Foundations of Rule-Based Query Answering

François Bry, Norbert Eisinger, Thomas Eiter, Tim Furche, Georg Gottlob, Clemens Ley, Benedikt Linse, Reinhard Pichler, Fang Wei

Institute for Informatics, University of Munich
Institute of Information Systems, Vienna University of Technology
Oxford University Computing Laboratory

Supported by the European Network of Excellence
http://rewerse.net/

Antoniou et al. (Eds.): Reasoning Web 2007
Springer LNCS 4636, pp.1–153

Monday, 3rd September, 2007

Session 8:30 – 10:30

- 1 Introduction
- 3 Syntax
- 4 Declarative Semantics: Fundamentals

Session 11:00 – 13:00

- 5 Declarative Semantics: Adaptations

Session 14:30 – 16:00

- 6 Operational Semantics: Positive
- 7 Operational Semantics: Negative

Session 16:30 – 18:00

- 8 Complexity and Expressive Power

Essential concepts and methods of rule-based query languages

Session 8:30 – 10:30

- 1 Introduction
- 3 Syntax: From First-Order Predicate Logic to Query Language Fragments of First-Order Predicate Logic
- 4 Declarative Semantics: Fundamentals of Classical Model Theory

What are Query Languages? Tentative Definitions

1. **What are ...** their purposes of use?
   selecting and retrieving data from "information systems"
2. **What are ...** their programming paradigms?
   declarative, hence related to logic
3. **What are ...** their major representatives?
   SQL (relational data), OQL (object-oriented data),
   XPath, XQuery (HTML and XML data),
   RQL, RDQL, SPARQL (RDF data),
   forthcoming ones (OWL ontologies)
4. **What are ...** their research issues?
   query paradigms, declarative semantics, complexity and expressive power, procedural semantics, implementations, optimisation, and many more ...

What are Query Languages? Tentative Definitions

1. **What are ...** their purposes of use?
   selecting and retrieving data from "information systems"
2. **What are ...** their programming paradigms?
   declarative, hence related to logic
3. **What are ...** their major representatives?
   SQL (relational data), OQL (object-oriented data),
   XPath, XQuery (HTML and XML data),
   RQL, RDQL, SPARQL (RDF data),
   forthcoming ones (OWL ontologies)
4. **What are ...** their research issues?
   query paradigms, declarative semantics, complexity and expressive power, procedural semantics, implementations, optimisation, and many more ...

What are Query Languages? Tentative Definitions

1. **What are ...** their purposes of use?
   selecting and retrieving data from "information systems"
2. **What are ...** their programming paradigms?
   declarative, hence related to logic
3. **What are ...** their major representatives?
   SQL (relational data), OQL (object-oriented data),
   XPath, XQuery (HTML and XML data),
   RQL, RDQL, SPARQL (RDF data),
   forthcoming ones (OWL ontologies)
4. **What are ...** their research issues?
   query paradigms, declarative semantics, complexity and expressive power, procedural semantics, implementations, optimisation, and many more ...

What are Query Languages? Tentative Definitions

1. **What are ...** their purposes of use?
   selecting and retrieving data from "information systems"
2. **What are ...** their programming paradigms?
   declarative, hence related to logic
3. **What are ...** their major representatives?
   SQL (relational data), OQL (object-oriented data),
   XPath, XQuery (HTML and XML data),
   RQL, RDQL, SPARQL (RDF data),
   forthcoming ones (OWL ontologies)
4. **What are ...** their research issues?
   query paradigms, declarative semantics, complexity and expressive power, procedural semantics, implementations, optimisation, and many more ...

Coverage of This Survey

- Foundations of query languages,
- Focus on logic, complexity and expressive power (query optimisation in proceedings only)

Limited coverage, but

- corner stone for most research
- already a large field
- unity of concerns and methods

Session 8:30 – 10:30

- 1 Introduction
- 3 Syntax: From First-Order Predicate Logic to Query Language Fragments of First-Order Predicate Logic
- 4 Declarative Semantics: Fundamentals of Classical Model Theory

(Computably) Enumerable Set

S enumerable → 2.1 p.4, Def.1

exists surjection $\mathbb{N} \to S$

computably enumerable

enumerable with algorithmically computable surjection

(Computably) Enumerable Set

S enumerable → 2.1 p.4, Def.1

exists surjection $\mathbb{N} \to S$

computably enumerable

enumerable with algorithmically computable surjection

Examples

- Set of all C programs

  enumerable and computably enumerable

- Set of all terminating C programs

  enumerable, not computably enumerable

3.1 Syntax of First-Order Predicate Logic

Formula → p.7, Def.6

$$person(Mary) \wedge person(Tom) \wedge company(Web5.0) \wedge \forall x \Big(company(x) \Rightarrow person(founder(x)) \Big) \wedge \Big(married(Mary, Tom) \vee \exists y \big[company(y) \wedge married(founder(y), Tom) \big] \Big)$$

3.1 Syntax of First-Order Predicate Logic

Logical Symbols → p.6, Def.2

punctuation, connectives, quantifiers, variables

Signature Symbols → p.6, Def.3

n-ary function symbols (0-ary = constant)

n-ary relation symbols

Formula → p.7, Def.6

$$\mathit{person}(\mathit{Mary}) \wedge \mathit{person}(\mathit{Tom}) \wedge \mathit{company}(\mathit{Web5.0}) \wedge \forall x \Big(\mathit{company}(x) \Rightarrow \mathit{person}(\mathit{founder}(x)) \Big) \wedge \Big(\mathit{married}(\mathit{Mary}, \mathit{Tom}) \vee \exists y \big[\mathit{company}(y) \wedge \mathit{married}(\mathit{founder}(y), \mathit{Tom}) \big] \Big)$$

3.1 Syntax of First-Order Predicate Logic

Logical Symbols → p.6, Def.2

punctuation, connectives, quantifiers, variables

Signature Symbols → p.6, Def.3

n-ary function symbols (0-ary = constant)

n-ary relation symbols

Term → p.6, Def.4

x $\quad$ $Mary$ $\quad$ $founder(x)$ $\quad$ $founder(Web5.0)$

Atom → p.6, Def.5

$married(Mary, Tom)$ $\quad$ $married(founder(y), Tom)$

Formula → p.7, Def.6

$$person(Mary) \wedge person(Tom) \wedge company(Web5.0) \wedge$$
$$\forall x \Big(company(x) \Rightarrow person(founder(x)) \Big) \wedge$$
$$\Big(married(Mary, Tom) \vee \exists y \big[company(y) \wedge married(founder(y), Tom) \big] \Big)$$

Standard Notions

Subformula		→ p.7, Def.7
Scope		→ p.7, Def.8
Bound/free	$\big(\forall x\big[\exists x\, p(x) \wedge q(x)\big] \Rightarrow \big[r(\boldsymbol{x}) \vee \forall x\, s(x)\big]\big)$	→ p.7, Def.8
Rectified	$\big(\forall u\big[\exists v\, p(v) \wedge q(u)\big] \Rightarrow \big[r(x) \vee \forall w\, s(w)\big]\big)$	→ p.8, Def.10
Closed	no free variables	→ p.8, Def.11
Ground	no variables	→ p.8, Def.11
Propositional	$\big(\big[p \wedge q\big] \Rightarrow \big[r \vee s\big]\big)$	→ p.8, Def.12

Universal Formula

Polarity

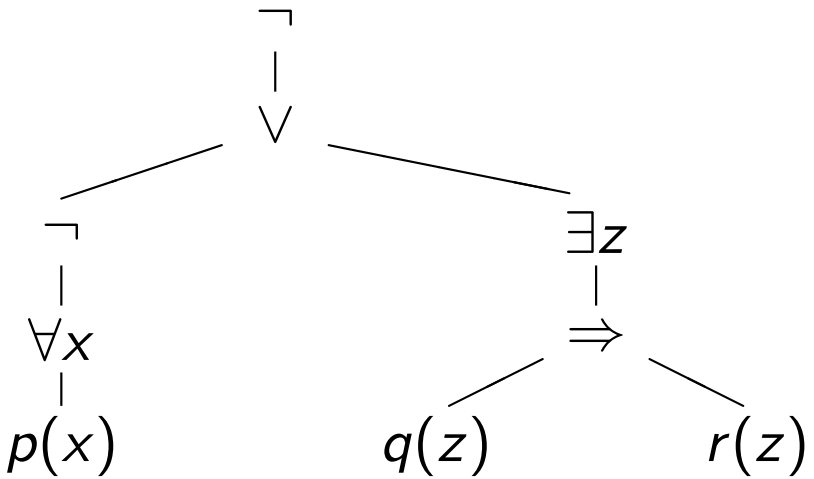
→ p.8, Def.13

$$\neg\Big(\neg\forall x\, p(x) \;\vee\; \exists z\Big(q(z) \Rightarrow r(z)\Big)\Big)$$

Universal Formula

Polarity

→ p.8, Def.13

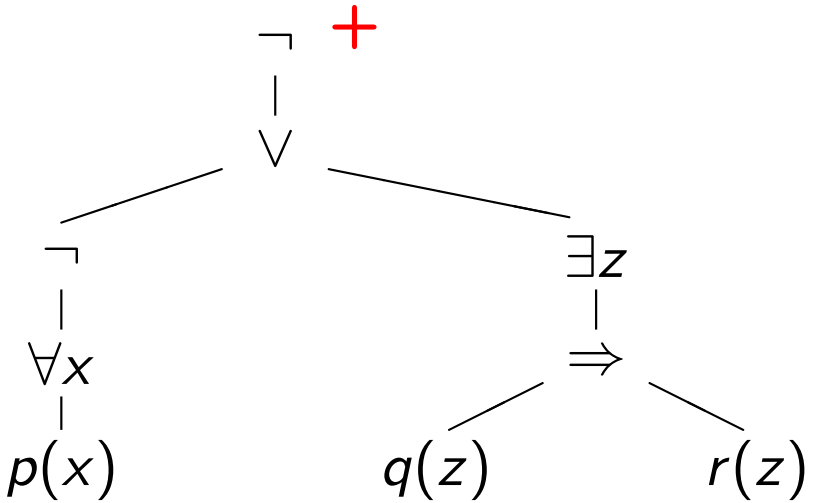


$$\neg\Big(\neg\forall x\, p(x) \;\vee\; \exists z\Big(q(z) \Rightarrow r(z)\Big)\Big)$$

Universal Formula

Polarity

→ p.8, Def.13

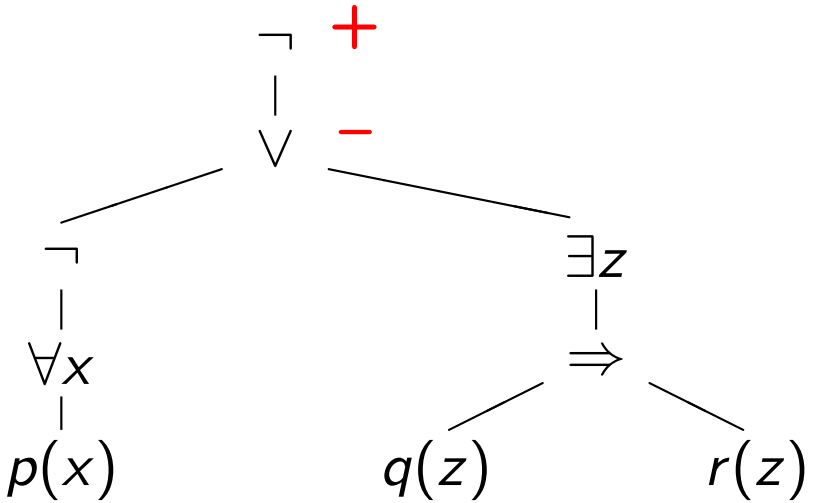


$$\neg\Big(\neg\forall x\, p(x) \;\vee\; \exists z\Big(q(z) \Rightarrow r(z)\Big)\Big)$$

Universal Formula

Polarity

→ p.8, Def.13

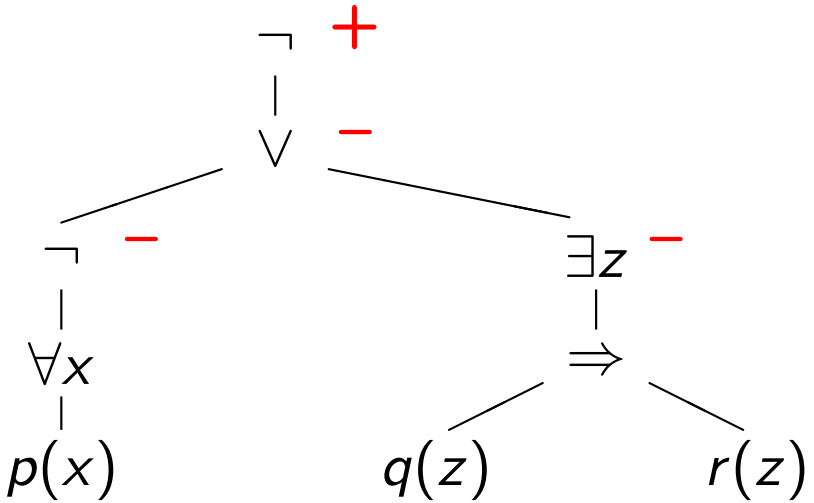


$$\neg\Big(\neg\forall x\, p(x) \;\vee\; \exists z\Big(q(z) \Rightarrow r(z)\Big)\Big)$$

Universal Formula

Polarity

→ p.8, Def.13

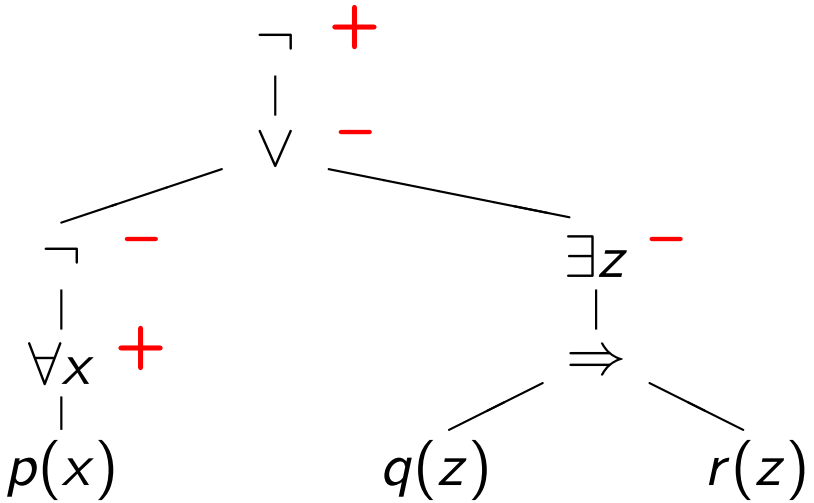


$$\neg\Big(\neg\forall x\, p(x) \;\vee\; \exists z\Big(q(z) \Rightarrow r(z)\Big)\Big)$$

Universal Formula

Polarity

→ p.8, Def.13

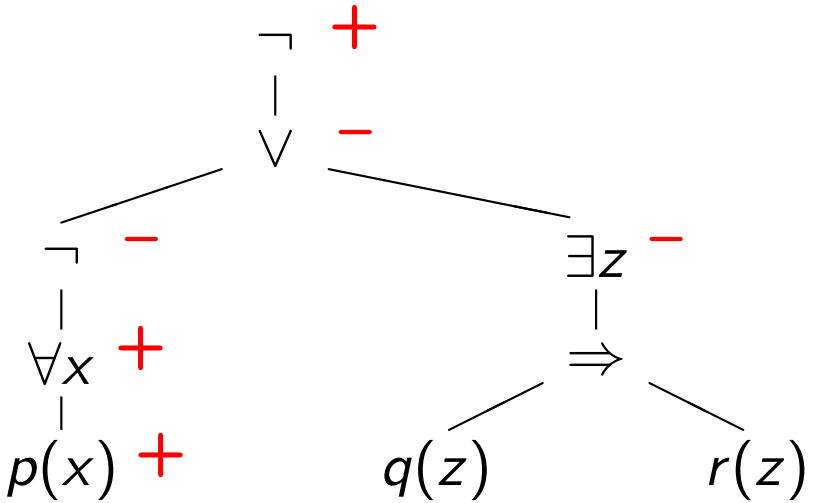


$$\neg\Big(\neg\forall x\, p(x) \;\vee\; \exists z\Big(q(z) \Rightarrow r(z)\Big)\Big)$$

Universal Formula

Polarity

→ p.8, Def.13

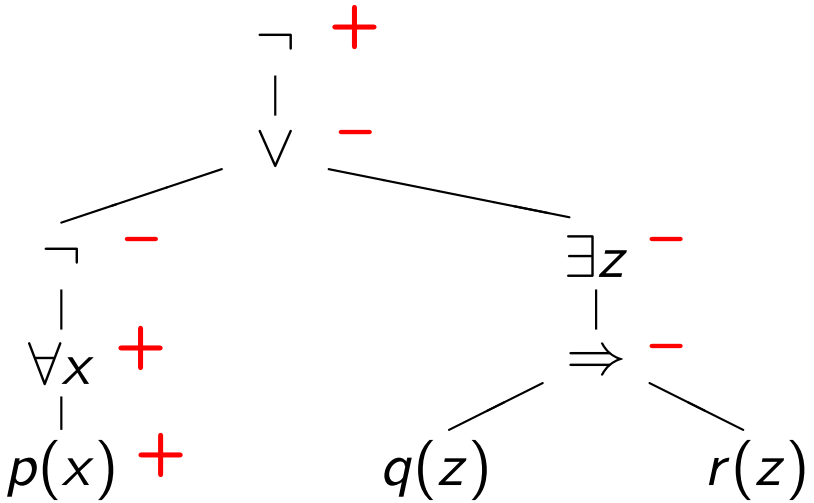


$$\neg\Big(\neg\forall x\, p(x) \;\vee\; \exists z\Big(q(z) \Rightarrow r(z)\Big)\Big)$$

Universal Formula

Polarity

→ p.8, Def.13



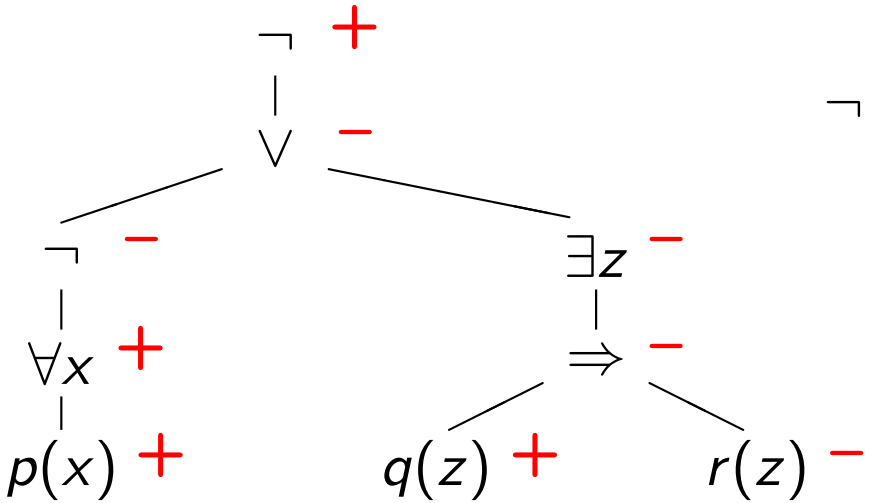
$$\neg\Big(\neg\forall x\, p(x) \;\vee\; \exists z\Big(q(z) \Rightarrow r(z)\Big)\Big)$$

Universal Formula

Polarity

→ p.8, Def.13

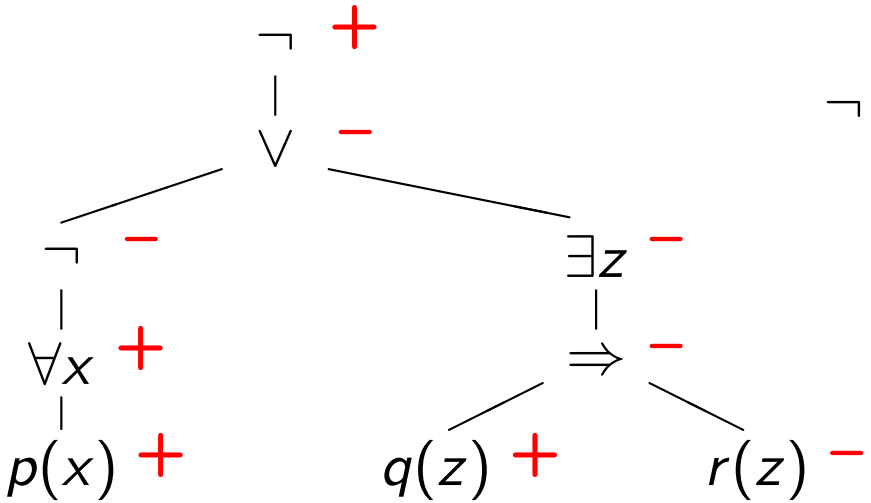
$$\neg\Big(\neg\forall x\, p(x) \;\vee\; \exists z\Big(q(z) \Rightarrow r(z)\Big)\Big)$$



Universal Formula

Polarity

→ p.8, Def.13

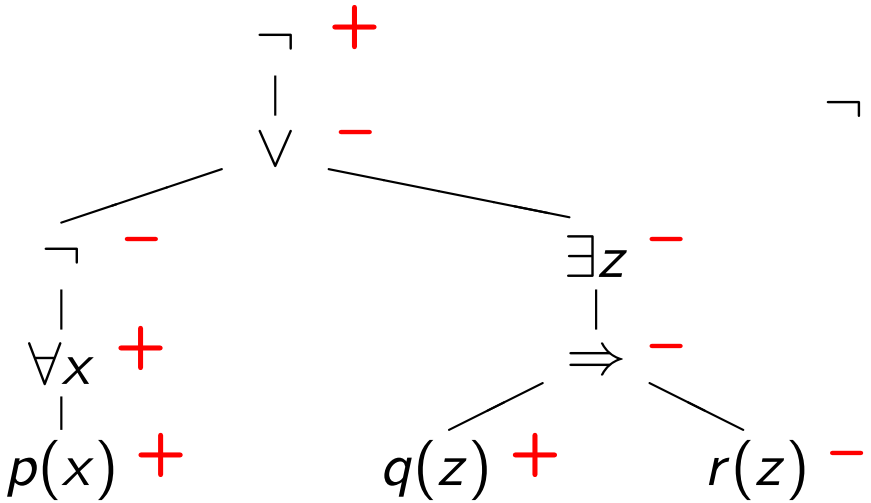


$$\neg\Big(\overset{+}{\neg\forall x\, p(x)} \;\vee\; \overset{-}{\exists z}\Big(q(z) \Rightarrow r(z)\Big)\Big)$$

Universal Formula

Polarity

→ p.8, Def.13



$$\neg\Big(\overset{+}{\neg\forall x\, p(x)} \;\vee\; \overset{-}{\exists z}\Big(q(z) \Rightarrow r(z)\Big)\Big)$$

Universal

→ p.9, Def.14

only $\forall^{+}$ and $\exists^{-}$

Prenex form

→ p.9, Def.15

$$\forall x\, \forall z\, \neg\Big(\neg p(x) \vee \Big(q(z) \Rightarrow r(z)\Big)\Big)$$

3.2 Query and Rule Language Fragments

Rule notation *consequent* $\leftarrow$ *antecedent* $\rightarrow$ p.9, Ntn.18

Range restricted

$p(x,z) \leftarrow p(x,y) \wedge p(y,z)$	yes	$\rightarrow$ p.11, Def.23
$p(x,z) \leftarrow p(x,x)$	no	

3.2 Query and Rule Language Fragments

Rule notation $\quad consequent \leftarrow antecedent$ → p.9, Ntn.18

stands for $\quad (antecedent \Rightarrow consequent)$

Range restricted

$p(x,z) \leftarrow p(x,y) \wedge p(y,z)$	yes
$p(x,z) \leftarrow p(x,x)$	no

→ p.11, Def.23

3.2 Query and Rule Language Fragments

Rule notation $consequent \leftarrow antecedent$ → p.9, Ntn.18

stands for $(antecedent \Rightarrow consequent)$

or for $\forall^*(antecedent \Rightarrow consequent)$

Range restricted

$p(x,z) \leftarrow p(x,y) \wedge p(y,z)$	yes
$p(x,z) \leftarrow p(x,x)$	no

→ p.11, Def.23

3.2 Query and Rule Language Fragments

Rule notation $consequent \leftarrow antecedent$ → p.9, Ntn.18

stands for $(antecedent \Rightarrow consequent)$

or for $\forall^*(antecedent \Rightarrow consequent)$

or for something else, various nonclassical semantics

Range restricted

$p(x,z) \leftarrow p(x,y) \wedge p(y,z)$	yes	→ p.11, Def.23
$p(x,z) \leftarrow p(x,x)$	no	

3.2 Query and Rule Language Fragments

Rule notation $consequent \leftarrow antecedent$ → p.9, Ntn.18

stands for $(antecedent \Rightarrow consequent)$

or for $\forall^*(antecedent \Rightarrow consequent)$

or for something else, various nonclassical semantics

Literal atom A, negated atom $\neg A$ → p.10, Def.19

Clause $A_1 \vee \ldots \vee A_k \leftarrow L_1 \wedge \ldots \wedge L_n$ → p.10, Def.20

atoms A_i, literals L_j, $k \geq 0$, $n \geq 0$

Range restricted $p(x,z) \leftarrow p(x,y) \wedge p(y,z)$ yes → p.11, Def.23

$p(x,z) \leftarrow p(x,x)$ no

Logic Programming Clauses

→ 3.2.1 p.10

	Name	Form	
Horn clause	definite clause	$A \leftarrow B_1 \wedge \ldots \wedge B_n$	$k = 1,\ n \geq 0$
	unit cl.	$A \leftarrow$	$k = 1,\ n = 0$
	definite goal	$\leftarrow B_1 \wedge \ldots \wedge B_n$	$k = 0,\ n \geq 0$
	empty cl.	$\leftarrow$	$k = 0,\ n = 0$
	normal clause	$A \leftarrow L_1 \wedge \ldots \wedge L_n$	$k = 1,\ n \geq 0$
	normal goal	$\leftarrow L_1 \wedge \ldots \wedge L_n$	$k = 0,\ n \geq 0$
	disjunctive clause	$A_1 \vee \ldots \vee A_k \leftarrow B_1 \wedge \ldots \wedge B_n$	$k \geq 0,\ n \geq 0$
	general clause	$A_1 \vee \ldots \vee A_k \leftarrow L_1 \wedge \ldots \wedge L_n$	$k \geq 0,\ n \geq 0$

atoms A, A_j, B_i, literals L_i, $k, n \in \mathbb{N}$

Datalog

→ 3.2.2 p.11

- definite clauses
- no function symbols except constants
- range restricted
- **extensional** relation symbols only in antecedents
  intensional relation symbols also in consequents

Datalog

→ 3.2.2 p.11

- definite clauses
- no function symbols except constants
- range restricted
- **extensional** relation symbols only in antecedents
  intensional relation symbols also in consequents

Monadic datalog 1-ary intensional relation symbols → p.12

Nonrecursive datalog no recursion

Linear datalog at most one intensional atom per antecedent

Disjunctive datalog disjunctive clauses

Datalog$^\neg$ normal clauses

Nonrecursive datalog$^\neg$ datalog$^\neg$, no direct or indirect recursion

Disjunctive datalog$^\neg$ datalog$^\neg$, general clauses

Conjunctive Queries

→ 3.2.3 p.12

Extensional: *parent*, *male*, *female*

$ans() \leftarrow parent(Mary, Tom)$	Is Mary a parent of Tom?
$ans() \leftarrow parent(Mary, y)$	Does Mary have children?
$ans(x) \leftarrow parent(x, Tom)$	Who are Tom's parents?
$ans(x) \leftarrow female(x) \wedge parent(x, y) \wedge parent(y, Tom)$	Who are Tom's grandmothers?
$ans(x, z) \leftarrow male(x) \wedge parent(x, y) \wedge parent(y, z)$	Who are grandfathers and their grandchildren?

