# 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]

- 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



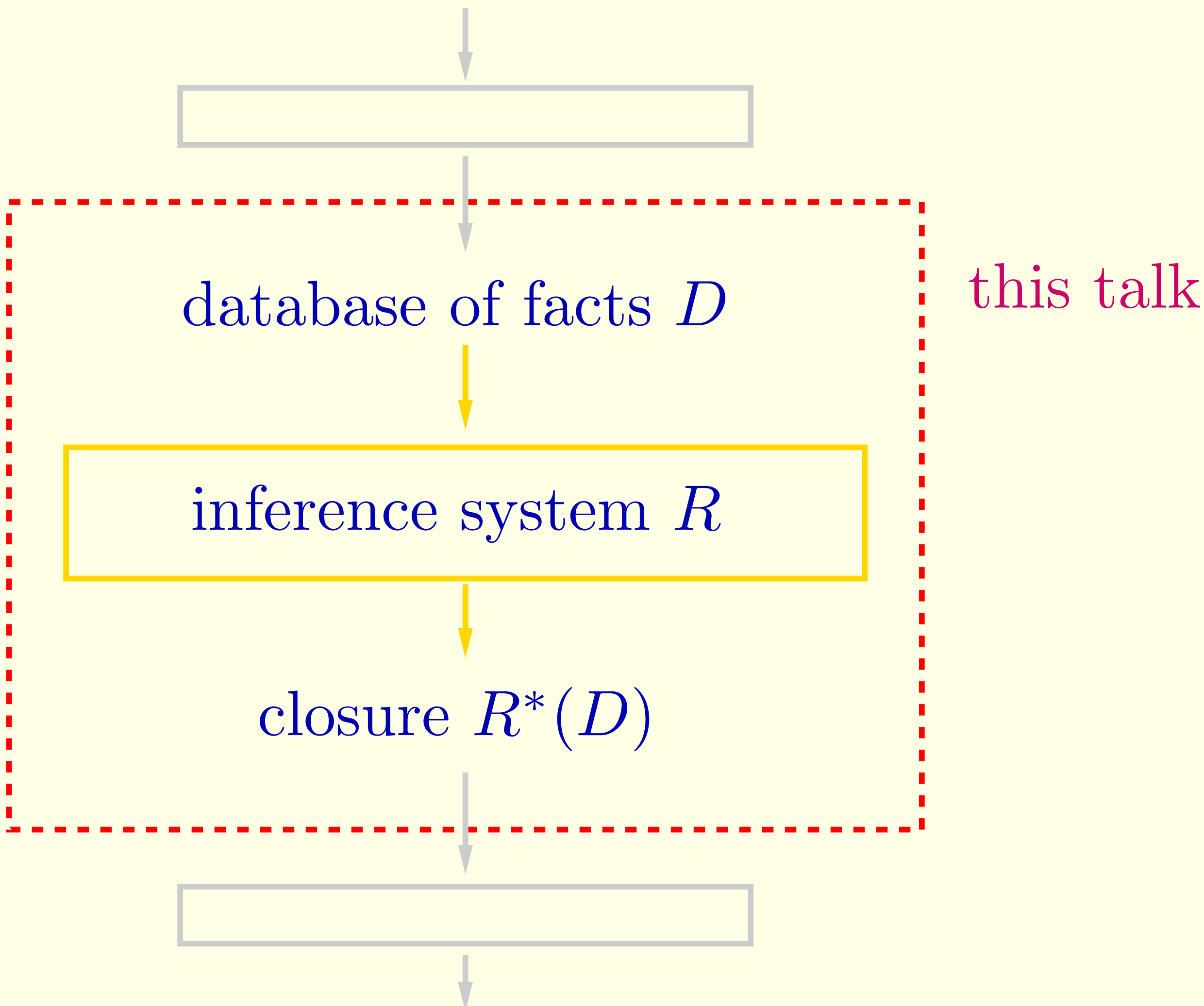
- logic programming of efficient algorithms
- complexity analysis through general meta-complexity theorems
- guaranteed execution time
- logical aspects of fundamental algorithmic paradigms (dynamic programming, union-find, congruence closure, priority queues)
- application to program analysis:
  type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

