Information Flow in Logic Programming

Antoun Yaacoub

Institut de Recherche en Informatique de Toulouse (IRIT)
LILaC Team

November 28th, 2012

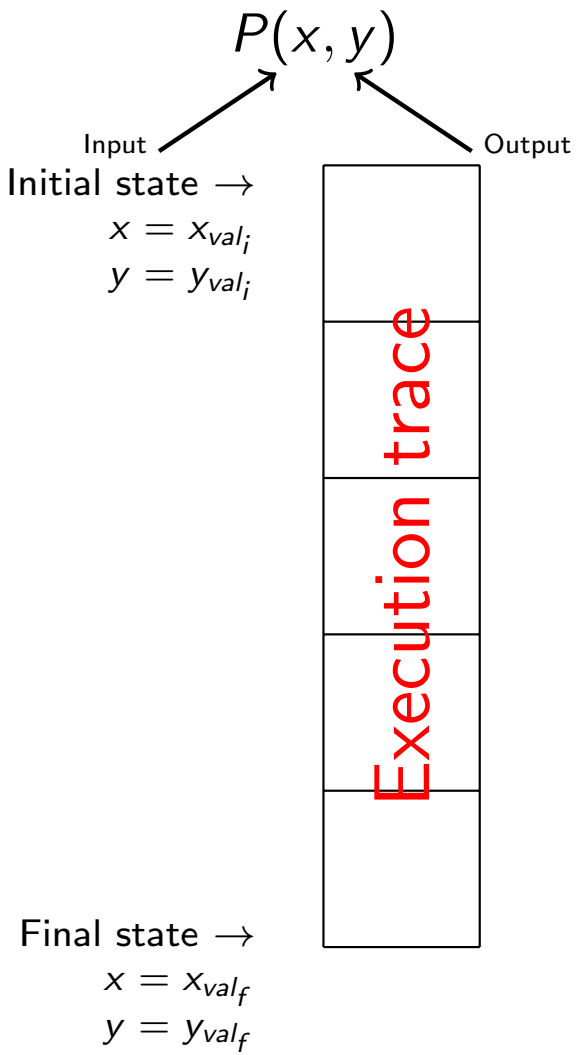
Introduction

Data security

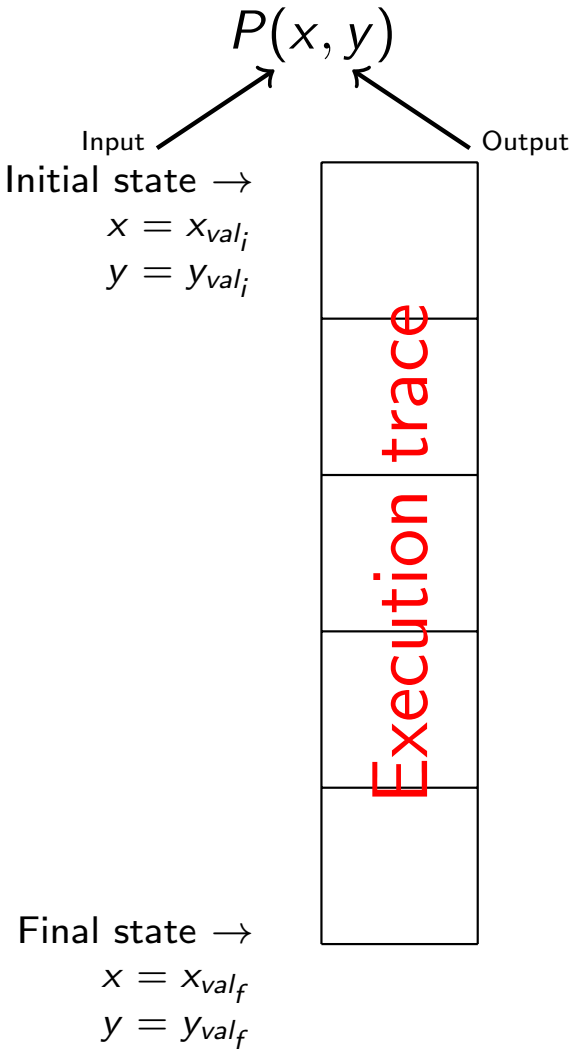
- Concepts
  - Confidentiality
  - Integrity
  - Availability
- Security policy
  - Bell-LaPadula model - 1976
  - Biba model - 1977
- Security mechanisms

Prevention	Access control
Detection	Inference control
Recovery	**Information flow control**

Introduction



Introduction



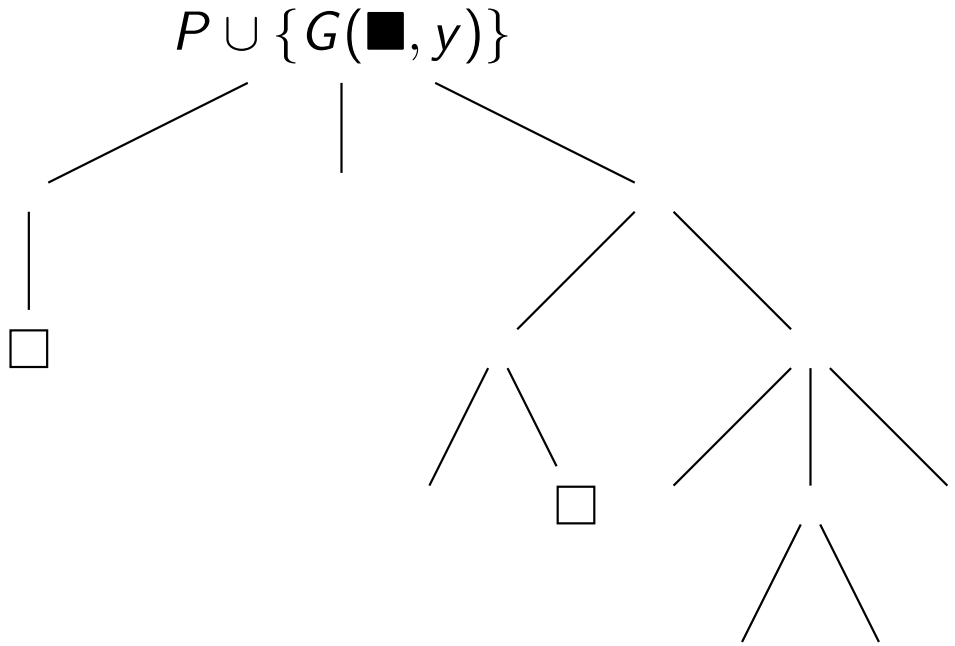
$$x \longrightarrow^{P} y$$

"The final value of y tells us (more or less) about the initial value of x".

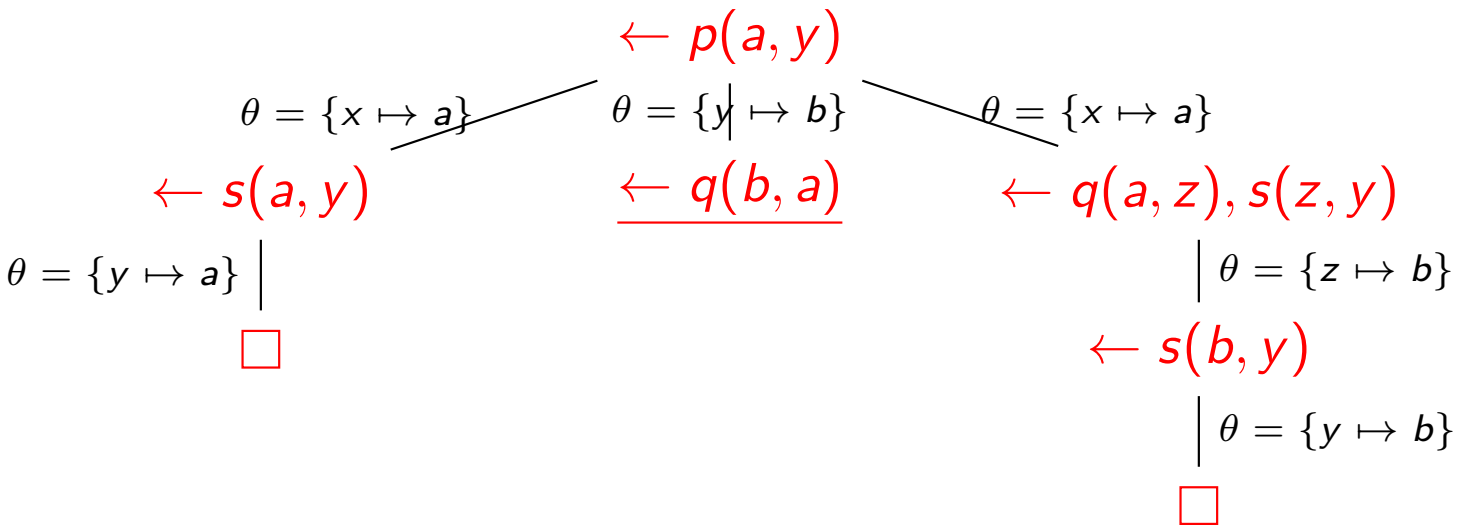
Examples

. $copy(F1, F2)$	$F1 \rightarrow F2$
. $y := x;$	$x \rightarrow y$
. $z := x;$	$x \rightarrow z$
$y := z;$	$x \rightarrow y$