Conjunctive Queries

→ 3.2.3 p.12

Extensional: *parent*, *male*, *female*

$ans() \leftarrow parent(Mary, Tom)$	Is Mary a parent of Tom?
$ans() \leftarrow parent(Mary, y)$	Does Mary have children?
$ans(x) \leftarrow parent(x, Tom)$	Who are Tom's parents?
$ans(x) \leftarrow female(x) \wedge parent(x, y) \wedge parent(y, Tom)$	Who are Tom's grandmothers?
$ans(x, z) \leftarrow male(x) \wedge parent(x, y) \wedge parent(y, z)$	Who are grandfathers and their grandchildren?

- nonrecursive datalog with extensional antecedents corresponds to SPC subclass of relational algebra queries
- combined with disjunction / negation / quantification corresponds to other subclasses of relational algebra queries

Conjunctive Queries

→ 3.2.3 p.12

Extensional: *parent*, *male*, *female*

$ans() \leftarrow parent(Mary, Tom)$	Is Mary a parent of Tom?
$ans() \leftarrow parent(Mary, y)$	Does Mary have children?
$ans(x) \leftarrow parent(x, Tom)$	Who are Tom's parents?
$ans(x) \leftarrow female(x) \wedge parent(x, y) \wedge parent(y, Tom)$	Who are Tom's grandmothers?
$ans(x, z) \leftarrow male(x) \wedge parent(x, y) \wedge parent(y, z)$	Who are grandfathers and their grandchildren?

- nonrecursive datalog with extensional antecedents
  corresponds to SPC subclass of relational algebra queries
- combined with disjunction / negation / quantification
  corresponds to other subclasses of relational algebra queries
- combined with recursion
  more expressive power than relational algebra queries

3.3 Syntactic Variations from

object-oriented / knowledge representation → 3.3.1 p.15

- record-like structures
- cyclic structures
- object identity
- roles (or slots)

relational databases → 3.3.2 p.16

- roles
- relational calculus

logic → 3.3.3 p.17

- range restricted quantification
- many-sorted first-order predicate logic

3.3 Syntactic Variations from

object-oriented / knowledge representation → 3.3.1 p.15

- record-like structures
- cyclic structures
- object identity
- roles (or slots)

relational databases → 3.3.2 p.16

- roles
- relational calculus

logic → 3.3.3 p.17

- range restricted quantification
- many-sorted first-order predicate logic

Can be considered as syntactic sugaring of first-order predicate logic

Session 8:30 – 10:30

4.1 Classical Tarski Model Theory

Principle of any Tarski-style semantics → 4 p.19

meaning of compound syntactic structure =
composition of meanings of immediate constituents

Advantage for computational treatment

simple recursive definition
well-defined, finite, and restricted computation scope

We'll come to disadvantages later...

Tarski-Interpretation

Signature

function symbols: 0-ary a, b 1-ary f

relation symbols: 1-ary p, q 2-ary r

Tarski-Interpretation

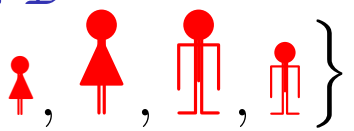
Signature

function symbols: 0-ary a, b 1-ary f

relation symbols: 1-ary p, q 2-ary r

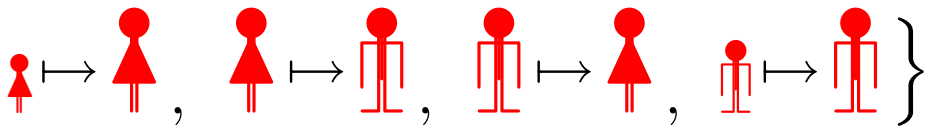
Interpretation $\mathcal{I}$

→ p.20, Def.29–31

$dom(\mathcal{I}) = \{$  $\}$

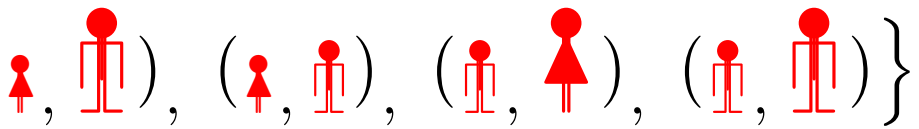
$a^{\mathcal{I}} =$ 

$b^{\mathcal{I}} =$ 

$f^{\mathcal{I}} = \{$  $\}$

$p^{\mathcal{I}} = \{$  $\}$

$q^{\mathcal{I}} = \{$  $\}$

$r^{\mathcal{I}} = \{$  $\}$

Tarski-Interpretation

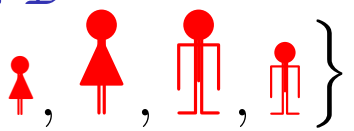
Signature

function symbols: 0-ary a, b 1-ary f

relation symbols: 1-ary p, q 2-ary r

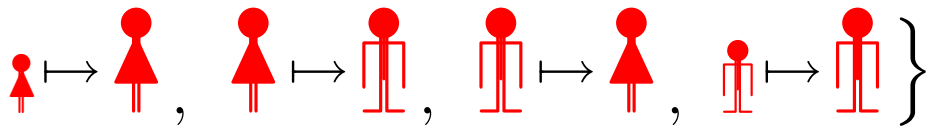
Interpretation $\mathcal{I}$

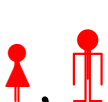
→ p.20, Def.29–31

$dom(\mathcal{I}) = \{$  $\}$

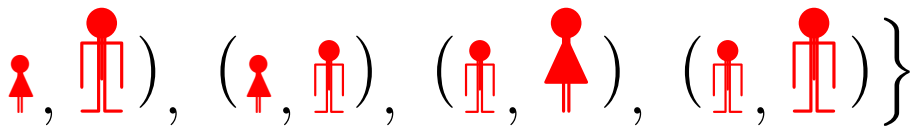
$a^{\mathcal{I}} =$ 

$b^{\mathcal{I}} =$ 

$f^{\mathcal{I}} = \{$  $\}$

$p^{\mathcal{I}} = \{$  $\}$

$q^{\mathcal{I}} = \{$  $\}$

$r^{\mathcal{I}} = \{$  $\}$

Model relationship $\models$

→ p.21, Def.32

$$\mathcal{I} \models q(a) \wedge r(a,b) \wedge \neg r(f(a),b) \wedge \forall x\Big(p(x) \Rightarrow r(x,f(x))\Big)$$

Semantic Properties

A formula is → p.21, Def.33

valid	iff	it is satisfied in each interpretation	$p \vee \neg p$
satisfiable	iff	it is satisfied in at least one interpretation	p
falsifiable	iff	it is falsified in at least one interpretation	p
unsatisfiable	iff	it is falsified in each interpretation	$p \wedge \neg p$

Semantic Properties, Entailment, Log. Equivalence

A formula is → p.21, Def.33

valid	iff	it is satisfied in each interpretation	$p \vee \neg p$
satisfiable	iff	it is satisfied in at least one interpretation	p
falsifiable	iff	it is falsified in at least one interpretation	p
unsatisfiable	iff	it is falsified in each interpretation	$p \wedge \neg p$

For formulas φ and ψ → p.21, Def.34

$\varphi \models \psi$	iff	for each interpretation $\mathcal{I}$: if $\mathcal{I} \models \varphi$ then $\mathcal{I} \models \psi$	$(p \wedge q) \models (p \vee q)$
$\varphi \models\!\!\models \psi$	iff	$\varphi \models \psi$ and $\psi \models \varphi$	$(p \wedge q) \models\!\!\models (q \wedge p)$

Semantic Properties, Entailment, Log. Equivalence

A formula is → p.21, Def.33

valid	iff	it is satisfied in each interpretation	$p \vee \neg p$
satisfiable	iff	it is satisfied in at least one interpretation	p
falsifiable	iff	it is falsified in at least one interpretation	p
unsatisfiable	iff	it is falsified in each interpretation	$p \wedge \neg p$

For formulas φ and ψ → p.21, Def.34

$\varphi \models \psi$ iff for each interpretation $\mathcal{I}$:
if $\mathcal{I} \models \varphi$ then $\mathcal{I} \models \psi$ $(p \wedge q) \models (p \vee q)$

$\varphi \models\!\models \psi$ iff $\varphi \models \psi$ and $\psi \models \varphi$ $(p \wedge q) \models\!\models (q \wedge p)$

Inter-translatable → p.21, Thm.35

Being able to determine one of validity, unsatisfiability, or entailment, is sufficient to determine all of them.

Significance for Query Languages

Logical understanding of yes/no query

does $(data \wedge rules) \models query$ hold?

However

depends by definition on all interpretations
(there are at least as many as there are sets)

no starting-point for algorithmic treatment

Results about Tarski Model Theory

Gödel, completeness theorem → p.23, Thm.38

entailment can be emulated by syntactic operations (derivability in a calculus)

Church-Turing, undecidability theorem → p.23, Thm.39

syntactic derivability is not decidable

Gödel-Malcev, finiteness or compactness theorem → p.23, Thm.40

a set S of closed formulas is unsatisfiable iff some finite subset of S is unsatisfiable

Results about Tarski Model Theory

In summary → p.23, Cor.41

- entailment, unsatisfiability, validity are semi-decidable, not decidable
- non-entailment, satisfiability, falsifiability are not semi-decidable

Key to Algorithmic Approaches

Herbrand interpretation $\mathcal{I}$ → p.27, Def.55–56

fixed $dom(\mathcal{I}) = HU =$ set of all ground terms

fixed $t^{\mathcal{I}} = t$ for each $t \in HU$

selectable $B \subseteq HB =$ set of all ground atoms. Uniquely determines $p^{\mathcal{I}}$

Observe HU and HB are computably enumerable (unless pathological)

Herbrand Interpretation $HI(B)$

Signature

function symbols: 0-ary a, b 1-ary f

relation symbols: 1-ary p, q 2-ary r

Herbrand Interpretation $HI(B)$

Signature

function symbols: 0-ary a, b 1-ary f

relation symbols: 1-ary p, q 2-ary r

$$
\begin{aligned}
B \;=\; \{\; & p(a),\quad p(b),\\
& q(a),\quad q(f(a)),\quad q(f(f(f(a)))),\quad \ldots\\
& \qquad q(f(f(b))),\quad q(f(f(f(f(b))))),\quad \ldots\\
& r(a,b),\quad r(a,f(a)),\quad r(a,f(f(a))),\quad r(a,f(f(f(a)))),\quad \ldots\\
& \qquad r(a,f(b)),\quad r(a,f(f(b))),\quad r(a,f(f(f(b)))),\quad \ldots\\
& \qquad r(b,f(a)),\quad r(b,f(f(a))),\quad r(b,f(f(f(a)))),\quad \ldots\\
& \qquad r(b,f(b)),\quad r(b,f(f(b))),\quad r(b,f(f(f(b)))),\quad \ldots\;\}
\end{aligned}
$$

Herbrand Interpretation $HI(B)$

Signature

function symbols: 0-ary a, b 1-ary f

relation symbols: 1-ary p, q 2-ary r

$$
\begin{aligned}
B \;=\; \{\; & p(a),\quad p(b),\\
& q(a),\quad q(f(a)),\quad q(f(f(f(a)))),\quad \ldots\\
& \qquad q(f(f(b))),\quad q(f(f(f(f(b))))),\quad \ldots\\
& r(a,b),\quad r(a,f(a)),\quad r(a,f(f(a))),\quad r(a,f(f(f(a)))),\quad \ldots\\
& \qquad r(a,f(b)),\quad r(a,f(f(b))),\quad r(a,f(f(f(b)))),\quad \ldots\\
& \qquad r(b,f(a)),\quad r(b,f(f(a))),\quad r(b,f(f(f(a)))),\quad \ldots\\
& \qquad r(b,f(b)),\quad r(b,f(f(b))),\quad r(b,f(f(f(b)))),\quad \ldots\;\}
\end{aligned}
$$

Model relationship $\models$

$$HI(B) \;\models\; q(a) \;\wedge\; r(a,b) \;\wedge\; \neg r(f(a),b) \;\wedge\; \forall x\Big(p(x) \Rightarrow r(x,f(x))\Big)$$

Tarski-Interpretation

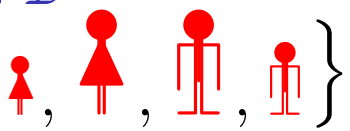
Signature

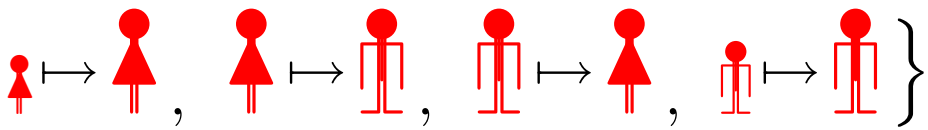
function symbols: 0-ary a, b 1-ary f

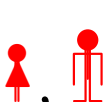
relation symbols: 1-ary p, q 2-ary r

Interpretation $\mathcal{I}$

→ p.20, Def.29–31

$dom(\mathcal{I}) = \{$  $\}$

$a^{\mathcal{I}} =$  $b^{\mathcal{I}} =$  $f^{\mathcal{I}} = \{$  $\}$

$p^{\mathcal{I}} = \{$  $\}$ $q^{\mathcal{I}} = \{$  $\}$

$r^{\mathcal{I}} = \{$  $\}$

Model relationship $\models$

→ p.21, Def.32

$$\mathcal{I} \models q(a) \wedge r(a,b) \wedge \neg r(f(a),b) \wedge \forall x\Big(p(x) \Rightarrow r(x,f(x))\Big)$$

Herbrand Theorem

S set of universal closed formulas
S_{ground} set of its ground instances

S is unsatisfiable → p.28, Cor.65
- iff S has no Herbrand model
- iff S_{ground} has no Herbrand model
- iff some finite subset of S_{ground} has no Herbrand model

Does not hold for non-universal formulas! → p.27, Ex.58–59

Dealing with non-universal Formulas

Skolemization → p.29, Cor.68

$\mathcal{L}$ signature, S set of closed formulas, computably enumerable

Constructs:

$\mathcal{L}_{sko}$ extension

S_{sko} set of universal closed formulas, computably enumerable

with: S is unsatisfiable iff S_{sko} is unsatisfiable

Assessment of Tarski Model Theory

For Logic and Mathematics in General

- domain of an interpretation may by any nonempty set

  first-order predicate logic can model statements about any arbitrary application domain
- excellent clarification of relationship syntax/semantics
- rich body of results
- quite successful for mathematics

Assessment of Tarski Model Theory

Inadequacy for Query Languages, 1

Unique name assumption

→ 4.1.8 (1) p.32

- different constants to be interpreted differently
- frequent requirement in applications
  a mechanism making it available by default would be useful
- not supported by Tarski model theory
  explicit formalisation is cumbersome

Assessment of Tarski Model Theory

Inadequacy for Query Languages, 2

Function symbols as term constructors

→ 4.1.8 (2) p.32

- grouping pieces of data that belong together
- makes sense in many applications
- terms as compound data structures

- not supported by Tarski model theory

Assessment of Tarski Model Theory

Inadequacy for Query Languages, 3

Closed world assumption

→ 4.1.8 (3) p.33

- nothing holds unless explicitly specified
- tacit understanding in many applications (transportation timetables)
- corresponds to an induction principle
  cannot be expressed in first-order predicate logic with Tarski model theory

Assessment of Tarski Model Theory

Inadequacy for Query Languages, 4

Disregard infinite models

→ 4.1.8 (4) p.33

- real-world query answering applications are often finite
- in this case infinite domains are irrelevent
- moreover, they cause "strange" phenomena
- restricting interpretations to finite ones is not possible
  finiteness cannot be expressed in first-order predicate logic with Tarski model theory

Assessment of Tarski Model Theory

Inadequacy for Query Languages, 5

Definability of transitive closure

→ 4.1.8 (5) p.33

- relevant in many query answering applications

  e.g., traffic application
  r represents direct connections between junctions
  t represents indirect connections

  t should be interpreted as the transitive closure of r

- cannot be expressed in first-order predicate logic with Tarski model theory

  $$\forall x \forall z \Big(t(x,z) \Leftrightarrow \big(r(x,z) \vee \exists y \left[t(x,y) \wedge t(y,z) \right] \big) \Big)$$

  does **not** do it!

Assessment of Tarski Model Theory

Inadequacy for Query Languages, 6

Application-specific restrictions

→ 4.1.8 (6) p.35

- e.g., to domains with a given cardinality, with odd cardinality, etc.
- cannot be expressed in first-order predicate logic with Tarski model theory

4.2 Herbrand Model Theory

Restricts interpretations to Herbrand interpretations → 4.2 p.36

- appealingly simple
- on universal formulas: coincides with Tarski model theory
- on non-universal formulas: Herbrand unsatisfiability and Herbrand entailment are not semi-decidable

4.3 Finite Model Theory

Restricts interpretations to interpretations with finite domain → 4.3 p.38

- amazingly different, many unexpected results, e.g.,
- finite non-entailment, finite satisfiability are semi-decidable, finite entailment, finite unsatisfiability are not

  (reversal of Results about Tarski Model Theory)
- 0-1 Laws → 4.3.2 p.41

Session 11:00 – 13:00

- 5 Declarative Semantics: Adapting Classical Model Theory to Rule Languages

5.1 Minimal Model Semantics of Definite Rules

A positive definite rule is

- a special universal formula → 3.1 p.9, Def.14
- a special inductive formula *(cf. infra)* → 5.1.3 p.47, Def.125

Interesting model-theoretic properties:

- If a set of universal formulas is satisfiable, then it is Herbrand satisfiable.
- If a set of inductive formulas is satisfiable, then the intersection of its models is also a model, provided that the models intersected are compatible.
- A set of definite inductive formulas is satisfiable.

Minimal Model Semantics of Positive Definite Rules

Thus, each set of positive definite rules has a unique minimal Herbrand model, the intersection of all its Herbrand models.

This minimal model can be taken as "the meaning" of the set of positive definite rules in a model-theoretic sense.

Intersection of Compatible Interpretations

Let $\{\mathcal{I}_i \mid i \in I\}$ be a set of interpretations.

$\{\mathcal{I}_i \mid i \in I\}$ is compatible iff → 5.1.1 p.43, Def.114

- $I \neq \emptyset$.
- $D = \bigcap\{dom(\mathcal{I}_i) \mid i \in I\} \neq \emptyset$.
- all interpretations of a function symbol coincide on D
- a variable is identically interpreted in all interpretations

If $\{\mathcal{I}_i \mid i \in I\}$ is compatible, then $\bigcap\{\mathcal{I}_i \mid i \in I\}$ → 5.1.1 p.44, Def.115
is the interpretation $\mathcal{I}$ with

- $dom(\mathcal{I}) = D = \bigcap\{dom(\mathcal{I}_i) \mid i \in I\}$.
- a function symbol is interpreted as the intersection of its interpretations
- a relation symbol is interpreted as the intersection of its interpretations
- a variable is interpreted like in all given interpretations

5.1.2 Universal Formulas and Theories

Recall

- polarity of a subformula
- universal formula
- Herbrand universe *HU*
- Herbrand base *HB*
- Herbrand interpretation

Inducers of Herbrand Models

Let $\{B_i \mid i \in I\}$ be a set of sets of ground atoms. → 5.1.2 p.45, Lem.116

- $\{HI(B_i) \mid i \in I\}$ is a compatible set of interpretations, i.e., its intersection is defined.
- $\bigcap\{HI(B_i) \mid i \in I\} = HI\big(\bigcap\{B_i \mid i \in I\}\big)$

Let S be a set of formulas.

- The set of inducers of the Herbrand models of S is $Mod_{HB}(S)$ $= \{B \subseteq HB \mid HI(B) \models S\}$. → 5.1.2 p.45, Def.117
- $Mod_{\cap}(S) = \begin{cases} \bigcap Mod_{HB}(S) & \text{if } Mod_{HB}(S) \neq \emptyset \\ HB & \text{if } Mod_{HB}(S) = \emptyset \end{cases}$ → 5.1.2 p.45, Ntn.118
- If S is universal, then $Mod_{\cap}(S) = \{A \in HB \mid S \models A\}$. → 5.1.2 p.45, Thm.119

Example

$\longrightarrow$ 5.1.2 p.45, Ex.120

Assume a signature consisting of a unary relation symbol p and constants a and b and no other symbols.

Let $S = \{p(a) \vee p(b)\}$.

Then $Mod_{HB}(S) = \{\ \{p(a)\},\ \{p(b)\},\ \{p(a), p(b)\}\ \}$.

But $HI(Mod_{\cap}(S)) = HI(\emptyset)$ is not a model of S.

Subinterpretation Property of Universal Formulas

Subinterpretation

→ 5.1.2 p.46, Def.121

$\mathcal{I}_1$ is a subinterpretation of $\mathcal{I}_2$ ($\mathcal{I}_1 \subseteq \mathcal{I}_2$) if

- $dom(\mathcal{I}_1) \subseteq dom(\mathcal{I}_2)$.
- the interpretations of a function symbol coincide on the common domain
- the interpretations of a relation symbol coincide on the common domain
- a variable is identically interpreted in the interpretations

If in addition $dom(\mathcal{I}_1) \neq dom(\mathcal{I}_2)$, then $\mathcal{I}_1$ is a proper subinterpretation of $\mathcal{I}_2$.

Property

→ 5.1.2 p.46, Thm.123

Let $\mathcal{I}_1$ and $\mathcal{I}_2$ be interpretations with $\mathcal{I}_1 \subseteq \mathcal{I}_2$.
For each universal closed formula φ, if $\mathcal{I}_2 \models \varphi$ then $\mathcal{I}_1 \models \varphi$.

5.1.3 Inductive Formulas and Theories

Positive and negative formulas → 5.1.3 p.46, Def.124

A formula φ is positive (or negative, respectively) iff every atom occurring in φ has positive (or negative, respectively) polarity in φ.

Inductive formula → 5.1.3 p.47, Def.125

- A generalised definite rule or definite inductive formula is a formula of the form $\forall^*((A_1 \wedge \ldots \wedge A_n) \leftarrow \varphi)$ where φ is positive and the A_i are atoms for $1 \leq i \leq n$.
- A generalised definite goal or integrity constraint is a formula of the form $\forall^* \varphi$ where φ is negative.
- An inductive formula is either a generalised definite rule or a generalised definite goal.
- A (definite) inductive theory is a theory axiomatised by a set of (definite) inductive formulas.

Order on Interpretations

→ 5.1.3 p.47, Def.126

$\mathcal{I}_1 \leq \mathcal{I}_2$ for interpretations $\mathcal{I}_1$ and $\mathcal{I}_2$ if

- $dom(\mathcal{I}_1) = dom(\mathcal{I}_2)$.
- the interpretations of a function symbol coincide on the common domain
- the "smaller" interpretation of a relation symbol is a restriction of the other
- a variable is identically interpreted in the interpretations

If in addition $p^{\mathcal{I}_1} \neq p^{\mathcal{I}_2}$ for at least one p, then $\mathcal{I}_1 < \mathcal{I}_2$.

Models of Inductive Formulas

Maximal Herbrand Model → 5.1.3 p.48, Thm.128

For each set S of generalised definite rules, $HI(HB) \models S$.

Intersection-Closedness → 5.1.3 p.48, Thm.129

Let S be a set of inductive formulas. If $\{\mathcal{I}_i \mid i \in I\}$ is a set of compatible models of S with the same domain D, then $\mathcal{I} = \bigcap\{\mathcal{I}_i \mid i \in I\}$ is also a model of S.

Intersection Model → 5.1.3 p.48, Cor.130

If S is a set of inductive formulas and $\{B_i \subseteq HB \mid i \in I\}$ is a nonempty set with $HI(B_i) \models S$ for each $i \in I$, then $HI\big(\bigcap\{B_i \mid i \in I\}\big) \models S$.

5.1.4 Minimal Models

Minimal Model

→ 5.1.4 p.48, Lem.131

A minimal model of a set of formulas is a $\leq$-minimal member $\mathcal{I}$ of the set of all its models with domain $dom(\mathcal{I})$.

