union-find, congruence closure, Henglein’s subtype analysis

3rd meta-complexity theorem
Language: logic programs with deletion and instance priorities
Algorithmic ingredients: priority queues
Examples: shortest paths, minimal spanning trees
1st meta-complexity theorem
Language: bottom-up logic programs
Algorithmic ingredients: dynamic programming, indexing
Examples: (interprocedural) reachability

2nd meta-complexity theorem
Language: logic programs with deletion and priorities
Logical basis: saturation up to redundancy
Examples: union-find, congruence closure, Henglein’s subtype analysis

3rd meta-complexity theorem
Language: logic programs with deletion and instance priorities
Algorithmic ingredients: priority queues
Examples: shortest paths, minimal spanning trees
1st meta-complexity theorem
Language: bottom-up logic programs
Algorithmic ingredients: dynamic programming, indexing
Examples: (interprocedural) reachability

2nd meta-complexity theorem
Language: logic programs with deletion and priorities
Logical basis: saturation up to redundancy
Examples: union-find, congruence closure, Henglein’s subtype analysis

3rd meta-complexity theorem
Language: logic programs with deletion and instance priorities
Algorithmic ingredients: priority queues
Examples: shortest paths, minimal spanning trees
database of facts $D$

inference system $R$

closure $R^*(D)$

this talk
Paradigm

input

pre-processor

database of facts $D$

inference system $R$

closure $R^*(D)$

post-processor

output

this talk
Paradigm

- **Input**
  - Pre-processor
  - Database of facts $D$
  - Inference system $R$
  - Closure $R^*(D)$
  - Post-processor
  - Output

- Paige, Yang 1997
Database:

\[ D = \{ e(u, v) \mid (u, v) \in E \} \cup \{ s(u) \mid u \text{ a source node} \} \]
Reachability in Graphs

Database:

\[ D = \{ e(u, v) \mid (u, v) \in E \} \cup \{ s(u) \mid u \text{ a source node} \} \]

Inference system:

\[
\begin{align*}
  s(u) & \quad r(u) \\
  e(u, v) & \quad r(v)
\end{align*}
\]
Reachability in Graphs

Database:

\[ D = \{ e(u, v) \mid (u, v) \in E \} \cup \{ s(u) \mid u \text{ a source node} \} \]

Inference system:

\[
\begin{align*}
\frac{s(u) \quad e(u, v)}{r(u) \quad r(v)} \quad \frac{r(u)}{r(u)}
\end{align*}
\]

Clause notation:  \( s(u) \supset r(u) \quad r(u), e(u, v) \supset r(v) \)
Database:

\[ D = \{ e(u, v) \mid (u, v) \in E \} \cup \{ s(u) \mid u \text{ a source node} \} \]

Inference system:

\[
\begin{align*}
    s(u) & \quad r(u) \\
    \frac{s(u)}{r(u)} & \quad \frac{r(u)}{e(u, v)} \\
    \frac{r(u)}{r(v)} &
\end{align*}
\]

Clause notation:  \( s(u) \supset r(u) \)  \( r(u), e(u, v) \supset r(v) \)

Closure:

\[ R^*(D) = D \cup \{ r(u) \mid u \text{ reachable from a source} \} \]
Database
$s(1), e(1, 3), e(1, 4), e(2, 3), e(3, 4), e(4, 3)$
Example

Database

$s(1), e(1, 3), e(1, 4), e(2, 3), e(3, 4), e(4, 3), r(1)$
Database

\[s(1), e(1, 3), e(1, 4), e(2, 3), e(3, 4), e(4, 3), r(1), r(3)\]
Database

s(1), e(1, 3), e(1, 4), e(2, 3), e(3, 4), e(4, 3), r(1), r(3), r(4)
Example

Database

$s(1), e(1, 3), e(1, 4), e(2, 3), e(3, 4), e(4, 3), r(1), r(3), r(4)$

⇒ saturated.
Bottom-up computation: match prefixes of antecedents against database and fire conclusions
Bottom-up computation: match prefixes of antecedents against database and fire conclusions

prefix firings:

$$\pi_R(D) = | \{(r\sigma,i) \mid r = A_1 \land \ldots \land A_i \land \ldots \land A_n \supset A_0 \in R \land A_j \sigma \in D, \text{ for } 1 \leq j \leq i\} |$$
**First Meta-Complexity Theorem**

**Bottom-up computation:** match prefixes of antecedents against database and fire conclusions

**Prefix firings:**

\[
\pi_R(D) = \left| \{(r\sigma, i) \mid r = A_1 \land \ldots \land A_i \land \ldots \land A_n \supset A_0 \in R \right.
\]

\[
A_j\sigma \in D, \text{ for } 1 \leq j \leq i \}
\]

**Theorem** [McAllester 1999] Let \( R \) be an inference system such that \( R^*(D) \) is finite. Then \( R^*(D) \) can be computed in time \( O(\|D\| + \pi_R(R^*(D))) \).
First Meta-Complexity Theorem

Bottom-up computation: match prefixes of antecedents against database and fire conclusions

prefix firings:

$$
\pi_R(D) = | \{(r\sigma, i) \mid r = A_1 \land \ldots \land A_i \land \ldots \land A_n \supset A_0 \in R \\
A_j \sigma \in D, \text{ for } 1 \leq j \leq i\} |
$$

**Theorem** [McAllester 1999] Let $R$ be an inference system such that $R^*(D)$ is finite. Then $R^*(D)$ can be computed in time $O(\|D\| + \pi_R(R^*(D)))$.