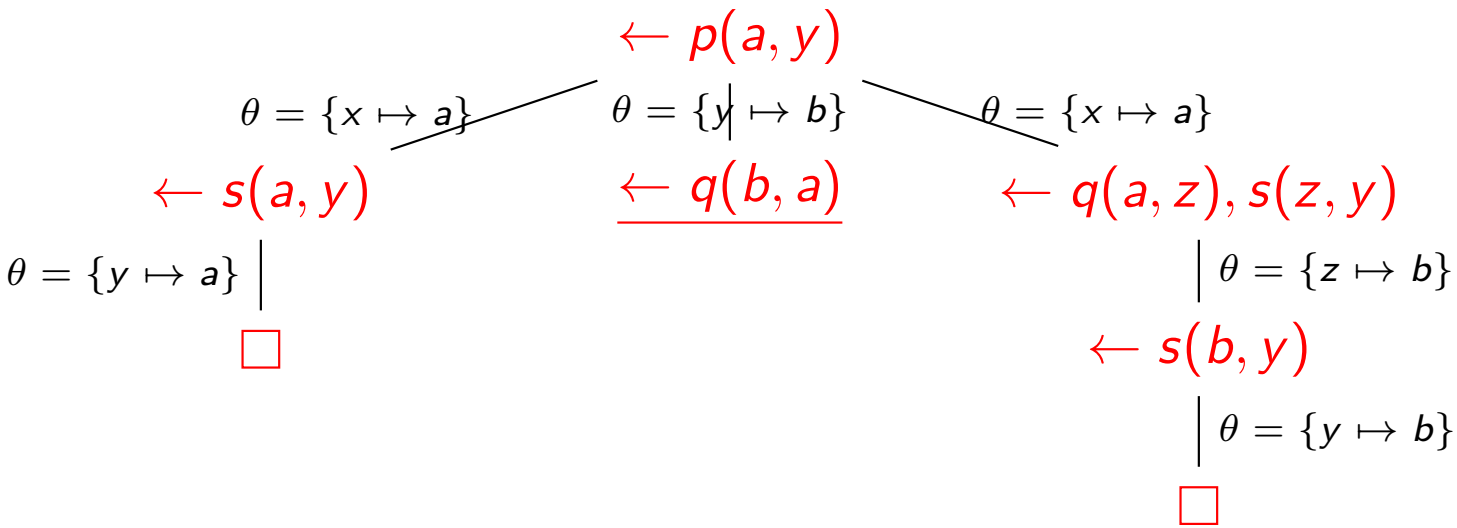
Information flow in logic programming

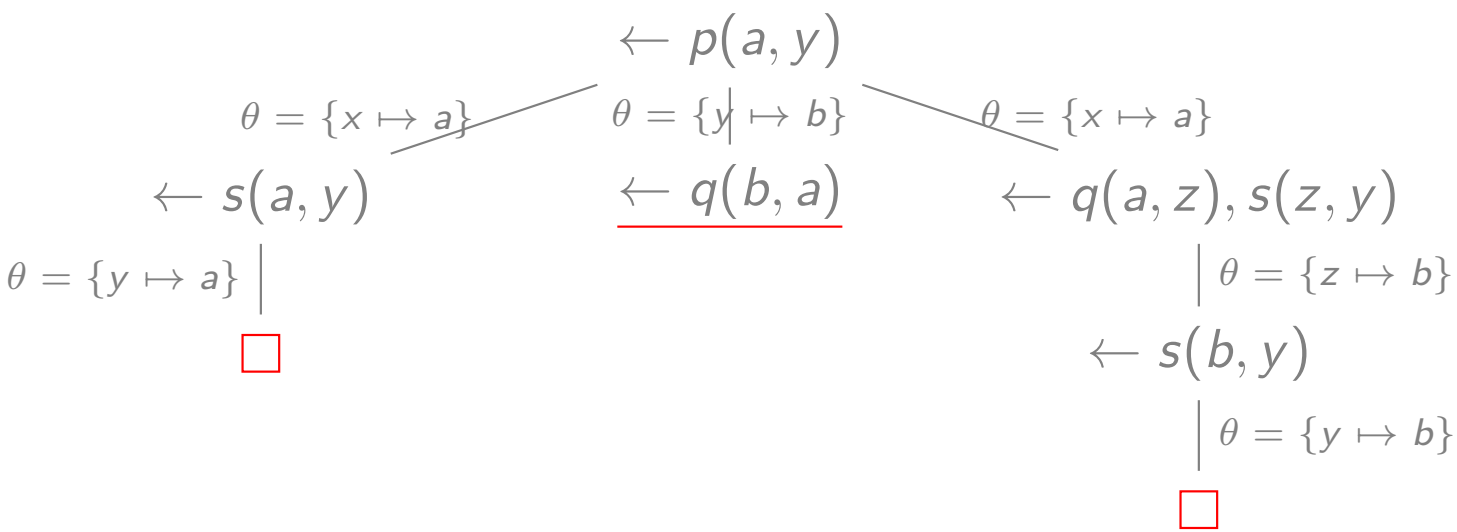


Information flow in logic programming

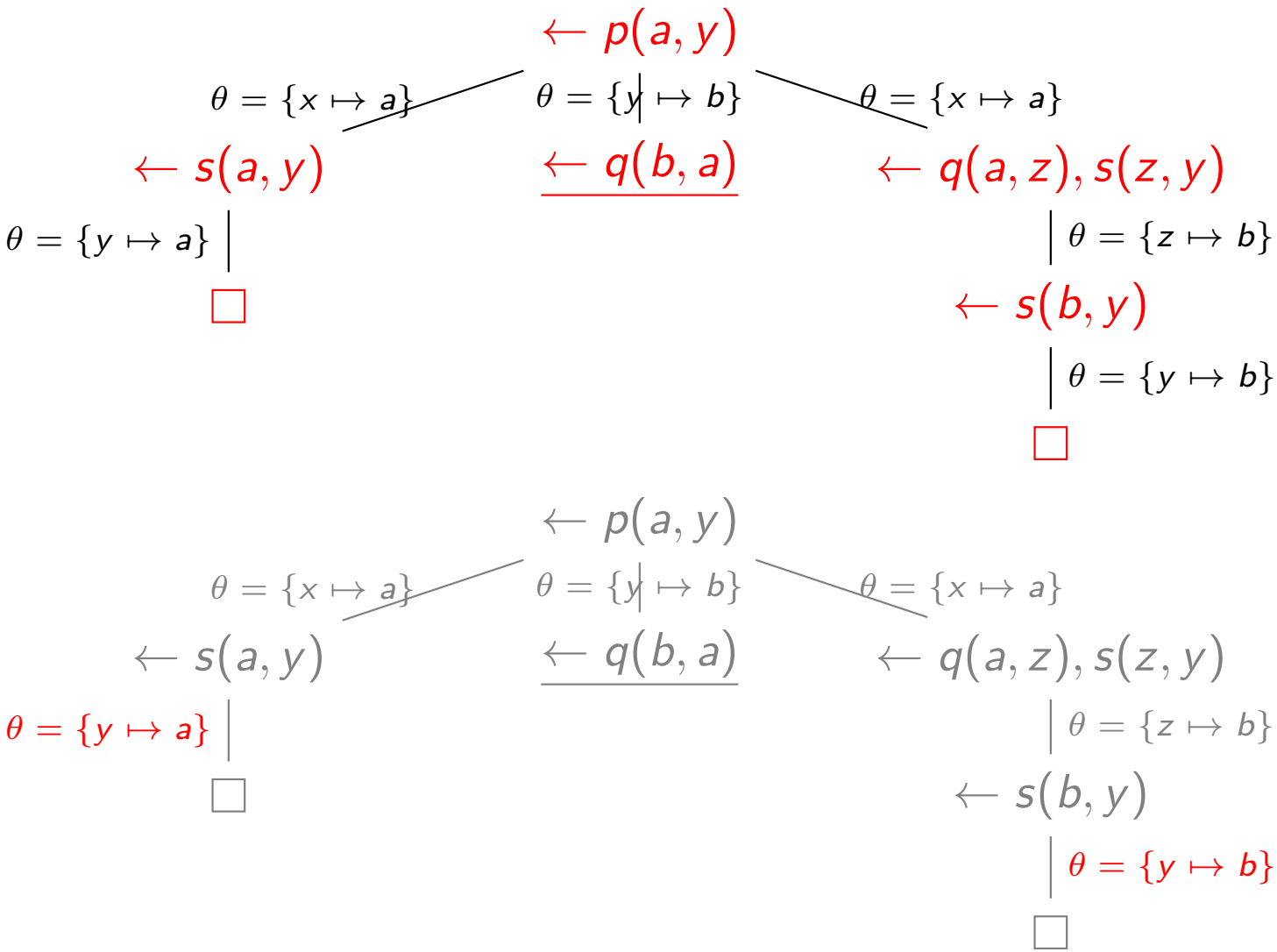


Information flow in logic programming

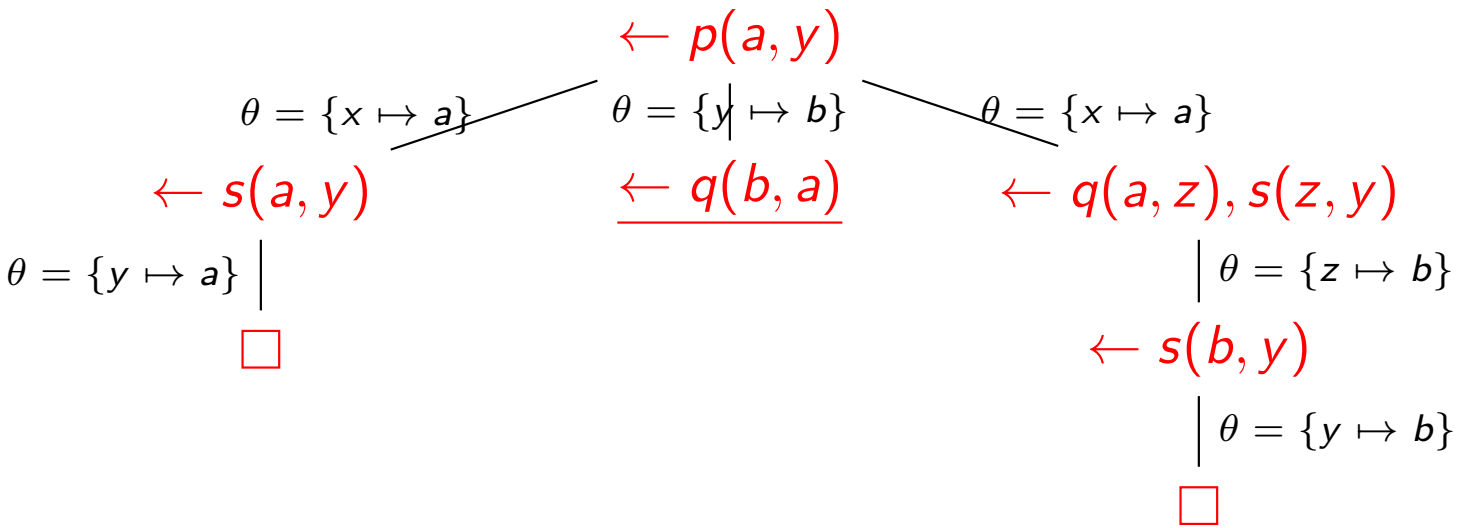


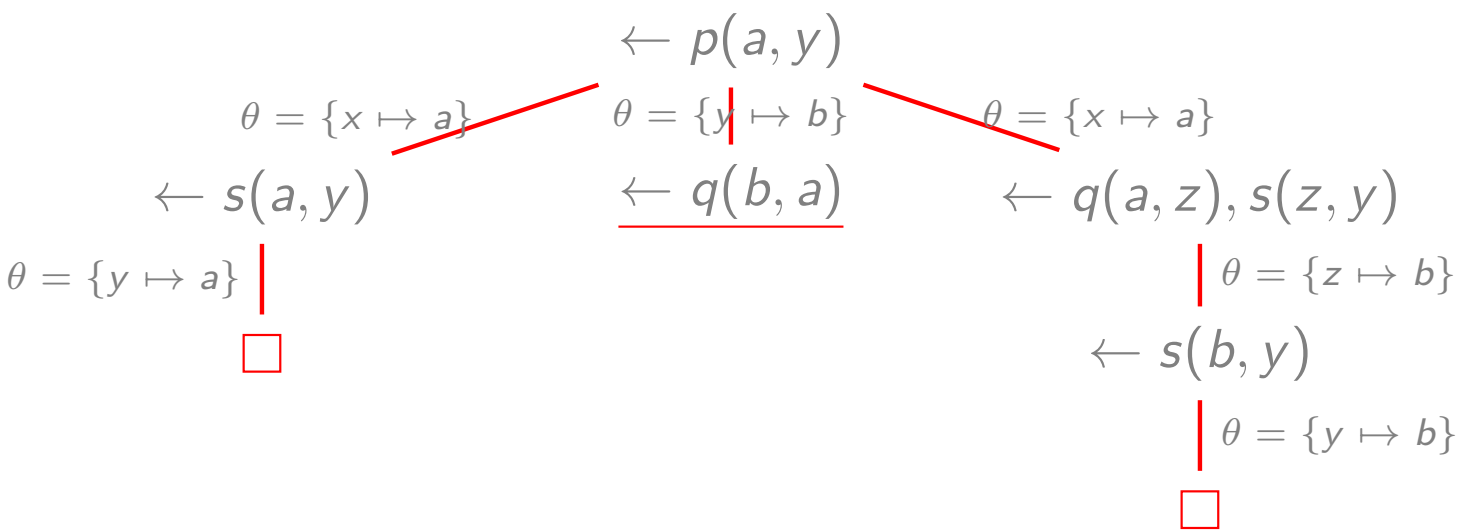


Information flow in logic programming



Information flow in logic programming





Outline of the presentation

Outline of the presentation

Datalog programs

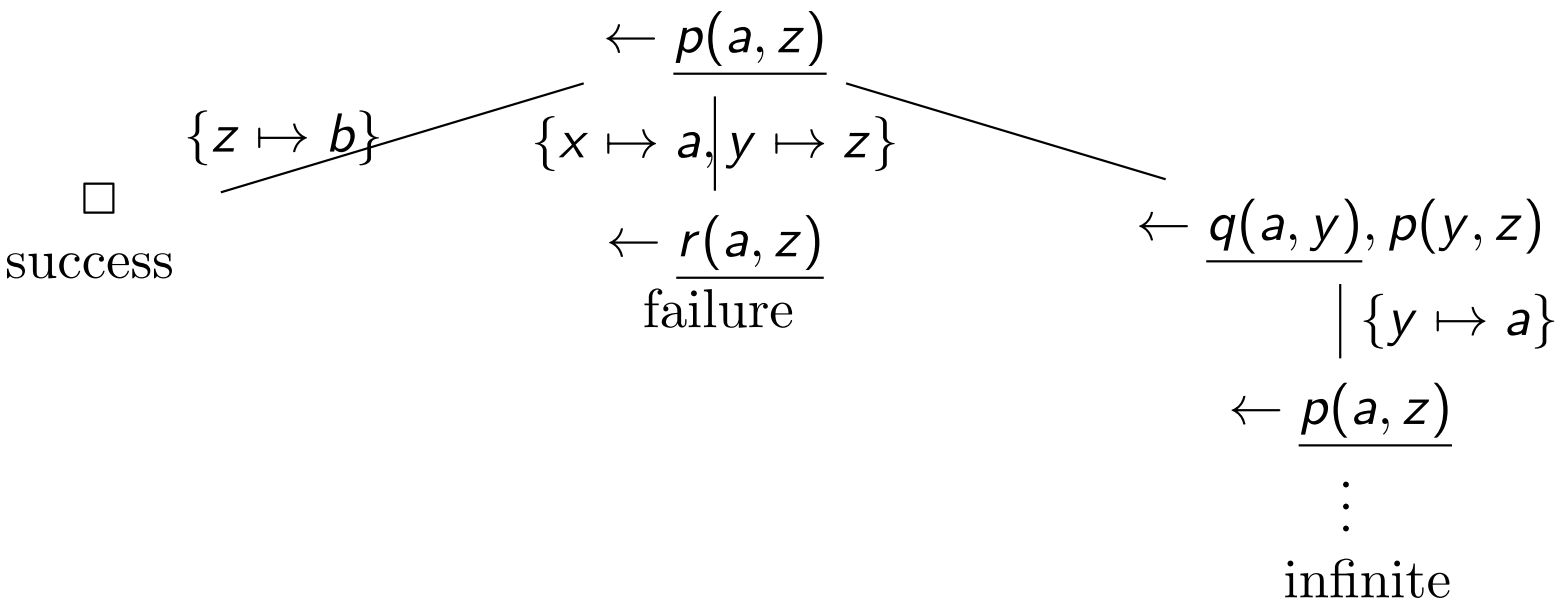
- Language L=Datalog language
- clause: $A_0 \leftarrow A_1, \cdots, A_i, \cdots, A_n$
  A_i: $p(t_1, \cdots, t_k)$.
  Term in Datalog: a constant or a variable.