Minimal Herbrand Model

→ 5.1.4 p.49, Lem.132

Let S be a set of formulas.

- An Herbrand model of S is minimal iff it is induced by a $\subseteq$-minimal member of $Mod_{HB}(S)$.
- If $HI(Mod_{\cap}(S))$ is a model of S, it is a minimal Herbrand model of S and it is the only minimal Herbrand model of S.

Minimal Herbrand Model of Inductive Formulas

→ 5.1.4 p.49, Thm.133

Let S be a set of inductive formulas. If either each member of S is definite, or S is satisfiable and each member of S is universal, then $HI(Mod_{\cap}(S))$ is the unique minimal Herbrand model of S.

Minimal Herbrand Model

Definite Program → 5.1.4 p.49, Cor.134

Each set S of positive definite rules (i.e., each definite program) has a unique minimal Herbrand model.
This model is the intersection of all Herbrand models of S.
It satisfies precisely those ground atoms that are logical consequences of S.

Generalisation of inductive formulas for which 'minimal models' remains useful:

Generalised Rule → 5.1.4 p.49, Def.135

A generalised rule is a formula of the form $\forall^*(\psi \leftarrow \varphi)$ where φ is positive and quantifier-free.

Implicant of a Positive Quantifier-Free Formula

Pre-Implicant and Implicant

→ 5.1.4 p.49, Def.136

Let ψ be a positive quantifier-free formula. The set $\mathit{primps}(\psi)$ of pre-implicants of ψ is defined as follows:

- $\mathit{primps}(\psi) = \{\{\psi\}\}$ if ψ is an atom or $\top$ or $\bot$.
- $\mathit{primps}(\neg\psi_1) = \mathit{primps}(\psi_1)$.
- $\mathit{primps}(\psi_1 \wedge \psi_2) = \{\ C_1 \cup C_2 \mid C_1 \in \mathit{primps}(\psi_1),\ C_2 \in \mathit{primps}(\psi_2)\ \}$.
- $\mathit{primps}(\psi_1 \vee \psi_2) = \mathit{primps}(\psi_1 \Rightarrow \psi_2) = \mathit{primps}(\psi_1) \cup \mathit{primps}(\psi_2)$.

The set of implicants of ψ is obtained from $\mathit{primps}(\psi)$ by removing all sets containing $\bot$ and by removing $\top$ from the remaining sets.

Implicants and Entailement

→ 5.1.4 p.50, Lem.137

1. If C is an implicant of ψ, then $C \models \psi$.
2. For any interpretation $\mathcal{I}$, if $\mathcal{I} \models \psi$ then there exists an implicant C of ψ with $\mathcal{I} \models C$.

Supported Atom

Supported Atom

→ 5.1.4 p.50, Def.138

Let $\mathcal{I}$ be an interpretation, V a variable assignment in $dom(\mathcal{I})$ and $A = p(t_1, \ldots, t_n)$ an atom, $n \geq 0$.

- an atom B supports A in $I[V]$ iff
  $I[V] \models B$ and $B = p(s_1, \ldots, s_n)$ and $s_i^{I[V]} = t_i^{I[V]}$ for $1 \leq i \leq n$.
- a set C of atoms supports A in $I[V]$ iff
  $I[V] \models C$ and there is an atom in C that supports A in $I[V]$.
- a generalised rule $\forall^*(\psi \leftarrow \varphi)$ supports A in I iff for each variable assignment V with $I[V] \models \varphi$ there is an implicant C of ψ that supports A in $I[V]$.

Minimal Models and Supported Ground Atom

Minimal Models Satisfy Only Supported Ground Atom

→ 5.1.4 p.50, Thm.139

Let S be a set of generalised rules. Let $\mathcal{I}$ be an interpretation with domain D. If $\mathcal{I}$ is a minimal model of S, then: For each ground atom A with $\mathcal{I} \models A$ there is a generalised rule in S that supports A in $\mathcal{I}$.

Example

→ 5.1.4 p.51, Ex.140

Consider a signature containing a unary relation symbol p and constants a and b. Let $S = \{ (p(b) \leftarrow \top) \}$.
The interpretation $\mathcal{I}$ with $dom(\mathcal{I}) = \{1\}$ and $a^{\mathcal{I}} = b^{\mathcal{I}} = 1$ and $p^{\mathcal{I}} = \{(1)\}$ is a minimal model of S.
(Note that the only smaller interpretation interprets p with the empty relation and does not satisfy the rule.)
Moreover, $\mathcal{I} \models p(a)$. By the theorem, $p(a)$ is supported in $\mathcal{I}$ by $p(b)$, which can be confirmed by applying the definition.

Unique Name Property

Unique Name Property → 5.1.4 p.51, Def.141

An interpretation $\mathcal{I}$ has the unique name property, if for each term s, ground term t, and variable assignment V in $dom(\mathcal{I})$ with $s^{I[V]} = t^{I[V]}$ there exists a substitution σ with $s\sigma = t$.

Herbrand interpretations have the unique name property.

Non-Minimal Models Supporting Ground Atoms

The converse of Thm. 139 is refuted by counter-examples with indefinite rules such as $\{ (p(a) \vee p(b) \leftarrow \top) \}$ because the definition of supported cannot distinguish between implicants of rule consequent. Both atoms are supported in the Herbrand model $HI(\{p(a), p(b)\})$ of this set, although the model is not minimal.

The converse of Theorem 139 is also refuted by (simple) counter-examples with definite rules:

Example → 5.1.4 p.52, Ex.142

Consider $S = \{ (p \leftarrow p) \}$ and its Herbrand model $HI(\{p\})$. The only ground atom satisfied by $HI(\{p\})$ is p, which is supported in $HI(\{p\})$ by the rule. But $HI(\{p\})$ is not minimal because $HI(\emptyset)$ is also a model of S.

5.2 Fixpoint Semantics of Positive Definite Rules

Operator → 5.2.1 p.52, Def.143

Let X be a set. An operator Γ on a set X is a mapping: $\mathcal{P}(X) \to \mathcal{P}(X)$.

Monotonic Operator → 5.2.1 p.52, Def.144

Let X be a set. An operator Γ on X is monotonic, iff for all subset $M \subseteq M' \subseteq X$ holds: $\Gamma(M) \subseteq \Gamma(M')$.

Continuous operator → 5.2.1 p.52, Def.145

Let X be a nonempty set.

A set $Y \subseteq \mathcal{P}(X)$ of subsets of X is directed, if every finite subset of Y has an upper bound in Y, i.e., for each finite $Y_{fin} \subseteq Y$, there is a set $M \in Y$ such that $\bigcup Y_{fin} \subseteq M$.

An operator Γ on X is continuous, iff for each directed set $Y \subseteq \mathcal{P}(X)$ of subsets of X holds: $\Gamma(\bigcup Y) = \bigcup\{\Gamma(M) \mid M \in Y\}$.

A continuous operator on a nonempty set is monotonic.

→ 5.2.1 p.52, Lem.146

5.2.2 Fixpoints of Monotonic and Continuous Operators

Fixpoint → 5.2.2 p.52, Def.147

Let Γ be an operator on a set X. A subset $M \subseteq X$ is a fixpoint of Γ iff $\Gamma(M) = M$.

Knaster-Tarski Theorem → 5.2.2 p.52, Thm.148

Let Γ be a monotonic operator on a nonempty set X. Then Γ has a least fixpoint $lfp(\Gamma)$ and a greatest fixpoint $gfp(\Gamma)$ with

$$lfp(\Gamma) = \bigcap\{M \subseteq X \mid \Gamma(M) = M\} = \bigcap\{M \subseteq X \mid \Gamma(M) \subseteq M\}.$$
$$gfp(\Gamma) = \bigcup\{M \subseteq X \mid \Gamma(M) = M\} = \bigcup\{M \subseteq X \mid \Gamma(M) \subseteq M\}.$$

Ordinal Powers of a Monotonic Operator

→ 5.2.1 p.53, Def.149

Let Γ be a monotonic operator on a nonempty set X.

- The upward power of Γ is defined as:

$$\begin{array}{lcll} \Gamma \uparrow 0 & = & \emptyset & \text{(base case)} \\ \Gamma \uparrow \alpha + 1 & = & \Gamma(\Gamma \uparrow \alpha) & \text{(successor case)} \\ \Gamma \uparrow \lambda & = & \bigcup\{\Gamma \uparrow \beta \mid \beta < \lambda\} & \text{(limit case)} \end{array}$$

- The downward power of Γ is defined as:

$$\begin{array}{lcll} \Gamma \downarrow 0 & = & X & \text{(base case)} \\ \Gamma \downarrow \alpha + 1 & = & \Gamma(\Gamma \downarrow \alpha) & \text{(successor case)} \\ \Gamma \downarrow \lambda & = & \bigcap\{\Gamma \downarrow \beta \mid \beta < \lambda\} & \text{(limit case)} \end{array}$$

Let Γ be an operator on a nonempty set X.

Theorem

→ 5.2.1 p.53, Thm.151

If Γ is monotonic, then there exists α such that $\Gamma \uparrow \alpha = \mathit{lfp}(\Gamma)$.

Kleene Theorem

→ 5.2.1 p.53, Thm.152

If Γ is continuous, then $\mathit{lfp}(\Gamma) = \Gamma \uparrow \omega$.

5.2.3 Immediate Consequence Operator

Immediate Consequence Operator → 5.2.3 p.54, Def.153

Let S be a set of universal generalised definite rules. Let $B \subseteq HB$ be a set of ground atoms. The immediate consequence operator $\mathbf{T}_S : \mathcal{P}(HB) \to \mathcal{P}(HB)$ is defined by:
$\mathbf{T}_S(B) = \{A \in HB \mid$ there is a ground instance $A_1 \wedge \ldots \wedge A_n \leftarrow \varphi$ of a member of S with $HI(B) \models \varphi$ and $A = A_i$ for some i with $1 \leq i \leq n\}$

Theorem → 5.2.3 p.54, Thm.156

The immediate consequence operator of a set of positive definite rules is continuous and monotonic.

Least Fixpoint of a Definite Program → 5.2.4 p.56, Ntn.158

For a set S of universal generalised definite rules, the least fixpoint of S is $lfp(S) = lfp(\mathbf{T}_S)$.

5.3 Declarative Semantics of Rules with Negation

If a database of students does not list "Mary", then one may conclude that "Mary" is not a student. The principle underlying this is called closed world assumption (CWA).

Two approaches to coping with this form of negation:

- axiomatization within first-oder predicate logic
- deduction methods not requiring specific axioms conveying the CWA

The second approach is desirable but it poses the problem of the declarative semantics, or model theory.

Minimal Models don't all convey the CWA

Example → 5.3 p.57, Ex.159

$S_1 = \{ (q \leftarrow r \wedge \neg p),\ (r \leftarrow s \wedge \neg t),\ (s \leftarrow \top) \}$
Minimal Herbrand models: $HI(\{s, r, q\})$, $HI(\{s, r, p\})$, and $HI(\{s, t\})$.

Example → 5.3 p.57, Ex.160

$S_2 = \{ (p \leftarrow \neg q),\ (q \leftarrow \neg p) \}$
Minimal Herbrand models: $HI(\{p\})$, $HI(\{q\})$.

Example → 5.3 p.57, Ex.161

$S_3 = \{ (p \leftarrow \neg p) \}$ Minimal Herbrand model: $HI(\{p\})$.

Example → 5.3 p.58, Ex.162

$S_4 = \{ (p \leftarrow \neg p),\ (p \leftarrow \top) \}$
Minimal Herbrand model: $HI(\{p\})$.

Justification vs. Consistency

Justification postulate

dependable justifications for derived truths.

Consistency postulate

every syntactically correct set of normal clauses is consistent (hence has a model).

$Th_{\text{canonical}}$ is not Monotonic

Example → 5.3 p.58, Ex.163

$S_5 = \{ (q \leftarrow \neg p) \}$
Minimal Herbrand models: $HI(\{p\})$ and $HI(\{q\})$.
Only the latter conveys the intuitive meaning under the CWA and should be retained as (the only) canonical model.
Therefore, $q \in Th_{canonical}(S_5)$.

$S_5' = S_5 \cup \{ (p \leftarrow \top) \}$
Minimal Herbrand model: $HI(\{p\})$, which also conveys the CWA.
Therefore, $q \notin Th_{canonical}(S_5')$.

$S_5 \subseteq S_5'$ but $Th_{canonical}(S_5) \not\subseteq Th_{canonical}(S_5')$.

Justification vs. Consistency Cont'd

Non-monotonicity is independent of the choice Justification vs. Consistency.

Any semantics not complying with Consistency is non-monotonic in an even stronger sense: Consistency (defined as usual as the existence of models) is not inherited by subsets.

5.3.1 Stratifiable Rule Sets

Stratification

→ 5.3.1 p.59, Def.164

Let S be a set of normal clauses. A stratification of S is a partition $S_0, \ldots, S_k$ of S such that

- For each relation symbol p there is a stratum S_i, such that all clauses of S containing p in their consequent are members of S_i. (p is defined in stratum S_i.)
- For each stratum S_j and for each positive literal A in the antecedents of members of S_j, the relation symbol of A is defined in a stratum S_i with $i \leq j$.
- For each stratum S_j and for each negative literal $\neg A$ in the antecedents of members of S_j, the relation symbol of A is defined in a stratum S_i with $i < j$.

A set of normal clauses is called stratifiable if there exists a stratification of it.

Stratifiable Rule Sets - Examples

$S = \{ (r \leftarrow \top),\ (q \leftarrow r),\ (p \leftarrow q \wedge \neg r) \}$ is stratifiable.
$S = \{ (p \leftarrow \neg p) \}$ is not stratifiable. More generally, any set of normal clauses with a cycle of recursion through negation is not stratifiable.

Stratifiable Rule Sets - Model

By definition the stratum S_0 always consists of definite clauses. Hence the truth values of all atoms of stratum S_0 can be determined without negation being involved.

After that the clauses of stratum S_1 refer only to such negative literals whose truth values have already been determined.

And so on.

5.3.2 Stable Model Semantics

Let S be a (possibly infinite) set of ground normal clauses, i.e., of formulas of the form $A \leftarrow L_1 \wedge \ldots \wedge L_n$ where $n \geq 0$ and A is a ground atom and the L_i for $1 \leq i \leq n$ are ground literals.

Gelfond-Lifschitz Transformation → 5.3.2 p.59, Def.165

Let $B \subseteq HB$. The Gelfond-Lifschitz transform $GL_B(S)$ of S with respect to B is obtained from S as follows:

1. remove each clause whose antecedent contains a literal $\neg A$ with $A \in B$.
2. remove from the antecedents of the remaining clauses all negative literals.

Stable Model → 5.3.2 p.60, Def.166

An Herbrand interpretation $HI(B)$ is a stable model of S iff it is the unique minimal Herbrand model of $GL_B(S)$.

A stable model of a set S of normal clauses is a stable model of the (possibly infinite) set of ground instances of S.

Stable Model Semantics - Properties

Let S be a set of ground normal clauses.

Lemma → 5.3.2 p.60, Lem.167

Let $HI(B)$ be an Herbrand interpretation.
$HI(B) \models S$ iff $HI(B) \models GL_B(S)$.

Theorem → 5.3.2 p.60, Thm.168

Each stable model of S is a minimal Herbrand model of S.

Stable Model Semantics - Example

Example → 5.3.2 p.61, Ex.169

$S_1 = \{ (q \leftarrow r \wedge \neg p),\ (r \leftarrow s \wedge \neg t),\ (s \leftarrow \top) \}$ has one stable model: $HI(\{s, r, q\})$

$S_2 = \{ (p \leftarrow \neg q),\ (q \leftarrow \neg p) \}$ has two stable models: $HI(\{p\})$ and $HI(\{q\})$

$S_3 = \{ (p \leftarrow \neg p) \}$ has no stable model.

$S_4 = \{ (p \leftarrow \neg p),\ (p \leftarrow \top) \}$ has one stable model: $HI(\{p\})$

Stable Model Semantics - Evaluation

- The stable model semantics coincides with the intuitive understanding based on the "Justification Postulate".
- A set may have several stable models or exactly one or none. Each stratifiable set has exactly one stable model.

5.3.3 Well-Founded Semantics

Notation

→ 5.3.3 p.61, Ntn.170

For a set I of ground literals:
$\overline{I} = \{ \overline{L} \mid L \in I \}$ and $pos(I) = I \cap HB$ and $neg(I) = \overline{I} \cap HB$.
$(I = pos(I) \cup \overline{neg(I)}.)$

(In)Consistent Sets of Literals

→ 5.3.3 p.61, Def.171

A set I of ground literals is consistent iff $pos(I) \cap neg(I) = \emptyset$.
Otherwise, I is inconsistent.
Two sets I_1 and I_2 of ground literals are (in)consistent iff $I_1 \cup I_2$ is.
A literal L and a set I of ground literals are (in)consistent iff $\{L\} \cup I$ is.

Partial Interpretation

→ 5.3.3 p.61, Def.172

A partial interpretation is a consistent set of ground literals.
A partial interpretation I is total iff $pos(I) \cup neg(I) = HB$ (i.e. for each ground atom A either $A \in I$ or $\neg A \in I$).
For a total interpretation I, the Herbrand interpretation induced by I is defined as $HI(I) = HI(pos(I))$.

Well-Founded Semantics - Models

Let I be a partial interpretation.

Model Relationship

→ 5.3.3 p.61, Def.173

$\top$ is satisfied in I and $\bot$ is falsified in I.

A ground literal L is

satisfied or true in I	iff $L \in I$.
falsified or false in I	iff $\overline{L} \in I$.
undefined in I	iff $L \notin I$ and $\overline{L} \notin I$.

A conjunction $L_1 \wedge \ldots \wedge L_n$ of ground literals, $n \geq 0$, is

satisfied or true in I	iff each L_i for $1 \leq i \leq n$ is satisfied in I.
falsified or false in I	iff at least one L_i for $1 \leq i \leq n$ is falsified in I.
undefined in I	iff each L_i for $1 \leq i \leq n$ is satisfied or undefined in I and at least one of them is undefined in I.

Well-Founded Semantics - Models Cont'd

Let I be a partial interpretation.

Model Relationship

→ 5.3.3 p.61, Def.173

A ground normal clause $A \leftarrow \varphi$ is

satisfied or true in I	iff A is satisfied in I or φ is falsified in I.
falsified or false in I	iff A is falsified in I and φ is satisfied in I.
weakly falsified in I	iff A is falsified in I and φ is satisfied or undefined in I.

A normal clause is

satisfied or true in I	iff each of its ground instances is.
falsified or false in I	iff at least one of its ground instances is.
weakly falsified in I	iff at least one of its ground instances is.

A set of normal clauses is

satisfied or true in I	iff each of its members is.
falsified or false in I	iff at least one of its members is.
weakly falsified in I	iff at least one of its members is.

Well-Founded Semantics - Models Cont'd

Total and Partial Models

→ 5.3.3 p.62, Def.174

Let S be a set of normal clauses.

A total interpretation I is a total model of S, iff S is satisfied in I.

A partial interpretation I is a partial model of S, iff there exists a total model I' of S with $I \subseteq I'$.

Well-Founded Semantics - Unfoundedness

Let S be a set of normal clauses, I a partial interpretation, and $U \subseteq HB$ a set of ground atoms.

Unfounded Sets of Atoms

→ 5.3.3 p.63, Def.176

U is an unfounded set with respect to S and I, if for each $A \in U$ and for each ground instance $A \leftarrow L_1 \wedge \ldots \wedge L_n$, $n \geq 1$, of a member of S at least one of the following holds:

1. $L_i \in \overline{I}$ for some positive or negative L_i with $1 \leq i \leq n$. (L_i is falsified in I)
2. $L_i \in U$ for some positive L_i with $1 \leq i \leq n$. (L_i is unfounded)

A literal fulfilling one of these conditions is a witness of unusability for the ground instance of a clause.

U is a maximal unfounded set with respect to S and I, iff U is an unfounded set with respect to S and I and no proper superset of U is.

Well-Founded Semantics - Operators

Let $\mathcal{PI} = \{\, I \subseteq HB \cup \overline{HB} \mid I \text{ is consistent} \,\}$, and note that $\mathcal{P}(HB) \subseteq \mathcal{PI}$.
Let S be a set of normal clauses.

Operators

→ 5.3.3 p.64, Def.181

$$
\begin{array}{lrcl}
\mathbf{T}_S: & \mathcal{PI} & \to & \mathcal{P}(HB) \\
 & I & \mapsto & \{\, A \in HB \mid \text{there is a ground instance } (A \leftarrow \varphi) \\
 & & & \quad \text{of a member of } S \text{ such that} \\
 & & & \quad \varphi \text{ is satisfied in } I \,\} \\
\mathbf{U}_S: & \mathcal{PI} & \to & \mathcal{P}(HB) \\
 & I & \mapsto & \text{the maximal subset of } HB \text{ that is} \\
 & & & \quad \text{unfounded with respect to } S \text{ and } I \\
\mathbf{W}_S: & \mathcal{PI} & \to & \mathcal{PI} \\
 & I & \mapsto & \mathbf{T}_S(I) \cup \overline{\mathbf{U}_S(I)}
\end{array}
$$

Lemma

→ 5.3.3 p.65, Lem.183

$\mathbf{T}_S$, $\mathbf{U}_S$, and $\mathbf{W}_S$ are monotonic. (but not in general continuous!)

Well-Founded Semantics

→ 5.3.3 p.64, Ex.182

Assume a signature with $HB = \{p, q, r, s, t\}$, and let $I_0 = \emptyset$ and $S = \{\, (q \leftarrow r \wedge \neg p),\ (r \leftarrow s \wedge \neg t),\ (s \leftarrow \top)\,\}$.

$$\begin{aligned} \mathbf{T}_S(I_0) &= \{s\} \\ \mathbf{U}_S(I_0) &= \{p, t\} \\ \mathbf{W}_S(I_0) &= \{s, \neg p, \neg t\} = I_1 \end{aligned}$$

$$\begin{aligned} \mathbf{T}_S(I_1) &= \{s, r\} \\ \mathbf{U}_S(I_1) &= \{p, t\} \\ \mathbf{W}_S(I_1) &= \{s, r, \neg p, \neg t\} = I_2 \end{aligned}$$

$$\begin{aligned} \mathbf{T}_S(I_2) &= \{s, r, q\} \\ \mathbf{U}_S(I_2) &= \{p, t\} \\ \mathbf{W}_S(I_2) &= \{s, r, q, \neg p, \neg t\} \end{aligned}$$

Well-Founded Semantics - Properties

Well-Founded Model

→ 5.3.3 p.65, Def.185

Let S be a set of normal clauses. The well-founded model of S is its partial model $lfp(\mathbf{W}_S)$.

Examples

→ 5.3.3 p.65, Ex.186

$S_1 = \{ (q \leftarrow r \wedge \neg p),\ (r \leftarrow s \wedge \neg t),\ (s \leftarrow \top) \}$ has the well-founded model $\{s, r, q, \neg p, \neg t\}$. It is total.

$S_2 = \{ (p \leftarrow \neg q),\ (q \leftarrow \neg p) \}$ has the well-founded model $\emptyset$. It is partial.

$S_3 = \{ (p \leftarrow \neg p) \}$ has the well-founded model $\emptyset$. It is partial.

$S_4 = \{ (p \leftarrow \neg p),\ (p \leftarrow \top) \}$ has the well-founded model $\{p\}$. It is total.

Well-Founded Semantics

→ 5.3.3 p.66, Ex.187

Assume a signature containing no other symbols than those occurring in the following set of normal clauses.

$$S = \{\, p(a) \leftarrow \top,\quad p(f(x)) \leftarrow p(x),\quad q(y) \leftarrow p(y),\quad s \leftarrow p(z) \wedge \neg q(z),\quad r \leftarrow \neg s \,\}$$

$$\mathit{lfp}(\mathbf{W}_S) = \mathbf{W}_S \uparrow \omega + 2 =$$
$$\{\, p(a), \ldots, p(f^n(a)), \ldots \} \cup \{\, q(a), \ldots, q(f^n(a)), \ldots \} \cup \{\, \neg s,\ r \,\}$$

S is the (standard) translation into normal clauses of the following set of generalised rules:

$$\{\, p(a) \leftarrow \top,\quad p(f(x)) \leftarrow p(x),\quad q(y) \leftarrow p(y),\quad r \leftarrow \forall z \big(p(z) \Rightarrow q(z)\big) \,\}$$

Well-Founded Semantics - Evaluation

- The well-founded semantics coincides with an intuitive understanding based on the "Justification Postulate".
- A set always has exactly one model but some ground atoms might be "undefined" in this model. Thus, the well-founded semantics coincides with the "Consistency Postulate".
- The well-founded model might not be computable (in those not unfrequent cases where the fixpoint is reached after more than ω steps).

5.3.4 Stable and Well-Founded Semantics Compared

- If a rule set is stratifiable, then it has a unique minimal model, which is its only stable model and is also its well-founded model and total.
- If a rule set S has a total well-founded model, then this model is also the single stable model of S .
- If a rule set S has a single stable model, then this model is also the well-founded model of S and it is total.
- If a rule set S has a partial well-founded model I that is not total, then S has either no stable model or more than one stable model. In the latter case, a ground atom is true in all stable models of S if it is true in I.
- Stable model entailment does not imply well-founded entailment:

  → missing in text!

  $S = \{p \leftarrow \neg q, \quad q \leftarrow \neg p, \quad r \leftarrow p, \quad r \leftarrow q\}$

  r is true in all stable models but it is undefined in the well-founded model.

5.3.5 Inflationary Semantics

Attention restricted to datalog$^\neg$ programs, i.e. finite sets of normal clauses.

- The Herbrand universe **dom** is finite
- The Herbrand base HB is finite

Normal clauses are rule r of the form $A \leftarrow L_1, \ldots, L_m$ where $m \geq 0$ and A is an atom. They are assumed to be range restricted.