**Corollary** [Dowling, Gallier 1984] If $R$ is ground, $R^*(D)$ can be computed in time $O(\|D\| + \|R\|)$. 
First Meta-Complexity Theorem

Bottom-up computation: match prefixes of antecedents against database and fire conclusions

Prefix firings:

$$\pi_R(D) = | \{(r\sigma, i) \mid r = A_1 \wedge \ldots \wedge A_i \wedge \ldots \wedge A_n \supset A_0 \in R \wedge A_j\sigma \in D, \text{ for } 1 \leq j \leq i\} |$$

**Theorem** [McAllester 1999] Let $R$ be an inference system such that $R^*(D)$ is finite. Then $R^*(D)$ can be computed in time $O(\|D\| + \pi_R(R^*(D)))$.

**Corollary** [Dowling, Gallier 1984] If $R$ is ground, $R^*(D)$ can be computed in time $O(\|D\| + \|R\|)$.

**Extension:** constraints for which each solution can be computed in time $O(1)$.
Reachability in Graphs

\[
\begin{align*}
\text{s}(u) & \quad \text{r}(u) \\
\hline
\text{r}(u) & \\
\text{e}(u, v) & \\
\hline
\text{r}(v) & 
\end{align*}
\]
Reachability in Graphs

\[
\begin{align*}
&\frac{s(u)}{r(u)} \quad O(|V|) \\
&\frac{r(u)}{r(v)} \quad O(|V|) \\
&\frac{e(u, v)}{r(v)}
\end{align*}
\]
Reachability in Graphs

\[
\begin{align*}
\frac{s(u)}{r(u)} & \quad O(|V|) & \quad \frac{r(u)}{r(v)} & \quad O(|V|) \\
\frac{e(u, v)}{r(v)} & + O(|E|)
\end{align*}
\]

**Theorem** Reachability can be decided in linear time.
Interprocedural Reachability: Database

program

1  procedure main
2    begin
3      declare x: int
4      read(x)
5      call p(x)
6    end

7  procedure p(a:int)
8    begin
9      if a>0 then
10         read(g)
11         a:=a-g
12         call p(a)
13         print(a)
14      fi
15    end

facts

proc(main,2,6)
next(main,2,5)
call(main,p,5,6)

proc(p,8,15)
next(p,8,12)
call(p,p,12,13)
next(p,13,15)
next(p,8,15)
Interprocedural Reachability: Rules

Read “$P \Rightarrow L$” as “in procedure $P$ label $L$ can be reached”.

\[
\text{proc}(P, B_P, E_P) \\
\hline
P \Rightarrow B_P
\]

\[
\text{next}(Q, L, L') \\
Q \Rightarrow L \\
\hline
Q \Rightarrow L'
\]

\[
\text{call}(Q, P, L_c, R_r) \\
\text{proc}(P, B_P, E_P) \\
P \Rightarrow E_P \\
Q \Rightarrow L_c \\
\hline
Q \Rightarrow L_r
\]
Interprocedural Reachability: Rules

Read “$P \Rightarrow L$” as “in procedure $P$ label $L$ can be reached”.

\[
\text{proc}(P, B_P, E_P) \quad O(n)
\]

\[
\overline{P \Rightarrow B_P}
\]

\[
\text{next}(Q, L, L') \quad Q \Rightarrow L
\]

\[
\overline{Q \Rightarrow L'}
\]

\[
\text{call}(Q, P, L_c, R_r) \quad \text{proc}(P, B_P, E_P) \quad P \Rightarrow E_P \quad Q \Rightarrow L_c
\]

\[
\overline{Q \Rightarrow L_r}
\]
Interprocedural Reachability: Rules

Read “\( P \Rightarrow L \)” as “in procedure \( P \) label \( L \) can be reached”.

\[
\begin{align*}
\text{proc}(P, B_P, E_P) & \quad O(n) \\
\hline
P & \Rightarrow B_P \\
\end{align*}
\]

\[
\begin{align*}
\text{next}(Q, L, L') & \quad O(n) \\
Q & \Rightarrow L \\
\hline
Q & \Rightarrow L' \\
\end{align*}
\]

\[
\begin{align*}
\text{call}(Q, P, L_c, R_r) & \\
\text{proc}(P, B_P, E_P) & \\
P & \Rightarrow E_P \\
Q & \Rightarrow L_c \\
\hline
Q & \Rightarrow L_r \\
\end{align*}
\]
Interprocedural Reachability: Rules

Read “\( P \Rightarrow L \)” as “in procedure \( P \) label \( L \) can be reached”.

\[
\begin{align*}
\text{proc}(P, B_P, E_P) & \quad O(n) \\
\hline
P \Rightarrow B_P
\end{align*}
\]

\[
\begin{align*}
\text{next}(Q, L, L') & \quad O(n) \\
Q \Rightarrow L & \quad \ast O(1) \\
\hline
Q \Rightarrow L'
\end{align*}
\]

\[
\begin{align*}
\text{call}(Q, P, L_c, R_r) & \\
\text{proc}(P, B_P, E_P) & \\
P \Rightarrow E_P & \\
Q \Rightarrow L_c & \\
\hline
Q \Rightarrow L_r
\end{align*}
\]
Interprocedural Reachability: Rules