- Fact: $A_0 \leftarrow$
- Goal: $G = \leftarrow A_1, \cdots, A_m$.
- Empty goal: $\square$

- Inference rule: SLD-resolution
- Computation rule: selection of the leftmost atom in a goal

Datalog programs - example

Let P be the program:
$r(b,b) \leftarrow$,
$q(a,a) \leftarrow$,
$p(a,b) \leftarrow$,
$p(x,y) \leftarrow r(x,y)$,
$p(x,z) \leftarrow q(x,y), p(y,z)$
and G be the goal $\leftarrow p(a,z)$.

Example of goals:
$\leftarrow s(a,y), r(x,y)$
$\leftarrow q(x,a)$
$\leftarrow p(a,z)$



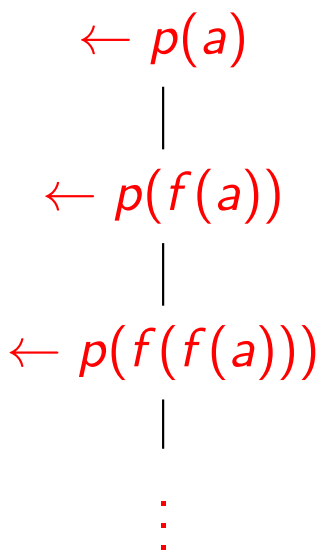
Prolog programs

- The alphabet of Prolog includes function symbols (constants, variables and expression of the form $f(...)$ are terms)

Example:

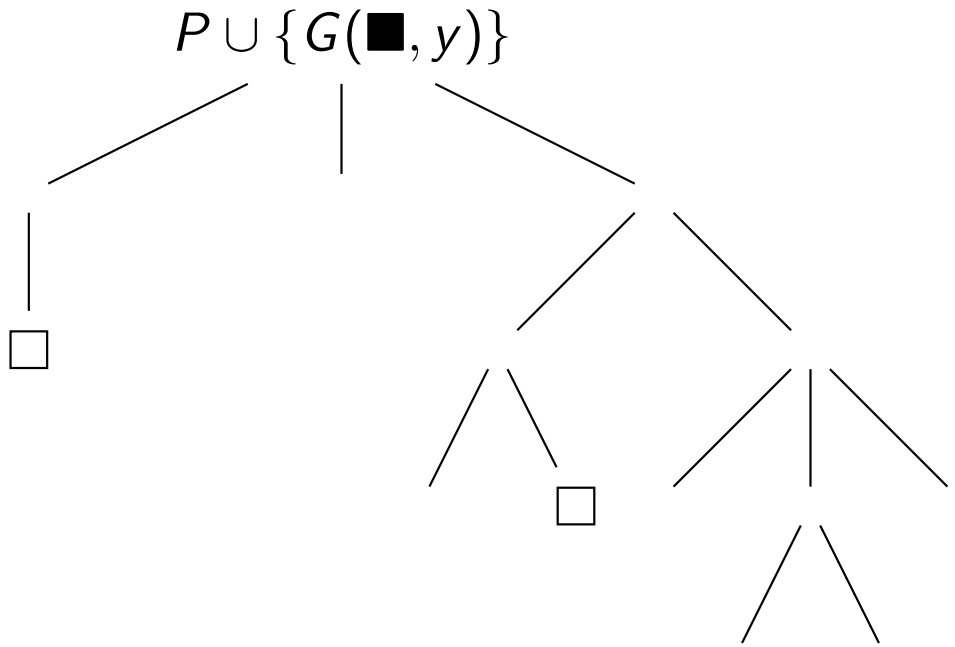
Let P be:
$p(x) \leftarrow p(f(x))$

and let $G = \leftarrow p(a)$

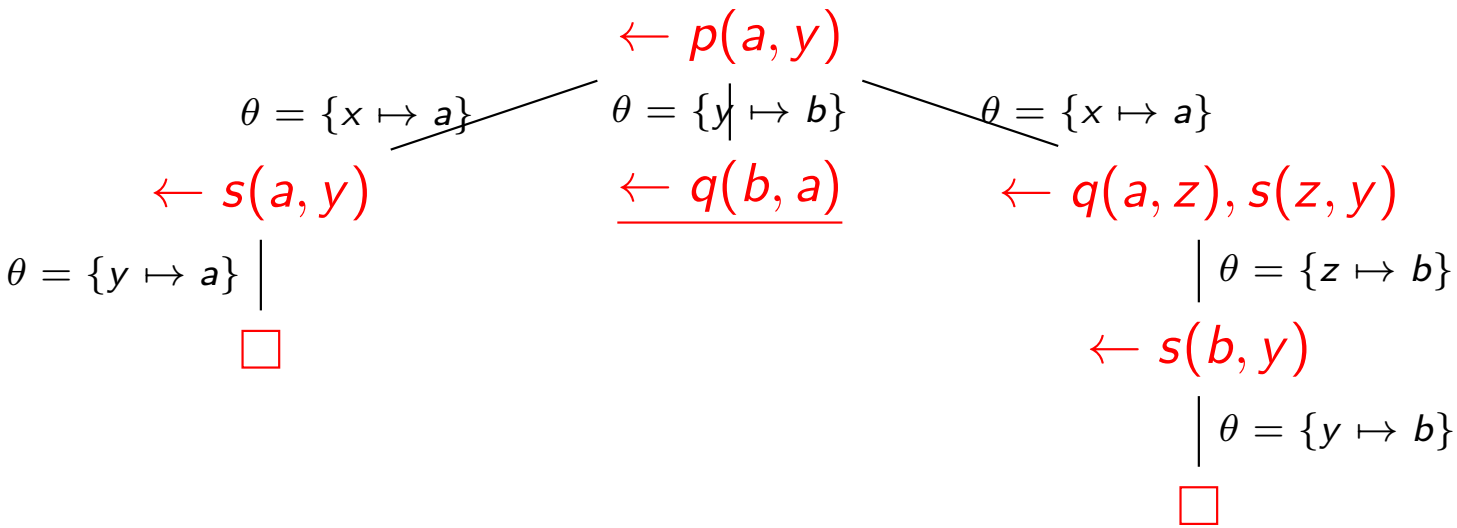


Outline of the presentation

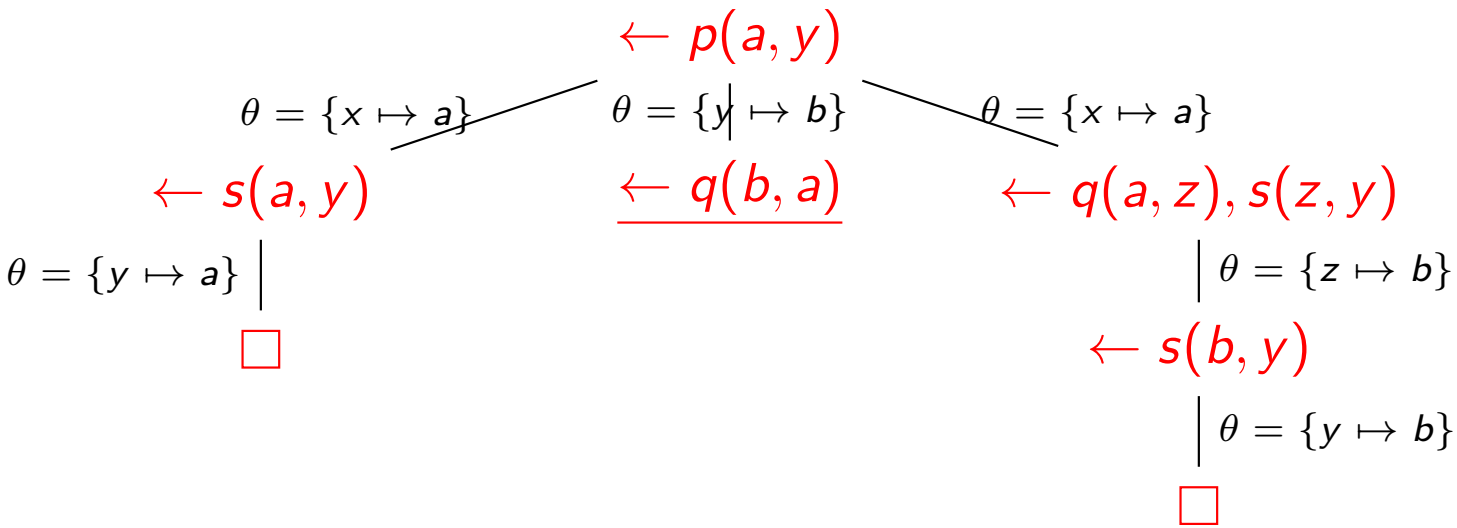
Information flow in logic programming

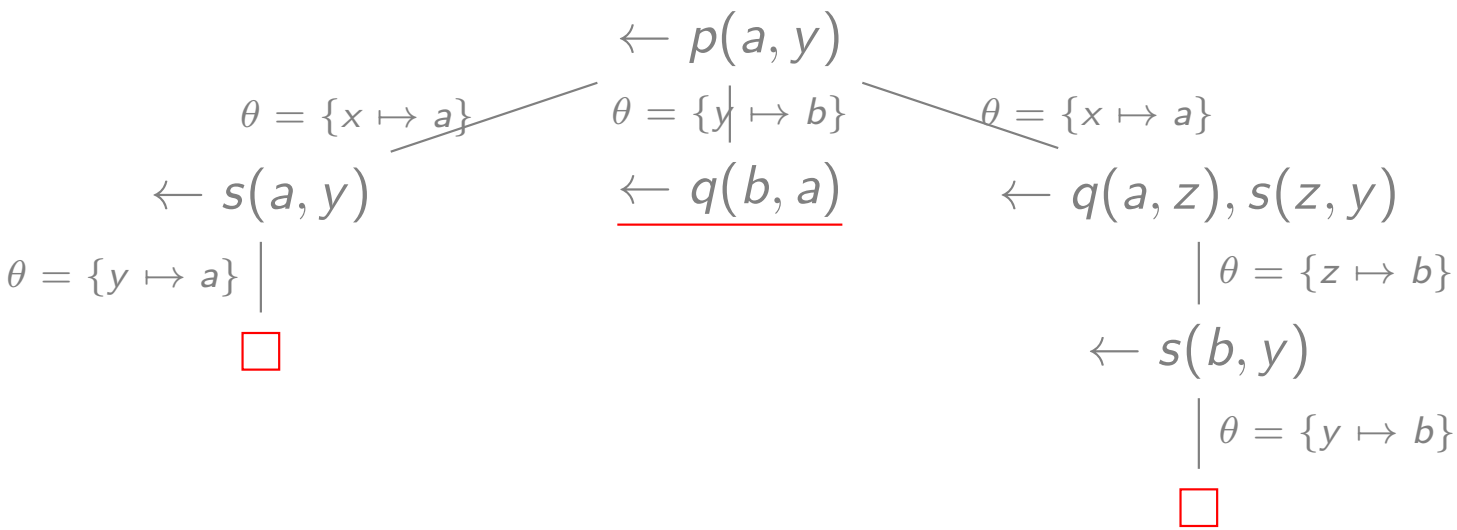


Information flow in logic programming

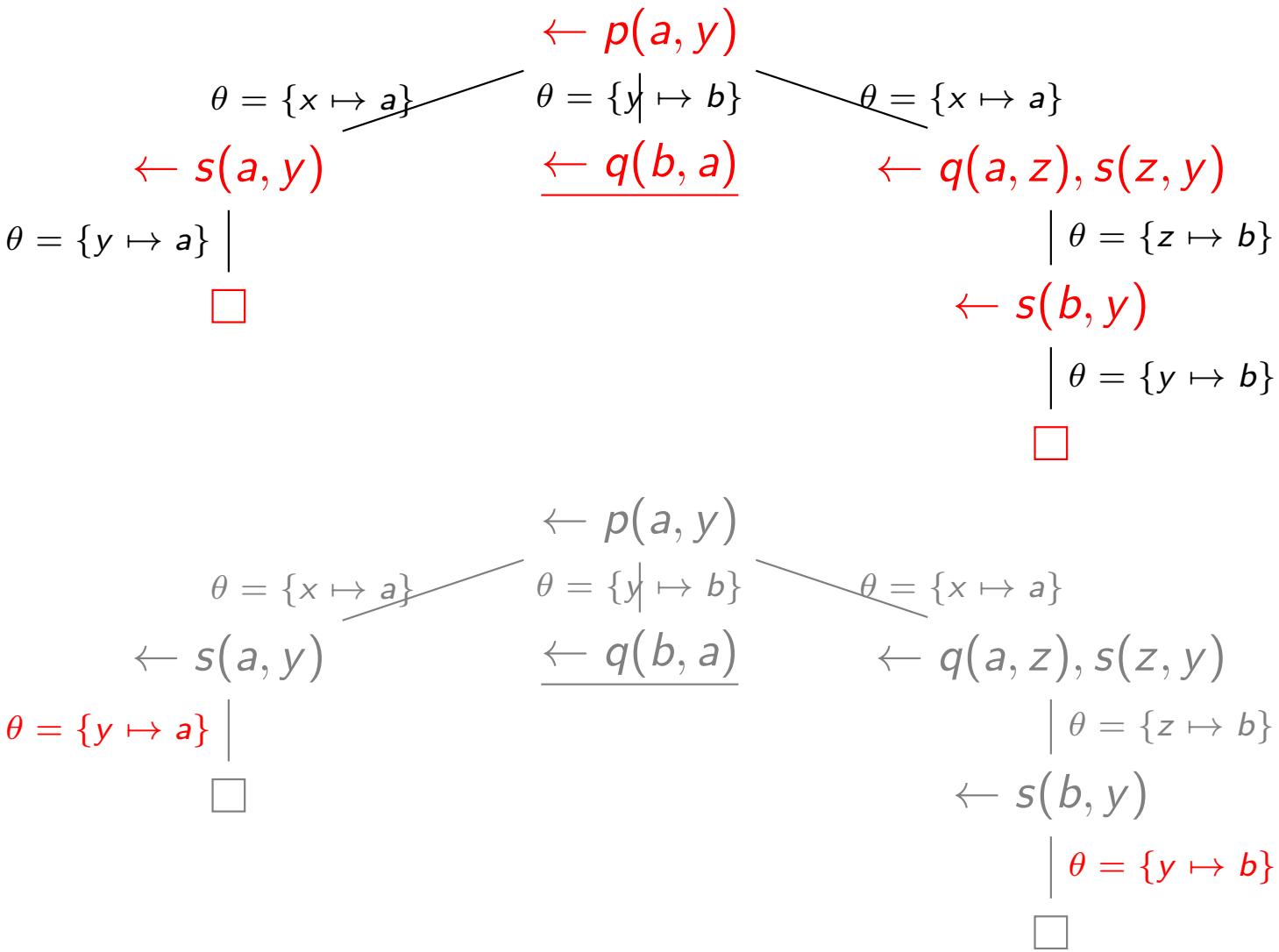


Information flow in logic programming

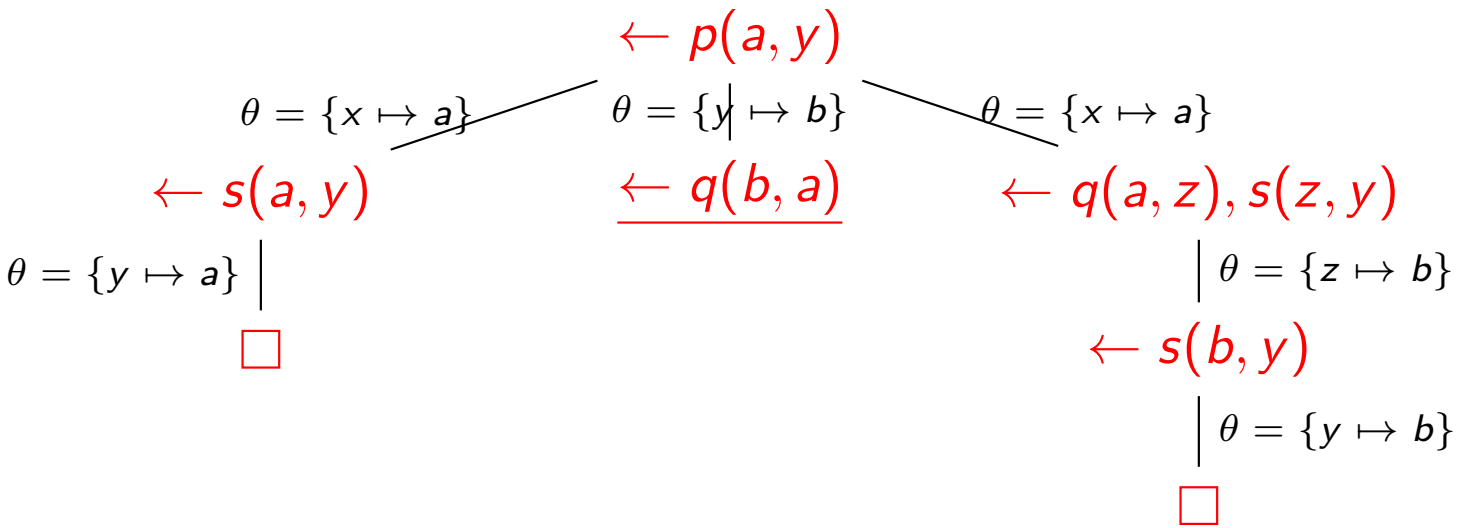




Information flow in logic programming



Information flow in logic programming



Information flow in logic programming

$$x \longrightarrow^{P}_{G(x,y)} y$$

Information flow in logic programming

Definition 1 of information flow in logic programming - Success / Failure

- For a logic program P and a goal $G(x, y)$ of arity 2, $x \xrightarrow{SF}{}^{P}_{G} y$ iff $\exists a, b \in U_{L(P)}$ such that: $P \cup \{G(a, y)\}$ succeeds and $P \cup \{G(b, y)\}$ fails

Example 1

Let P_1 be the following program:
$p(a, b) \leftarrow$
and let $G_1(x, y)$ be the goal: $\leftarrow p(x, y)$.

$$\begin{array}{l} P_1 \cup \{G_1(a, y)\} \text{ succeeds} \\ P_1 \cup \{G_1(b, y)\} \text{ fails} \end{array} \quad \text{then } x \xrightarrow{SF}{}^{P_1}_{G_1} y$$

Information flow in logic programming

Definition 2 of information flow in logic programming - Substitution answers

- For a logic program P and a goal $G(x, y)$ of arity 2, $x \xrightarrow[G]{SA} y$ iff $\exists a, b \in U_{L(P)}$ such that: $\Theta(P \cup \{G(a, y)\}) \neq \Theta(P \cup \{G(b, y)\})$

Example 2

Let P_2 be the following program:
$p(a, y) \leftarrow$
and let $G_2(x, y)$ be the goal: $\leftarrow p(x, y)$.

$$\Theta(P_2 \cup \{G_2(a, y)\}) = \{\epsilon\} \quad \Theta(P_2 \cup \{G_2(b, y)\}) = \emptyset \quad \text{then } x \xrightarrow[G_2]{SA} y$$

Bisimulation - Definition

Bisimulation

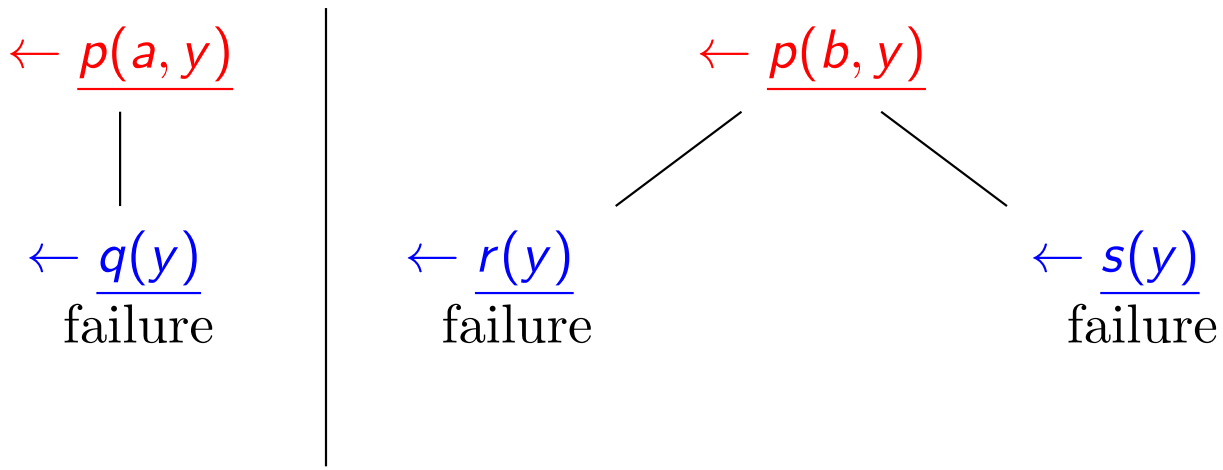
Let P be a logic program. A binary relation $\mathcal{Z}$ between logic goals is said to be a P-bisimulation iff it satisfies the following conditions for all goals F_1, G_1 such that $F_1 \mathcal{Z} G_1$:

- **TEST:** $F_1 = \square$ iff $G_1 = \square$,
- **BCK cond**: For each resolvent F_2 of F_1 and a clause in P, there exists a resolvent G_2 of G_1 and a clause in P such that $F_2 \mathcal{Z} G_2$,
- **FWD cond**: For each resolvent G_2 of G_1 and a clause in P, there exists a resolvent F_2 of F_1 and a clause in P such that $F_2 \mathcal{Z} G_2$.

We prove that there exists a maximal P-bisimulation, noted $\mathcal{Z}^P_{max}$. $\mathcal{Z}^P_{max}$ is an equivalence relation.

Example

Let P be the following program:
$p(a, y) \leftarrow q(y), \qquad p(b, y) \leftarrow r(y), \qquad p(b, y) \leftarrow s(y)$
and let $F = \leftarrow p(a, y)$ and $G = \leftarrow p(b, y)$.



Let $\mathcal{Z}$ be the binary relation between goals such that:
$\leftarrow p(a, y)\ \mathcal{Z} \leftarrow p(b, y)$,
$\leftarrow q(y)\ \mathcal{Z} \leftarrow r(y)$,
$\leftarrow q(y)\ \mathcal{Z} \leftarrow s(y)$.

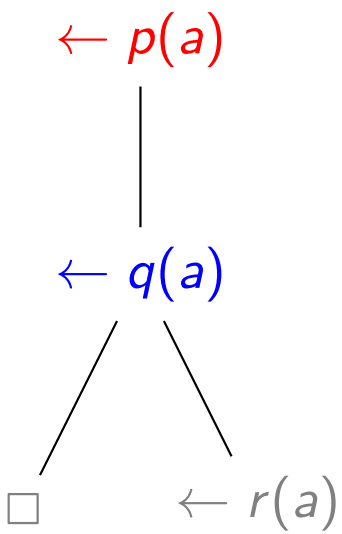
$\mathcal{Z}$ is a *P-bisimulation*. Since $F\ \mathcal{Z}\ G$, then $F\ \mathcal{Z}^P_{max}\ G$.

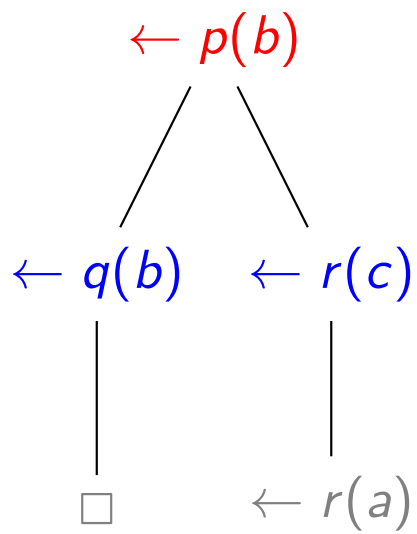
Example

Let P be the following program:

$p(a) \leftarrow q(a)$, $q(a) \leftarrow$, $p(a) \leftarrow r(a)$,
$p(b) \leftarrow q(b)$, $p(b) \leftarrow r(c)$, $q(b) \leftarrow$
$r(c) \leftarrow r(a)$

and let $F = \leftarrow p(a)$ and $G = \leftarrow p(b)$.





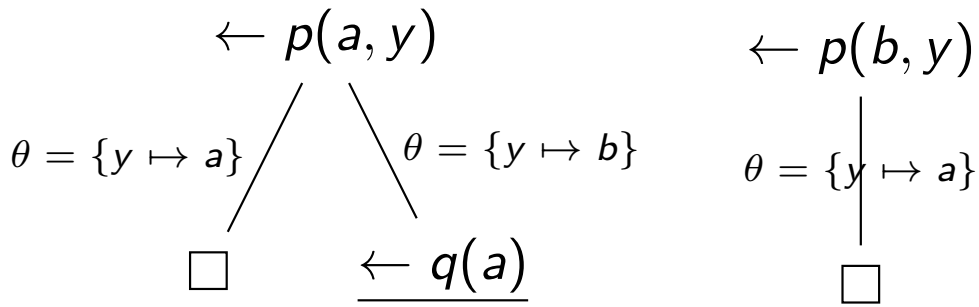
Information flow in logic programming

Definition 3 of information flow in logic programming - Bisimulation

- For a logic program P and a goal $G(x, y)$ of arity 2, $x \xrightarrow{BI}{}^{P}_{G} y$ iff $\exists a, b \in U_{L(P)}$ such that: $non\ G(a, y) \mathcal{Z}^{P}_{max} G(b, y)$

Example 3

Let P_3 be the following program:
$p(x, a) \leftarrow$
$p(a, b) \leftarrow q(a)$
and let $G_3(x, y)$ be the goal: $\leftarrow p(x, y)$.



$$x \xrightarrow{BI}{}^{P_3}_{G_3} y$$

Information flow in logic programming

Results

For a program P and a goal $G(x, y)$, we prove that:

- $x \xrightarrow{SF}{}^{P}_{G} y \Rightarrow x \xrightarrow{SA}{}^{P}_{G} y$
- $x \xrightarrow{SF}{}^{P}_{G} y \Rightarrow x \xrightarrow{BI}{}^{P}_{G} y$