$\rightarrow$ 3.2.2 p.11, Def.23

Notation

Head (consequent) of a rule r: $H(r)$

Body (antecedent) of r: $B(r)$

$$B^+(r) = \{R(\vec{x}) \mid \exists i\, L_i = R(\vec{x})\} \qquad B^-(r) = \{R(\vec{x}) \mid \exists i\, L_i = \neg R(\vec{x})\}$$

Inflationary Semantics - Operator

Let P be a datalog$^\neg$ program and $\mathbf{I}$ an instance or interpretation ($\mathbf{I} \subseteq HB$)

Immediate Consequence Operator

→ 5.3.5 p.68, Def.188

$R(\vec{t})$ is an *immediate consequence* for $\mathbf{I}$ and P ($R(\vec{t}) \in \mathbf{T}_P(\mathbf{I})$), if either $R\ R(\vec{t}) \in \mathbf{I}$, or there exists some ground instance r of a rule in P such that

- $H(r) = R(\vec{t})$,
- $B^+(r) \subseteq \mathbf{I}$, and
- $B^-(r) \cap \mathbf{I} = \emptyset$.

Inflationary Operator and Semantics

→ 5.3.5 p.68, Def.189

$\widetilde{\mathbf{T}}_P(\mathbf{I}) = \mathbf{I} \cup \mathbf{T}_P(\mathbf{I})$

The inflationary semantics of P w.r.t. $\mathbf{I}$ ($P_{inf}(\mathbf{I})$) is the limit of $\{\widetilde{\mathbf{T}}_P^i(\mathbf{I})\}_{i \geq 0}$, where $\widetilde{\mathbf{T}}_P^0(\mathbf{I}) = \mathbf{I}$ and $\widetilde{\mathbf{T}}_P^{i+1}(\mathbf{I}) = \widetilde{\mathbf{T}}_P(\widetilde{\mathbf{T}}_P^i(\mathbf{I}))$.

Inflationary Semantics

- By definition of $\widetilde{\mathbf{T}}_P$: $\widetilde{\mathbf{T}}_P^0(\mathbf{I}) \subseteq \widetilde{\mathbf{T}}_P^1(\mathbf{I}) \subseteq \widetilde{\mathbf{T}}_P^2(\mathbf{I}) \subseteq \ldots$
- Each set in this sequence is a subset of the finite set HB. Therefore, the sequence reaches a fixpoint $\widetilde{\mathbf{T}}_P(\mathbf{I})$ after a finite number of steps.
- $HI(P_{inf}(\mathbf{I}))$ is a model of P containing $\mathbf{I}$ but not necessarily a minimal model containing $\mathbf{I}$.

Example → 5.3.5 p.68, Ex.190

$P = \{ (p \leftarrow s \wedge \neg q),\ (q \leftarrow s \wedge \neg p) \}$
$\mathbf{I} = \{s\}$.

$P_{inf}(\mathbf{I}) = \{s, p, q\}$
Although $HI(P_{inf}(\mathbf{I}))$ is a model of P, it is not minimal.

$\widetilde{\mathbf{T}}_P$ is not monotonic: $\widetilde{\mathbf{T}}_P(\{s\}) = \{s, p, q\}$ and $\widetilde{\mathbf{T}}_P(\{s, p\}) = \{s, p\}$

Inflationary Semantics - Examples

$S_1 = \{ (q \leftarrow r \wedge \neg p),\ (r \leftarrow s \wedge \neg t),\ (s \leftarrow \top) \}$ → 5.3.5 p.69, Ex.191

$\widetilde{\mathbf{T}}^1_{S_1}(\emptyset) = \{s\},\ \widetilde{\mathbf{T}}^2_{S_1}(\emptyset) = \{s, r\},\ \widetilde{\mathbf{T}}^3_{S_1}(\emptyset) = \{s, r, q\} = \widetilde{\mathbf{T}}^4_{S_1}(\emptyset).$

$S_2 = \{ (p \leftarrow \neg q),\ (q \leftarrow \neg p) \}$ → 5.3.5 p.69, Ex.192

$\widetilde{\mathbf{T}}^1_{S_2}(\emptyset) = \{p, q\} = \widetilde{\mathbf{T}}^2_{S_2}(\emptyset)$

$S_3 = \{ (p \leftarrow \neg p) \}$ → 5.3.5 p.69, Ex.193

$\widetilde{\mathbf{T}}^1_{S_3}(\emptyset) = \{p\} = \widetilde{\mathbf{T}}^2_{S_3}(\emptyset)$

$S_4 = \{ (p \leftarrow \neg p),\ (p \leftarrow \top) \}$ → 5.3.5 p.69, Ex.194

$\widetilde{\mathbf{T}}^1_{S_4}(\emptyset) = \{p\} = \widetilde{\mathbf{T}}^2_{S_4}(\emptyset)$

$S_5 = \{ (r \leftarrow \neg q),\ (q \leftarrow \neg p) \}$ → 5.3.5 p.69, Ex.195

S_5 is stratifiable, its minimal models are $HI(\{q\})$ and $HI(\{p, r\})$.

$\widetilde{\mathbf{T}}^1_{S_5}(\emptyset) = \{q, r\}$

Inflationary Semantics - Evaluation

The inflationary semantics gives up a fundamental principle, that models are preserved when adding logical consequences.

$S_5 = \{ (r \leftarrow \neg q),\ (q \leftarrow \neg p) \}$

S_5 is stratifiable, its minimal models are $HI(\{q\})$ and $HI(\{p, r\})$.

q is true in the only inflationary model $HI(\{q, r\})$ of S_5 but $HI(\{q, r\})$ is not an inflationary model of $S_5 \cup \{q\}$.

Session 14:30 – 16:00

- 6 Operational Semantics: Positive Rule Sets
- 7 Operational Semantics: Rule Sets with Non-monotonic Negation

6 Operational Semantics: Positive

6.1 Semi-naive Evaluation of Datalog Programs

```
feeds_milk(betty).
lays_eggs(betty).
has_spines(betty).
monotreme(X)←
      lays_eggs(X), feeds_milk(X).
echidna(X)←
      monotreme(X), has_spines(X).
```

6.1 Semi-naive Evaluation of Datalog Programs

```
feeds_milk(betty).
lays_eggs(betty).
has_spines(betty).
monotreme(X)←
      lays_eggs(X), feeds_milk(X).
echidna(X)←
      monotreme(X), has_spines(X).
```

Terminology:

- Extensional Predicate Symbols $ext(P)$
- Intensional Predicate Symbols $int(P)$
- Semantics of a LP: Mapping from extensions over $ext(P)$ to extensions over $int(P)$

6.1 Semi-naive Evaluation of Datalog Programs

```
feeds_milk(betty).
lays_eggs(betty).
has_spines(betty).
monotreme(X)←
      lays_eggs(X), feeds_milk(X).
echidna(X)←
      monotreme(X), has_spines(X).
```

- Schema of P:
  $\{$ feeds_milk, lays_eggs, has_spines, monotreme, echidna $\}$
- Fixpoint calculation:
  $\{betty\}_{feeds}, \{betty\}_{lays}, \{betty\}_{spines}, \{\}_{monotreme}, \{\}_{echidna}$
  $\{betty\}_{feeds}, \{betty\}_{lays}, \{betty\}_{spines}, \{betty\}_{monotreme}, \{\}_{echidna}$
  $\{betty\}_{feeds}, \{betty\}_{lays}, \{betty\}_{spines}, \{betty\}_{monotreme}, \{betty\}_{echidna}$

Further Optimizations for Evaluating Positive Rule Sets

- Goal directedness
- Storing partially instantiated rules
- Sharing of instantiated premises among similar rules

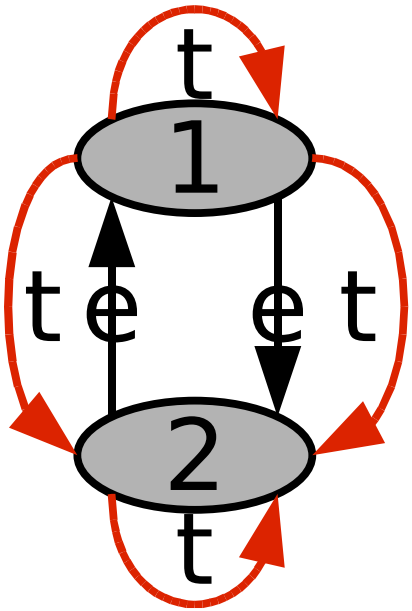
6.4 SLD-Resolution: Principles

- goal driven evaluation of LPs
- instead of showing that $P \models q$, show that $P \cup \neg q$ is unsatisfiable
- resolution: a mechanical method for proving statements in FOL
- uses unification
- elimination of a literal that occurs positive in one clause and negative in another
- recall that $Q \leftarrow P$ is equivalent to $Q \vee \neg P$
- SLD resolution: Linear resolution with a Selection function for Definite clauses
- resolution with backtracking as control mechanism in Prolog

6.4 SLD-Resolution

```
1: t(X,Y)←e(X,Y).
2: t(X,Z)←t(X,Y),e(Y,Z).
3: e(1,2).
4: e(2,1).
5: ←t(1,A).
```

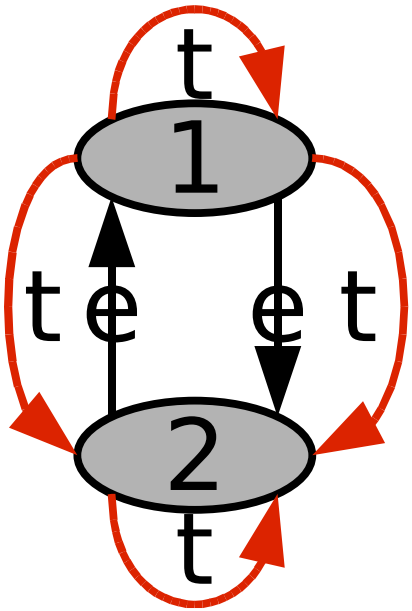
```
:- t(1,A)
```



6.4 SLD-Resolution

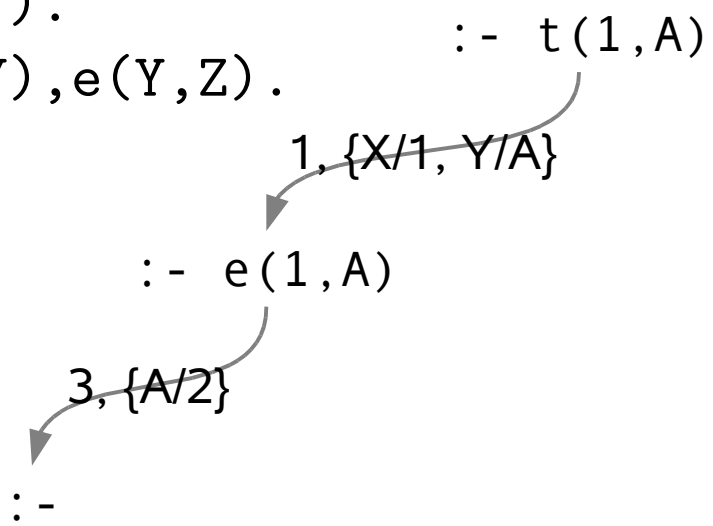
```
1: t(X,Y)←e(X,Y).
2: t(X,Z)←t(X,Y),e(Y,Z).
3: e(1,2).
4: e(2,1).
5: ←t(1,A).
```

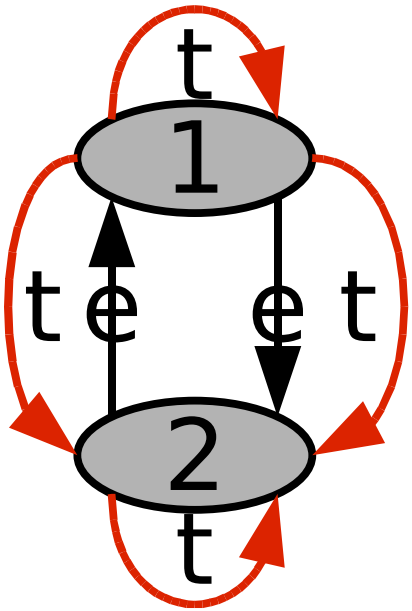
```
:- t(1,A)
        1,{X/1, Y/A}
:- e(1,A)
```



6.4 SLD-Resolution

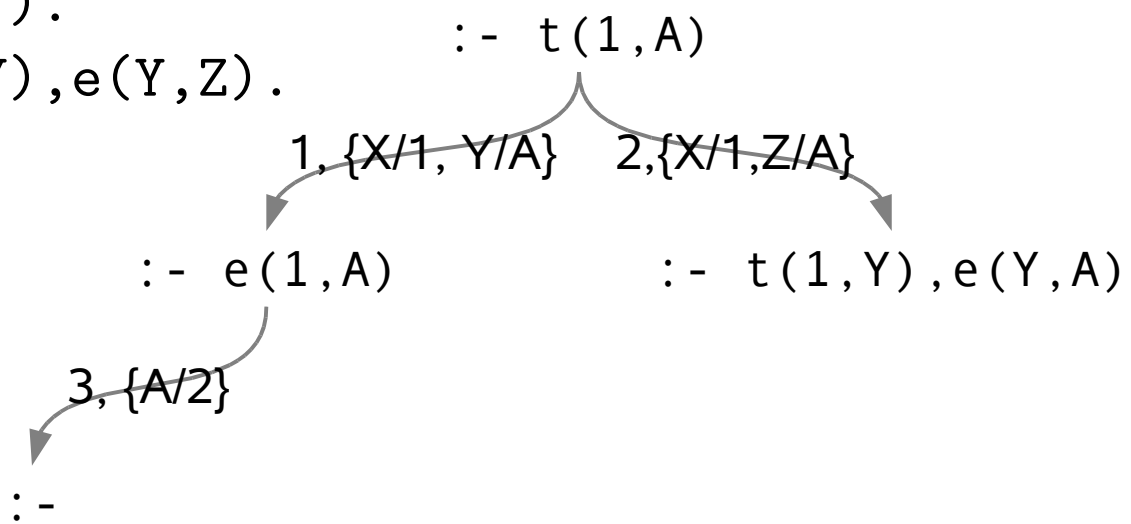
```
1: t(X,Y)←e(X,Y).
2: t(X,Z)←t(X,Y),e(Y,Z).
3: e(1,2).
4: e(2,1).
5: ←t(1,A).
```

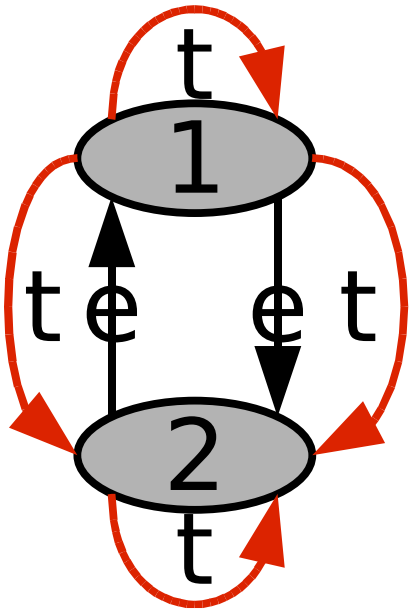




6.4 SLD-Resolution

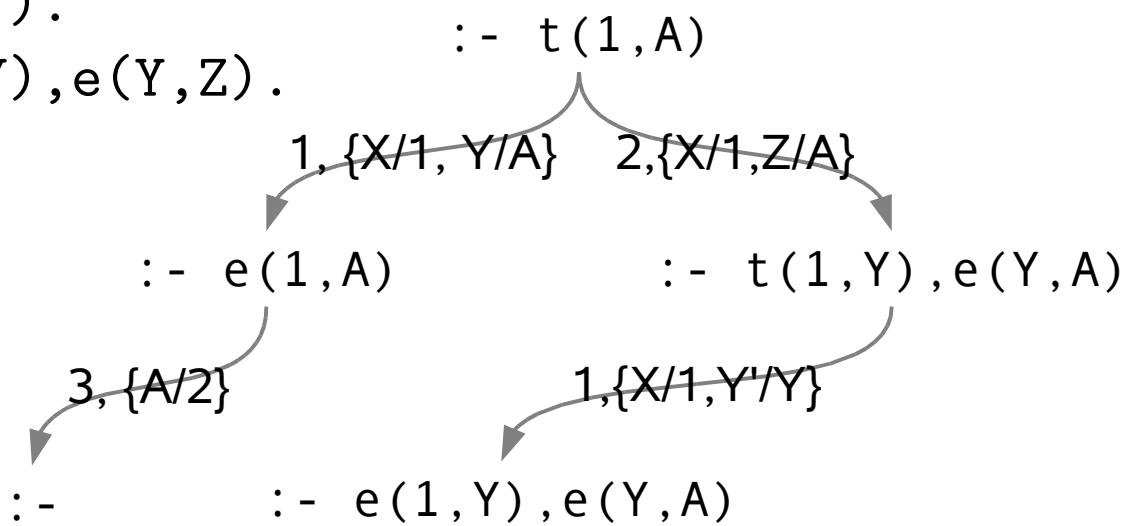
```
1: t(X,Y)←e(X,Y).
2: t(X,Z)←t(X,Y),e(Y,Z).
3: e(1,2).
4: e(2,1).
5: ←t(1,A).
```

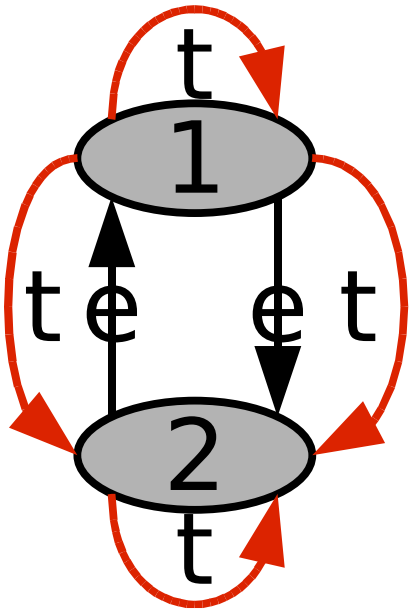




6.4 SLD-Resolution

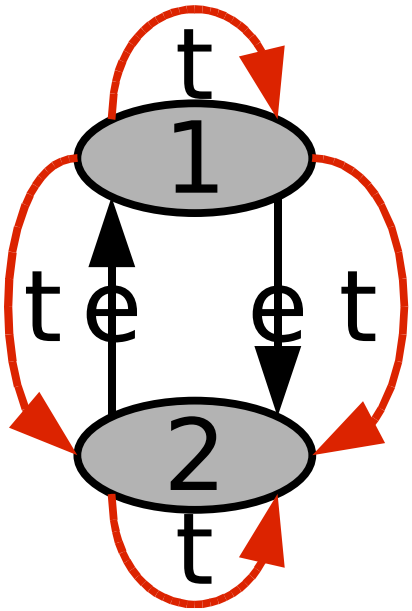
```
1: t(X,Y)←e(X,Y).
2: t(X,Z)←t(X,Y),e(Y,Z).
3: e(1,2).
4: e(2,1).
5: ←t(1,A).
```

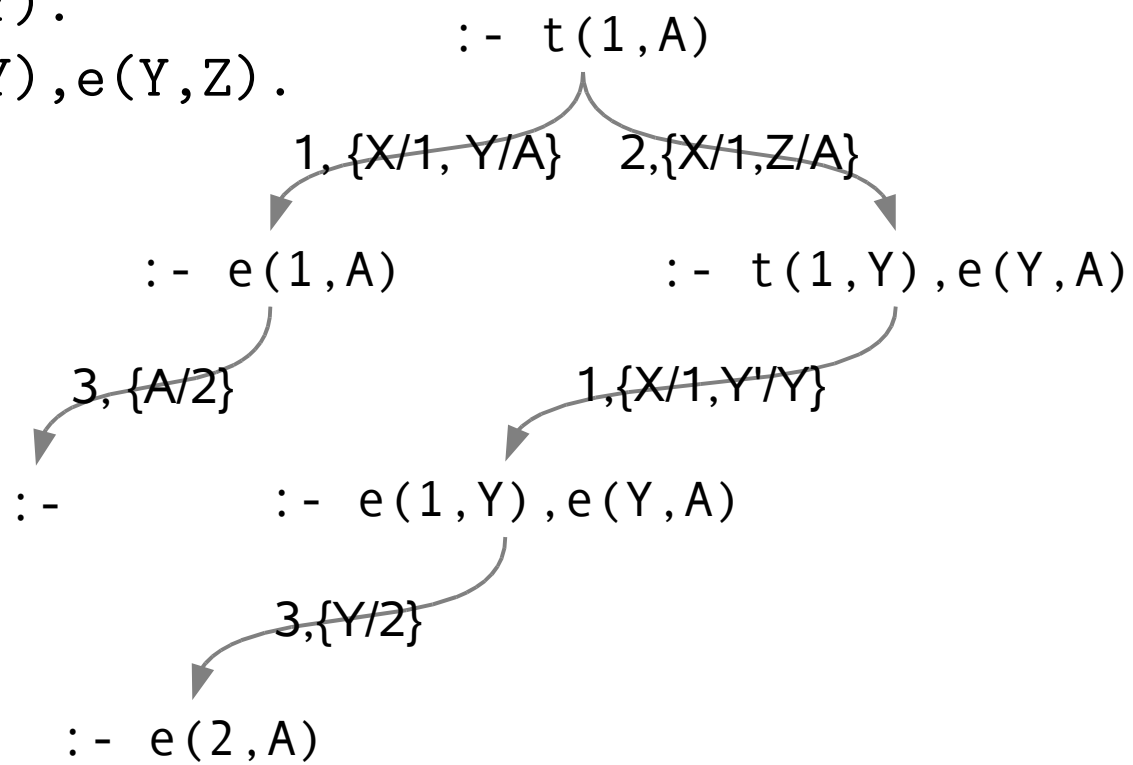




6.4 SLD-Resolution

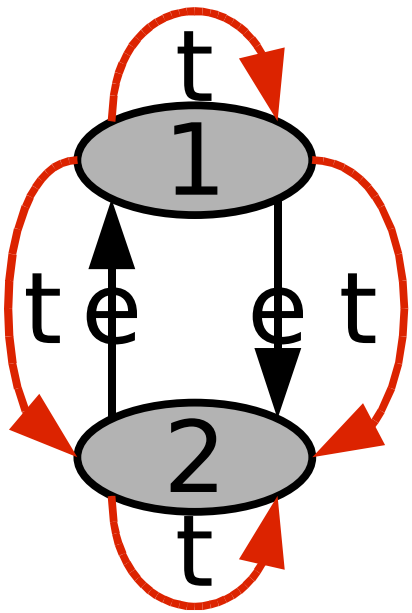
```
1: t(X,Y)←e(X,Y).
2: t(X,Z)←t(X,Y),e(Y,Z).
3: e(1,2).
4: e(2,1).
5: ←t(1,A).
```

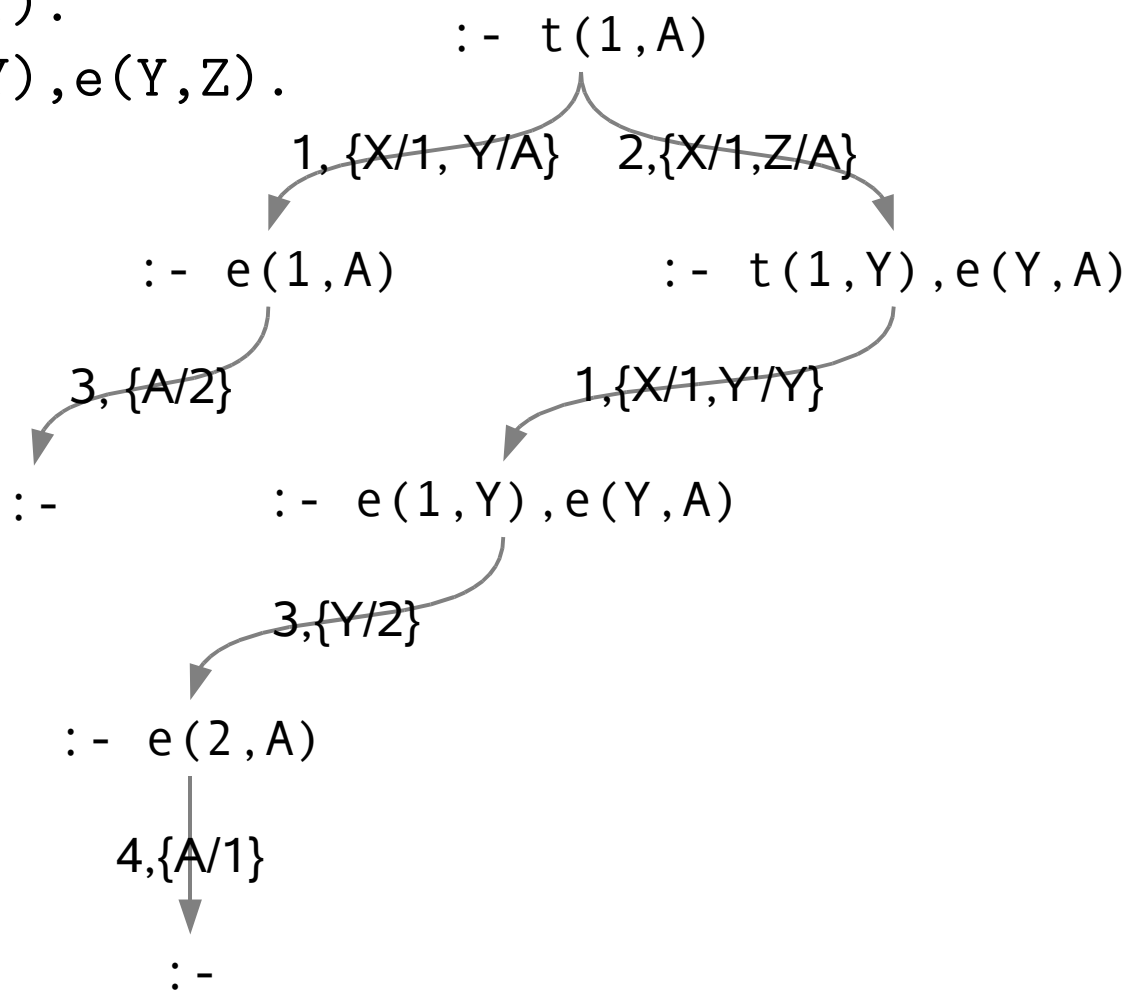




6.4 SLD-Resolution

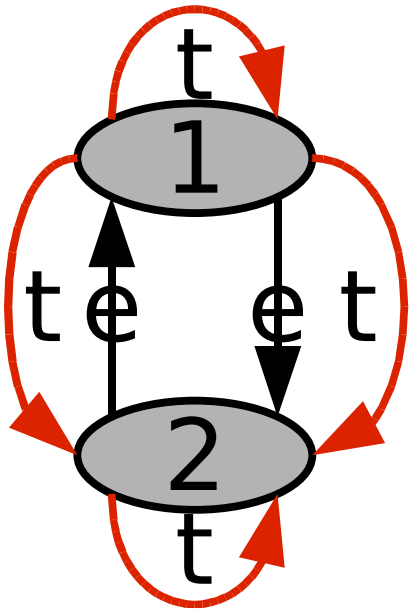
```
1: t(X,Y)←e(X,Y).
2: t(X,Z)←t(X,Y),e(Y,Z).
3: e(1,2).
4: e(2,1).
5: ←t(1,A).
```

