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
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
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:

- \( s(u) \)
- \( e(u, v) \)
- \( r(u) \)
- \( r(v) \)
Reachability in Graphs

Database:

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

Inference system:

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

Clause notation: \( s(u) \supset r(u) \quad r(u), e(u, v) \supset r(v) \)
Reachability in Graphs

Database:

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

Inference system:

\[
\frac{s(u)}{r(u)} \quad \frac{r(u)}{r(v)} \\
\frac{e(u, v)}{r(v)}
\]

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

Closure:

\[ R^*(D) = D \cup \{r(u) \mid u \text{ reachable from a source}\} \]
Example
Example

Database

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

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

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
First Meta-Complexity Theorem

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

Prefix firings:

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

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

prefix firings:

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

**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:**

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

**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 \( O(\|D\| + \|R\|) \).

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

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>$s(u)$</td>
<td>$r(u)$</td>
</tr>
<tr>
<td>$r(u)$</td>
<td>$r(v)$</td>
</tr>
<tr>
<td>$e(u,v)$</td>
<td></td>
</tr>
</tbody>
</table>
Reachability in Graphs

\[ s(u) \quad O(|V|) \quad r(u) \quad O(|V|) \]

\[ e(u, v) \quad r(u) \quad r(v) \]
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”.

\[
\begin{array}{c}
\text{proc}(P, B_P, E_P) \\
\hline
P \Rightarrow B_P
\end{array}
\]

\[
\begin{array}{c}
\text{next}(Q, L, L') \\
Q \Rightarrow L \\
\hline
Q \Rightarrow L'
\end{array}
\]

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

Read “P ⇒ L” as “in procedure P label L can be reached”.

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

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

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

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

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

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

$$P \Rightarrow B_P$$

$$\text{next}(Q, L, L') \quad O(n)$$

$$Q \Rightarrow L \quad \ast O(1)$$

$$Q \Rightarrow L'$$

$$\text{call}(Q, P, L_c, R_r)$$

$$\text{proc}(P, B_P, E_P)$$

$$P \Rightarrow E_P$$

$$Q \Rightarrow L_c$$

$$Q \Rightarrow L_r$$

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”.

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

**Theorem** \( IPR^*(D) \) can be computed in time \( O(n) \), with \( n = \|D\| \).
Assumption: all terms in fully shared form
Assumption: all terms in fully shared form

Matching: in $O(1)$ (for atoms in rules against atoms in $D$)
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 $O(|R(D)|)$
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
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 $p(X, Y) \land q(Y, Z) \supset r(X, Y, Z)$
Proof of the Meta-Complexity Theorem II

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

\[
\begin{align*}
\rho[Y] & \quad \rho[Y] \\
p\text{-list of } \rho[t] & \quad q\text{-list of } \rho[t]
\end{align*}
\]

\begin{align*}
p(a, t) & \quad p(b, t) \\
p(c, t) & \quad p(d, t) \\
p(e, t) & \quad \text{were not in the diagram}
\end{align*}

\begin{align*}
q(t, u) & \quad q(t, v) \\
q(t, w) & \quad q(t, s)
\end{align*}
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 $A[t]$. 
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 $A[t]$. The inference system can be transformed (maintaining $\pi$) so that it contains only unary rules and binary rules of the form $\rho$. 
Remarks

- memory consumption often much smaller
Remarks

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

• in the presence of transitivity laws, complexity is in $\Omega(n^3)$
II. Redundancy, Deletion, and Priorities
• redundant information causes inefficiency

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

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

- redundant information causes inefficiency

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

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

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

- 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

- 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 \[\ldots\]

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

- 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
• 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)
$\Gamma \supset B$ applicable in $S$ if
- each atom in $\Gamma$ is visible in $S$, and
- rule application changes $S$ (by adding $B$ or some $\neg A$)
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 $S$
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
• 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: $\pi_R(D) = \max\{|p.f.(\mathcal{C})| \mid \mathcal{C} \text{ a computation from } D\}$
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 $1 \leq j \leq i$

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

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

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

**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

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

<table>
<thead>
<tr>
<th>Operation</th>
<th>Rule</th>
<th>Equivalent</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>find(x)</code></td>
<td>(Ref1)</td>
<td><code>x \Rightarrow! x</code></td>
</tr>
<tr>
<td><code>x \Rightarrow! y</code></td>
<td>(N)</td>
<td><code>x \Rightarrow! z</code></td>
</tr>
<tr>
<td><code>y \Rightarrow z</code></td>
<td>(Comm)</td>
<td><code>x \Rightarrow z</code></td>
</tr>
<tr>
<td><code>x \Rightarrow y</code></td>
<td></td>
<td><code>union(y, z)</code></td>
</tr>
</tbody>
</table>
### Union-Find