For some program P and a goal $G(x, y)$, we have:

- $x \xrightarrow{SA}{}^{P}_{G} y$ and not $x \xrightarrow{SF}{}^{P}_{G} y$
- $x \xrightarrow{BI}{}^{P}_{G} y$ and not $x \xrightarrow{SF}{}^{P}_{G} y$

Information flow decision problems in logic programming

Consider the following decision problems:

Information flow decision problems for logic programs

$$\pi_{SF} \begin{cases} \text{Input: A logic program } P\text{, a goal } G(x,y) \\ \text{Output: Determine if } x \xrightarrow{SF}{}^{P}_{G} y \end{cases}$$

$$\pi_{SA} \begin{cases} \text{Input: A logic program } P\text{, a goal } G(x,y) \\ \text{Output: Determine if } x \xrightarrow{SA}{}^{P}_{G} y \end{cases}$$

$$\pi_{BI} \begin{cases} \text{Input: A logic program } P\text{, a goal } G(x,y) \\ \text{Output: Determine if } x \xrightarrow{BI}{}^{P}_{G} y \end{cases}$$

Information flow decision problems in logic programming - Undecidability

For logic program (in general), we prove that:

- π_{SF} is undecidable.
  Sketch of the proof:
  Reduce the following undecidable problem π_1 to π_{SF}
  Input: a logic program P,
  a (ground) goal $\leftarrow G$
  Output: $P \cup \{\leftarrow G\}$ succeeds

  Reduction

π_1	π_{SF}
$(P, \leftarrow G)$	$P' = P \cup \{p(a, y) \leftarrow G\}$

- π_{SA} is undecidable.

Information flow decision problems in logic programming - Undecidability

Information flow decision problems in logic programming - Undecidability

For logic program (in general), we prove that:

- π_{BI} is undecidable.
  Sketch of the proof:
  Reduce the following undecidable problem π_2 to π_{BI}
  Input: a binary program with exactly one clause, a goal $\leftarrow G$
  Output: SLD-tree for $P \cup \{\leftarrow G\}$ contains an infinite branch

Reduction

π_2	π_{BI}
$(P, \leftarrow G)$	$P' = P \cup \{p(a,y) \leftarrow G$ $p(b,y) \leftarrow G$ $p(b,y) \leftarrow p(c,y)$ $p(c,y) \leftarrow p(c,y)\}$

Information flow decision problems in logic programming - Decidability

Information flow decision problems in logic programming - Decidability

For Datalog programs, we prove that:

- π_{SF} is EXPTIME-complete.
  hardness: Reduce the following problem to π_{SF}
  Input: A Datalog program P, a ground atom A
  Output: $P?A$ (A is a logical consequence of P)
- π_{SA} is EXPTIME-complete.
- π_{BI} ?

Hierarchical Datalog programs

Clark (1977), Sheperdson (1985)

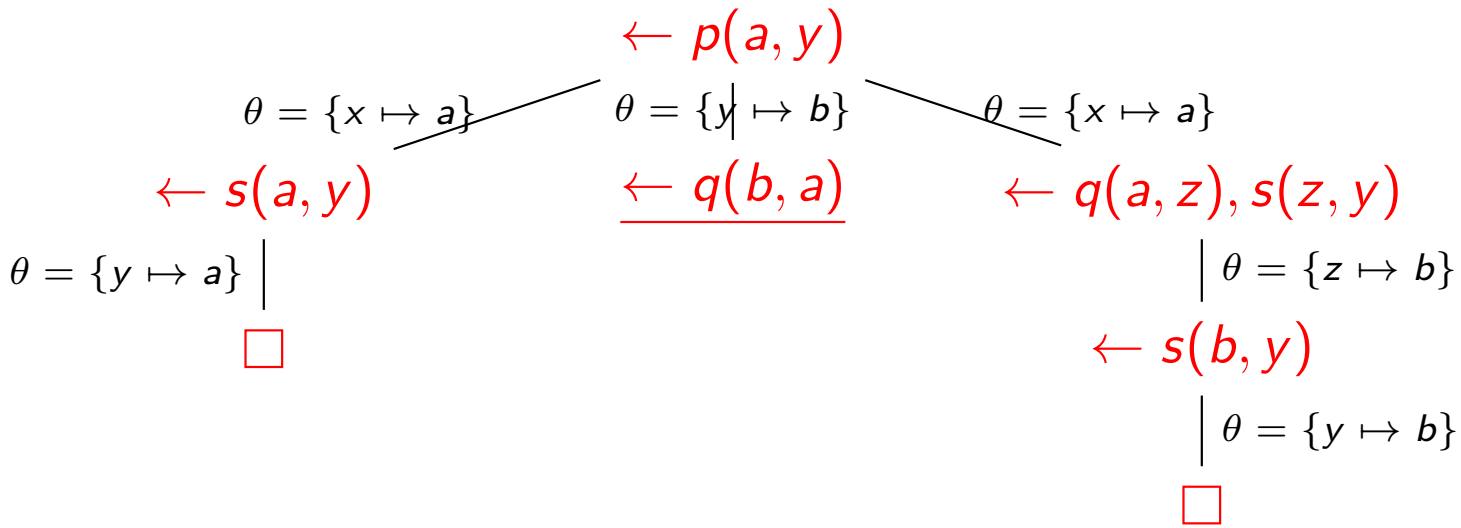
- for every predicate symbol p in P, associate a positive integer $l(p)$
- for all clauses of the form $p_0(\cdots) \leftarrow p_1(\cdots), \cdots, p_n(\cdots)$ $l(p_0) > l(p_1), \cdots, l(p_n)$.

Hierarchical Datalog programs

Example of a program:

$s(a,a) \leftarrow$
$s(b,b) \leftarrow$
$q(a,b) \leftarrow$
$p(x,y) \leftarrow s(x,y)$
$p(a,b) \leftarrow q(b,a)$
$p(x,y) \leftarrow q(x,z), s(z,y)$

$l(q) = 1$
$l(s) = 1$
$l(p) = 2$



Information flow decision problems in logic programming - Decidability

For hierarchical Datalog binary (body of the clause contains at most one atom) programs, we prove that:

- π_{SF} is in $\Delta_2 P$.
  Sketch of the proof: For all $a, b \in U_{L(P)}$
  If $(P \cup \{G(a, y)\} \in$ SUCCESSES and $P \cup \{G(b, y)\} \in$ FAILURES) **then** accept
  else reject
- π_{SA} is in EXPTIME.
- π_{BI} is in EXPTIME.

Bisimulation for hierachical Datalog programs

(π_{hie}): given an hierarchical Datalog program P and Datalog goals F_1, G_1, determine whether $F_1 \mathcal{Z}^P_{max} G_1$.

We can write a decision procedure $bisim1(F_1, G_1)$:

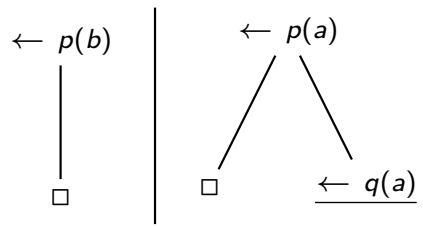
Bisimulation for hierachical Datalog programs

function bisim1(F_1,G_1)

if *bothempty*(F_1, G_1) **or** *bothfail*(F_1, G_1) **then**
  return *true*

else
  $SF \longleftarrow$ *successor*(F_1)
  $SG \longleftarrow$ *successor*(G_1)
  if $SF \neq \emptyset$ **and** $SG \neq \emptyset$ **then**
    $SF' \longleftarrow SF$
    while $SF' \neq \emptyset$ **do**
      $F_2 \longleftarrow$ *get-element*(SF')
      found-bisim $\longleftarrow$ *false*
      $SG' \longleftarrow SG$
      while $SG' \neq \emptyset$ **and** *found-bisim* $=$ *false* **do**
        $G_2 \longleftarrow$ *get-element*(SG')
        found-bisim $\longleftarrow$ *bisim1*(F_2, G_2)
      if *found-bisim* $=$ *false* **then**
        return *false*
    $SG' \longleftarrow SG$
    while $SG' \neq \emptyset$ **do**
      $G_2 \longleftarrow$ *get-element*(SG')
      found-bisim $\longleftarrow$ *false*
      $SF' \longleftarrow SF$
      while $SF' \neq \emptyset$ **and** *found-bisim* $=$ *false* **do**
        $F_2 \longleftarrow$ *get-element*(SF')
        found-bisim $\longleftarrow$ *bisim1*(G_2, F_2)
      if *found-bisim* $=$ *false* **then**
        return *false*
    return *true*
  else
    return *false*

Example:
Let P be the following hierarchical Datalog program:
$p(a) \leftarrow$
$p(b) \leftarrow$
$p(a) \leftarrow q(a)$
and let $F_1 = \leftarrow p(b)$, $G_1 = \leftarrow p(a)$



Bisimulation for hierachical Datalog programs

(π_{hie}): given an hierarchical Datalog program P and Datalog goals F_1, G_1, determine whether $F_1 \mathcal{Z}^P_{max} G_1$.

We can write a decision procedure $bisim1(F_1, G_1)$ satisfying:

Termination: $bisim1(F_1, G_1)$ terminates.

Completeness: If $F_1 \mathcal{Z}^P_{max} G_1$, then $bisim1(F_1, G_1)$ returns *true*.

Soundness: If $bisim1(F_1, G_1)$ returns *true*, then $F_1 \mathcal{Z}^P_{max} G_1$.

Bisimulation for hierachical Datalog programs

(π_{hie}) is in 2EXPTIME.

Bisimulation for hierachical Datalog programs

(π_{hie}): given an hierarchical Datalog program P and Datalog goals F_1, G_1, determine whether $F_1 \mathcal{Z}^P_{max} G_1$.

We can write a decision procedure $bisim1(F_1, G_1)$ satisfying:

Termination: $bisim1(F_1, G_1)$ terminates.
Proof done by $\ll$-induction on (F_1, G_1).
Let $\ll$ be the binary relation on the set of all pairs of Datalog goals defined by: $(F_2, G_2) \ll (F_1, G_1)$ iff

- the SLD-tree for F_1 is deeper than the SLD-tree for F_2,
- the SLD-tree for G_1 is deeper than the SLD-tree for G_2.