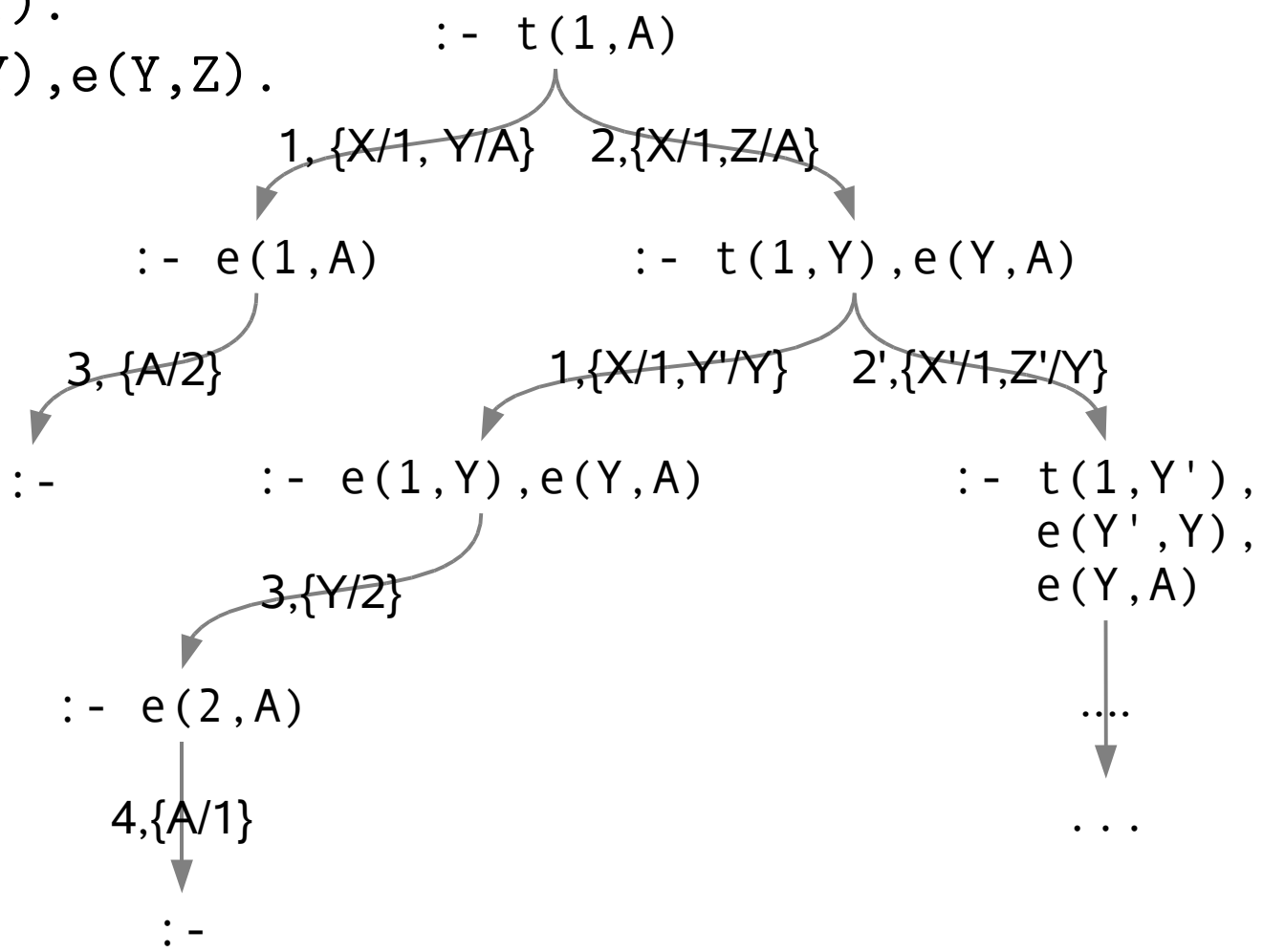




6.4 SLD-Resolution

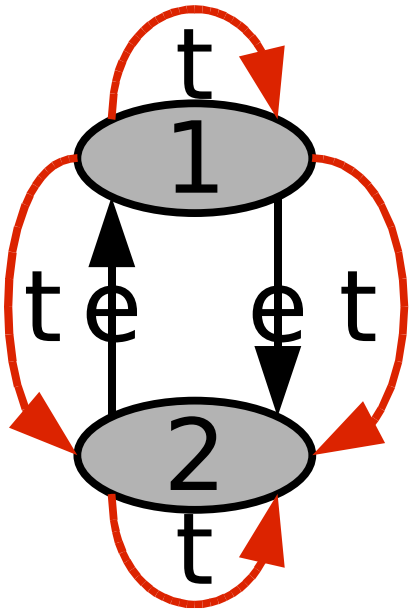
1: t(X,Y)←e(X,Y).
2: t(X,Z)←t(X,Y),e(Y,Z).
3: e(1,2).
4: e(2,1).
5: ←t(1,A).

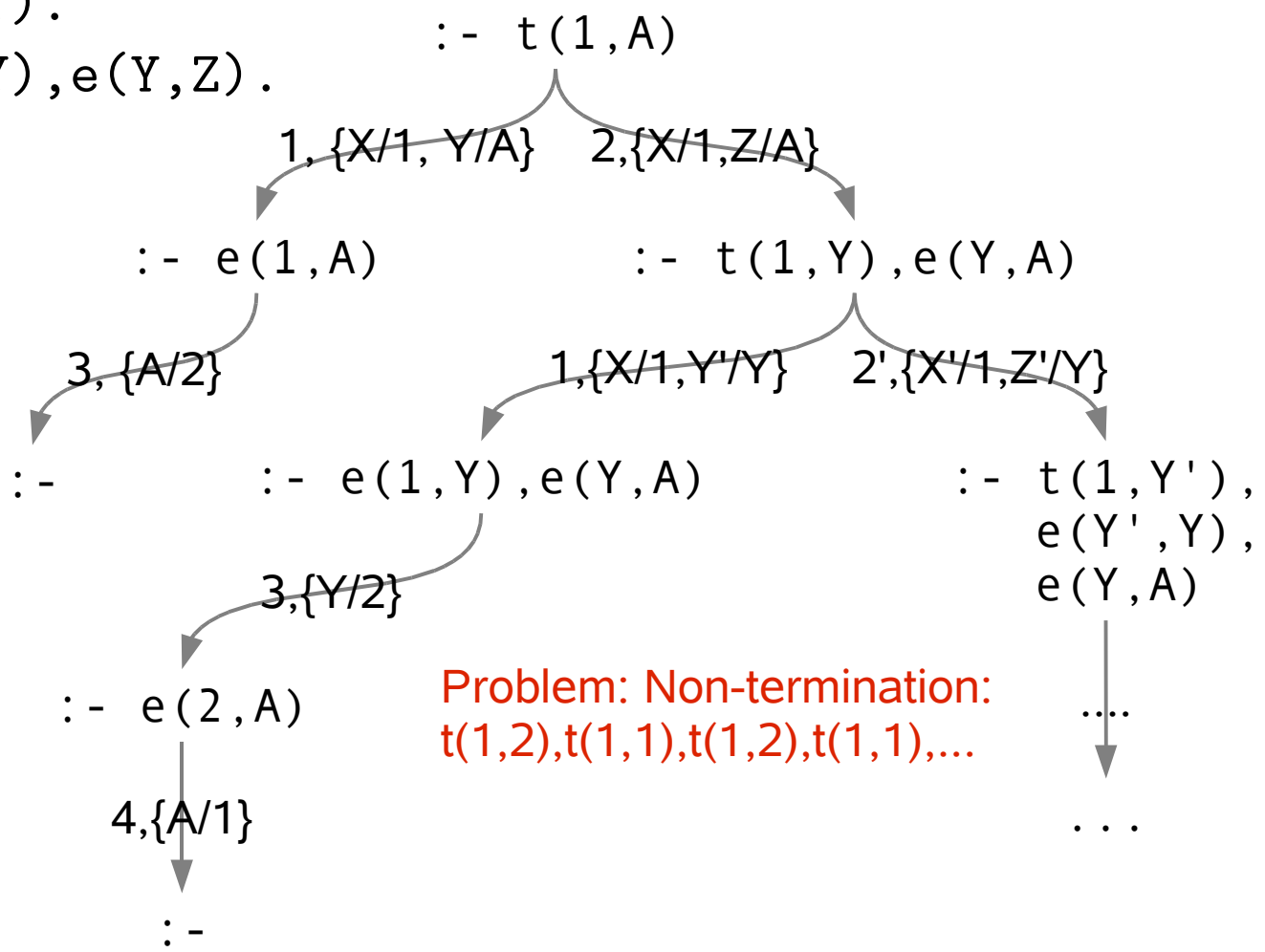




6.4 SLD-Resolution

```
1: t(X,Y)←e(X,Y).
2: t(X,Z)←t(X,Y),e(Y,Z).
3: e(1,2).
4: e(2,1).
5: ←t(1,A).
```

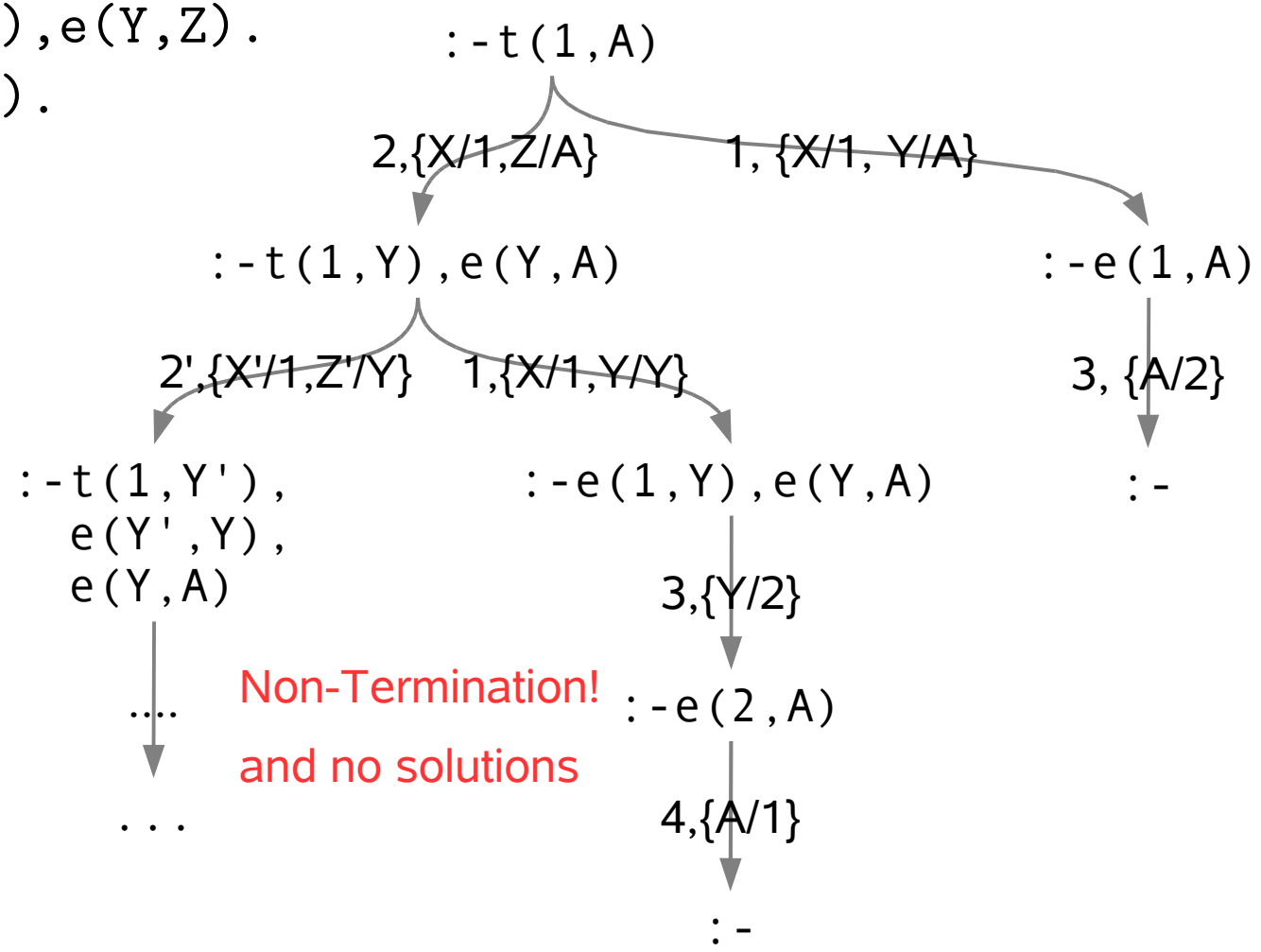




Problem: Non-termination:
t(1,2),t(1,1),t(1,2),t(1,1),...

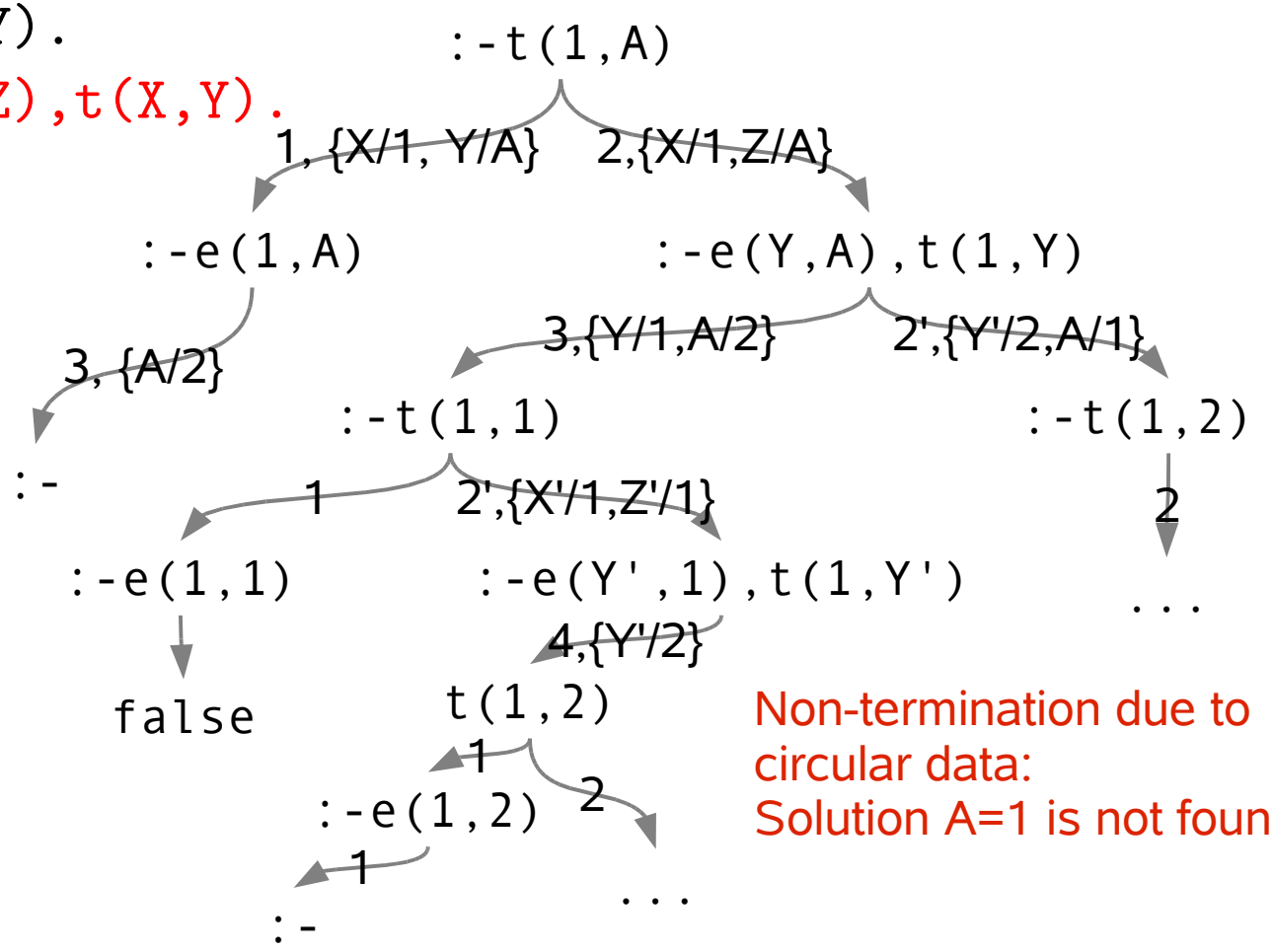
6.4 SLD-Resolution

2: t(X,Z)←t(X,Y),e(Y,Z).
1: t(X,Y)←e(X,Y).
3: e(1,2).
4: e(2,1).
5: ←t(1,A).



6.4 SLD-Resolution

```
1: t(X,Y)←e(X,Y).
2: t(X,Z)←e(Y,Z),t(X,Y).
3: e(1,2).
4: e(2,1).
5: ←t(1,A).
```



Non-termination due to circular data:
Solution A=1 is not found

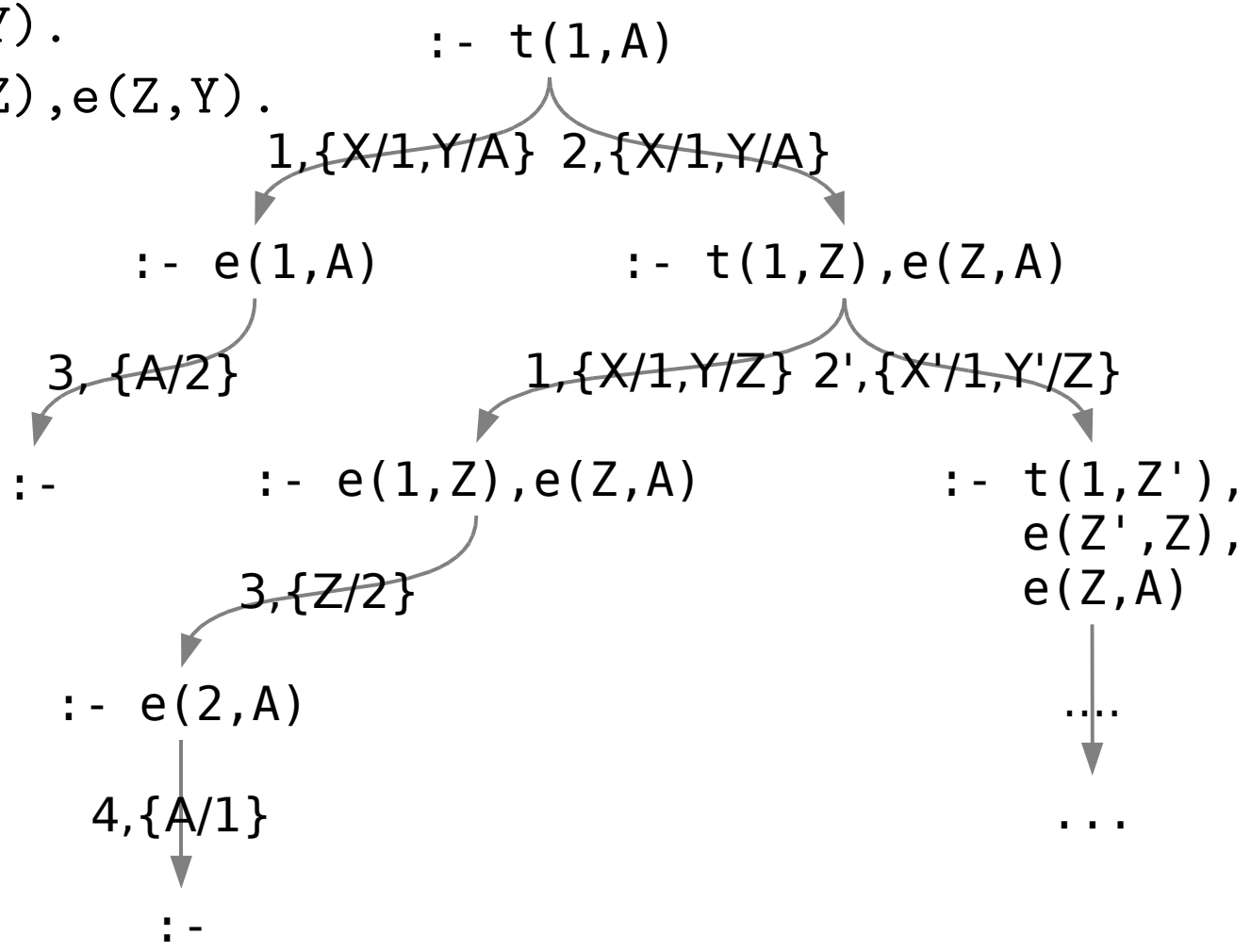
6.5 OLDT-Resolution

Ideas:

- Non-termination due to infinite branches
- Infinite branches due to
  - variants of the same goal on the infinite branch or
  - subsuming goals on the infinite branch
- Avoidance of repeated evaluation of a subgoal on the same computation path
- Side effect: No repeated evaluations of subgoals at all
- Distinction of tabled predicates
- Distinction between solution- and lookup nodes.

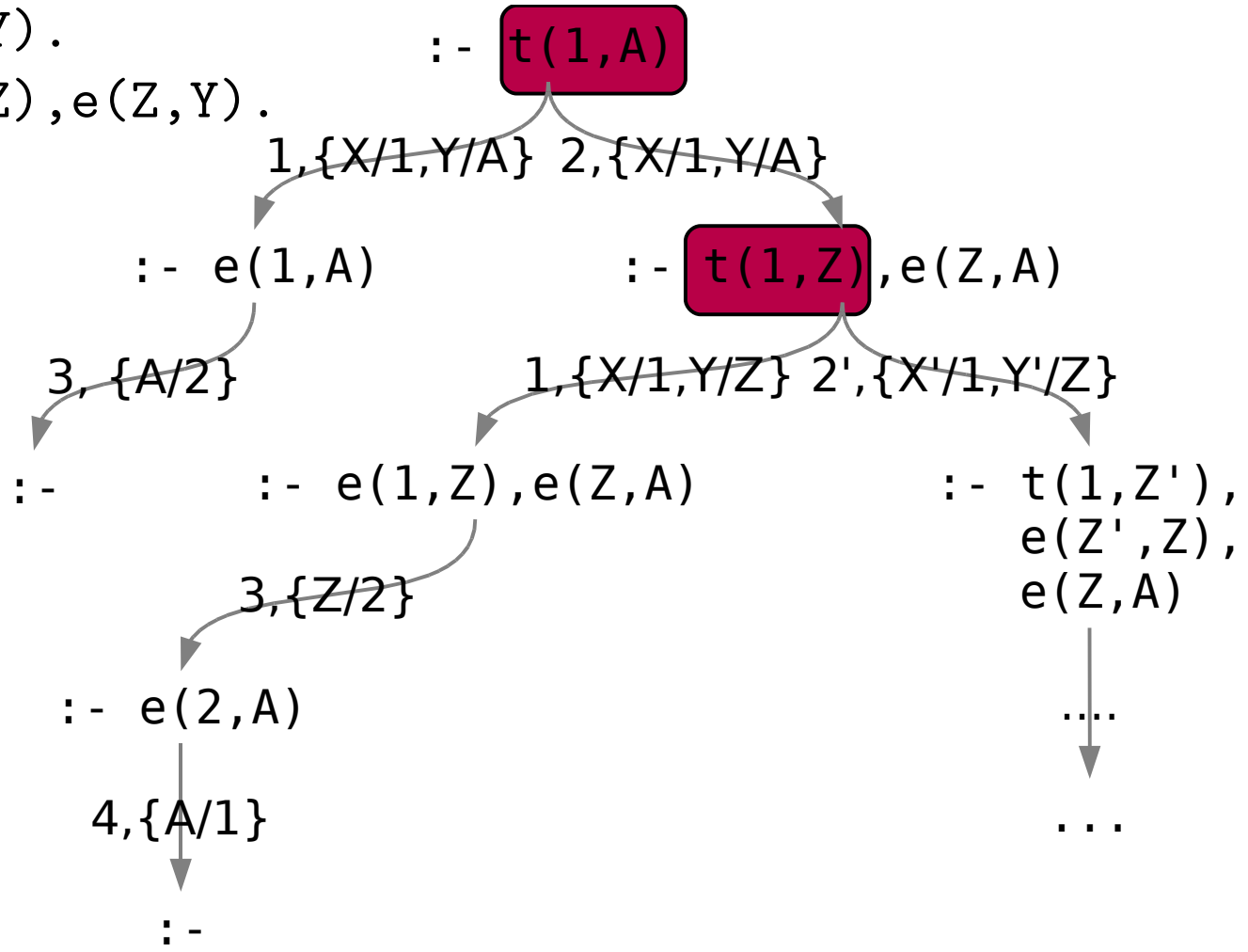
6.5 OLDT-Resolution

1: t(X,Y)←e(X,Y).
2: t(X,Y)←t(X,Z),e(Z,Y).
3: e(1,2).
4: e(2,1).
5: ←t(1,A).