# Contents

# Contents

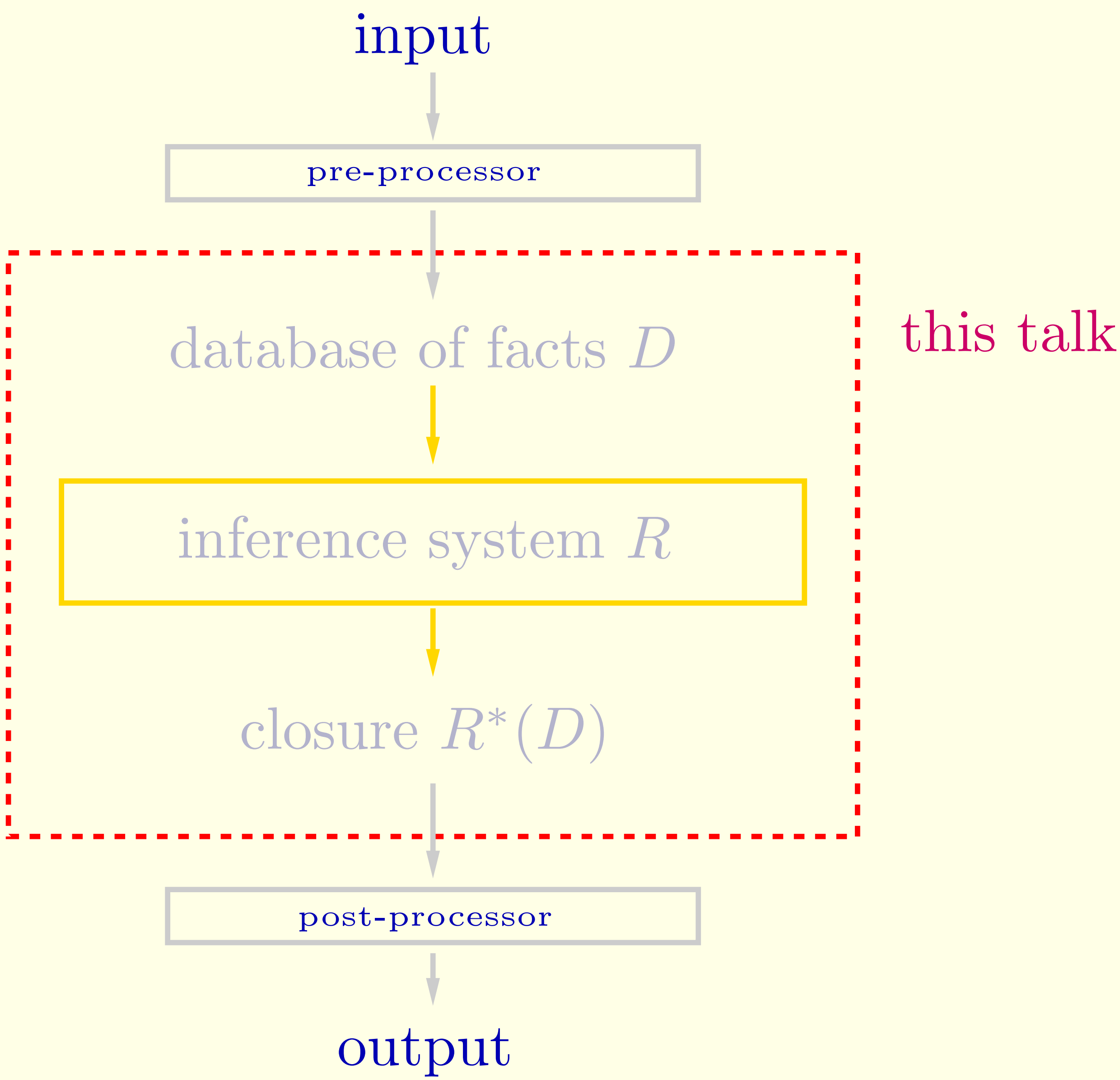
# Contents 

# Paradigm



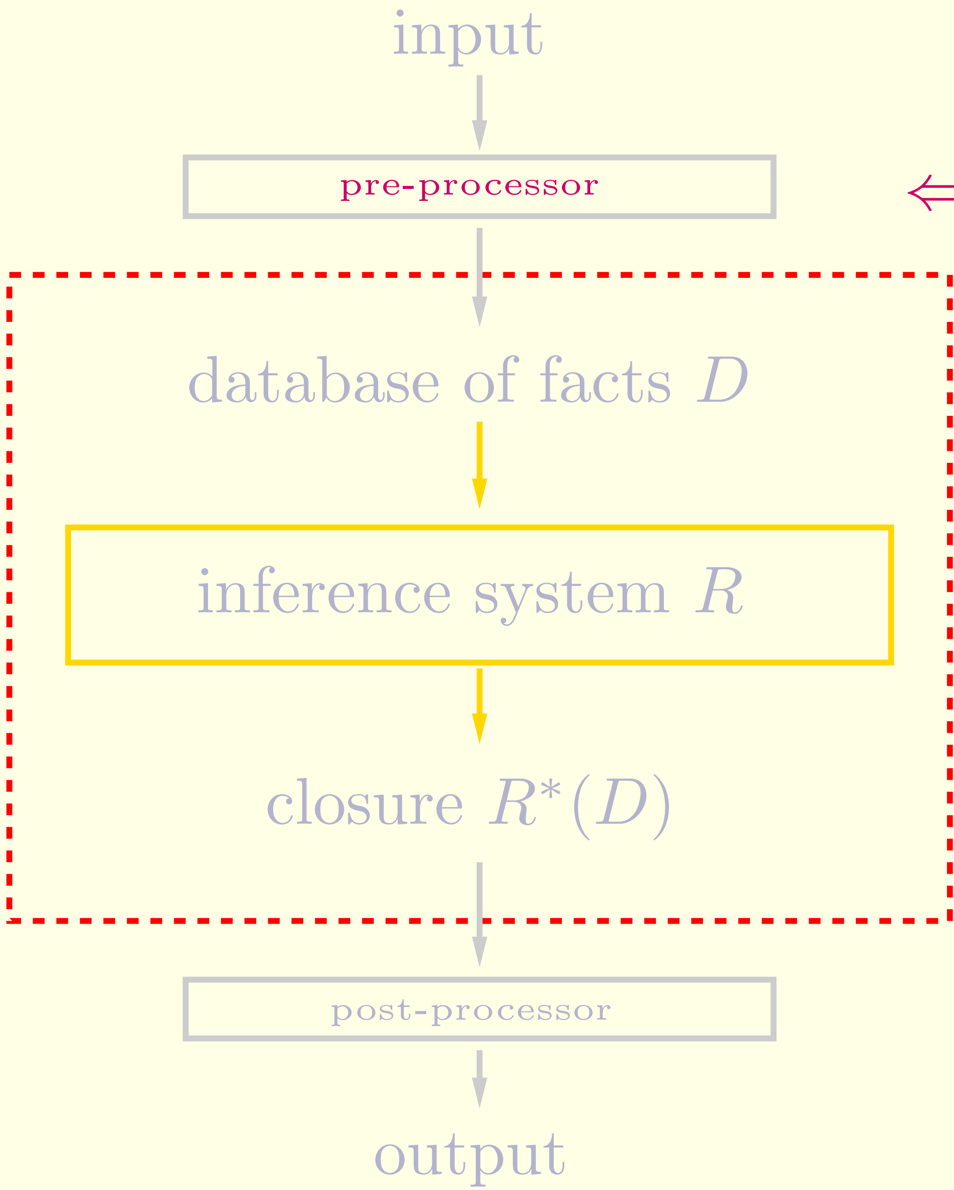
# Paradigm




input
pre-processor
database of facts $D$
this talk
inference system $R$
closure $R^*(D)$
post-processor
output

# Paradigm






$\Longleftarrow$ Paige, Yang 1997

# Reachability in Graphs 

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:

$$\frac{s(u)}{r(u)} \qquad\qquad \frac{\begin{array}{c} r(u) \\ e(u,v) \end{array}}{r(v)}$$

# Reachability in Graphs



Database:

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

Inference system:

$$\frac{s(u)}{r(u)} \qquad\qquad \frac{\begin{array}{c} r(u) \\ e(u,v) \end{array}}{r(v)}$$

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

# Reachability in Graphs 

Database:

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

Inference system:

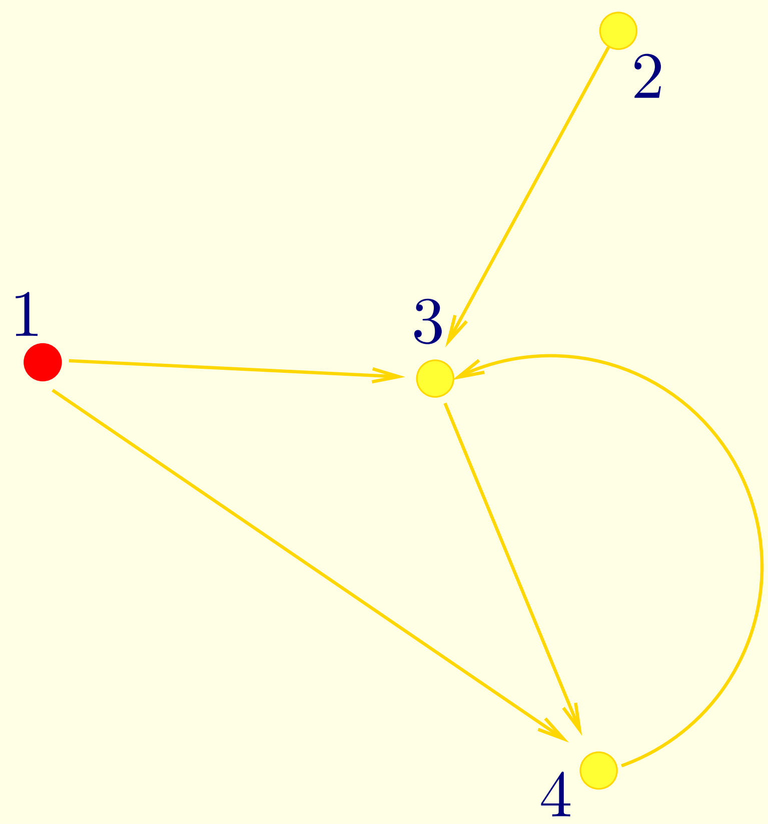
$$\frac{s(u)}{r(u)} \qquad\qquad \frac{\begin{array}{c} r(u) \\ e(u,v) \end{array}}{r(v)}$$

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

Closure:

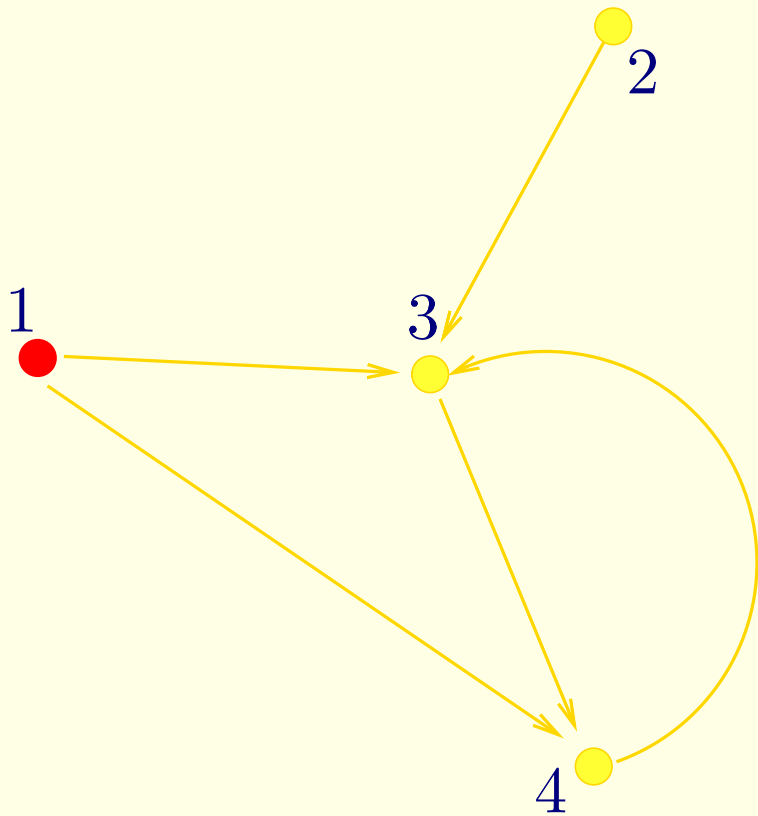
$$R^*(D) = D \cup \{r(u) \mid u \text{ reachable from a source}\}$$