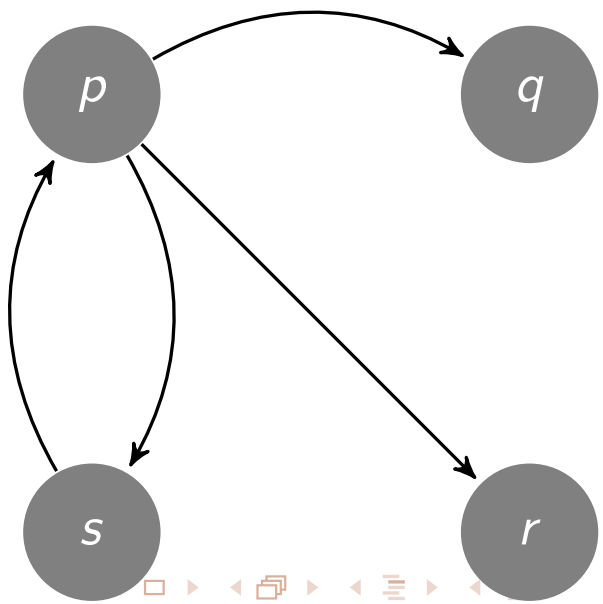
$\ll$ is a well-founded partial order on the set of all pairs of goals.

Restricted Datalog programs

Bol, Apt and Klop. (1991)

- Consider dependency graph (N, E) of a Datalog program P.
  N is the set of the predicate symbols in P.
  pEq if P contains a clause $p(\cdots) \leftarrow \cdots, q(\cdots), \cdots$
  E^* reflexive transitive closure of E.
- for all clauses of the form $p_0(\cdots) \leftarrow p_1(\cdots), \cdots, p_n(\cdots)$, and for all $1 \leq i \leq n-1$, p_i does not depend on p_0

Example:
Let P be:
$p(x, y) \leftarrow q(x), r(y), s(y, x)$
$s(x, y) \leftarrow p(x, y)$
$q(a) \leftarrow$
$r(b) \leftarrow$



Example

Let P be the following restricted Datalog program:
$q(a) \leftarrow$
$p(x) \leftarrow q(x), p(x)$

and let $F = \leftarrow p(a)$

$$\leftarrow p(a)$$
$$|$$
$$\leftarrow q(a), p(a)$$
$$|$$
$$\leftarrow p(a)$$
$$|$$
$$\vdots$$

Loop check

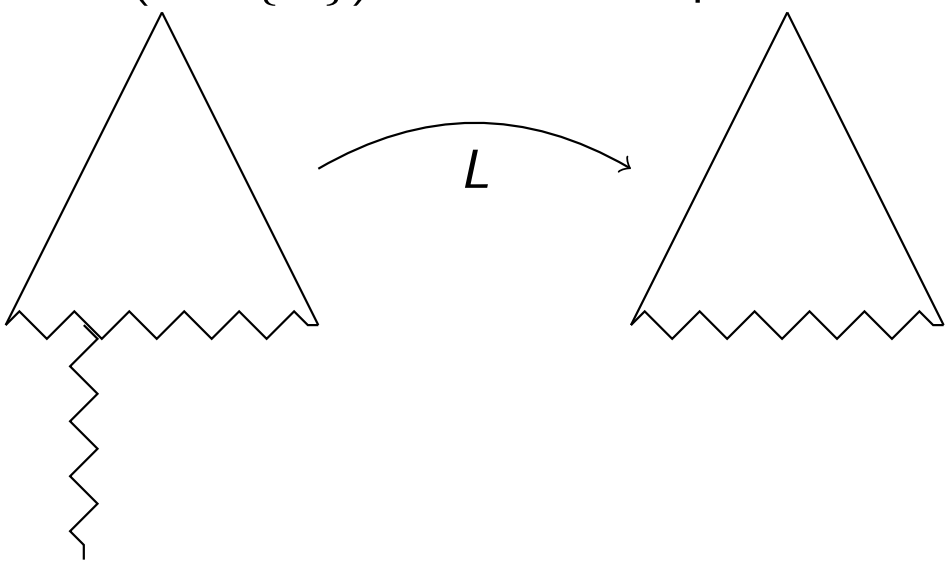
Apt *et al.* (1989), Bol *et al.* (1991), Bol (1990), Smith *et al.* (1986), Van Gelder (1987), Vieille (1989), Besnard (1989), Convington (1985), Sahlin (1993), Brough & Walker (1984), Shen (1997).

- Modify the computation mechanism by adding a capability of **pruning**.
- At some point, the interpreter is forced to stop its search through a certain part of the SLD-tree.
- The pruning of a node in an SLD-tree must depend only on its ancestors.

Loop check

P a program
G a goal
L a loop check

T =SLD-tree($P \cup \{G\}$) T' =pruned SLD-tree($P \cup \{G\}$)



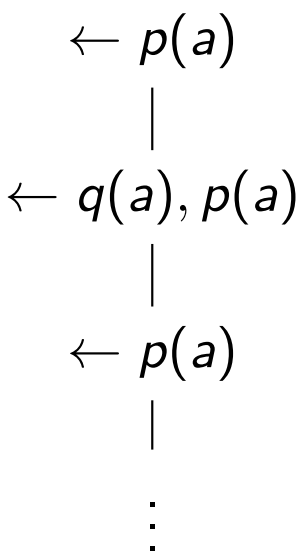
Example

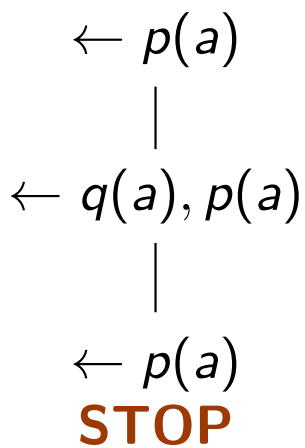
Let P be the following restricted Datalog program:

$q(a) \leftarrow$

$p(x) \leftarrow q(x), p(x)$

and let $F = \leftarrow p(a)$





Bisimulation for restricted Datalog programs

(π_{res}): given a restricted Datalog program P and Datalog goals F_1, G_1, determine whether $F_1 \mathcal{Z}^P_{max} G_1$.

We can write a decision procedure $bisim2((F_1),(G_1))$:

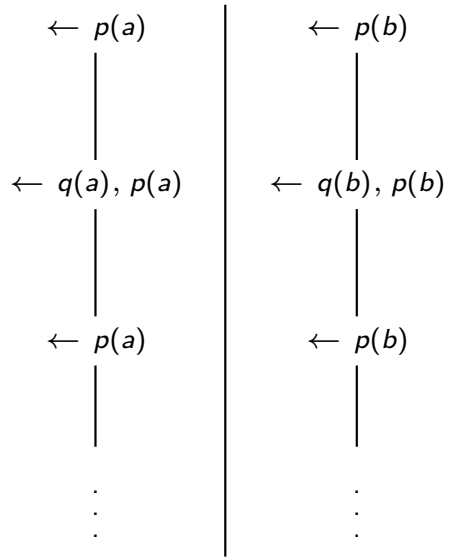
Bisimulation for restricted Datalog programs

function bisim2$((F_1 \Rightarrow \cdots \Rightarrow F_i),(G_1 \Rightarrow \cdots \Rightarrow G_i))$

if $bothempty(F_i, G_i)$ **or** $bothfail(F_i, G_i)$ **or** $occur((F_1 \Rightarrow \cdots \Rightarrow F_i), (G_1 \Rightarrow \cdots \Rightarrow G_i))$ **then**
  return *true*

else
  $SF \longleftarrow successor(F_i)$
  $SG \longleftarrow successor(G_i)$
  if $SF \neq \emptyset$ **and** $SG \neq \emptyset$ **then**
    $SF' \longleftarrow SF$
    while $SF' \neq \emptyset$ **do**
      $F' \longleftarrow get\text{-}element(SF')$
      $found\text{-}bisim \longleftarrow false$
      $SG' \longleftarrow SG$
      while $SG' \neq \emptyset$ **and** $found\text{-}bisim = false$ **do**
        $G' \longleftarrow get\text{-}element(SG')$
        $found\text{-}bisim \longleftarrow bisim2((F_1 \Rightarrow \cdots \Rightarrow F_i \Rightarrow F'), (G_1 \Rightarrow \cdots \Rightarrow G_i \Rightarrow G'))$
      if $found\text{-}bisim = false$ **then**
        return *false*
    $SG' \longleftarrow SG$
    while $SG' \neq \emptyset$ **do**
      $G' \longleftarrow get\text{-}element(SG')$
      $found\text{-}bisim \longleftarrow false$
      $SF' \longleftarrow SF$
      while $SF' \neq \emptyset$ **and** $found\text{-}bisim = false$ **do**
        $F' \longleftarrow get\text{-}element(SF')$
        $found\text{-}bisim \longleftarrow bisim2((G_1 \Rightarrow \cdots \Rightarrow G_i \Rightarrow G'), (F_1 \Rightarrow \cdots \Rightarrow F_i \Rightarrow F'))$
      if $found\text{-}bisim = false$ **then**
        return *false*
    return *true*
  else
    return *false*

Example:
Let P be the following restricted Datalog program:
$q(a) \leftarrow$
$q(b) \leftarrow$
$p(x) \leftarrow q(x), p(x)$
and let $F_1 = \leftarrow p(a)$, $G_1 = \leftarrow p(b)$



Bisimulation for restricted Datalog programs

(π_{res}): given a restricted Datalog program P and Datalog goals F_1, G_1, determine whether $F_1 \mathcal{Z}^P_{max} G_1$.

We can write a decision procedure $bisim2((F_1),(G_1))$ satisfying:

Termination: $bisim2((F_1),(G_1))$ terminates.

Completeness: If $F_1 \mathcal{Z}^P_{max} G_1$, then $bisim2((F_1),(G_1))$ returns *true*.

Soundness: If $bisim2((F_1),(G_1))$ returns *true*, then $F_1 \mathcal{Z}^P_{max} G_1$.

Bisimulation for restricted Datalog programs

(π_{res}) is in 2EXPTIME.

Bisimulation for restricted Datalog programs

(π_{res}): given a restricted Datalog program P and Datalog goals F_1, G_1, determine whether $F_1 \mathcal{Z}^P_{max} G_1$.

We can write a decision procedure $bisim2((F_1),(G_1))$ satisfying:

Termination: $bisim2((F_1),(G_1))$ terminates.
Proof done by $\prec$-induction on $((F_1),(G_1))$.
Let $\prec$ be the binary relation on the set of all pairs of SLD-derivations defined by: $((F_1 \Rightarrow \cdots \Rightarrow F_i),(G_1 \Rightarrow \cdots \Rightarrow G_i)) \prec ((F'_1 \Rightarrow \cdots \Rightarrow F'_j),(G'_1 \Rightarrow \cdots \Rightarrow G'_j))$ iff:

- $i > j$,
- $(F_1 \Rightarrow \cdots \Rightarrow F_j) = (F'_1 \Rightarrow \cdots \Rightarrow F'_j)$,
- $(G_1 \Rightarrow \cdots \Rightarrow G_j) = (G'_1 \Rightarrow \cdots \Rightarrow G'_j)$,
- there exists no substitutions σ, τ such that $F'_l = F'_k[\sigma]$, $G'_l = G'_k[\tau]$ for some $1 \leq k < l \leq j$.