Read “$P \Rightarrow L$” as “in procedure $P$ label $L$ can be reached”.

\[
\begin{align*}
\text{proc}(P, B_P, E_P) & \quad O(n) \\
P \Rightarrow B_P & \\
\text{next}(Q, L, L') & \quad O(n) \\
Q \Rightarrow L & \quad \ast O(1) \\
Q \Rightarrow L' & \\
\text{call}(Q, P, L_c, R_r) & \quad O(n) \\
\text{proc}(P, B_P, E_P) & \\
P \Rightarrow E_P & \\
Q \Rightarrow L_c & \\
Q \Rightarrow L_r & \\
\end{align*}
\]
Interprocedural Reachability: Rules

Read “$P \Rightarrow L$” as “in procedure $P$ label $L$ can be reached”.

\[
\begin{align*}
&\text{proc}(P, B_P, E_P) \quad O(n) \\
\hline
&P \Rightarrow B_P
\end{align*}
\]

\[
\begin{align*}
&\text{next}(Q, L, L') \quad O(n) \\
&Q \Rightarrow L \quad \ast O(1) \\
\hline
&Q \Rightarrow L'
\end{align*}
\]

\[
\begin{align*}
&\text{call}(Q, P, L_c, R_r) \quad O(n) \\
&\text{proc}(P, B_P, E_P) \quad \ast O(1) \\
&P \Rightarrow E_P \\
&Q \Rightarrow L_c \\
\hline
&Q \Rightarrow L_r
\end{align*}
\]

Theorem IPR $D$ can be computed in time $O(n)$, with $n = |D|$. 
Interprocedural Reachability: Rules

Read “$P \Rightarrow L$” as “in procedure $P$ label $L$ can be reached”.

\[
\begin{align*}
\text{proc}(P, B_P, E_P) & \quad O(n) \\
\hline
P \Rightarrow B_P \\
\end{align*}
\]

\[
\begin{align*}
\text{next}(Q, L, L') & \quad O(n) \\
Q \Rightarrow L \quad & \quad \ast O(1) \\
\hline
Q \Rightarrow L' \\
\end{align*}
\]

\[
\begin{align*}
\text{call}(Q, P, L_c, R_r) & \quad O(n) \\
\text{proc}(P, B_P, E_P) \quad & \quad \ast O(1) \\
P \Rightarrow E_P \quad & \quad \ast O(1) \\
Q \Rightarrow L_c \quad & \quad \ast O(1) \\
\hline
Q \Rightarrow L_r \\
\end{align*}
\]
Interprocedural Reachability: Rules

Read “$P \Rightarrow L$” as “in procedure $P$ label $L$ can be reached”.

\[
\text{proc}(P, B_P, E_P) \quad O(n)
\]

\[
P \Rightarrow B_P
\]

\[
\begin{align*}
\text{next}(Q, L, L') & \quad O(n) \\
Q \Rightarrow L & \quad \ast O(1)
\end{align*}
\]

\[
Q \Rightarrow L'
\]

\[
\begin{align*}
\text{call}(Q, P, L_c, R_r) & \quad O(n) \\
\text{proc}(P, B_P, E_P) & \quad \ast O(1) \\
P \Rightarrow E_P & \quad \ast O(1) \\
Q \Rightarrow L_c & \quad \ast O(1)
\end{align*}
\]

\[
Q \Rightarrow L_r
\]

**Theorem** $IPR^*(D)$ can be computed in time $O(n)$, with $n = \|D\|$. 
Proof of the Meta-Complexity Theorem I

**Assumption:** all terms in fully shared form
Proof of the Meta-Complexity Theorem I

Assumption: all terms in fully shared form

Matching: in $O(1)$ (for atoms in rules against atoms in $D$)
Assumption: all terms in fully shared form

Matching: in $O(1)$ (for atoms in rules against atoms in $D$)

**Unary Rules** $A \supset B$: matching of $A$ against each atom in $R(D)$, plus construction of $B$, costs total time $O(|R(D)|)$
Proof of the Meta-Complexity Theorem I

Assumption: all terms in fully shared form

Matching: in $O(1)$ (for atoms in rules against atoms in $D$)

Unary Rules $A \supset B$: matching of $A$ against each atom in $R(D)$, plus construction of $B$, costs total time $O(|R(D)|)$

Note: programs not cons-free
Proof of the Meta-Complexity Theorem I

Assumption: all terms in fully shared form

Matching: in $O(1)$ (for atoms in rules against atoms in $D$)

Unary Rules $A \supset B$: matching of $A$ against each atom in $R(D)$, plus construction of $B$, costs total time $O(|R(D)|)$

Note: programs not cons-free

Problem: avoiding $O(|R(D)|^k)$ for rules of length $k$
Data structure for rules $\rho$ of the form $p(X, Y) \land q(Y, Z) \supset r(X, Y, Z)$
Data structure for rules $\rho$ of the form $p(X, Y) \land q(Y, Z) \supset r(X, Y, Z)$
Proof of the Meta-Complexity Theorem II

Data structure for rules $\rho$ of the form $p(X, Y) \land q(Y, Z) \supset r(X, Y, Z)$