# Example



# Example



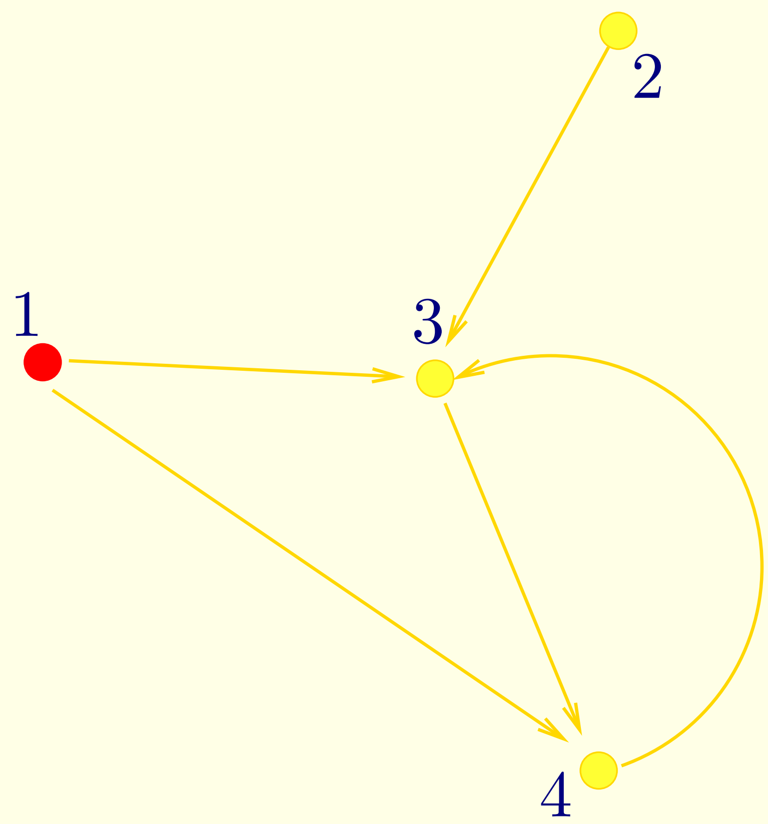



Database

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

# Example



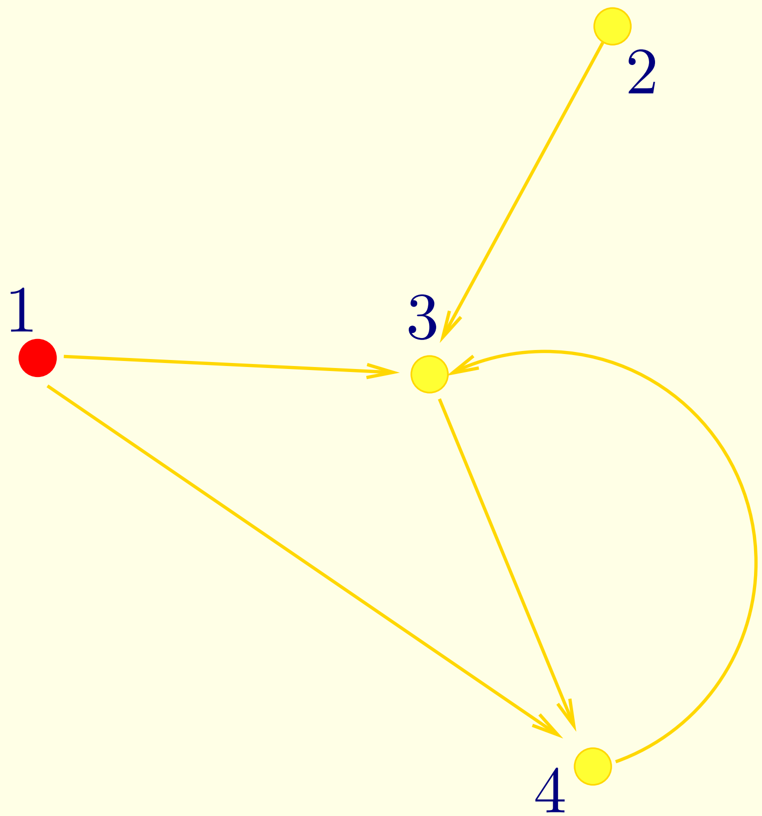



Database

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

# Example



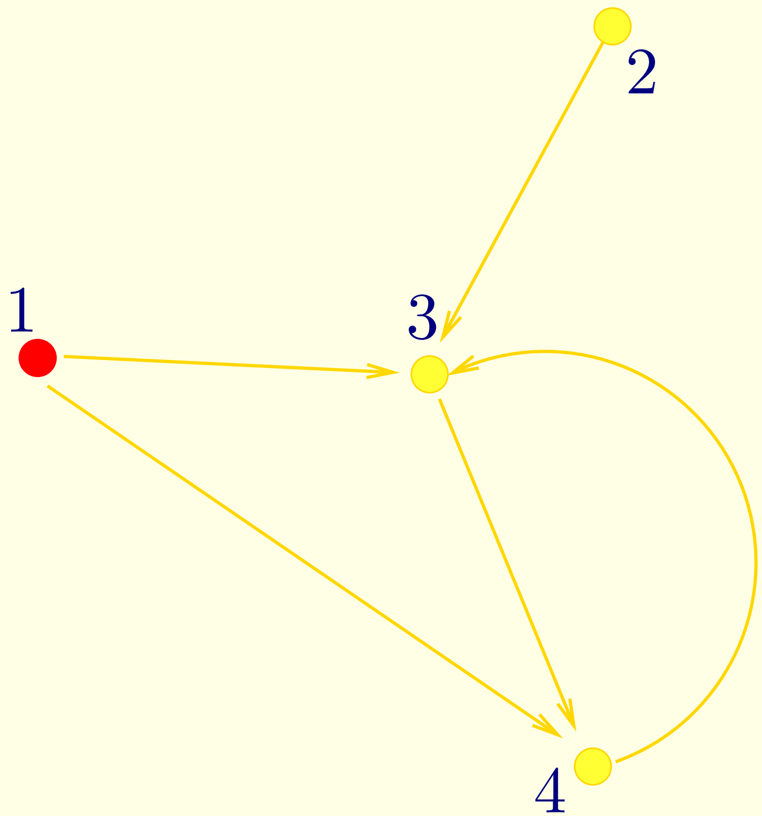



Database

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

# Example



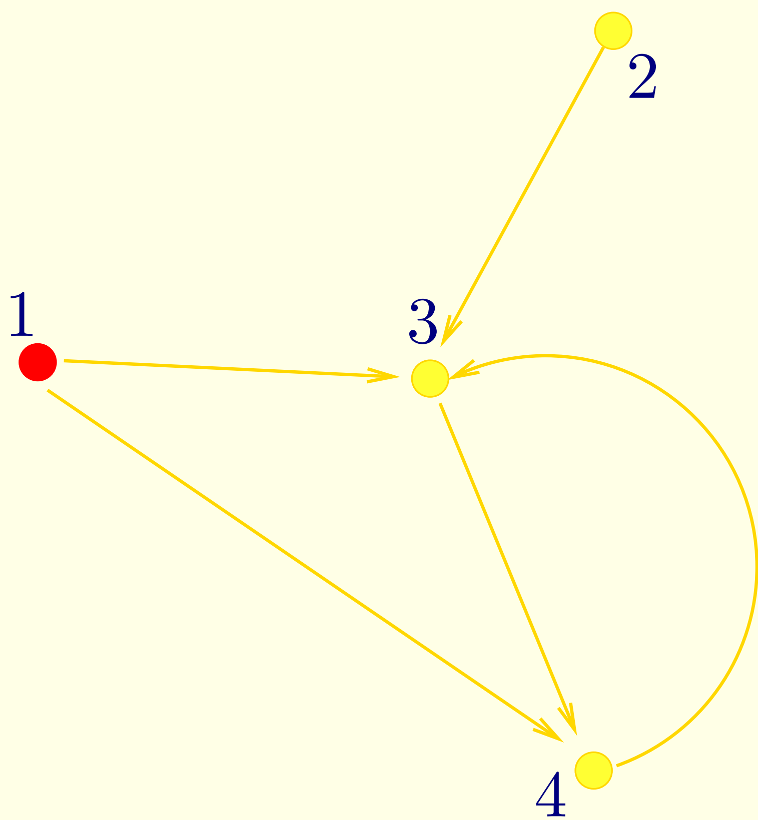



Database

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

# Example






Database

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

$\Rightarrow$ saturated.