$\prec$ is a well-founded partial order on the set of all pairs of SLD-derivations.

Outline of the presentation

Application - Level of a goal

Let P be a Datalog program, $G(x, y)$ be a goal. Let a, b be two elements of $U_{L(P)}$.

- $a \equiv_{SF} b$ iff both $P \cup \{\leftarrow p(a, y)\}$ and $P \cup \{\leftarrow p(b, y)\}$ succeed or both $P \cup \{\leftarrow p(a, y)\}$ and $P \cup \{\leftarrow p(b, y)\}$ do not succeed;
- $a \equiv_{SA} b$ iff $\theta(P \cup \{\leftarrow p(a, y)\}) = \theta(P \cup \{\leftarrow p(b, y)\})$;
- $a \equiv_{BI} b$ iff $P \cup \{\leftarrow p(a, y)\} Z^{P}_{max} P \cup \{\leftarrow p(b, y)\}$.

We prove that $\equiv_{SF}$, $\equiv_{SA}$ and $\equiv_{BI}$ are equivalence relations.

We define the **level of a goal** as the cardinality of the smallest equivalence class.

If the level of $G(x, y)$ is equal to 1, then the output variable y reveals information about the variable x.

Application - Example

Let P be the following program:
$C_1 : p(a, a) \leftarrow;$
$C_2 : p(a, b) \leftarrow;$
$C_3 : p(b, a) \leftarrow;$
and $\leftarrow p(x, y)$ a goal. $U_{L(P)}$ is equal to $\{a, b\}$.

For the definition of the flow based on success and failure:
$P \cup \{\leftarrow p(a, y)\}$ succeeds and $P \cup \{\leftarrow p(b, y)\}$ succeeds.
$\text{Level}(\leftarrow p(x, y)) = 2.$

For the definition of the flow based on substitution answers:
$\Theta[P \cup \{\leftarrow p(a, y)\}] = \{y \mapsto a, y \mapsto b\}$ and
$\Theta[P \cup \{\leftarrow p(b, y)\}] = \{y \mapsto a\}.$
$\text{Level}(\leftarrow p(x, y)) = 1.$

For the definition of the flow based on bisimulation:
$Tree(P \cup \{\leftarrow p(a, y)\})\ Z^P_{max}\ Tree(P \cup \{\leftarrow p(b, y)\}).$
$\text{Level}(\leftarrow p(x, y)) = 2.$

Application - Logic programs/Abstract functions

Logic programs as abstract functions:

Program P	**Function** f_P
$p(x, y)$	$f_p : U_{L(P)} \longmapsto R$ $a \longmapsto f_P(a)$
x: input position	$a \in U_{L(P)}$
y: output position	$f_P(a) \in \theta(P \cup \{\leftarrow p(a, y)\})$

Application - Example

Program

$age(ann, 56) \leftarrow$
$age(billy, 27) \leftarrow$
$age(carl, 34) \leftarrow$

Function

$age : U_{L(P)} \longmapsto R$

Let $\leftarrow age(x, y)$ be a goal.

Example:
For $x = ann$, $age(ann) = \theta(P \cup \{\leftarrow age(ann, y)\}) = \{y \mapsto 56\}$

Application - Protection mechanisms, Confidentiality policy

Protection mechanism produces:

1. the same value as the program for inputs not violating the policy
2. an error message for inputs revealing confidential information

For a program $f_P : U_{L(P)} \longmapsto R$.

$$m : \quad U_{L(P)} \longmapsto R \cup \{error_1, error_2, \cdots\}$$
$$a \in U_{L(P)} \longmapsto m(a) = f_P(a) \text{ OR } m(a) \in \{error_1, error_2, \cdots\}$$
$$E = \{error_1, error_2, \cdots\}$$

Confidentiality policy for P :

$$c : U_{L(P)} \longmapsto J \text{ such that } J \subseteq U_{L(P)}$$

m is **secure** iff there is a function $m' : J \to R \cup E$ such that, for all $a \in U_{L(P)}, m(a) = m'(c(a))$.

Application - Example

Program

$age(ann, 56) \leftarrow;$
$age(billy, 27) \leftarrow;$
$age(carl, 34) \leftarrow;$

Function

$age : U_{L(P)} \longmapsto R$

Protection mechanism:
$m : U_{L(P)} \longmapsto R \cup E$ for which:

- $m(a) = age(a)$ when $a \in U_{L(P)}$
- $m(a) = Error$, otherwise.

Confidentiality policy: bearing leaking information about *ann*.
$c : U_{L(P)} \longmapsto J$,
$U_{L(P)} = \{ann, billy, carl\}$, $J = \{billy, carl\}$
$c(billy) = billy, c(carl) = carl$ and $c(ann)$ is *undefined*.

Application - Example

Program

$age(ann, 56) \leftarrow;$
$age(billy, 27) \leftarrow;$
$age(carl, 34) \leftarrow;$

Function

$age : U_{L(P)} \longmapsto R$

Goal:	$P \cup \{\leftarrow age(billy, y)\}$	$\theta = \{y \mapsto 27\}$
Protection mec:	$m(billy)$	$\theta = \{y \mapsto 27\}$
Sec. mec:	$m(c(billy))$	$\theta = \{y \mapsto 27\}$

Goal:	$P \cup \{\leftarrow age(ann, y)\}$	$\theta = \{y \mapsto 56\}$
Protection mec:	$m(ann)$	$\theta = \{y \mapsto 56\}$
Sec. mec:	$m(c(ann))$	error

Application - Example

Program

$age(ann, 56) \leftarrow;$
$age(billy, 27) \leftarrow;$
$age(carl, 34) \leftarrow;$

Function

$age : U_{L(P)} \longmapsto R$

Protection mechanism:
$m_2 : U_{L(P)} \longmapsto R \cup E$ for which:

- $m(a) = age(a)$ when $\text{Level}(\leftarrow age(a, y)) > 1$
- $m(a) = Error$, otherwise.

Confidentiality policy: bearing leaking information about *ann*.
$c : U_{L(P)} \longmapsto J$,
$U_{L(P)} = \{ann, billy, carl\}$, $J = \{billy, carl\}$
$c(billy) = billy, c(carl) = carl$ and $c(ann)$ is *undefined*.

Application - Example

Program

$age(ann, 56) \leftarrow;$
$age(billy, 27) \leftarrow;$
$age(carl, 34) \leftarrow;$

Function

$age : U_{L(P)} \longmapsto R$

Goal:	$P \cup \{\leftarrow age(billy, y)\}$	$\theta = \{y \mapsto 27\}$
Protection mec:	$m(billy)$	error
Sec. mec:	$m(c(billy))$	error

Goal:	$P \cup \{\leftarrow age(ann, y)\}$	$\theta = \{y \mapsto 56\}$
Protection mec:	$m(ann)$	error
Sec. mec:	$m(c(ann))$	error

Application - Results

m_1, m_2 two distinct protection mechanisms for P under c.

m_1 is as precise as m_2 $(m_1 \succ m_2)$ if for all $a \in U_{L(P)}$, if $m_2(a) = f_P(a)$, then $m_1(a) = f_P(a)$.

m_1 is more precise than m_2 $(m_1 \gg m_2)$ if:

- $(m_1 \succ m_2)$ and
- $\exists b \in U_{L(P)}$ such that $m_1(b) = f_P(b)$ and $m_2(b) \neq f_P(b)$.

We can prove that:
$\succ$ is reflexive and transitive and that
$\gg$ is a strict ordering
on the protection mechanisms for a given P and c.

Application - Results

$m_3 = m_1 \cup m_2$ is defined as

$$m_3(a) \begin{cases} = f_P(a) \text{ when } m_1(a) = f_P(a) \text{ or } m_2(a) = f_P(a) \\ = m_1(a) \text{ otherwise.} \end{cases}$$

$$\begin{cases} m_1 \text{ secure for } P \text{ under } c \\ m_2 \text{ secure for } P \text{ under } c \end{cases} \Rightarrow m_1 \cup m_2 \text{ secure for } P \text{ under } c.$$

We can show that, $m_1 \cup m_2 \succ m_1$ and $m_1 \cup m_2 \succ m_2$.

There exists a precise, secure mechanism m^* such that, for all secure mechanisms m associated with p and c, $m^* \succ m$.

Outline of the presentation

Conclusion

- **Information flow in logic programming**:
  1. Three definitions of information flows in logic programs.
  2. Notion of bisimulation between goals.
  3. Links between these definitions.
  4. Generalization of the definitions for goals with arity higher than 2.
  5. Flow manipulation via program transformation.

- **Undecidability / decidability of the flow in logic programming**:
  For a logic program P and a two variables goal $\leftarrow G(x, y)$, determining whether there is a flow of information from x to y.

	General setting	Datalog programs	Binary hierarchical Datalog programs
π_{SF}	Undecidable	EXPTIME-complete	in $\Delta_2 P$
π_{SA}	Undecidable	EXPTIME-complete	in EXPTIME
π_{BI}	Undecidable	?	in EXPTIME

Conclusion

- **Bisimulation of logic goals**:

Logic programs	**Decidability and complexity results**
Prolog programs	Undecidable
Datalog programs	?
hierarchical Datalog programs	2EXPTIME
restricted Datalog programs	2EXPTIME
nvi Datalog programs	?
svo Datalog programs	?
hierarchical Datalog + negation	✓
restricted Datalog + negation	**ongoing**

- **Preventive inference control for deductive databases**:
  1. We proposed the notion of confidentiality policy, secure mechanism, precise security mechanism for logic programming.
  2. We gave a precise and secure mechanism for deductive databases.

Conclusion

Future Work

1. **More formal work**:
   - Looking for new types of logic programs for which the question of the existence of the flow could be decided too.
   - Continue investigating the decidability of the existence of the flow relatively to bisimulation in Datalog programs without considering loop checking techniques.
   - How to use the notion of bisimulation between goals in order to define bisimulation between logic programs.
2. **Implementation**: different algorithms.
3. **Real-Time Databases**:
   - how to embed our security mechanism framework in real-time databases.
   - how our framework can enforce databases security policies.
   - how to bring changes on our framework in order to balance real-time requirements with security.

Conclusion

Thank you ...