Upon adding a fact $p(e, t)$, fire all $r(e, t, z)$, for $z$ on the $q$-list of $A[t]$. 
Data structure for rules $\rho$ of the form $p(X, Y) \land q(Y, Z) \supset r(X, Y, Z)$

Upon adding a fact $p(e, t)$, fire all $r(e, t, z)$, for $z$ on the $q$-list of $A[t]$. The inference system can be transformed (maintaining $\pi$) so that it contains only unary rules and binary rules of the form $\rho$. 
- memory consumption often much smaller
Remarks

- memory consumption often much smaller
- if $R^*(D)$ infinite, consider $R^*(D) \cap \text{atoms} (\text{subterms}(D))$
  $\Rightarrow$ concept of local inference systems (Givan, McAllester 1993)
Remarks

- memory consumption often much smaller
- if $R^*(D)$ infinite, consider $R^*(D) \cap \text{atoms(subterms}(D))$
  $\Rightarrow$ concept of local inference systems (Givan, McAllester 1993)
- in the presence of transitivity laws, complexity is in $\Omega(n^3)$
II. Redundancy, Deletion, and Priorities
• redundant information causes inefficiency

\[ D = \{\ldots, \text{dist}(x) \leq d, \text{dist}(x) \leq d', d' < d, \ldots\} \]

\( \Rightarrow \) delete \( \text{dist}(x) \leq d \)
Removal of Redundant Information

- redundant information causes inefficiency

\[ D = \{ \ldots, \text{dist}(x) \leq d, \text{dist}(x) \leq d', d' < d, \ldots \} \]

\[ \Rightarrow \text{delete dist}(x) \leq d \]

- Notation: antecedents to be deleted in parenthesis [\ldots]

\[ \ldots, [A], \ldots, A', \ldots, [A''], \ldots \supset B \]
Removal of Redundant Information

- redundant information causes inefficiency

\[ D = \{\ldots, \text{dist}(x) \leq d, \text{dist}(x) \leq d', d' < d, \ldots\} \]

\[ \Rightarrow \text{delete } \text{dist}(x) \leq d \]

- Notation: antecedents to be deleted in parenthesis [\ldots]

\[ \ldots, [A], \ldots, A', \ldots, [A''], \ldots \supset B \]

- in the presence of deletion, computations are nondeterministic:

\[ P \supset Q \quad [Q] \supset S \quad [Q] \supset W \]

\[ \Rightarrow \text{either } S \text{ or } W \text{ can be derived, but not both} \]
removal of redundant information causes inefficiency

\[ D = \{\ldots, \text{dist}(x) \leq d, \text{dist}(x) \leq d', d' < d, \ldots \} \]

\[ \Rightarrow \text{delete } \text{dist}(x) \leq d \]

- Notation: antecedents to be deleted in parenthesis \([\ldots]\)

\[ \ldots, [A], \ldots, A', \ldots, [A''], \ldots \supset B \]

- in the presence of deletion, computations are nondeterministic:

\[ P \supset Q \quad [Q] \supset S \quad [Q] \supset W \]

\[ \Rightarrow \text{either } S \text{ or } W \text{ can be derived, but not both} \]

- non-determinism don’t-care and/or restricted by priorities
rules can have antecedents to be deleted after firing
• rules can have antecedents to be deleted after firing
• priorities assigned to rule schemes
Logic Programs with Priorities and Deletion

- rules can have antecedents to be deleted after firing
- priorities assigned to rule schemes
- computation states $S$ contain positive and negative (deleted) atoms
Logic Programs with Priorities and Deletion

- rules can have antecedents to be deleted after firing
- priorities assigned to rule schemes
- computation states $S$ contain positive and negative (deleted) atoms
  - $A$ visible in $S$ if $A \in S$ and $\neg A \not\in S$ (deletions are permanent)
• rules can have antecedents to be deleted after firing
• priorities assigned to rule schemes
• computation states $S$ contain positive and negative (deleted) atoms
• $A$ visible in $S$ if $A \in S$ and $\neg A \notin S$ (deletions are permanent)
• $\Gamma \triangleright B$ applicable in $S$ if
  – each atom in $\Gamma$ is visible in $S$, and
  – rule application changes $S$ (by adding $B$ or some $\neg A$)
rules can have antecedents to be deleted after firing

- priorities assigned to rule schemes

- computation states \( S \) contain positive and negative (deleted) atoms

- \( A \) visible in \( S \) if \( A \in S \) and \( \neg A \notin S \) (deletions are permanent)

- \( \Gamma \supset B \) applicable in \( S \) if
  - each atom in \( \Gamma \) is visible in \( S \), and
  - rule application changes \( S \) (by adding \( B \) or some \( \neg A \))

- \( S \) visible to a rule if no higher-priority rule is applicable in \( S \)
- Rules can have antecedents to be deleted after firing.
- Priorities assigned to rule schemes.
- Computation states \( S \) contain positive and negative (deleted) atoms.
- \( A \) visible in \( S \) if \( A \in S \) and \( \neg A \notin S \) (deletions are permanent).
- \( \Gamma \supset B \) applicable in \( S \) if
  - each atom in \( \Gamma \) is visible in \( S \), and
  - rule application changes \( S \) (by adding \( B \) or some \( \neg A \)).
- \( S \) visible to a rule if no higher-priority rule is applicable in \( S \).
- Computations are maximal sequences of applications of visible rules.
- rules can have antecedents to be deleted after firing
- priorities assigned to rule schemes
- computation states $S$ contain positive and negative (deleted) atoms
- $A$ visible in $S$ if $A \in S$ and $\neg A \not\in S$ (deletions are permanent)
- $\Gamma \supset B$ applicable in $S$ if
  - each atom in $\Gamma$ is visible in $S$, and
  - rule application changes $S$ (by adding $B$ or some $\neg A$)
- $S$ visible to a rule if no higher-priority rule is applicable in $S$
- computations are maximal sequences of applications of visible rules
- the final state of a computation starting with $D$ is called an $(R)$-saturation of $D$
Let $\mathcal{C} = S_0, S_1, \ldots, S_T$ be a computation.