# First Meta-Complexity Theorem



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:

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

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

prefix firings:

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

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

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

prefix firings:

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

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

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

prefix firings:

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

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 $O(1)$

$$\frac{s(u)}{r(u)} \qquad\qquad \frac{r(u) \quad e(u,v)}{r(v)}$$

$$r(u) \qquad O(|V|)$$

$$\frac{s(u)}{r(u)} \; O(|V|) \qquad \frac{e(u,v)}{r(v)}$$

$$\frac{s(u)}{r(u)} \quad O(|V|) \qquad\qquad \frac{\begin{array}{c} r(u) \\ e(u,v) \end{array}}{r(v)} \quad \begin{array}{r} O(|V|) \\ +\,O(|E|) \end{array}$$

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".

$$\frac{proc(P, B_P, E_P)}{P \Rightarrow B_P}$$

$$\frac{\begin{array}{l} next(Q, L, L') \\ Q \Rightarrow L \end{array}}{Q \Rightarrow L'} \qquad \frac{\begin{array}{l} call(Q, P, L_c, R_r) \\ proc(P, B_P, E_P) \\ P \Rightarrow E_P \\ Q \Rightarrow L_c \end{array}}{Q \Rightarrow L_r}$$

# Interprocedural Reachability: Rules 

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

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

$$\frac{\begin{array}{l} next(Q, L, L') \\ Q \Rightarrow L \end{array}}{Q \Rightarrow L'} \qquad \frac{\begin{array}{l} call(Q, P, L_c, R_r) \\ proc(P, B_P, E_P) \\ P \Rightarrow E_P \\ Q \Rightarrow L_c \end{array}}{Q \Rightarrow L_r}$$

# Interprocedural Reachability: Rules 

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

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

$$\frac{\begin{array}{l} next(Q, L, L') \quad O(n) \\ Q \Rightarrow L \end{array}}{Q \Rightarrow L'}$$

$$\frac{\begin{array}{l} call(Q, P, L_c, R_r) \\ proc(P, B_P, E_P) \\ P \Rightarrow E_P \\ Q \Rightarrow L_c \end{array}}{Q \Rightarrow L_r}$$

# Interprocedural Reachability: Rules



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

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

$$\frac{\begin{array}{ll} next(Q, L, L') & O(n) \\ Q \Rightarrow L & * O(1) \end{array}}{Q \Rightarrow L'} \qquad \frac{\begin{array}{l} call(Q, P, L_c, R_r) \\ proc(P, B_P, E_P) \\ P \Rightarrow E_P \\ Q \Rightarrow L_c \end{array}}{Q \Rightarrow L_r}$$

# Interprocedural Reachability: Rules



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

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

$$\frac{\begin{array}{ll} next(Q, L, L') & O(n) \\ Q \Rightarrow L & *\,O(1) \end{array}}{Q \Rightarrow L'}$$

$$\frac{\begin{array}{ll} call(Q, P, L_c, R_r) & O(n) \\ proc(P, B_P, E_P) & \\ P \Rightarrow E_P & \\ Q \Rightarrow L_c & \end{array}}{Q \Rightarrow L_r}$$

# Interprocedural Reachability: Rules 

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

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

$$\frac{\begin{array}{ll} next(Q, L, L') & O(n) \\ Q \Rightarrow L & *\, O(1) \end{array}}{Q \Rightarrow L'}$$

$$\frac{\begin{array}{ll} call(Q, P, L_c, R_r) & O(n) \\ proc(P, B_P, E_P) & *\, O(1) \\ P \Rightarrow E_P & \\ Q \Rightarrow L_c & \end{array}}{Q \Rightarrow L_r}$$

# Interprocedural Reachability: Rules 

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

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

$$\frac{\begin{array}{ll} next(Q, L, L') & O(n) \\ Q \Rightarrow L & * O(1) \end{array}}{Q \Rightarrow L'}$$

$$\frac{\begin{array}{ll} call(Q, P, L_c, R_r) & O(n) \\ proc(P, B_P, E_P) & * O(1) \\ P \Rightarrow E_P & * O(1) \\ Q \Rightarrow L_c & * O(1) \end{array}}{Q \Rightarrow L_r}$$

# Interprocedural Reachability: Rules 

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

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

$$\frac{\begin{array}{ll} next(Q, L, L') & O(n) \\ Q \Rightarrow L & * \, O(1) \end{array}}{Q \Rightarrow L'} \qquad \frac{\begin{array}{ll} call(Q, P, L_c, R_r) & O(n) \\ proc(P, B_P, E_P) & * \, O(1) \\ P \Rightarrow E_P & * \, O(1) \\ Q \Rightarrow L_c & * \, O(1) \end{array}}{Q \Rightarrow L_r}$$

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 $D$)

# 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

# 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

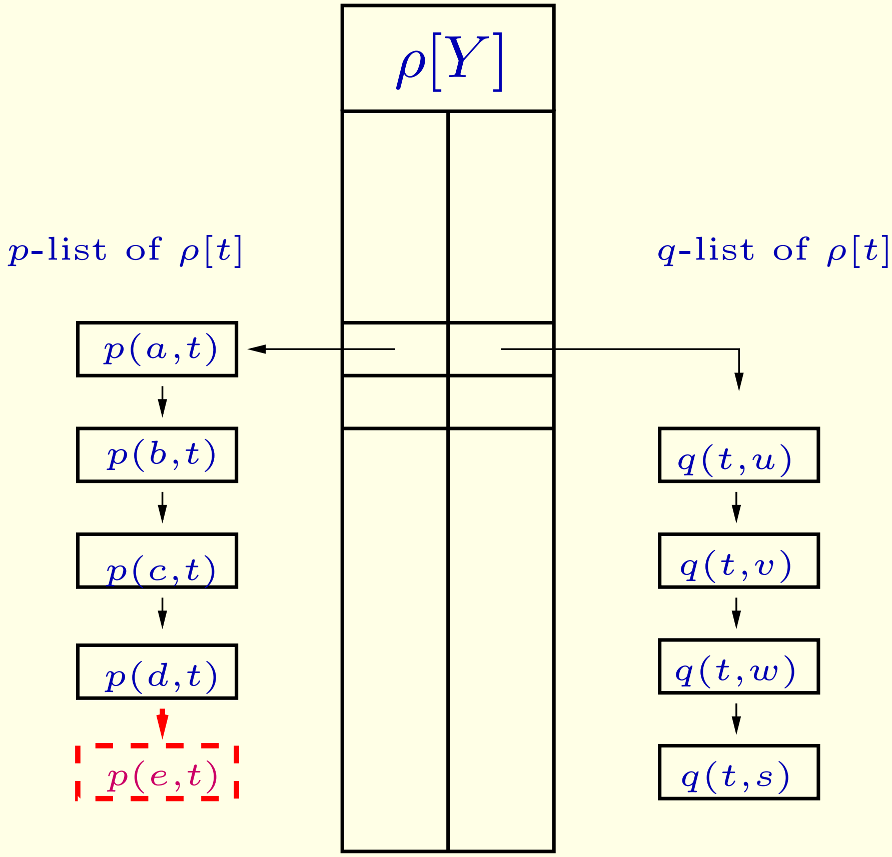
Problem: avoiding $O(|R(D)|^k)$ for rules of length $k$