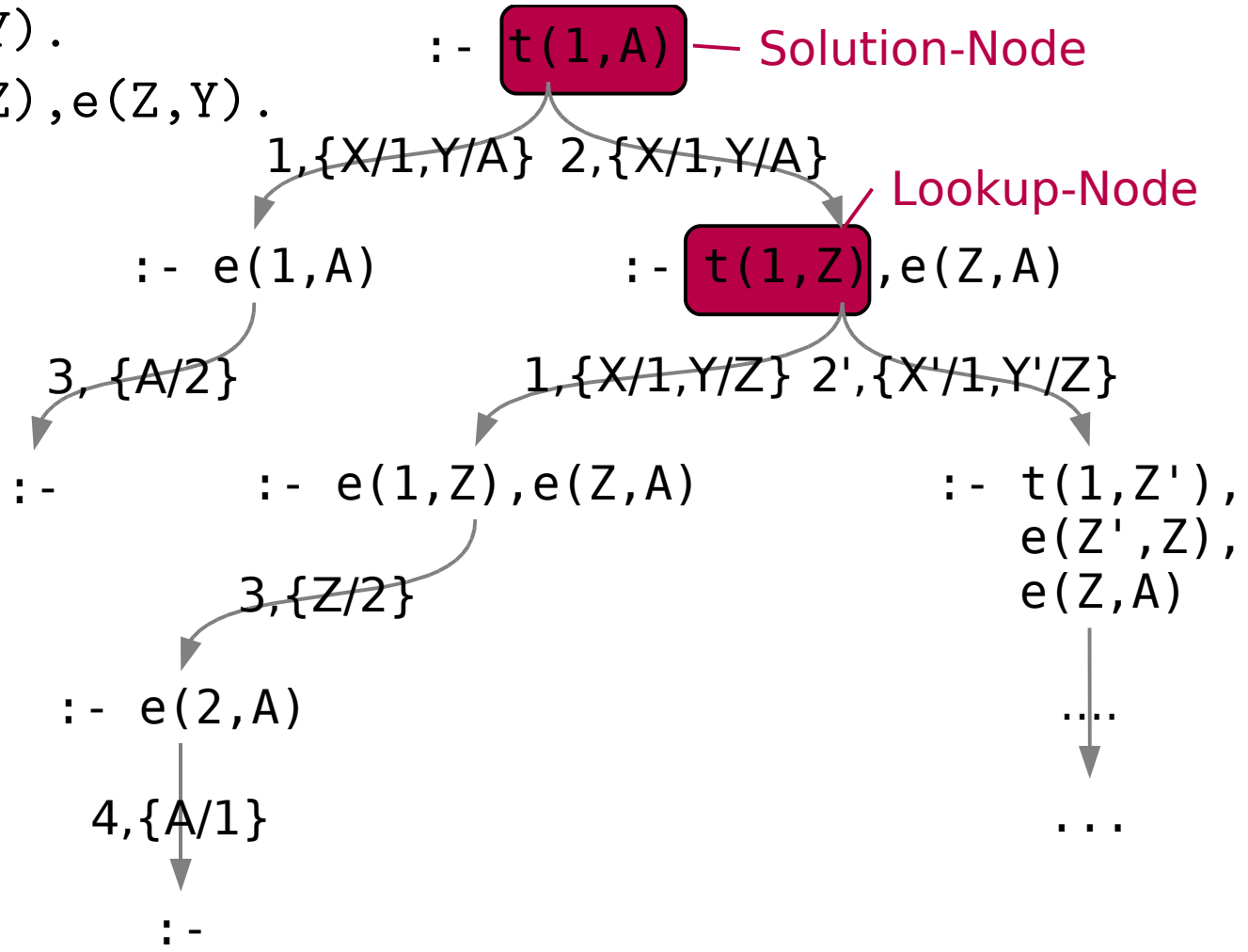
6.5 OLDT-Resolution

1: t(X,Y)←e(X,Y).
2: t(X,Y)←t(X,Z),e(Z,Y).
3: e(1,2).
4: e(2,1).
5: ←t(1,A).



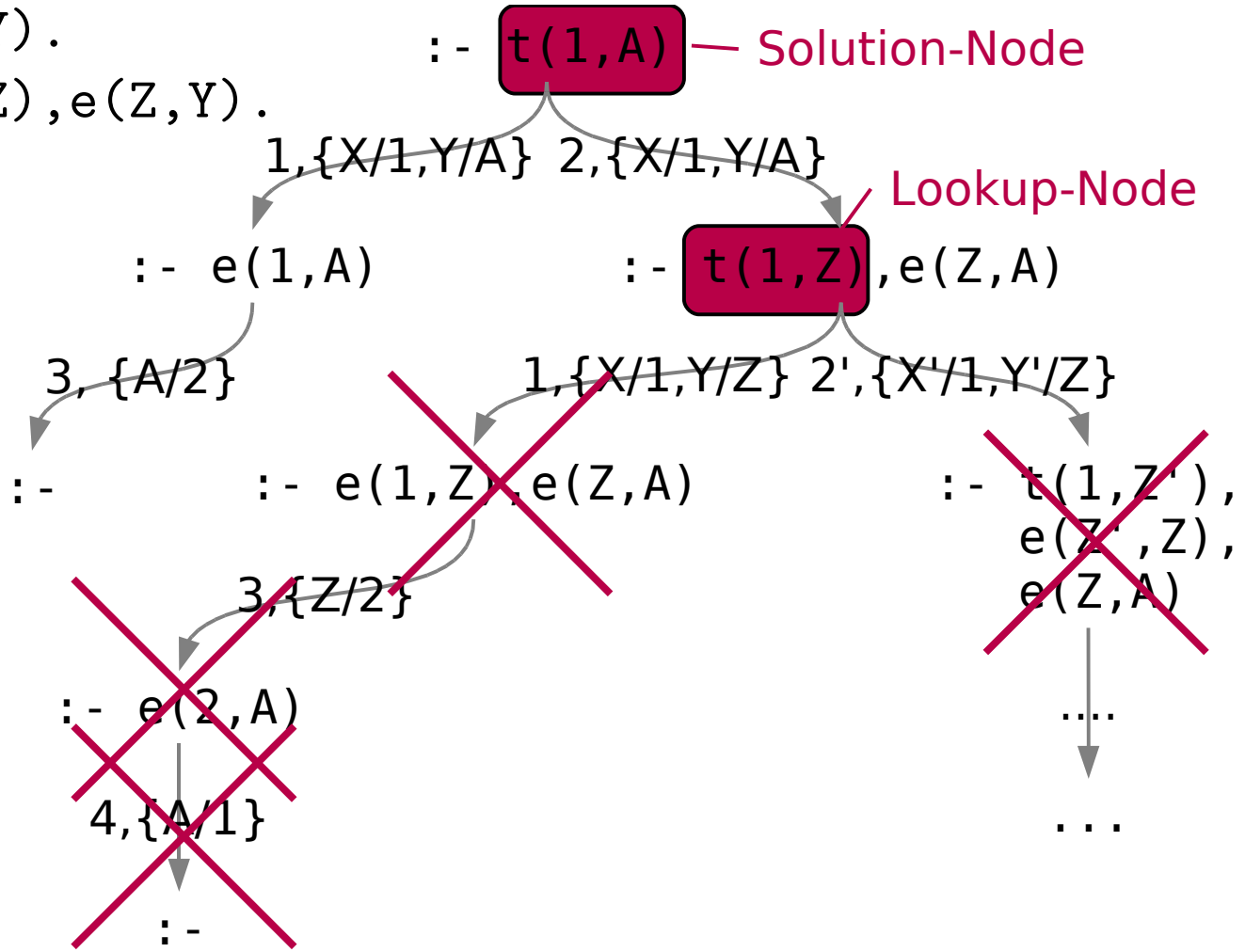
6.5 OLDT-Resolution

1: t(X,Y)←e(X,Y).
2: t(X,Y)←t(X,Z),e(Z,Y).
3: e(1,2).
4: e(2,1).
5: ←t(1,A).



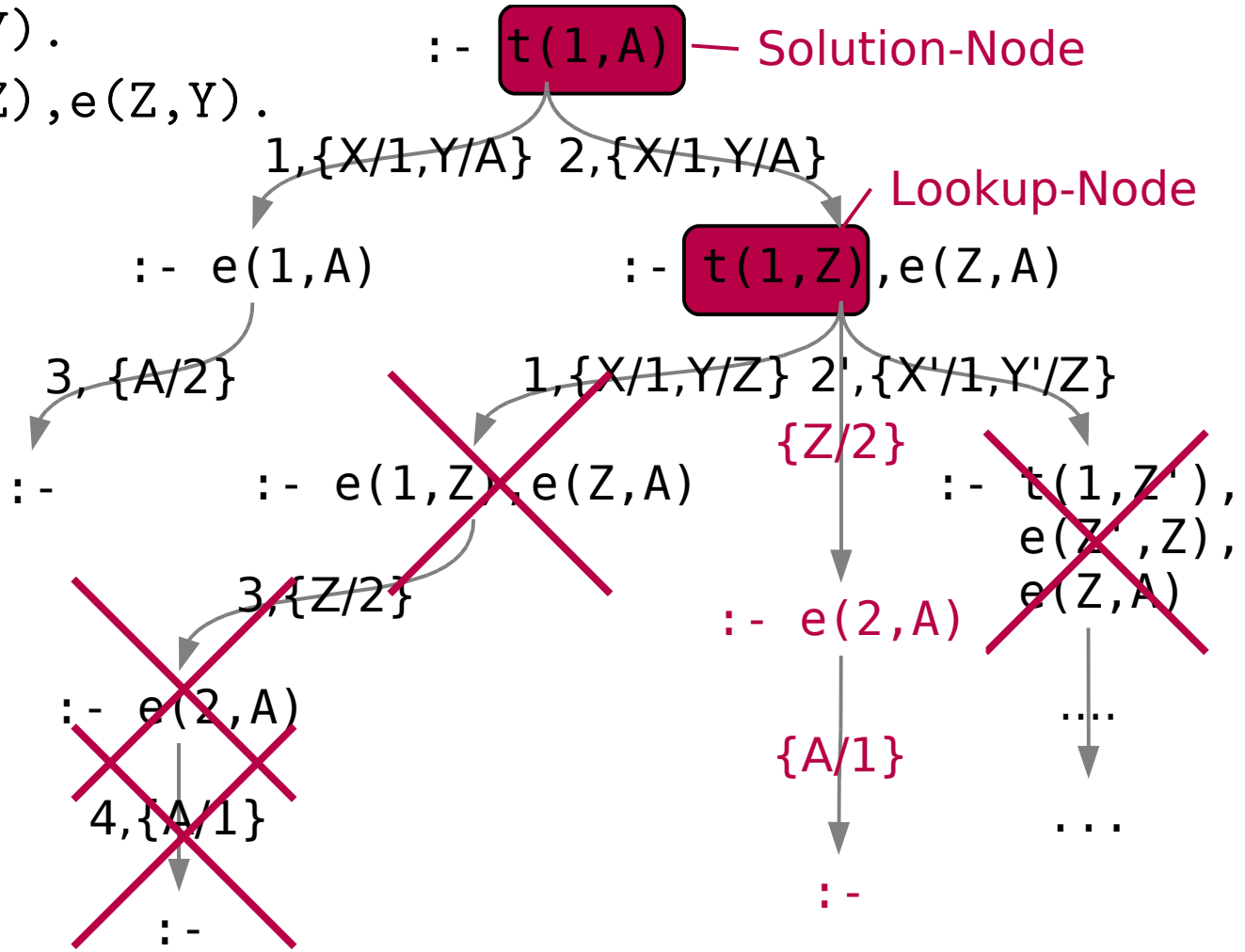
6.5 OLDT-Resolution

```
1: t(X,Y)←e(X,Y).
2: t(X,Y)←t(X,Z),e(Z,Y).
3: e(1,2).
4: e(2,1).
5: ←t(1,A).
```



6.5 OLDT-Resolution

1: t(X,Y)←e(X,Y).
2: t(X,Y)←t(X,Z),e(Z,Y).
3: e(1,2).
4: e(2,1).
5: ←t(1,A).



6.2 The Magic Templates Transformation Algorithm

Until now, we have seen:

- forward chaining (data driven) evaluation of LP
- backward chaining (goal driven) evaluation of LP
- improvement of backward chaining by tabling

Idea of the magic templates transformation:

- take the best of both worlds:
  - Efficiency of goal directedness
  - Good termination properties of forward chaining
  - Easy implementation of a forward chaining rule engine

6.2 The Magic Templates Transformation Algorithm

```
t(X,Y) ← r(X,Y)
t(X,Z) ← t(X,Y), t(Y,Z)
r(a,b).
r(b,c).
r(c,d).
```

6.2 The Magic Templates Transformation Algorithm

Goal-directed evaluation

```
t(X,Y) ← r(X,Y)
t(X,Z) ← t(X,Y), t(Y,Z)
r(a,b).
r(b,c).
r(c,d).
← t(b, Answer).
```

6.2 The Magic Templates Transformation Algorithm

```
t(X,Y) ← r(X,Y)
t1(X,Z) ← t2(X,Y), t3(Y,Z)
r(a,b).
r(b,c).
r(c,d).
← t(b, Answer).
```

Goal-directed evaluation

Information passing

- $\texttt{t} \hookrightarrow_X \texttt{r}$.
- $\texttt{t}_1 \hookrightarrow_X \texttt{t}_2$.
- $\texttt{t}_2 \hookrightarrow_Y \texttt{t}_3$.

6.2 The Magic Templates Transformation Algorithm

```
t(X,Y) ← r(X,Y)
t1^bf(X,Z) ← t2^bf(X,Y), t3^bf(Y,Z)
r(a,b).
r(b,c).
r(c,d).
← t(b, Answer).
```

Goal-directed evaluation

Information passing

- $\mathtt{t} \hookrightarrow_X \mathtt{r}$.
- $\mathtt{t}_1 \hookrightarrow_X \mathtt{t}_2$.
- $\mathtt{t}_2 \hookrightarrow_Y \mathtt{t}_3$.

Adornment (**b**ound, **f**ree)

6.2 The Magic Templates Transformation Algorithm

```
t(X,Y) ← r(X,Y)
t1^bf(X,Z) ← t2^bf(X,Y), t3^bf(Y,Z)
r(a,b).
r(b,c).
r(c,d).
← t(b, Answer).
```

Goal-directed evaluation

Information passing

- $\texttt{t} \hookrightarrow_X \texttt{r}$.
- $\texttt{t}_1 \hookrightarrow_X \texttt{t}_2$.
- $\texttt{t}_2 \hookrightarrow_Y \texttt{t}_3$.

Adornment (**b**ound, **f**ree)

$\texttt{mag\_t}^{bf}\texttt{(b)}$.

Magic Rules

Rewritten Rules

6.2 The Magic Templates Transformation Algorithm

```
t(X,Y) ← r(X,Y)
t1^bf(X,Z) ← t2^bf(X,Y), t3^bf(Y,Z)
r(a,b).
r(b,c).
r(c,d).
← t(b, Answer).
```

Goal-directed evaluation

Information passing

- $\texttt{t} \hookrightarrow_X \texttt{r}$.
- $\texttt{t}_1 \hookrightarrow_X \texttt{t}_2$.
- $\texttt{t}_2 \hookrightarrow_Y \texttt{t}_3$.

Adornment (**b**ound, **f**ree)

```
mag_t^bf(b).
mag_r^bf(X) ← mag_t^bf(X).
```

Magic Rules

Rewritten Rules

6.2 The Magic Templates Transformation Algorithm

```
t(X,Y) ← r(X,Y)
t1^bf(X,Z) ← t2^bf(X,Y), t3^bf(Y,Z)
r(a,b).
r(b,c).
r(c,d).
← t(b, Answer).
```

Goal-directed evaluation

Information passing

- $\texttt{t} \hookrightarrow_X \texttt{r}$.
- $\texttt{t}_1 \hookrightarrow_X \texttt{t}_2$.
- $\texttt{t}_2 \hookrightarrow_Y \texttt{t}_3$.

Adornment (**b**ound, **f**ree)

```
mag_t^bf(b).
mag_r^bf(X) ← mag_t^bf(X).
mag_t^bf(X) ← mag_t^bf(X).
```

Magic Rules

Rewritten Rules

6.2 The Magic Templates Transformation Algorithm

```
t(X,Y) ← r(X,Y)
t1^bf(X,Z) ← t2^bf(X,Y), t3^bf(Y,Z)
r(a,b).
r(b,c).
r(c,d).
← t(b, Answer).
```

Goal-directed evaluation

Information passing

- $\texttt{t} \hookrightarrow_X \texttt{r}$.
- $\texttt{t}_1 \hookrightarrow_X \texttt{t}_2$.
- $\texttt{t}_2 \hookrightarrow_Y \texttt{t}_3$.

Adornment (**b**ound, **f**ree)

```
mag_t^bf(b).
mag_r^bf(X) ← mag_t^bf(X).
mag_t^bf(X) ← mag_t^bf(X).
mag_t^bf(Y) ← mag_t^bf(X), t(X,Y).
```

Magic Rules

Rewritten Rules

6.2 The Magic Templates Transformation Algorithm

```
t(X,Y) ← r(X,Y)
t1^bf(X,Z) ← t2^bf(X,Y), t3^bf(Y,Z)
r(a,b).
r(b,c).
r(c,d).
← t(b, Answer).
```

Goal-directed evaluation

Information passing

- $t \hookrightarrow_X r$.
- $t_1 \hookrightarrow_X t_2$.
- $t_2 \hookrightarrow_Y t_3$.

Adornment (**b**ound, **f**ree)

```
mag_t^bf(b).
mag_r^bf(X) ← mag_t^bf(X).
mag_t^bf(X) ← mag_t^bf(X).
mag_t^bf(Y) ← mag_t^bf(X), t(X,Y).
t(X,Y) ← mag_t^bf(X), r(X,Y).
t(X,Z) ← mag_t^bf(X),t(X,Y),t(Y,Z)
r(a,b).  r(b,c).  r(c,d).
```

Magic Rules

Rewritten Rules

6.2 The Magic Templates Transformation Algorithm

```
t(X,Y) ← r(X,Y)
t1^bf(X,Z) ← t2^bf(X,Y), t3^bf(Y,Z)
r(a,b).
r(b,c).
r(c,d).
← t(b, Answer).
```

Goal-directed evaluation

Information passing

- $t \hookrightarrow_X r$.
- $t_1 \hookrightarrow_X t_2$.
- $t_2 \hookrightarrow_Y t_3$.

Adornment (**b**ound, **f**ree)

```
mag_t^bf(b).
mag_r^bf(X) ← mag_t^bf(X).
mag_t^bf(X) ← mag_t^bf(X).
mag_t^bf(Y) ← mag_t^bf(X), t(X,Y).
t(X,Y) ← mag_t^bf(X), r(X,Y).
t(X,Z) ← mag_t^bf(X),t(X,Y),t(Y,Z)
r(a,b).  r(b,c).  r(c,d).
```

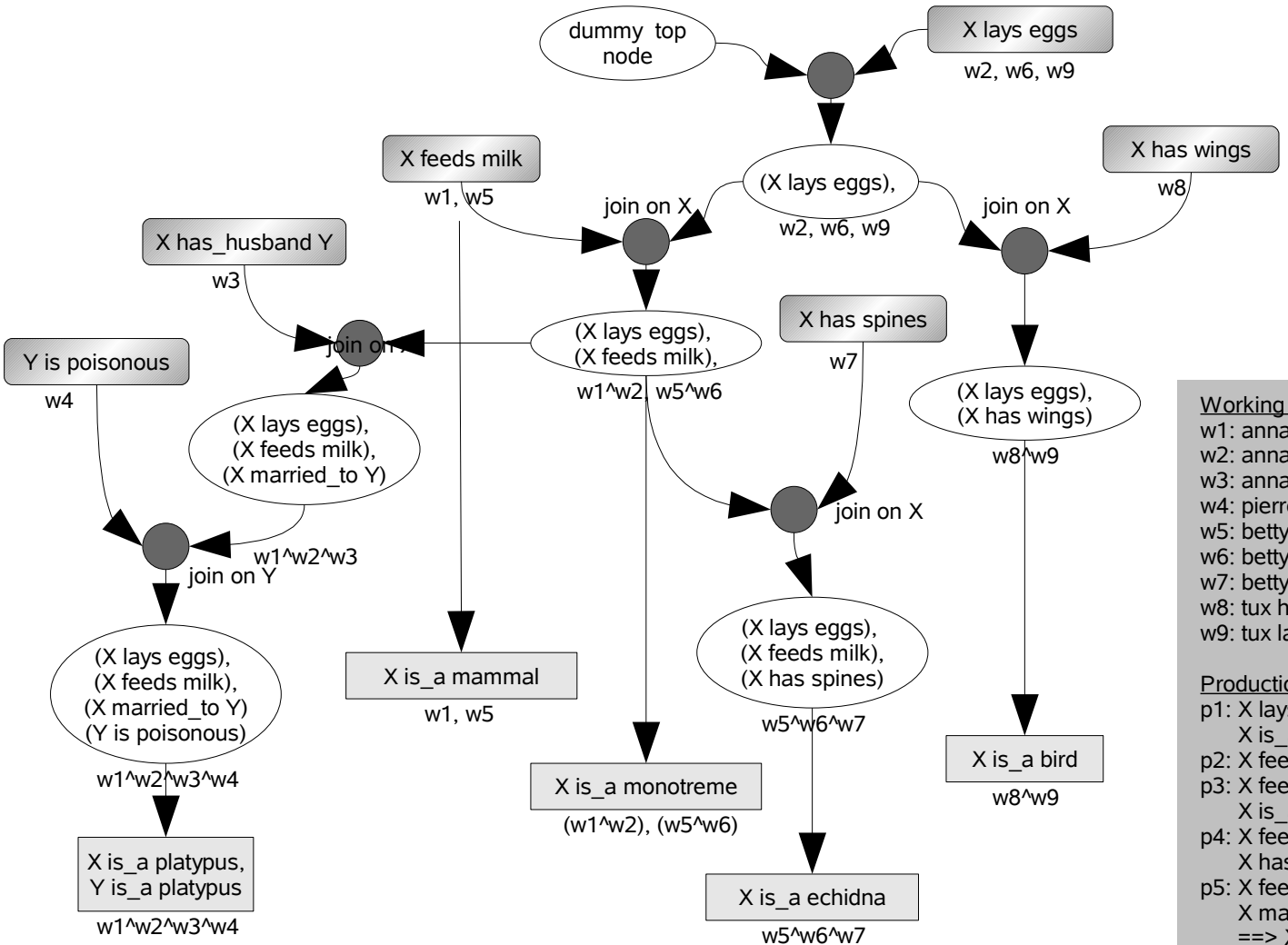
Evaluation:

```
mag_t^bf(b).
t(b,c).
mag_t^bf(c).
t(c,d).
mag_t^bf(d).
t(b,d).
```

6.3 The Rete Algorithm

- By Charles Forgy
- Forward chaining evaluation
- Storage of partially instantiated rules
- Sharing of instantiated literals among similar rules
- Suitable for reason maintenance
- Several optimizations, industrial use

6.3 The Rete Algorithm



Working memory:
w1: anna feeds milk
w2: anna lays eggs
w3: anna married_to pierre
w4: pierre is poisonous
w5: betty feeds milk
w6: betty lays eggs
w7: betty has spines
w8: tux has wings
w9: tux lays eggs

Production memory:
p1: X lays eggs, X has wings ==>
X is_a bird
p2: X feeds milk ==> X is_a mammal
p3: X feeds milk, X lays eggs ==>
X is_a monotreme
p4: X feeds milk, X lays eggs,
X has_spines ==> X is_a echidna
p5: X feeds milk, X lays eggs,
X married_to Y, Y is poisonous
==> X is_a platypus,
Y is_a platypus

6.6 The Backward Fixpoint Procedure

Ideas:

- Desirability of fixpoint computation
- Set-oriented fact processing
- bottom-up meta-interpretation
- Rewriting- resolution-based methods as implementations of the BFP
- Alexander and Magic Set methods as specializations of the BFP
- BFP as a logical specification of SLD-Resolution

6.6 The Backward Fixpoint Procedure

$$\texttt{fact(Q)} \leftarrow \texttt{query}_b\texttt{(Q)} \wedge \texttt{rule(Q} \leftarrow \texttt{B)} \wedge \texttt{evaluate(B)}.$$
$$\texttt{query}_b\texttt{(B)} \leftarrow \texttt{query}_b\texttt{(Q)} \wedge \texttt{rule(Q} \leftarrow \texttt{B)}.$$
$$\texttt{query}_b\texttt{(Q}_1\texttt{)} \leftarrow \texttt{query}_b\texttt{(Q}_1 \wedge \texttt{Q}_2\texttt{)}.$$
$$\texttt{query}_b\texttt{(Q}_2\texttt{)} \leftarrow \texttt{query}_b\texttt{(Q}_1 \wedge \texttt{Q}_2\texttt{)} \wedge \texttt{evaluate(Q}_1\texttt{)}.$$

- Rules of the object program
- Rules of the meta-interpeter
- bottom-up (forward chaining) evaluation of the meta-interpreter $\Rightarrow$ top-down (backward chaining) evaluation of the object program.
- `evaluate(B)` is true if all facts in `B` have been proven.