**Prefix firing in $\mathcal{C}$**:

- Pair $(r \sigma, i)$ such that for some $0 \leq t < T$:
  - $r = A_1 \land \ldots \land A_i \land \ldots \land A_n \supset A_0 \in R$
  - $S_t$ visible to $r$
  - $A_j \sigma$ visible in $S_t$, for $1 \leq j \leq i$
Let $\mathcal{C} = S_0, S_1, \ldots, S_T$ be a computation.

Prefix firing in $\mathcal{C}$: pair $(r\sigma, i)$ such that for some $0 \leq t < T$:
- $r = A_1 \land \ldots \land A_i \land \ldots \land A_n \supset A_0 \in R$
- $S_t$ visible to $r$
- $A_j\sigma$ visible in $S_t$, for $1 \leq j \leq i$

Prefix count: $\pi_R(D) = \max\{|\text{p.f.}(\mathcal{C})| \mid \mathcal{C} \text{ a computation from } D\}$
Let $C = S_0, S_1, \ldots, S_T$ be a computation.

Prefix firing in $C$: pair $(r\sigma, i)$ such that for some $0 \leq t < T$:
- $r = A_1 \land \ldots \land A_i \land \ldots \land A_n \supset A_0 \in R$
- $S_t$ visible to $r$
- $A_j\sigma$ visible in $S_t$, for $1 \leq j \leq i$

Prefix count: $\pi_R(D) = \max\{|\p.f.(C)| \mid C \text{ a computation from } D\}$

**Theorem** [Ganzinger/McAllester 2001] Let $R$ be an inference system such that $R(D)$ is finite. Then some $R(D)$ can be computed in time $O(\|D\| + \pi_R(D))$. 
Second Meta-Complexity Theorem

Let $\mathcal{C} = S_0, S_1, \ldots, S_T$ be a computation.

**Prefix firing in $\mathcal{C}$:** pair $(r\sigma, i)$ such that for some $0 \leq t < T$:
- $r = A_1 \land \ldots \land A_i \land \ldots \land A_n \supset A_0 \in R$
- $S_t$ visible to $r$
- $A_j\sigma$ visible in $S_t$, for $1 \leq j \leq i$

**Prefix count:** $\pi_R(D) = \max\{|p.f.((C))| \mid \mathcal{C} \text{ a computation from } D\}$

**THEOREM [Ganzinger/McAllester 2001]** Let $R$ be an inference system such that $R(D)$ is finite. Then some $R(D)$ can be computed in time $O(\|D\| + \pi_R(D))$.

**Proof** as before, but also using constant-length priority queues.
Second Meta-Complexity Theorem

Let $C = S_0, S_1, \ldots, S_T$ be a computation.

Prefix firing in $C$: pair $(r\sigma, i)$ such that for some $0 \leq t < T$:
- $r = A_1 \wedge \ldots \wedge A_i \wedge \ldots \wedge A_n \supset A_0 \in R$
- $S_t$ visible to $r$
- $A_j \sigma$ visible in $S_t$, for $1 \leq j \leq i$

Prefix count: $\pi_R(D) = \max\{|p.f.(C)| \mid C \text{ a computation from } D\}$

Theorem [Ganzinger/McAllester 2001] Let $R$ be an inference system such that $R(D)$ is finite. Then some $R(D)$ can be computed in time $O(\|D\| + \pi_R(D))$.

Proof as before, but also using constant-length priority queues

Note: again prefix firings count only once; priorities are for free
Union-Find

\[
\begin{align*}
\text{find}(x) & \quad x \Rightarrow! y & \quad x \Rightarrow y \\
(\text{Ref1}) & \quad y \Rightarrow z & \quad x \Rightarrow z \\
(\text{N}) & \quad x \Rightarrow! z & \quad (\text{Comm}) \quad \text{union}(y, z)
\end{align*}
\]
We are interested in $x \dot{=} y$ defined as $\exists z (x \Rightarrow! z \land y \Rightarrow! z)$
Union-Find

\[
\begin{align*}
\text{find}(x) & \quad x \Rightarrow! y \quad O(n^2) \\
(\text{Ref}) & \quad y \Rightarrow z \quad * \quad O(n) \\
\text{find}(y) & \quad x \Rightarrow y \quad O(n^2) \\
\text{(N)} & \quad x \Rightarrow z \quad * \quad O(n) \\
\text{find}(x), & \quad \text{union}(x, y) \\
\text{find}(y) & \quad \text{union}(y, z) \\
\end{align*}
\]