# Proof of the Meta-Complexity Theorem II 

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

# Proof of the Meta-Complexity Theorem II 

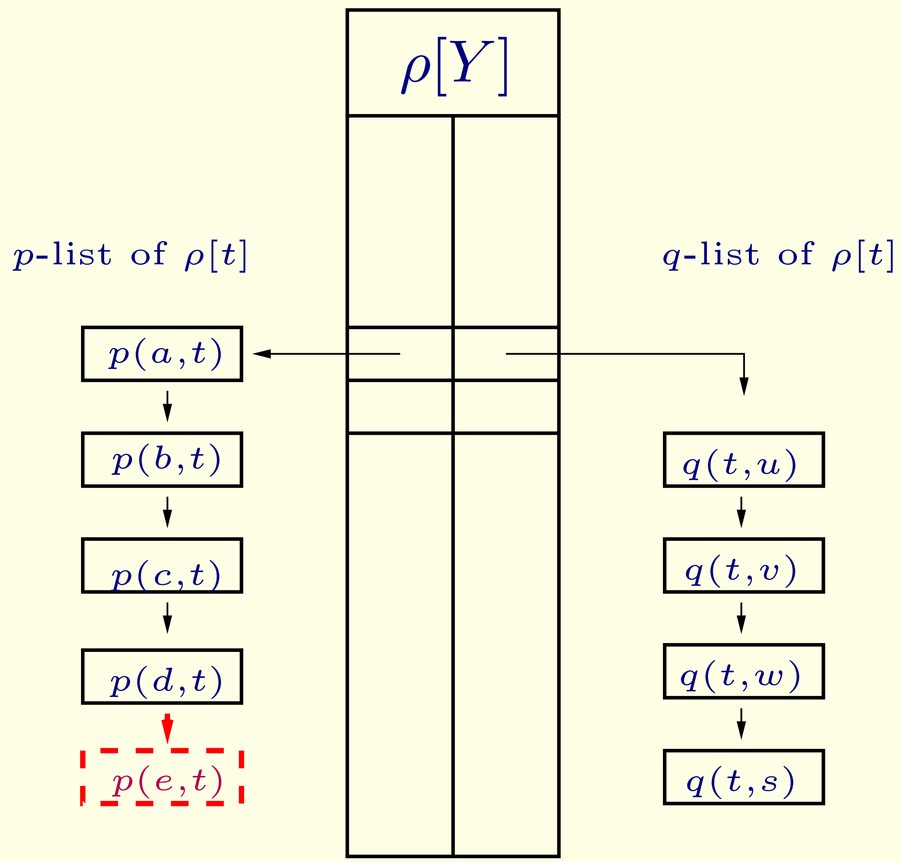
Data structure for rules $\rho$ of the form $p(X,Y) \wedge q(Y,Z) \supset r(X,Y,Z)$



# Proof of the Meta-Complexity Theorem II



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

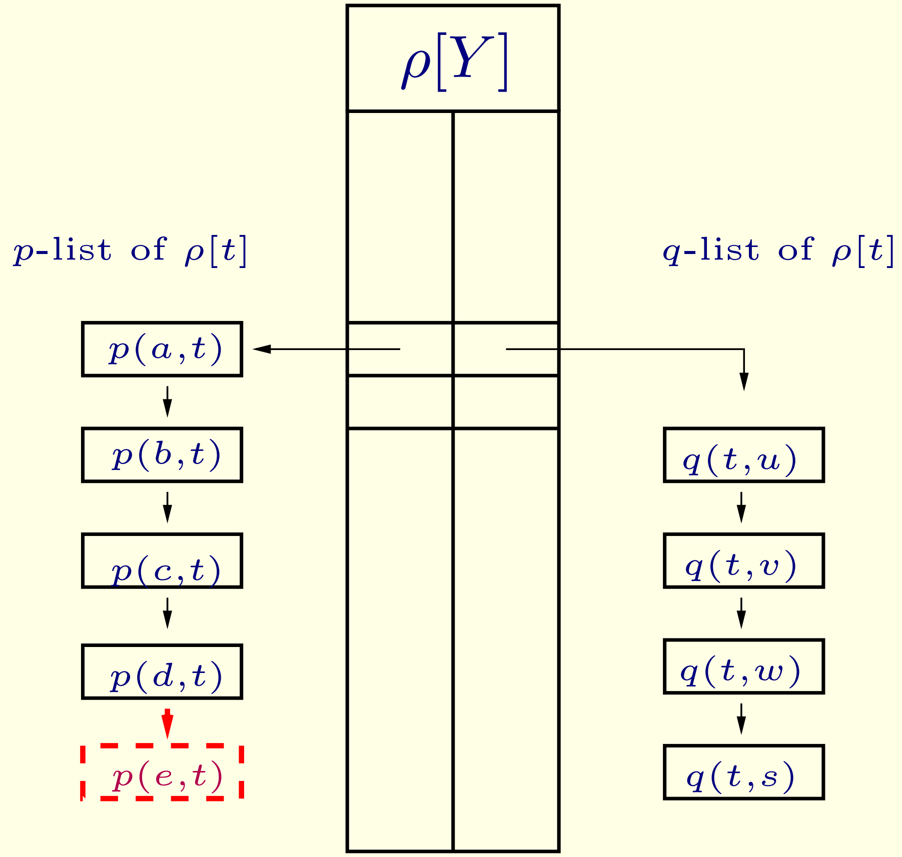



Upon adding a fact $p(e, t)$, fire all $r(e, t, z)$, for $z$ on the $q$-list of $A[t]$.