6.6 The Backward Fixpoint Procedure

$$\mathtt{fact(Q)} \leftarrow \mathtt{query}_b\mathtt{(Q)} \wedge \mathtt{rule(Q} \leftarrow \mathtt{B)} \wedge \mathtt{evaluate(B)}.$$
$$\mathtt{query}_b\mathtt{(B)} \leftarrow \mathtt{query}_b\mathtt{(Q)} \wedge \mathtt{rule(Q} \leftarrow \mathtt{B)}.$$
$$\mathtt{query}_b\mathtt{(Q}_1\mathtt{)} \leftarrow \mathtt{query}_b\mathtt{(Q}_1 \wedge \mathtt{Q}_2\mathtt{)}.$$
$$\mathtt{query}_b\mathtt{(Q}_2\mathtt{)} \leftarrow \mathtt{query}_b\mathtt{(Q}_1 \wedge \mathtt{Q}_2\mathtt{)} \wedge \mathtt{evaluate(Q}_1\mathtt{)}.$$

```
t(X,Y) ← r(X,Y).
t(X,Z) ← t(X,Y), t(Y,Z).
r(a,b).
r(b,c).
r(c,d).
← t(b, Answer).
```

6.6 The Backward Fixpoint Procedure

$$\mathtt{fact(Q)} \leftarrow \mathtt{query}_b\mathtt{(Q)} \wedge \mathtt{rule(Q} \leftarrow \mathtt{B)} \wedge \mathtt{evaluate(B)}.$$
$$\mathtt{query}_b\mathtt{(B)} \leftarrow \mathtt{query}_b\mathtt{(Q)} \wedge \mathtt{rule(Q} \leftarrow \mathtt{B)}.$$
$$\mathtt{query}_b\mathtt{(Q}_1\mathtt{)} \leftarrow \mathtt{query}_b\mathtt{(Q}_1 \wedge \mathtt{Q}_2\mathtt{)}.$$
$$\mathtt{query}_b\mathtt{(Q}_2\mathtt{)} \leftarrow \mathtt{query}_b\mathtt{(Q}_1 \wedge \mathtt{Q}_2\mathtt{)} \wedge \mathtt{evaluate(Q}_1\mathtt{)}.$$

```
rule( t(X,Y) ← r(X,Y) ).
rule( t(X,Z) ← t(X,Y), t(Y,Z) ).
fact( r(a,b) ).
fact( r(b,c) ).
fact( r(c,d) ).
query_b( t(b,Answer) ).
```

6.6 The Backward Fixpoint Procedure

$\texttt{fact(Q)} \leftarrow \texttt{query}_b\texttt{(Q)} \wedge \texttt{rule(Q} \leftarrow \texttt{B)} \wedge \texttt{evaluate(B)}.$
$\texttt{query}_b\texttt{(B)} \leftarrow \texttt{query}_b\texttt{(Q)} \wedge \texttt{rule(Q} \leftarrow \texttt{B)}.$
$\texttt{query}_b\texttt{(Q}_1\texttt{)} \leftarrow \texttt{query}_b\texttt{(Q}_1 \wedge \texttt{Q}_2\texttt{)}.$
$\texttt{query}_b\texttt{(Q}_2\texttt{)} \leftarrow \texttt{query}_b\texttt{(Q}_1 \wedge \texttt{Q}_2\texttt{)} \wedge \texttt{evaluate(Q}_1\texttt{)}.$

```
rule( t(X,Y) ← r(X,Y) ).
rule( t(X,Z) ← t(X,Y), t(Y,Z) ).
fact( r(a,b) ).
fact( r(b,c) ).
fact( r(c,d) ).
query_b( t(b,Answer) ).
```

Evaluation:

```
fact( t(b,c) ).
query_b( t(b,Y), t(Y,Z) ).
query_b( t(b,Y) ).
query_b( t(c,Z) ).
fact( t(c,d) ).
fact( t(b,d) ).
```

Session 14:30 – 16:00

- 6 Operational Semantics: Positive Rule Sets
- 7 Operational Semantics: Rule Sets with Non-monotonic Negation

7 Operational Semantics: Negative

7.1 Iterative Fixpoint Semantics, Stratified

```
q ← p.
p ← not q.
```

This Section shows algorithms for the computation of

- Stratified Semantics
- Well-Founded Semantics
- Stable Model Semantics

7.1 Iterative Fixpoint Semantics, Stratified

Recapitulation of Stratification:

- Partitioning of the set of rules S of a program
- C_1, C_2 define the same predicate $P \Rightarrow C_1$, C_2 must be in the same stratum
- $p(X)$ is a positive body literal of a rule in layer i, then p must be defined in a layer $j \leq i$.
- $\neg p(X)$ is a negative body literal of a rule in layer i, then p must be defined in layer $j < i$.

7.1 Iterative Fixpoint Semantics, Stratified

Recapitulation of Stratification:

- Partitioning of the set of rules S of a program
- C_1, C_2 define the same predicate $P \Rightarrow C_1$, C_2 must be in the same stratum
- $p(X)$ is a positive body literal of a rule in layer i, then p must be defined in a layer $j \leq i$.
- $\neg p(X)$ is a negative body literal of a rule in layer i, then p must be defined in layer $j < i$.

Iterative fixpoint semantics only provides a semantics for *stratifiable* programs

Stratification Example

```
human(john).
male(john).
plays_the_piano(john).
has_hobbies(X) ← plays_the_piano(X).
has_child(X) ← human(X), not has_hobbies(X).
married(X) ← human(X), has_child(X).
bachelor(X) ← male(X), not married(X).
```

Evaluation:

- $M_0 :=$ `human(john), male(john), plays_the_piano(john)`
- stratum 1: $M_1 := \mathbf{T}^{\omega}_{S_1}(M_0) = M_0 \cup$ {`has_hobbies(john)`}.
- stratum 2: $M_2 := \mathbf{T}^{\omega}_{S_2}(M_1) = M_1 \cup \emptyset$
- stratum 3: $M_3 := \mathbf{T}^{\omega}_{S_3}(M_2) = M_2 \cup$ {`bachelor(john)`}.

Stratification Example

```
human(john).
male(john).
plays_the_piano(john).
has_hobbies(X) ← plays_the_piano(X).
has_child(X) ← human(X), not has_hobbies(X).
married(X) ← human(X), has_child(X).
bachelor(X) ← male(X), not married(X).
```

Evaluation:

- $M_0 :=$ `human(john), male(john), plays_the_piano(john)`
- stratum 1: $M_1 := \mathbf{T}^{\omega}_{S_1}(M_0) = M_0 \cup$ {`has_hobbies(john)`}.
- stratum 2: $M_2 := \mathbf{T}^{\omega}_{S_2}(M_1) = M_1 \cup \emptyset$
- stratum 3: $M_3 := \mathbf{T}^{\omega}_{S_3}(M_2) = M_2 \cup$ {`bachelor(john)`}.

Stratification Example

```
human(john).
male(john).
plays_the_piano(john).
has_hobbies(X) ← plays_the_piano(X).                    stratum 1
has_child(X) ← human(X), not has_hobbies(X).
married(X) ← human(X), has_child(X).
bachelor(X) ← male(X), not married(X).
```

Evaluation:

- $M_0 :=$ `human(john), male(john), plays_the_piano(john)`
- stratum 1: $M_1 := \mathbf{T}^{\omega}_{S_1}(M_0) = M_0 \cup$ {`has_hobbies(john)`}.
- stratum 2: $M_2 := \mathbf{T}^{\omega}_{S_2}(M_1) = M_1 \cup \emptyset$
- stratum 3: $M_3 := \mathbf{T}^{\omega}_{S_3}(M_2) = M_2 \cup$ {`bachelor(john)`}.

Stratification Example

```
human(john).
male(john).
plays_the_piano(john).
has_hobbies(X) ← plays_the_piano(X).                    stratum 1
has_child(X) ← human(X), not has_hobbies(X).            stratum 2
married(X) ← human(X), has_child(X).
bachelor(X) ← male(X), not married(X).
```

Evaluation:

- $M_0 :=$ `human(john), male(john), plays_the_piano(john)`
- stratum 1: $M_1 := \mathbf{T}^{\omega}_{S_1}(M_0) = M_0 \cup$ {`has_hobbies(john)`}.
- stratum 2: $M_2 := \mathbf{T}^{\omega}_{S_2}(M_1) = M_1 \cup \emptyset$
- stratum 3: $M_3 := \mathbf{T}^{\omega}_{S_3}(M_2) = M_2 \cup$ {`bachelor(john)`}.

Stratification Example

```
human(john).
male(john).
plays_the_piano(john).
has_hobbies(X) ← plays_the_piano(X).                    stratum 1
has_child(X) ← human(X), not has_hobbies(X).            stratum 2
married(X) ← human(X), has_child(X).                    stratum 2
bachelor(X) ← male(X), not married(X).
```

Evaluation:

- $M_0 :=$ `human(john), male(john), plays_the_piano(john)`
- stratum 1: $M_1 := \mathbf{T}^{\omega}_{S_1}(M_0) = M_0 \cup$ {`has_hobbies(john)`}.
- stratum 2: $M_2 := \mathbf{T}^{\omega}_{S_2}(M_1) = M_1 \cup \emptyset$
- stratum 3: $M_3 := \mathbf{T}^{\omega}_{S_3}(M_2) = M_2 \cup$ {`bachelor(john)`}.

Stratification Example

```
human(john).
male(john).
plays_the_piano(john).
has_hobbies(X) ← plays_the_piano(X).                    stratum 1
has_child(X) ← human(X), not has_hobbies(X).            stratum 2
married(X) ← human(X), has_child(X).                    stratum 2
bachelor(X) ← male(X), not married(X).                  stratum 3
```

Evaluation:

- $M_0 :=$ `human(john), male(john), plays_the_piano(john)`
- stratum 1: $M_1 := \mathbf{T}^{\omega}_{S_1}(M_0) = M_0 \cup$ {`has_hobbies(john)`}.
- stratum 2: $M_2 := \mathbf{T}^{\omega}_{S_2}(M_1) = M_1 \cup \emptyset$
- stratum 3: $M_3 := \mathbf{T}^{\omega}_{S_3}(M_2) = M_2 \cup$ {`bachelor(john)`}.

Stratification Example

```
human(john).
male(john).
plays_the_piano(john).
has_hobbies(X) ← plays_the_piano(X).          stratum 1
has_child(X) ← human(X), not has_hobbies(X).  stratum 2
married(X) ← human(X), has_child(X).          stratum 2
bachelor(X) ← male(X), not married(X).        stratum 3
```

Evaluation:

- $M_0 :=$ `human(john), male(john), plays_the_piano(john)`
- stratum 1: $M_1 := \mathbf{T}^{\omega}_{S_1}(M_0) = M_0 \cup$ {`has_hobbies(john)`}.
- stratum 2: $M_2 := \mathbf{T}^{\omega}_{S_2}(M_1) = M_1 \cup \emptyset$
- stratum 3: $M_3 := \mathbf{T}^{\omega}_{S_3}(M_2) = M_2 \cup$ {`bachelor(john)`}.

Stratification Example

```
human(john).
male(john).
plays_the_piano(john).
has_hobbies(X) ← plays_the_piano(X).          stratum 1
has_child(X) ← human(X), not has_hobbies(X).  stratum 2
married(X) ← human(X), has_child(X).          stratum 2
bachelor(X) ← male(X), not married(X).        stratum 3
```

Evaluation:

- $M_0 :=$ `human(john), male(john), plays_the_piano(john)`
- stratum 1: $M_1 := \mathbf{T}^{\omega}_{S_1}(M_0) = M_0 \cup$ {`has_hobbies(john)`}.
- stratum 2: $M_2 := \mathbf{T}^{\omega}_{S_2}(M_1) = M_1 \cup \emptyset$
- stratum 3: $M_3 := \mathbf{T}^{\omega}_{S_3}(M_2) = M_2 \cup$ {`bachelor(john)`}.

Stratification Example

```
human(john).
male(john).
plays_the_piano(john).
has_hobbies(X) ← plays_the_piano(X).          stratum 1
has_child(X) ← human(X), not has_hobbies(X).  stratum 2
married(X) ← human(X), has_child(X).          stratum 2
bachelor(X) ← male(X), not married(X).        stratum 3
```

Evaluation:

- $M_0 :=$ `human(john), male(john), plays_the_piano(john)`
- stratum 1: $M_1 := \mathbf{T}^{\omega}_{S_1}(M_0) = M_0 \cup$ {`has_hobbies(john)`}.
- stratum 2: $M_2 := \mathbf{T}^{\omega}_{S_2}(M_1) = M_1 \cup \emptyset$
- stratum 3: $M_3 := \mathbf{T}^{\omega}_{S_3}(M_2) = M_2 \cup$ {`bachelor(john)`}.

Stratification Example

```
human(john).
male(john).
plays_the_piano(john).
has_hobbies(X) ← plays_the_piano(X).          stratum 1
has_child(X) ← human(X), not has_hobbies(X).  stratum 2
married(X) ← human(X), has_child(X).          stratum 2
bachelor(X) ← male(X), not married(X).        stratum 3
```

Evaluation:

- $M_0 :=$ `human(john), male(john), plays_the_piano(john)`
- stratum 1: $M_1 := \mathbf{T}^{\omega}_{S_1}(M_0) = M_0 \cup$ {`has_hobbies(john)`}.
- stratum 2: $M_2 := \mathbf{T}^{\omega}_{S_2}(M_1) = M_1 \cup \emptyset$
- stratum 3: $M_3 := \mathbf{T}^{\omega}_{S_3}(M_2) = M_2 \cup$ {`bachelor(john)`}.

7.2 Magic Set Transformation, Stratified

Problem with the magic set transformation for programs with negation:

- The Magic Set Transformation of stratified programs may have unstratified outcome.

Causes for unstratification of the MST:

- positive and negative occurrence of a literal in a rule body
- multiple negative occurrences of a literal in a rule body
- negative literal in a recursive rule

Solution:

- distinction of contexts of problematic atoms

7.2 Magic Set Transformation, Stratified

Problem with the magic set transformation for programs with negation:

- The Magic Set Transformation of stratified programs may have unstratified outcome.

Causes for unstratification of the MST:

- positive and negative occurrence of a literal in a rule body
- multiple negative occurrences of a literal in a rule body
- negative literal in a recursive rule

Solution:

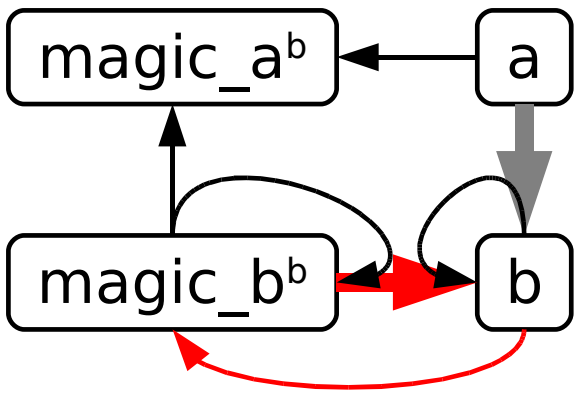
- distinction of contexts of problematic atoms

7.2 Magic Set Transformation, Stratified

```
a(x) ← not b(x), c(x,y), b(y).
b(x) ← c(x,y), b(y).
```

```
magic_a^b(1).
magic_b^b(x) ← magic_a^b(x)
magic_b^b(y) ←
   magic_a^b(x), not b(x), c(x,y).
a(x) ←
   magic_a^b(x), not b(x), c(x,y), b(y).
magic_b^b(y) ← magic_b^b(x), c(x,y).
b(x) ← magic_b^b(x), c(x,y), b(y).
```

- b occurs both negatively and positively in the first rule.



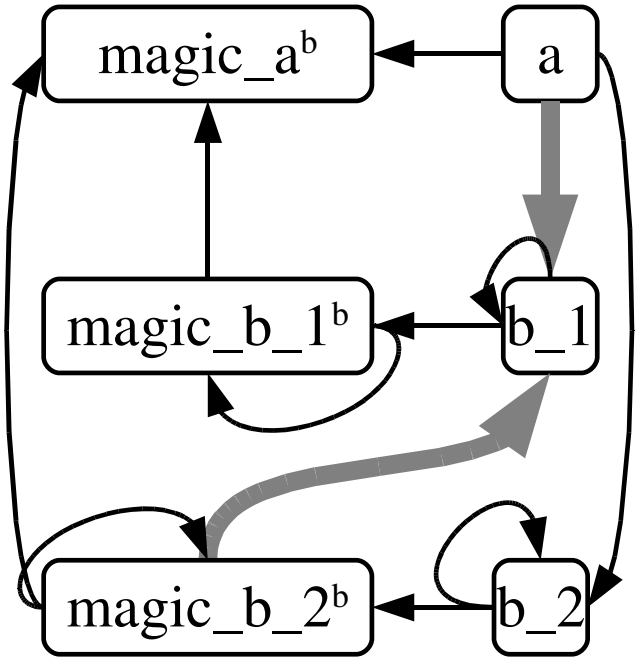
- Resulting program unstratifiable!

7.2 Magic Set Transformation, Stratified

```
a(x) ← not b_1(x), c(x,y), b_2(y).
b_1(x) ← c(x,y), b_1(y).
b_2(x) ← c(x,y), b_2(y).
```

```
magic_a^b(1).
magic_b_1^b(x) ← magic_a^b(x).
magic_b_2^b(y) ←
   magic_a^b(x), not b_1(x), c(x,y).
magic_b^b(y) ←
   magic_a^b(x), not b(x), c(x,y).
a(x) ←
   magic_a^b(x), not b(x), c(x,y), b(y).
magic_b^b(y) ← magic_b^b(x), c(x,y).
b(x) ← magic_b^b(x), c(x,y), b(y).
```

- Context labelling of predicates
- Rule replication



- Result is stratifiable!

7.2 Magic Set Transformation, Stratified: Exercise

- Consider the Program in Listing → p.99, Lst.7_17.
- Assume the facts `c(1,2)`, `b(2)`, `c(0,1)`.
- Consider the query `?- a(1)`.
- Compare the results of the forward chaining evaluation of the original program with the one of the rewritten program. → p.100, Lst.7_20

7.3 Stable Model Semantics: Recapitulation

```
married(john, mary).
male(X) ←married(X,Y), not male(Y)
```

Gelfond Lifschitz transformation with respect to the set
$M = \{\texttt{married(john, mary)}, \texttt{male(John)}\}$

```
married(john, mary).
male(john) ← married(john, mary), not male(mary).
male(mary) ← married(mary, john), not male(john).
male(mary) ← married(mary, mary), not male(mary).
male(john) ← married(john, john), not male(john).
```

7.3 Stable Model Semantics: An algorithm

Ideas:

- construction of *full sets* from a set of *negative antecedents*
- backtracking generates all possible candidates for full sets
- use of heuristics to limit the search space
- straight-forward derivation of stable models from full sets

7.3 Stable Model Semantics: An algorithm

- Negative antecedents $NAnt(P)$: → 7.4 p.101, Def.217
- Reduct $R(P, L)$, Deductive closure $DCL(P, L)$: → 7.4 p.101, Def.218
- Full sets: → 7.4 p.101, Def.219

Example: $P := \{q \leftarrow \neg r; q \leftarrow \neg p, r \leftarrow q\}$

- Negative antecedents of P: { r, p }
- Reduct of P with respect to $L := \{p, \neg p\}$: $\{q \leftarrow; r \leftarrow q\}$
- Deductive closure $Dcl(P, L) = \{p, q, r\}$
- First try: $\Lambda_1 := \{\neg p, \neg r\}$
  - Reduct $R(P, \Lambda_1) = \{q; r \leftarrow q\}$
  - Deductive closure $Dcl(P, \Lambda_1) = \{q, r\}$. Λ_1 not a full set since $r \in Dcl(P, \Lambda_1)$ and $\neg r \in \Lambda_1$.
- Second try: $\Lambda_2 := \{\neg p\}$
  - Reduct $R(P, \Lambda_2) = \{q; r \leftarrow q\}$
  - Deductive closure $Dcl(P, \Lambda_2) = \{q, r\}$. Hence Λ_2 is a full set for P.

7.3 Stable Model Semantics: An algorithm

full sets vs. stable models → 7.4 p.102, Def.221

- P: ground program
- Λ: set of negative literals
- if Λ is a full set wrt P then $Dcl(P, \Lambda)$ is a stable model of P
- if Δ is a stable model of P, then $\Lambda = not(NAnt(P) - \Delta)$ is a full set wrt P such that $Dcl(P, \Lambda) = \Delta$.

Problem: Full sets are still guessed!

7.3 Stable Model Semantics: An algorithm

```
function stable_model(P,B,φ)
  let B' = expand(P,B) in
    if conflict(P,B') then false
    else
      if (B' covers NAnt(P)) then test(Dcl(P,B'),φ)
      else
        take some χ ∈ NAnt(P) not covered by B'
        if stable_model(P, B'∪{not(χ)}, φ) then true
           else stable_model(P, B'∪{χ}, φ)
```

- Good choice for expand(P,B) is the least fix point of the Fitting operator

→ 7.4 p.102, Def.223

7.5 Well-Founded Model Semantics

Recall:

- unfounded set → 5.3.3 p.63, Def.176
- union of unfounded sets is unfounded
- existence of unique maximal unfounded set for any program
- monotonic operators $\mathbf{T}_S$, $\mathbf{U}_S$, $\mathbf{W}_S$
- well-founded semantics may be partial or total → 5.3.3 p.64, Def.181

Problem: unfounded sets must be guessed!

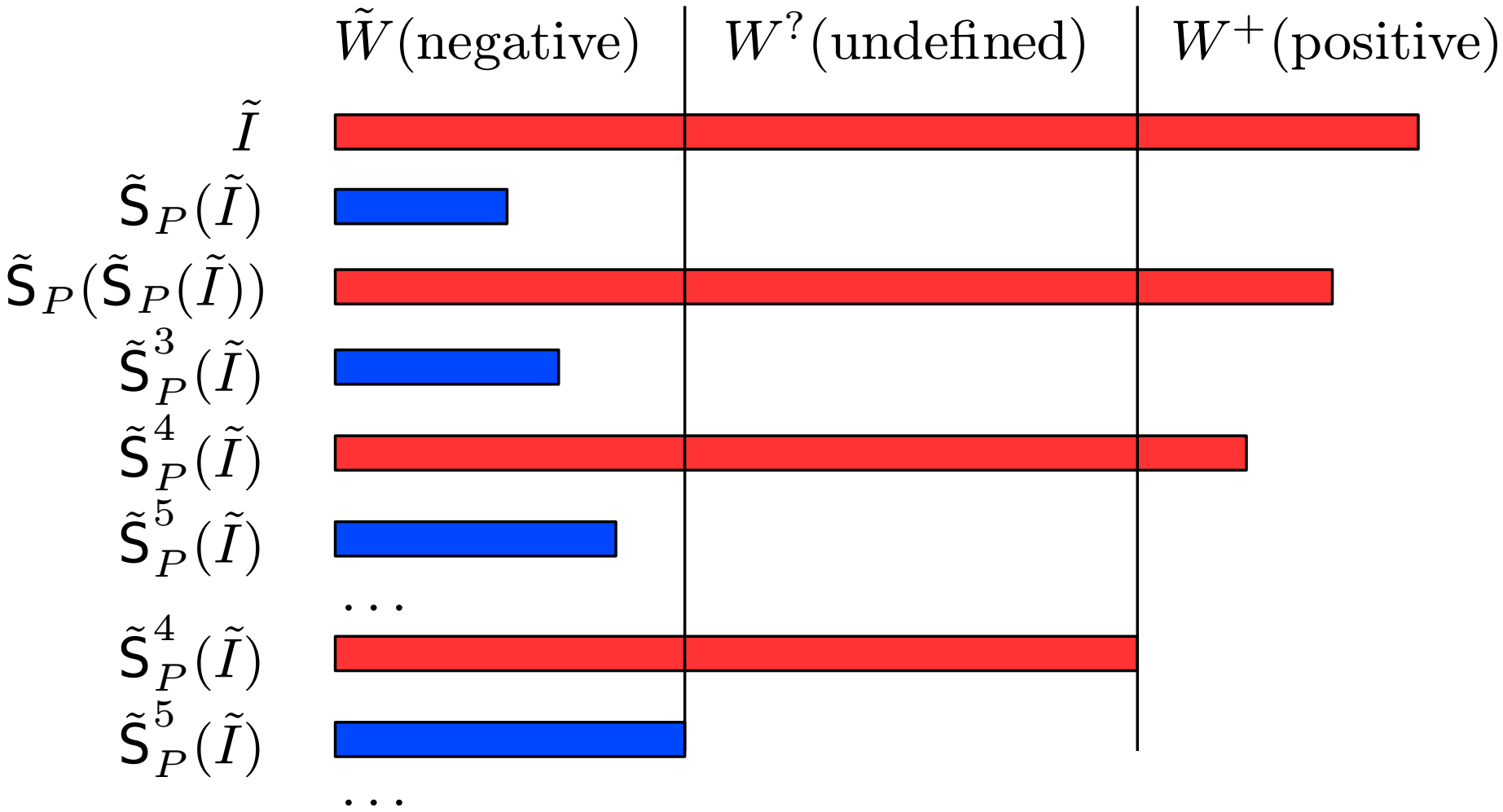
A possible solution: *Alternating Fixpoint Procedure*

The Alternating Fixpoint Procedure

Ideas:

- P_H : Herbrand instantiation of a Program P
- $\mathbf{T}_P(I)$: immediate consequence operator
- $\tilde{I}$: complement of the set of literals "known to be false"
- iteratively build up a set of negative conclusions $\tilde{A}$
- straight-forward derivation of positive conclusions from $\tilde{A}$ at the end
- nested fixpoint calculation
- each iteration is a two-phase process:
  1. Transformation of an underestimate $\tilde{I}$ of negative conclusions into a temporary overestimate $\tilde{\mathbf{S}}_P(\tilde{I}) := \overline{\mathbf{S}_P(\tilde{I})} := \neg \cdot (H - \mathbf{S}_P(\tilde{I}))$.
  2. Transformation of the overestimate back to an underestimate $\mathbf{A}_P(\tilde{I}) := \tilde{\mathbf{S}}_P(\tilde{\mathbf{S}}_P(\tilde{I}))$

Alternating Fixpoint Procedure



Alternating Fixpoint Procedure: Example

$a \leftarrow c, \neg b.$
$b \leftarrow \neg a.$
$c.$

$p \leftarrow q, \neg s.$
$p \leftarrow r, \neg s.$
$p \leftarrow t.$
$q \leftarrow p.$
$r \leftarrow q.$
$r \leftarrow \neg c.$