Naive Knuth/Bendix completion
Naive Knuth/Bendix completion
+ normalization (eager path compression)
Union-Find

\[
\begin{align*}
\text{find}(x) &
\quad \text{(Ref1)} \quad \text{[} x \Rightarrow ! y \text{]} \quad O(n \log n) \\
\quad x \Rightarrow ! x
\end{align*}
\]

\[
\begin{align*}
\text{weight}(x, 1) &
\quad \text{(N)} \quad \text{[} x \Rightarrow ! z \text{]} \quad \text{O}(1) \\
\quad x \Rightarrow ! z
\end{align*}
\]

\[
\begin{align*}
\text{union}(x, y) &
\quad \text{(Init)} \quad \text{[} \text{union}(x, y) \text{]} \\
\quad \text{find}(x), \text{find}(y)
\end{align*}
\]

\[
\begin{align*}
\text{union}(x, y) &
\quad \text{(Triv)} \quad \text{[} \text{union}(x, y) \text{]} \\
\quad x \Rightarrow ! z \\
\quad y \Rightarrow ! z
\end{align*}
\]

\[
\begin{align*}
\text{union}(x, y) &
\quad \text{(Comm)} \quad \text{union}(y, z)
\end{align*}
\]

\[
\begin{align*}
\text{union}(x, y) &
\quad \text{(Orient)} \quad \text{[} \text{union}(x, y) \text{]} \\
\quad z_1 \Rightarrow z_2 \\
\quad \text{weight}(z_2, w_1 + w_2)
\end{align*}
\]

\[
\begin{align*}
\text{Naive Knuth/Bendix completion} &
\quad + \text{ symmetric variant of (Orient)}
\end{align*}
\]

\[
\begin{align*}
\text{Naive Knuth/Bendix completion} &
\quad + \text{ normalization (eager path compression)} \quad + \text{ logarithmic merge}
\end{align*}
\]
Extension to congruence closure: 7 more rules, guaranteed optimal complexity $O(m + n \log n)$, where $m = |\text{union assertions}|$, $n = |(\text{sub})\text{terms}|$
Congruence Closure for Ground Horn Clauses

Extension to congruence closure: 7 more rules, guaranteed optimal complexity $O(m + n \log n)$, where $m = |\text{union assertions}|$, $n = |\text{(sub)terms}|$

Extension to ground Horn clauses with equality: 13 more rules
Extension to congruence closure: 7 more rules, guaranteed optimal complexity $O(m + n \log n)$, where $m = |\text{union assertions}|$, $n = |(\text{sub})\text{terms}|$

Extension to ground Horn clauses with equality: 13 more rules

**Theorem** [Ganzinger/McAllester 01] Satisfiability of a set $D$ of ground Horn clauses with equality can be decided in time $O(\|D\| + n \log n + \min(m \log n, n^2))$ where $m$ is the number of antecedents and input clauses and $n$ is the number of terms. This is optimal ($= O(\|D\|)$) whenever $m$ is in $\Omega(n^2)$. 
Congruence Closure for Ground Horn Clauses

Extension to congruence closure: 7 more rules, guaranteed optimal complexity $O(m + n \log n)$, where $m = |\text{union assertions}|$, $n = |\text{(sub)terms}|$

Extension to ground Horn clauses with equality: 13 more rules

**Theorem** [Ganzinger/McAllester 01] Satisfiability of a set $D$ of ground Horn clauses with equality can be decided in time $O(\|D\| + n \log n + \min(m \log n, n^2))$ where $m$ is the number of antecedents and input clauses and $n$ is the number of terms. This is optimal ($= O(\|D\|)$) whenever $m$ is in $\Omega(n^2)$.

**Logic View:** We can (partly) deal with logic programs with equality
Congruence Closure for Ground Horn Clauses

Extension to congruence closure: 7 more rules, guaranteed optimal complexity $O(m + n \log n)$, where $m = |\text{union assertions}|$, $n = |\text{(sub)terms}|$

Extension to ground Horn clauses with equality: 13 more rules

**Theorem** [Ganzinger/McAllester 01] Satisfiability of a set $D$ of ground Horn clauses with equality can be decided in time $O(\|D\| + n \log n + \min(m \log n, n^2))$ where $m$ is the number of antecedents and input clauses and $n$ is the number of terms. This is optimal ($= O(\|D\|)$) whenever $m$ is in $\Omega(n^2)$.

Logic View: We can (partly) deal with logic programs with equality

Applications: several program analysis algorithms (Steensgaard, Henglein)
Let $\succ$ a well-founded ordering on ground atoms.

**Definition**  $A$ is **redundant** in $S$ (denoted $A \in Red(S)$) whenever $A_1, \ldots, A_n \models_R A$, with $A_i$ in $S$ such that $A_i \prec A$. 
Let $\succ$ a well-founded ordering on ground atoms.

**Definition** $A$ is **redundant** in $S$ (denoted $A \in \text{Red}(S)$) whenever $A_1, \ldots, A_n \models_R A$, with $A_i$ in $S$ such that $A_i \prec A$.

**Properties** stable under enrichments and under deletion of redundant atoms
Formal Notion of Redundancy

Let $\succ$ a well-founded ordering on ground atoms.

**Definition**  $A$ is redundant in $S$ (denoted $A \in \text{Red}(S)$) whenever $A_1, \ldots, A_n \models R A$, with $A_i$ in $S$ such that $A_i \prec A$.