# Proof of the Meta-Complexity Theorem II



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




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 $R^*(D) \cap \text{atoms}(\text{subterms}(D))$
  $\Rightarrow$ concept of local inference systems (Givan, McAllester 1993)

# Remarks



- memory consumption often much smaller
- if $R^*(D)$ infinite, consider $R^*(D) \cap \mathrm{atoms}(\mathrm{subterms}(D))$

  $\Rightarrow$ 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

# Removal of Redundant Information



- redundant information causes inefficiency

$$D = \{\ldots,\ dist(x) \leq d,\ dist(x) \leq d',\ d' < d,\ \ldots\}$$

$\Rightarrow$ delete $dist(x) \leq d$

# Removal of Redundant Information 

- redundant information causes inefficiency

$$D = \{\ldots,\ dist(x) \leq d,\ dist(x) \leq d',\ d' < d,\ \ldots\}$$

$\Rightarrow$ 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

$$D = \{\ldots,\ dist(x) \leq d,\ dist(x) \leq d',\ d' < d,\ \ldots\}$$

$\Rightarrow$ 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:

$$P \ \supset \ Q \qquad [Q] \ \supset \ S \qquad [Q] \ \supset \ W$$

$\Rightarrow$ either $S$ or $W$ can be derived, but not both

# Removal of Redundant Information 

- redundant information causes inefficiency

$$D = \{\ldots,\ dist(x) \leq d,\ dist(x) \leq d',\ d' < d,\ \ldots\}$$

$\Rightarrow$ 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:

$$P \ \supset \ Q \qquad [Q] \ \supset \ S \qquad [Q] \ \supset \ W$$

$\Rightarrow$ either $S$ or $W$ can be derived, but not both

- non-determinism don't-care and/or restricted by priorities

# Logic Programs with Priorities and Deletion 

- rules can have antecedents to be deleted after firing

# Logic Programs with Priorities and Deletion



- 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 $\neg A \notin S$ (deletions are permanent)

# 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 $\neg A \notin S$ (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 $\neg A$)

# 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 $\neg A \notin S$ (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 $S$

# 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 $\neg A \notin S$ (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

# 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 $\neg A \notin S$ (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
- the final state of a computation starting with $D$ is called an ($R$-) saturation of $D$

# 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 $0 \leq t < T$:

- $r = A_1 \wedge \ldots \wedge A_i \wedge \ldots \wedge A_n \supset A_0 \in R$
- $S_t$ visible to $r$
- $A_j\sigma$ visible in $S_t$, for $1 \leq j \leq i$

## 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 $0 \leq t < T$:

- $r = A_1 \wedge \ldots \wedge A_i \wedge \ldots \wedge A_n \ \supset \ A_0 \in R$
- $S_t$ visible to $r$
- $A_j\sigma$ visible in $S_t$, for $1 \leq j \leq i$

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

# 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 $0 \leq t < T$:

- $r = A_1 \wedge \ldots \wedge A_i \wedge \ldots \wedge A_n \ \supset \ A_0 \in R$
- $S_t$ visible to $r$
- $A_j\sigma$ visible in $S_t$, for $1 \leq j \leq i$

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

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 $0 \leq t < T$:

– $r = A_1 \wedge \ldots \wedge A_i \wedge \ldots \wedge A_n \ \supset \ A_0 \in R$

– $S_t$ visible to $r$

– $A_j\sigma$ visible in $S_t$, for $1 \leq j \leq i$

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

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

# 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 $0 \leq t < T$:

- $r = A_1 \wedge \ldots \wedge A_i \wedge \ldots \wedge A_n \supset A_0 \in R$
- $S_t$ visible to $r$
- $A_j\sigma$ visible in $S_t$, for $1 \leq j \leq i$

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

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

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

# Union-Find



$$\texttt{(Refl)}\ \frac{\mathrm{find}(x)}{x \Rightarrow ! \ x}$$

$$\texttt{(N)}\ \frac{x \Rightarrow ! \ y \qquad y \Rightarrow z}{x \Rightarrow ! \ z}$$

$$\texttt{(Comm)}\ \frac{x \Rightarrow y \qquad x \Rightarrow z}{\mathrm{union}(y,\ z)}$$

# Union-Find 

$$\texttt{(Refl)}\ \frac{\text{find}(x)}{x \Rightarrow! \, x}$$

$$\texttt{(N)}\ \frac{x \Rightarrow! \, y \quad y \Rightarrow z}{x \Rightarrow! \, z}$$

$$\texttt{(Comm)}\ \frac{x \Rightarrow y \quad x \Rightarrow z}{\text{union}(y,\ z)}$$

$$\texttt{(Init)}\ \frac{\text{union}(x,\ y)}{\text{find}(x),\ \text{find}(y)}$$

$$\texttt{(Orient)}\ \frac{\text{union}(x,\ y) \quad x \Rightarrow! \, z_1 \quad y \Rightarrow! \, z_2}{z_1 \Rightarrow z_2}$$

We are interested in $x \doteq y$ defined as $\exists z(x \Rightarrow! \, z \wedge y \Rightarrow! \, z)$

$$\texttt{(Refl)}\frac{\mathrm{find}(x)}{x \Rightarrow! \, x}$$

$$\texttt{(N)}\frac{\begin{array}{ll} x \Rightarrow! \, y & O(n^2) \\ y \Rightarrow z & * \, O(n) \end{array}}{x \Rightarrow! \, z}$$

$$\texttt{(Comm)}\frac{\begin{array}{ll} x \Rightarrow y & O(n^2) \\ x \Rightarrow z & * \, O(n) \end{array}}{\mathrm{union}(y,\ z)}$$

$$\texttt{(Init)}\frac{\mathrm{union}(x,\ y)}{\begin{array}{l} \mathrm{find}(x), \\ \mathrm{find}(y) \end{array}}$$

$$\texttt{(Orient)}\frac{\begin{array}{l} \mathrm{union}(x,\ y) \\ x \Rightarrow! \, z_1 \\ y \Rightarrow! \, z_2 \end{array}}{z_1 \Rightarrow z_2}$$