<table>
<thead>
<tr>
<th>Rule</th>
<th>Syntax</th>
</tr>
</thead>
<tbody>
<tr>
<td>Refl</td>
<td>( x \Rightarrow! x )</td>
</tr>
<tr>
<td>N</td>
<td>( x \Rightarrow! z )</td>
</tr>
<tr>
<td>Comm</td>
<td>( \text{union}(y, z) )</td>
</tr>
</tbody>
</table>

We are interested in \( x \equiv y \) defined as \( \exists z (x \Rightarrow! z \land y \Rightarrow! z) \)
**Union-Find**

<table>
<thead>
<tr>
<th>Rule</th>
<th>Description</th>
<th>Complexity</th>
</tr>
</thead>
<tbody>
<tr>
<td>find(x)</td>
<td>( x \Rightarrow! y ) ( O(n^2) )</td>
<td>( x \Rightarrow y ) ( O(n^2) )</td>
</tr>
<tr>
<td>(Ref1)</td>
<td>( y \Rightarrow z ) ( *O(n) )</td>
<td>( x \Rightarrow z ) ( *O(n) )</td>
</tr>
<tr>
<td>(N)</td>
<td>( x \Rightarrow! z )</td>
<td>( \text{union}(y, z) )</td>
</tr>
<tr>
<td>(Init)</td>
<td>union(x, y)</td>
<td>union(x, y)</td>
</tr>
<tr>
<td>find(x), find(y)</td>
<td></td>
<td>x \Rightarrow! z_1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>y \Rightarrow! z_2</td>
</tr>
</tbody>
</table>

**Naive Knuth/Bendix completion**
Union-Find

find(x) (Ref1) \[x \Rightarrow! y\]  \(O(n^2)\) \[x \Rightarrow z\]  \(O(1)\) 
\(x \Rightarrow y\)  \(O(n)\) \(x \Rightarrow z\)  \(O(1)\) 

union(x, y) (Triv) \[\text{union}(x, y)\] \[x \Rightarrow! z\]  \[y \Rightarrow! z\] 
\[\text{union}(x, y)\] \[x \Rightarrow! z_1\]  \[y \Rightarrow! z_2\] 

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

\[
\begin{align*}
\text{find}(x) & \rightarrow [x \Rightarrow! y] & O(n \log n) & \quad x \Rightarrow y \\
(\text{Ref1}) & & y \Rightarrow z \quad \ast O(1) & \quad x \Rightarrow z \\
x \Rightarrow! x & \quad \text{(N)} & \quad x \Rightarrow! z \\
\text{weight}(x, 1) & \quad \text{(Comm)} & \quad \text{union}(y, z) \\
\end{align*}
\]

\[
\begin{align*}
\text{union}(x, y) & \rightarrow [\text{union}(x, y)] & \quad x \Rightarrow! z_1, \quad \text{weight}(z_1, w_1) \\
(\text{Init}) & & y \Rightarrow! z_2, [\text{weight}(z_2, w_2)] \\
\text{find}(x), \quad \text{find}(y) & & w_1 \leq w_2 \\
(\text{Triv}) & & [\text{union}(x, y)] \\
\top & & \text{z}_1 \Rightarrow z_2 \\
(\text{Orient}) & & \text{weight}(z_2, w_1 + w_2) \\
+ \text{symmetric variant of (Orient)}
\end{align*}
\]

Naive Knuth/Bendix completion
+ normalization (eager path compression) + logarithmic merge
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 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
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 $\Omega(n^2)$. 
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
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 $A \in \text{Red}(S)$) whenever $A_1, \ldots, A_n \models_R A$, with $A_i$ in $S$ such that $A_i \prec A$. 
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
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 $A \in \text{Red}(S)$) 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 $R(S \setminus \text{Red}(S)) \subseteq S \cup \text{Red}(S)$.

**Theorem** If deletion is based on redundancy then the result of every computation is saturated wrt $R$ up to 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 $R(S \setminus \text{Red}(S)) \subseteq S \cup \text{Red}(S)$.

**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

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

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

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

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

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

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

Correctness: obvious; deletion is based on redundancy
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

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

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

**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

(Init) > (Upd) > (Add)[n/d] > (Add)[m/d], for \( m > n \)

**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*}
\llbracket x \leftrightarrow y \rrbracket \\
x \Rightarrow! z \\
y \Rightarrow! z \\
\text{(Del)} \quad T \\
mst(x, c, y) \\
\text{union}(x, y)
\end{align*}
```
Minimum Spanning Tree

Basis: Union-find module

\[
\begin{align*}
\{x \leftrightarrow y\} \\
x \Rightarrow! z \\
y \Rightarrow! z \\
\text{(Del)} \\
T
\end{align*}
\]

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

Priorities: (here needed for correctness)

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

**Basis:** Union-find module

\[
[x \leftrightarrow y]
\]
\[
x \Rightarrow! z
\]
\[
y \Rightarrow! z
\]
\[
(Del)
\]
\[
T
\]

\[
[x \leftrightarrow y]
\]
\[
(Add)
\]
\[
mst(x, c, y)
\]
\[
union(x, y)
\]

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

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

**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
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