**Properties** stable under enrichments and under deletion of redundant atoms

**Definition**  $S$ is saturated up to redundancy wrt $R$ if $R(S \setminus \text{Red}(S)) \subseteq S \cup \text{Red}(S)$. 
Let $\succ$ a well-founded ordering on ground atoms.

**Definition** $A$ is redundant in $S$ (denoted $A \in Red(S)$) whenever $A_1, \ldots, A_n \models_R A$, with $A_i$ in $S$ such that $A_i \prec A$.

**Properties** stable under enrichments and under deletion of redundant atoms

**Definition** $S$ is saturated up to redundancy wrt $R$ if $R(S \setminus Red(S)) \subseteq S \cup Red(S)$.

**Theorem** If deletion is based on redundancy then the result of every computation is saturated wrt $R$ up to redundancy.
Formal Notion of Redundancy

Let $\succ$ a well-founded ordering on ground atoms.

**Definition** $A$ is redundant in $S$ (denoted $A \in Red(S)$) whenever $A_1, \ldots, A_n \models_R A$, with $A_i$ in $S$ such that $A_i \prec A$.

**Properties** stable under enrichments and under deletion of redundant atoms

**Definition** $S$ is saturated up to redundancy wrt $R$ if $R(S \setminus Red(S)) \subseteq S \cup Red(S)$.

**Theorem** If deletion is based on redundancy then the result of every computation is saturated wrt $R$ up to redundancy.

**Corollary** Priorities are irrelevant logically $\Rightarrow$ choose them so as to minimize prefix firings
Deletions based on Redundancy

Criterion: If

\[ r = [A_1], \ldots, [A_k], B_1, \ldots, B_m \supset B \]

and if \( S \cup \{A_1\sigma, \ldots, A_k\sigma, B_1\sigma, \ldots, B_m\sigma\} \) is visible to \( r \) then

\[ A_i\sigma \in Red(S \cup \{B_1\sigma, \ldots, B_m\sigma, B\sigma\}). \]
Deletions based on Redundancy

Criterion: If

\[ r = [A_1], \ldots, [A_k], B_1, \ldots, B_m \supset B \]

and if \( S \cup \{A_1\sigma, \ldots, A_k\sigma, B_1\sigma, \ldots, B_m\sigma\} \) is visible to \( r \) then

\[ A_i\sigma \in \text{Red}(S \cup \{B_1\sigma, \ldots, B_m\sigma, B\sigma\}). \]

Union-find example: not so easy to check, need proof orderings à la Bachmair and Dershowitz
Deletions based on Redundancy

Criterion: If

\[ r = [A_1], \ldots, [A_k], B_1, \ldots, B_m \supset B \]

and if \( S \cup \{A_1\sigma, \ldots, A_k\sigma, B_1\sigma, \ldots, B_m\sigma\} \) is visible to \( r \) then

\[ A_i\sigma \in \text{Red}(S \cup \{B_1\sigma, \ldots, B_m\sigma, B\sigma\}). \]

Union-find example: not so easy to check, need proof orderings à la Bachmair and Dershowitz

Note: redundancy should also be efficiently decidable
III. Instance-based Priorities
(Init) \[ \text{dist(src)} \leq 0 \]

(Upd) \[ \begin{align*}
\text{dist}(x) & \leq d' \\
\text{dist}(x) & \leq d' \\
d' & < d
\end{align*} \]

(Add) \[ \text{dist}(y) \leq c + d \]

[dist(x) \leq d]

\[ \text{dist}(x) \leq d \]

\[ x \xrightarrow{c} y \]
Shortest Paths

(Init) \[ \text{dist(src)} \leq 0 \]

(Upd) \[ [\text{dist}(x) \leq d] \]
\[ \text{dist}(x) \leq d' \]
\[ d' < d \]
\[ \text{dist}(x) \leq d \]
\[ x \xrightarrow{c} y \]

(Add) \[ \text{dist}(y) \leq c + d \]

Correctness: obvious; deletion is based on redundancy
## Shortest Paths

\[
\begin{align*}
(\text{Init}) \quad \text{dist(src)} & \leq 0 \\
\text{(Upd)} \quad \text{dist}(x) & \leq d' \\
\text{(Add)} \quad \text{dist}(y) & \leq c + d
\end{align*}
\]

### Correctness:

obvious; deletion is based on redundancy

### Priorities (Dijkstra):
always choose an instance of (Add) where \(d\) is minimal \(\implies\) allow for instance-based rule priorities

\((\text{Init}) > (\text{Upd}) > (\text{Add})[n/d] > (\text{Add})[m/d]\), for \(m > n\)
Shortest Paths

Correctness: obvious; deletion is based on redundancy

Priorities (Dijkstra): always choose an instance of (Add) where $d$ is minimal \Rightarrow allow for instance-based rule priorities

Prefix firing count: $O(|E|)$, but Dijkstra’s algorithm runs in time $O(|E| + |V| \log |V|)$ \Rightarrow one cannot expect a linear-time meta-complexity theorem for instance-based priorities
Minimum Spanning Tree

**Basis:** Union-find module
Minimum Spanning Tree

Basis: Union-find module

\[
\begin{align*}
[x \leftrightarrow y] \\
x & \rightarrow! z \\
y & \rightarrow! z \\
(T) & \overset{\text{(Del)}}{\longrightarrow} \\
\text{mst}(x, c, y) & \overset{\text{(Add)}}{\longrightarrow} \\
\text{union}(x, y)
\end{align*}
\]
Minimum Spanning Tree