Naive Knuth/Bendix completion

# Union-Find

$$\texttt{(Refl)}\ \frac{\mathrm{find}(x)}{x \Rightarrow! \, x}$$

$$\texttt{(N)}\ \frac{\begin{array}{ll} [\![x \Rightarrow! \, y]\!] & O(n^2) \\ y \Rightarrow z & *\, O(1) \end{array}}{x \Rightarrow! \, z}$$

$$\texttt{(Comm)}\ \frac{\begin{array}{ll} x \Rightarrow y & O(n) \\ x \Rightarrow z & *\, O(1) \end{array}}{\mathrm{union}(y,\ z)}$$

$$\texttt{(Init)}\ \frac{\mathrm{union}(x,\ y)}{\begin{array}{l} \mathrm{find}(x), \\ \mathrm{find}(y) \end{array}}$$

$$\texttt{(Triv)}\ \frac{\begin{array}{l} [\![\mathrm{union}(x,\ y)]\!] \\ x \Rightarrow! \, z \\ y \Rightarrow! \, z \end{array}}{\top}$$

$$\texttt{(Orient)}\ \frac{\begin{array}{l} [\![\mathrm{union}(x,\ y)]\!] \\ x \Rightarrow! \, z_1 \\ y \Rightarrow! \, z_2 \end{array}}{z_1 \Rightarrow z_2}$$

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

# Union-Find



$$\texttt{(Refl)}\ \frac{\mathrm{find}(x)}{x \Rightarrow! \ x \quad \mathrm{weight}(x, 1)}$$

$$\texttt{(N)}\ \frac{[\![x \Rightarrow! \ y]\!] \quad O(n \log n) \qquad y \Rightarrow z \quad * \, O(1)}{x \Rightarrow! \ z}$$

$$\texttt{(Comm)}\ \frac{x \Rightarrow y \qquad x \Rightarrow z}{\mathrm{union}(y,\ z)}$$

$$\texttt{(Init)}\ \frac{\mathrm{union}(x,\ y)}{\mathrm{find}(x), \quad \mathrm{find}(y)}$$

$$\texttt{(Triv)}\ \frac{[\![\mathrm{union}(x,\ y)]\!] \qquad x \Rightarrow! \ z \qquad y \Rightarrow! \ z}{\top}$$

$$\texttt{(Orient)}\ \frac{[\![\mathrm{union}(x,\ y)]\!] \qquad x \Rightarrow! \ z_1, \quad \mathrm{weight}(z_1, w_1) \qquad y \Rightarrow! \ z_2, \quad [\![\mathrm{weight}(z_2, w_2)]\!] \qquad w_1 \leq w_2}{z_1 \Rightarrow z_2 \quad \mathrm{weight}(z_2, w_1 + w_2)}$$

+ symmetric variant of (Orient)

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

$m = |\text{union assertions}|$, $n = |\text{(sub)terms}|$

## 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

# 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 $\Omega(n^2)$.

# 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)

# 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 $A_i \prec A$.

# 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

# 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 $R(S \setminus Red(S)) \ \subseteq \ S \cup Red(S)$.

# 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.

# 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

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

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

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

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

$$\textbf{(Init)}\ \frac{}{\mathrm{dist}(\mathrm{src}) \le 0}$$

$$\textbf{(Upd)}\ \frac{\begin{array}{l}[\![\mathrm{dist}(x) \le d]\!]\\ \mathrm{dist}(x) \le d'\\ d' < d\end{array}}{\top}$$

$$\textbf{(Add)}\ \frac{\begin{array}{l}\mathrm{dist}(x) \le d\\ x \xrightarrow{c} y\end{array}}{\mathrm{dist}(y) \le c + d}$$

# Shortest Paths

$$\texttt{(Init)}\ \frac{}{\mathrm{dist}(\mathrm{src}) \le 0} \qquad \texttt{(Upd)}\ \frac{\begin{array}{l}[\![\mathrm{dist}(x) \le d]\!] \\ \mathrm{dist}(x) \le d' \\ d' < d\end{array}}{\top} \qquad \texttt{(Add)}\ \frac{\begin{array}{l}\mathrm{dist}(x) \le d \\ x \xrightarrow{c} y\end{array}}{\mathrm{dist}(y) \le c + d}$$

Correctness: obvious; deletion is based on redundancy

# Shortest Paths

$$\texttt{(Init)}\ \frac{}{\mathrm{dist}(\mathrm{src}) \le 0} \qquad \texttt{(Upd)}\ \frac{\begin{array}{l}[\![\mathrm{dist}(x) \le d]\!] \\ \mathrm{dist}(x) \le d' \\ d' < d\end{array}}{\top} \qquad \texttt{(Add)}\ \frac{\begin{array}{l}\mathrm{dist}(x) \le d \\ x \xrightarrow{c} y\end{array}}{\mathrm{dist}(y) \le c + d}$$

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

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

$$\texttt{(Init)}\ \frac{}{\mathrm{dist}(\mathrm{src}) \leq 0} \qquad \texttt{(Upd)}\ \frac{\begin{array}{l}[\![\mathrm{dist}(x) \leq d]\!]\\ \mathrm{dist}(x) \leq d'\\ d' < d\end{array}}{\top} \qquad \texttt{(Add)}\ \frac{\begin{array}{l}\mathrm{dist}(x) \leq d\\ x \xrightarrow{c} y\end{array}}{\mathrm{dist}(y) \leq c + d}$$

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 $(\texttt{Init}) > (\texttt{Upd}) > (\texttt{Add})[n/d] > (\texttt{Add})[m/d]$, 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

Basis: Union-find module

Basis: Union-find module

$$
\texttt{(Del)}\ \frac{[\![x \overset{c}{\leftrightarrow} y]\!] \quad x \Rightarrow!\, z \quad y \Rightarrow!\, z}{T}
\qquad\qquad
\texttt{(Add)}\ \frac{[\![x \overset{c}{\leftrightarrow} y]\!]}{\mathrm{mst}(x, c, y) \quad \mathrm{union}(x, y)}
$$

Basis: Union-find module

$$\texttt{(Del)}\;\frac{[\![x \stackrel{c}{\leftrightarrow} y]\!] \quad x \Rightarrow! \, z \quad y \Rightarrow! \, z}{T} \qquad\qquad \texttt{(Add)}\;\frac{[\![x \stackrel{c}{\leftrightarrow} y]\!]}{\mathrm{mst}(x, c, y) \quad \mathrm{union}(x, y)}$$

Priorities: (here needed for correctness)

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

Basis: Union-find module

$$\texttt{(Del)}\ \frac{[\![x \overset{c}{\leftrightarrow} y]\!] \quad x \Rightarrow!\ z \quad y \Rightarrow!\ z}{T} \qquad\qquad \texttt{(Add)}\ \frac{[\![x \overset{c}{\leftrightarrow} y]\!]}{\mathrm{mst}(x, c, y) \quad \mathrm{union}(x, y)}$$

Priorities: (here needed for correctness)

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

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

# 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