Basis: Union-find module

\[
\begin{align*}
[x \leftrightarrow y] \\
x &\Rightarrow! z \\
y &\Rightarrow! z \\
(x \leftrightarrow y] &\text{(Add)} \\
T &\text{mst}(x, c, y) \\
T &\text{union}(x, y)
\end{align*}
\]

Priorities: (here needed for correctness)

\[
\text{union–find} > (\text{Del}) > (\text{Add})[n/c] > (\text{Add})[m/c], \text{ for } m > n
\]
Minimum Spanning Tree

**Basis:** Union-find module

\[
\begin{align*}
\{x \leftrightarrow y\} & \quad \text{(Add)} \quad \text{mst}(x, c, y) \\
\text{union}(x, y) & \quad \text{(Del)} \quad \text{mst}(x, c, y) \\
\end{align*}
\]

**Priorities:** (here needed for correctness)

union-find > (Del) > (Add)[n/c] > (Add)[m/c], for \( m > n \)

**Prefix firing count:** \( O(|E| + |V| \log |V|) \)
3rd Meta-Complexity Theorem

Programs: as before but priorities of rule instances depend on first atom in antecedent and can be computed from the atom in constant time
3rd Meta-Complexity Theorem

Programs: as before but priorities of rule instances depend on first atom in antecedent and can be computed from the atom in constant time

**Theorem** [in preparation] Let $R$ be an inference system such that $R^*(D)$ is finite. Then some $R(D)$ can be computed in time $O(\|D\| + \pi_R(D) \log p)$ where $p$ is the number of different priorities assigned to atoms in $R^*(D)$. 
3rd Meta-Complexity Theorem

Programs: as before but priorities of rule instances depend on first atom in antecedent and can be computed from the atom in constant time

**Theorem**  [in preparation] Let $R$ be an inference system such that $R^*(D)$ is finite. Then some $R(D)$ can be computed in time $O(\|D\| + \pi_R(D) \log p)$ where $p$ is the number of different priorities assigned to atoms in $R^*(D)$.

**Corollary**  2nd meta-complexity theorem is a special case
3rd Meta-Complexity Theorem

Programs: as before but priorities of rule instances depend on first atom in antecedent and can be computed from the atom in constant time

Theorem [in preparation] Let $R$ be an inference system such that $R^*(D)$ is finite. Then some $R(D)$ can be computed in time $O(\|D\| + \pi_R(D) \log p)$ where $p$ is the number of different priorities assigned to atoms in $R^*(D)$.

Corollary 2nd meta-complexity theorem is a special case

Proof technically involved; uses priority queues with log time operations; memory usage worse
Further Issues and Questions

- a concept for modules needed (cf. IJCAR paper)
- deletion not always based on redundancy
- “real equality” (on the meta-level)
- how far do we get?
- is deduction without deletion inherently less efficient?
- implementation of instance-based priorities with schematic priorities?
- bounds for memory consumption
- improved meta-complexity theorems
Further Issues and Questions

- a concept for modules needed (cf. IJCAR paper)
- deletion not always based on redundancy
- “real equality” (on the meta-level)
- how far do we get?
- is deduction without deletion inherently less efficient?
- implementation of instance-based priorities with schematic priorities?
- bounds for memory consumption
- improved meta-complexity theorems
Further Issues and Questions

- a concept for modules needed (cf. IJCAR paper)
- deletion not always based on redundancy
- “real equality” (on the meta-level)
- how far do we get?
- is deduction without deletion inherently less efficient?
- implementation of instance-based priorities with schematic priorities?
- bounds for memory consumption
- improved meta-complexity theorems
Further Issues and Questions

- a concept for modules needed (cf. IJCAR paper)
- deletion not always based on redundancy
- “real equality” (on the meta-level)
- **how far do we get?**
- is deduction without deletion inherently less efficient?
- implementation of instance-based priorities with schematic priorities?
- bounds for memory consumption
- improved meta-complexity theorems
Further Issues and Questions

- a concept for modules needed (cf. IJCAR paper)
- deletion not always based on redundancy
- “real equality” (on the meta-level)
- how far do we get?
- is deduction without deletion inherently less efficient?
- implementation of instance-based priorities with schematic priorities?
- bounds for memory consumption
- improved meta-complexity theorems
Further Issues and Questions

- a concept for modules needed (cf. IJCAR paper)
- deletion not always based on redundancy
- “real equality” (on the meta-level)
- how far do we get?
- is deduction without deletion inherently less efficient?
- implementation of instance-based priorities with schematic priorities?
- bounds for memory consumption
- improved meta-complexity theorems
Further Issues and Questions

- a concept for modules needed (cf. IJCAR paper)
- deletion not always based on redundancy
- “real equality” (on the meta-level)
- how far do we get?
- is deduction without deletion inherently less efficient?
- implementation of instance-based priorities with schematic priorities?
- bounds for memory consumption
- improved meta-complexity theorems
Further Issues and Questions

- A concept for modules needed (cf. IJCAR paper)
- Deletion not always based on redundancy
- "Real equality" (on the meta-level)
- How far do we get?
- Is deduction without deletion inherently less efficient?
- Implementation of instance-based priorities with schematic priorities?
- Bounds for memory consumption
- Improved meta-complexity theorems