- $H = \{a, b, c, p, q, r, s, t\}$
- $\tilde{I}_0 = \emptyset$
- $\mathbf{S}_P(\emptyset) = \{c\}$
- $\tilde{I}_1 = \tilde{\mathbf{S}}_P(\emptyset) = \{\neg a, \neg b, \neg p, \neg q, \neg r, \neg s, \neg t\}$
- $\mathbf{S}_P(\tilde{I}_1) = \{c, a, b\}$
- $\tilde{I}_2 = \tilde{\mathbf{S}}_P(\tilde{I}_1) = \{\neg p, \neg q, \neg r, \neg s, \neg t\}$
- $\tilde{I}_3 = \tilde{I}_1$ and $\tilde{I}_4 = \tilde{I}_2$. Fixpoint reached!
- Well founded partial model is $\{c, \neg p, \neg q, \neg r, \neg s, \neg t\}$

7.7 Other Query Answering Methods for Negation

- SLD resolution with negation as failure – SLDNF (completion semantics)
- SLS resolution (perfect model semantics)
- global SLS resolution (well founded model semantics)

Session 16:30 – 18:00

- 8 Complexity and Expressive Power of Logic Programming Formalisms

The Story So far

- Query languages with the form of logics
- Syntax, declarative and operational semantics

- How much resource (time, space) do we need for the computation of these semantics? $\Rightarrow$ Complexity

- What kind of properties can a given query language express?
- Is Q_1 more expressive than Q_2? $\Rightarrow$ Expressive power

A dream query language should have:

- lower complexity, and
- more expressive power

The Story So far

- Query languages with the form of logics
- Syntax, declarative and operational semantics
- How much resource (time, space) do we need for the computation of these semantics? $\Rightarrow$ Complexity
- What kind of properties can a given query language express?
- Is Q_1 more expressive than Q_2? $\Rightarrow$ Expressive power

A dream query language should have:

- lower complexity, and
- more expressive power

The Story So far

- Query languages with the form of logics
- Syntax, declarative and operational semantics
- How much resource (time, space) do we need for the computation of these semantics? $\Rightarrow$ Complexity
- What kind of properties can a given query language express?
- Is Q_1 more expressive than Q_2? $\Rightarrow$ Expressive power

A dream query language should have:

- lower complexity, and
- more expressive power

The Story So far

- Query languages with the form of logics
- Syntax, declarative and operational semantics

- How much resource (time, space) do we need for the computation of these semantics? $\Rightarrow$ Complexity

- What kind of properties can a given query language express?
- Is Q_1 more expressive than Q_2? $\Rightarrow$ Expressive power

A dream query language should have:

- lower complexity, and
- more expressive power

The Story So far

- Query languages with the form of logics
- Syntax, declarative and operational semantics

- How much resource (time, space) do we need for the computation of these semantics? $\Rightarrow$ Complexity

- What kind of properties can a given query language express?
- Is Q_1 more expressive than Q_2? $\Rightarrow$ Expressive power

A dream query language should have:

- lower complexity, and
- more expressive power

The Story So far

- Query languages with the form of logics
- Syntax, declarative and operational semantics

- How much resource (time, space) do we need for the computation of these semantics? $\Rightarrow$ Complexity

- What kind of properties can a given query language express?
- Is Q_1 more expressive than Q_2? $\Rightarrow$ Expressive power

A dream query language should have:

- lower complexity, and
- more expressive power

The Results Overview

Query	Data Complexity	Program Complexity
Conjunctive query	AC_0	NP-complete
FO	AC_0	PSPACE-complete
Prop. LP		P-complete
Datalog	P-complete	EXPTIME-complete
Stratified Datalog	P-complete	EXPTIME-complete
Datalog(WFM)	P-complete	EXPTIME-complete
Datalog(INF)	P-complete	EXPTIME-complete
Datalog(Stable Model)	co-NP-complete	co-NEXPTIME-complete
Disjun. Datalog	Π_2^p-complete	co-NEXPTIME$^{\text{NP}}$-complete

The Results Overview

Today we shall concentrate on

Query	Data Complexity	Program Complexity
Conjunctive query	AC_0	NP-complete
FO	AC_0	PSPACE-complete
Prop. LP		P-complete
Datalog	P-complete	EXPTIME-complete
Stratified Datalog	P-complete	EXPTIME-complete
Datalog(WFM)	P-complete	EXPTIME-complete
Datalog(INF)	P-complete	EXPTIME-complete
Datalog(Stable Model)	co-NP-complete	co-NEXPTIME-complete
Disjun. Datalog	Π_2^p-complete	co-NEXPTIME$^{\text{NP}}$-complete

The Goal of this Lecture

- Basic concept of Turing machine, reduction, data complexity and program complexity
- How to prove completeness, Logspace reduction
- Get a taste of the hardness proofs of logic programming via nice encoding of a Turing machine
- Learn basics about expressive power

Decision Problems

- Problems where the answer is "yes" or "no"
- Formally,
  - A language L over some alphabet Σ.
  - An *instance* is given as a word $x \in \Sigma^*$.
  - Question: whether $x \in L$ holds
- The resources (i.e., either time or space) required in the worst case to find the correct answer for any instance x of a problem L is referred to as the *complexity* of the problem L

Complexities

Let P be a program with some query language, D_{in} input database and A a ground atom.

- **data complexity**
  Let P be *fixed*

  Instance. D_{in} and A.
  Question. Does $D_{in} \cup P \models A$ hold?
- **program complexity** (a.k.a. **expression complexity**)
  Let D_{in} be *fixed*.

  Instance. P and A.
  Question. Does $D_{in} \cup P \models A$ hold?
- **combined complexity**

  Instance. P, D_{in} and A.
  Question. Does $D_{in} \cup P \models A$ hold?

Complexity classes

$$\mathsf{L} \subseteq \mathsf{NL} \subseteq \mathsf{P} \subseteq \mathsf{NP} \subseteq \mathsf{PSPACE} \subseteq \mathsf{EXPTIME} \subseteq \mathsf{NEXPTIME}$$

These are the classes of problems which can be solved in

- logarithmic space (L),
- non-deterministic logarithmic space (NL),
- polynomial time (P),
- non-deterministic polynomial time (NP),
- polynomial space (PSPACE),
- exponential time (EXPTIME), and
- non-deterministic exponential time (NEXPTIME).

we shall encounter in this course: P, NP, PSPACE, EXPTIME

Complexity classes

$$L \subseteq NL \subseteq P \subseteq NP \subseteq PSPACE \subseteq EXPTIME \subseteq NEXPTIME$$

These are the classes of problems which can be solved in

- logarithmic space (L),
- non-deterministic logarithmic space (NL),
- polynomial time (P),
- non-deterministic polynomial time (NP),
- polynomial space (PSPACE),
- exponential time (EXPTIME), and
- non-deterministic exponential time (NEXPTIME).

we shall encounter in this course: P, NP, PSPACE, EXPTIME

Complexity classes – co Problems

- Any complexity class $\mathcal{C}$ has its *complementary class* denoted by co-$\mathcal{C}$
- For every language $L \subseteq \Sigma^*$, let $\overline{L}$ denote its *complement*, i.e. the set $\Sigma^* \setminus L$. Then co-$\mathcal{C}$ is $\{\overline{L} \mid L \in \mathcal{C}\}$.
- Every deterministic complexity class is closed under complement, because one can simply add a last step to the algorithm which reverses the answer. (co-P?)

Complexity classes – Reductions

- **Logspace Reduction**
  - Let L_1 and L_2 be decision problems (languages over some alphabet Σ).
  - $R : \Sigma^* \longrightarrow \Sigma^*$ be a function which can be computed in logarithmic space
  - The following property holds: for every $x \in \Sigma^*$, $x \in L_1$ iff $R(x) \in L_2$.
  - Then R is called a *logarithmic-space reduction* from L_1 to L_2 and we say that L_1 is *reducible* to L_2.
- **Hardness, Completeness**

  Let $\mathcal{C}$ be a set of languages. A language L is called *$\mathcal{C}$-hard* if any language L' in $\mathcal{C}$ is reducible to L. If L is $\mathcal{C}$-hard and $L \in \mathcal{C}$ then L is called *complete for $\mathcal{C}$* or simply *$\mathcal{C}$-complete*.

Turing machines

A *deterministic Turing machine (DTM)* is defined as a quadruple

$$(S, \Sigma, \delta, s_0)$$

- S is a finite set of *states*,
- Σ is a finite alphabet of *symbols*, which contains a special symbol $\sqcup$ called the *blank*.
- δ is a *transition function*,
- and $s_0 \in S$ is the *initial state*.

The transition function δ is a map

$$\delta : \quad S \times \Sigma \quad \rightarrow \quad (S \cup \{\texttt{yes}, \texttt{no}\}) \times \Sigma \times \{\texttt{-1}, \ \texttt{0}, \ \texttt{+1}\},$$

where `yes`, and `no` denote two additional states not occurring in S, and `-1`, `0`, `+1` denote *motion directions*.

Turing machines

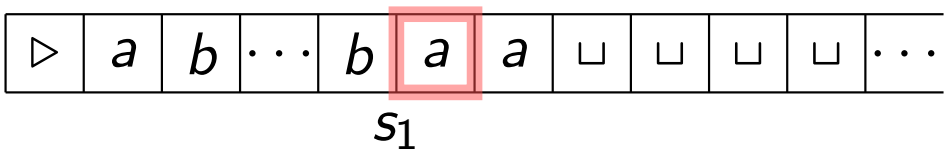
DTM quadruple:

$$(\Sigma, S, \delta, s_0)$$

Transition function:

$$\delta(s, \sigma) = (s', \sigma', d).$$

The tape of the TM



Turing machines

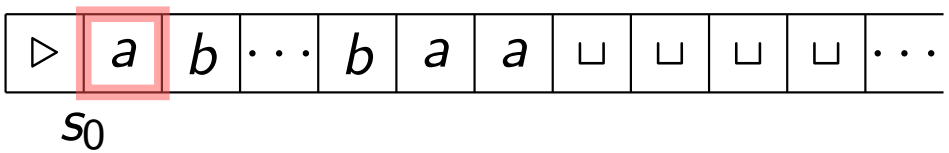
DTM quadruple:

$$(\Sigma, S, \delta, s_0)$$

Transition function:

$$\delta(s, \sigma) = (s', \sigma', d).$$

The tape of the TM



Turing machines

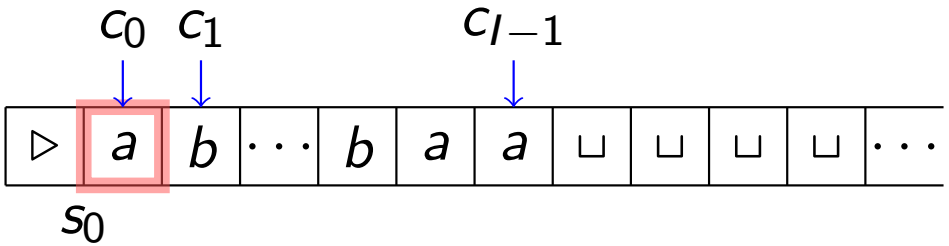
DTM quadruple:

$$(\Sigma, S, \delta, s_0)$$

Transition function:

$$\delta(s, \sigma) = (s', \sigma', d).$$

The tape of the TM



Turing machines

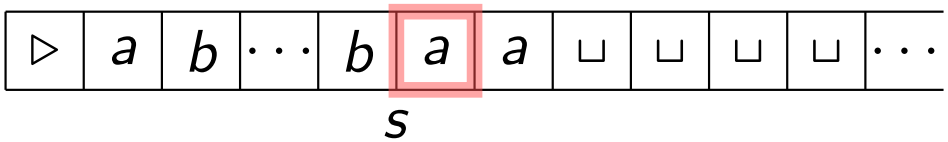
DTM quadruple:

$$(\Sigma, S, \delta, s_0)$$

Transition function:

$$\delta(s, \sigma) = (s', \sigma', d).$$

The tape of the TM



Transition Function example:

$$\delta(s, a) = (s', b, -1)$$

Turing machines

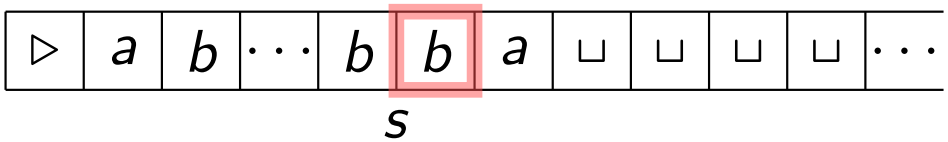
DTM quadruple:

$$(\Sigma, S, \delta, s_0)$$

Transition function:

$$\delta(s, \sigma) = (s', \sigma', d).$$

The tape of the TM



Transition Function example:

$$\delta(s, a) = (s', b, -1)$$

Turing machines

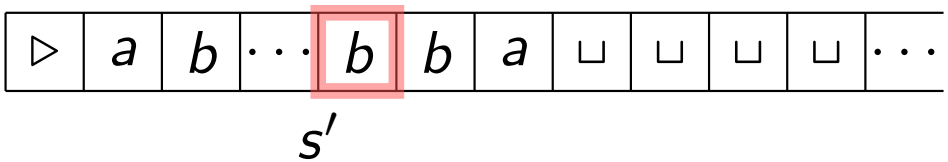
DTM quadruple:

$$(\Sigma, S, \delta, s_0)$$

Transition function:

$$\delta(s, \sigma) = (s', \sigma', d).$$

The tape of the TM



Transition Function example:

$$\delta(s, a) = (s', b, -1)$$

Turing machines

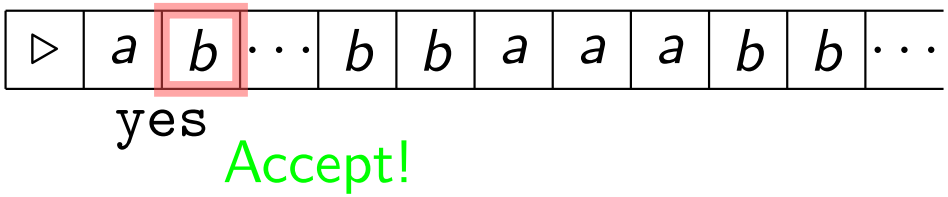
DTM quadruple:

$$(\Sigma, S, \delta, s_0)$$

Transition function:

$$\delta(s, \sigma) = (s', \sigma', d).$$

The tape of the TM



T halts, when any of the states `yes` or `no` is reached

Turing machines

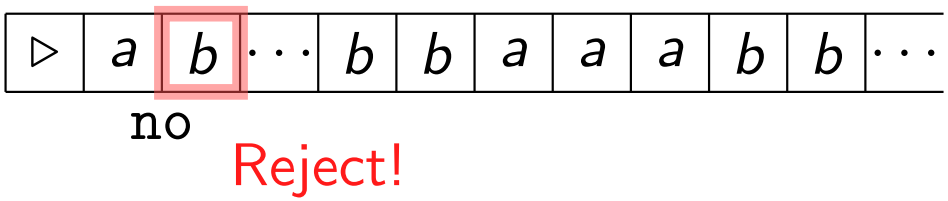
DTM quadruple:

$$(\Sigma, S, \delta, s_0)$$

Transition function:

$$\delta(s, \sigma) = (s', \sigma', d).$$

The tape of the TM



T halts, when any of the states `yes` or `no` is reached

NDTM

A *non-deterministic Turing machine (NDTM)* is defined as a quadruple

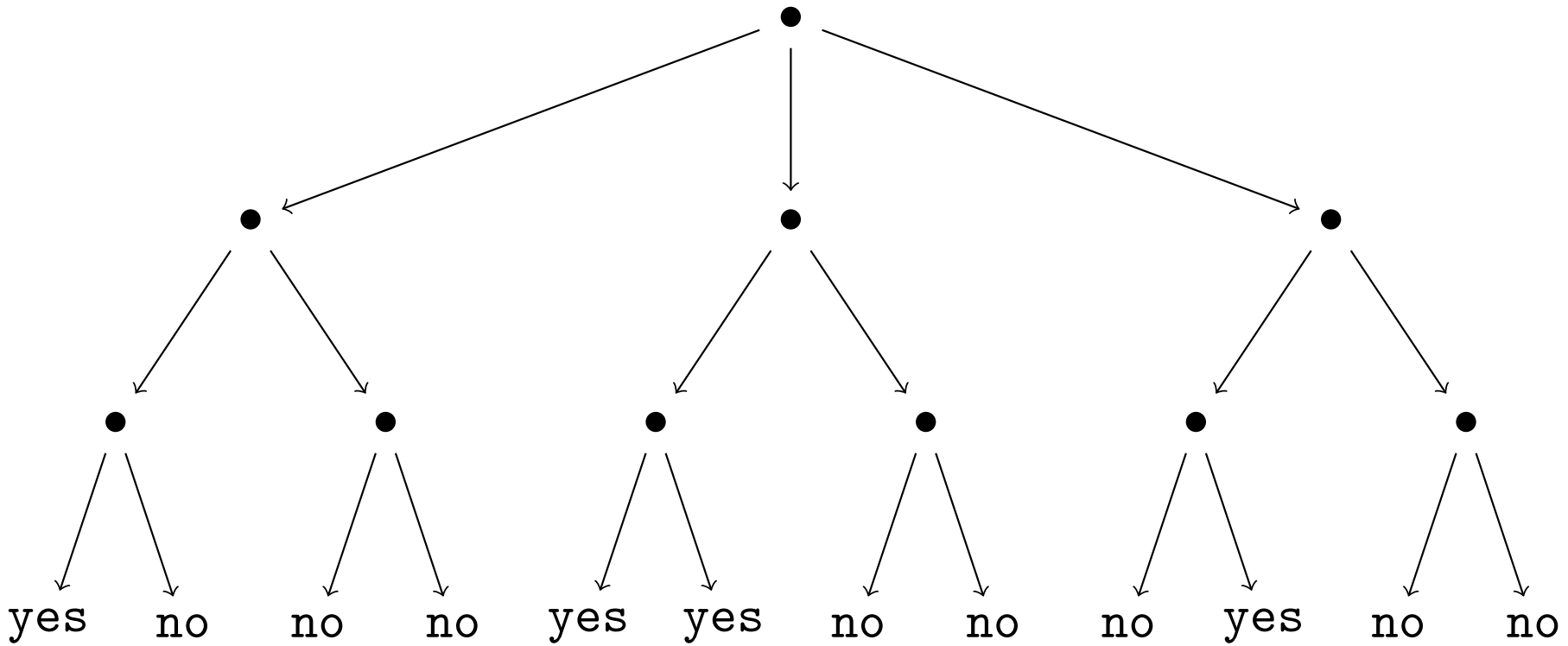
$$(S, \Sigma, \Delta, s_0)$$

- S, Σ, s_0 are the same as DTM
- Δ is no longer a function, but a relation:

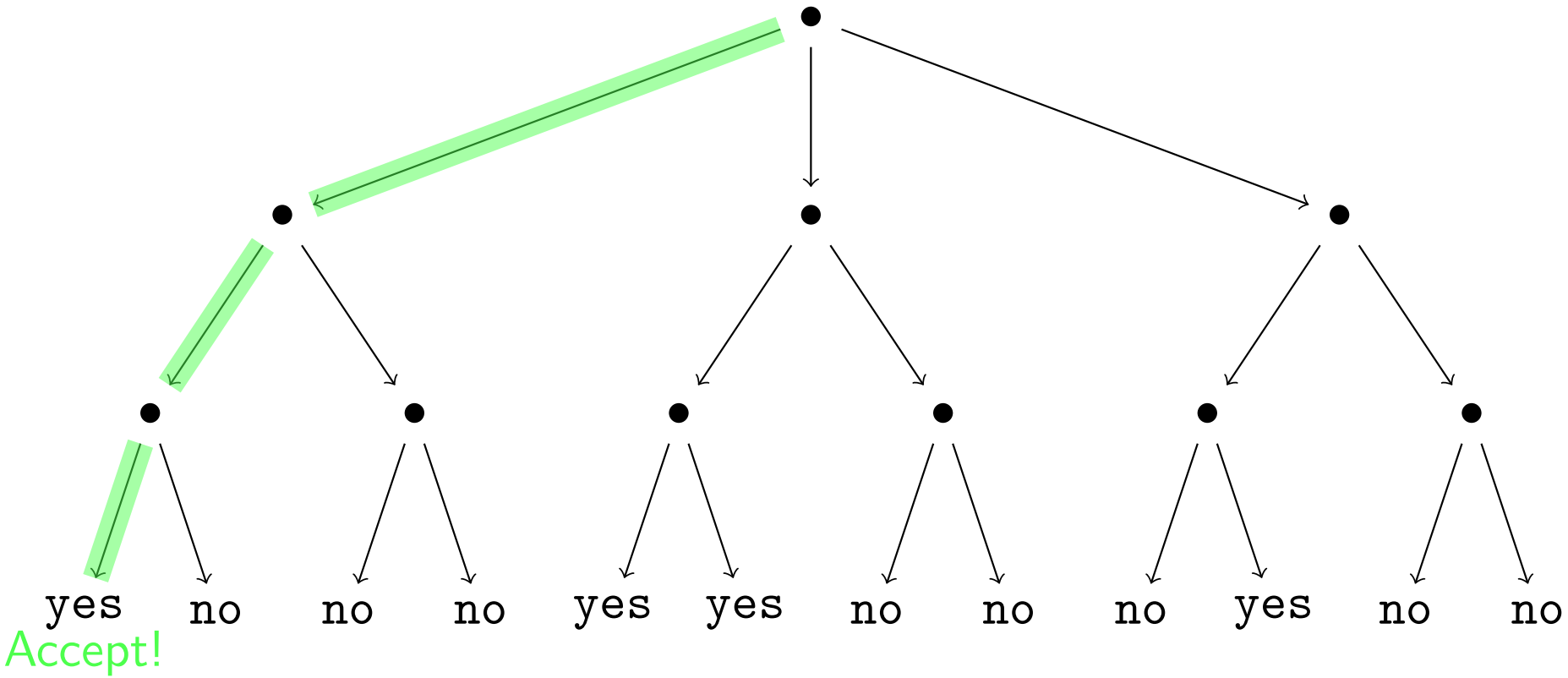
$$\Delta \subseteq (S \times \Sigma) \times (S \cup \{\texttt{yes}, \texttt{no}\}) \times \Sigma \times \{\texttt{-1}, \texttt{ 0}, \texttt{ +1}\}.$$

- A tuple with s and σ. If the number of such tuples is greater than one, the NDTM non-deterministically chooses any of them and operates accordingly.
- Unlike the case of a DTM, the definition of acceptance and rejection by a NDTM is asymmetric.

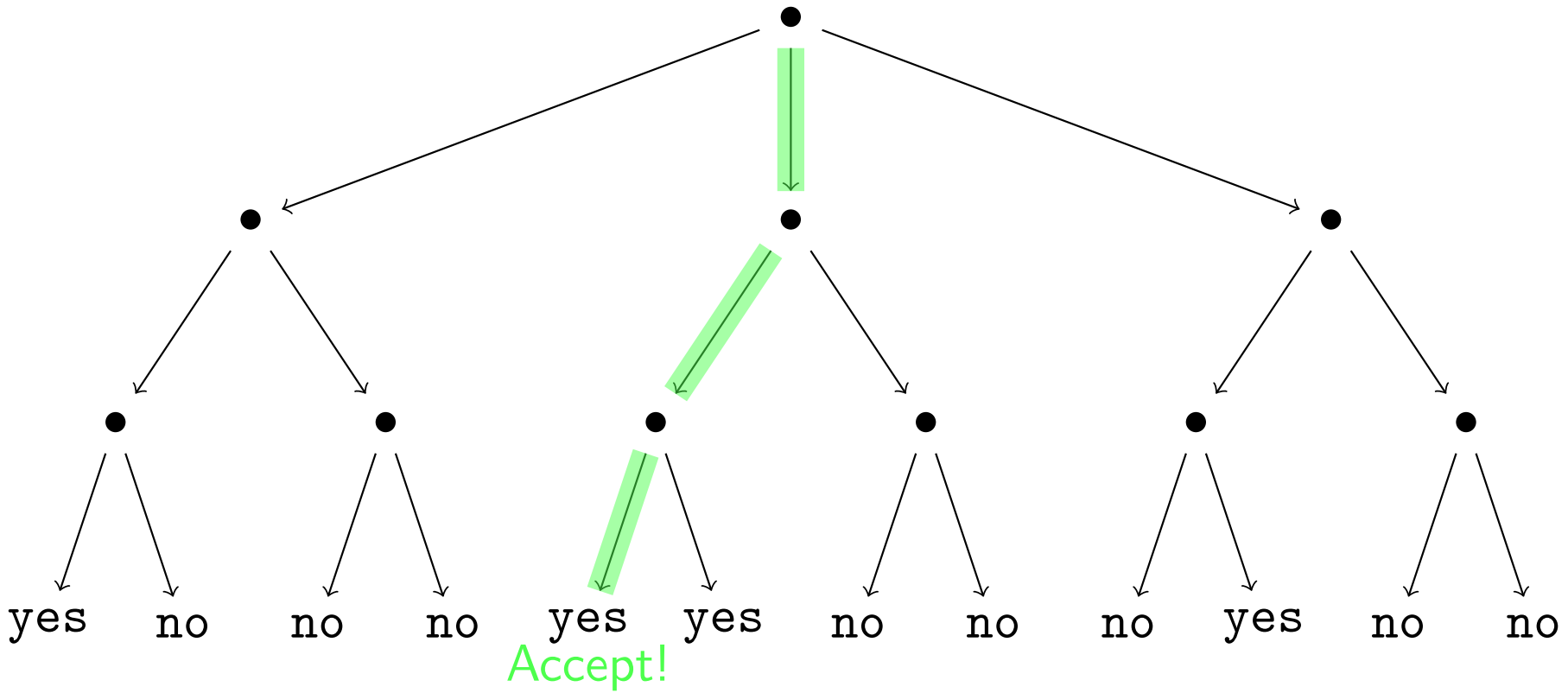
Nondeterministic Computation (Accept)



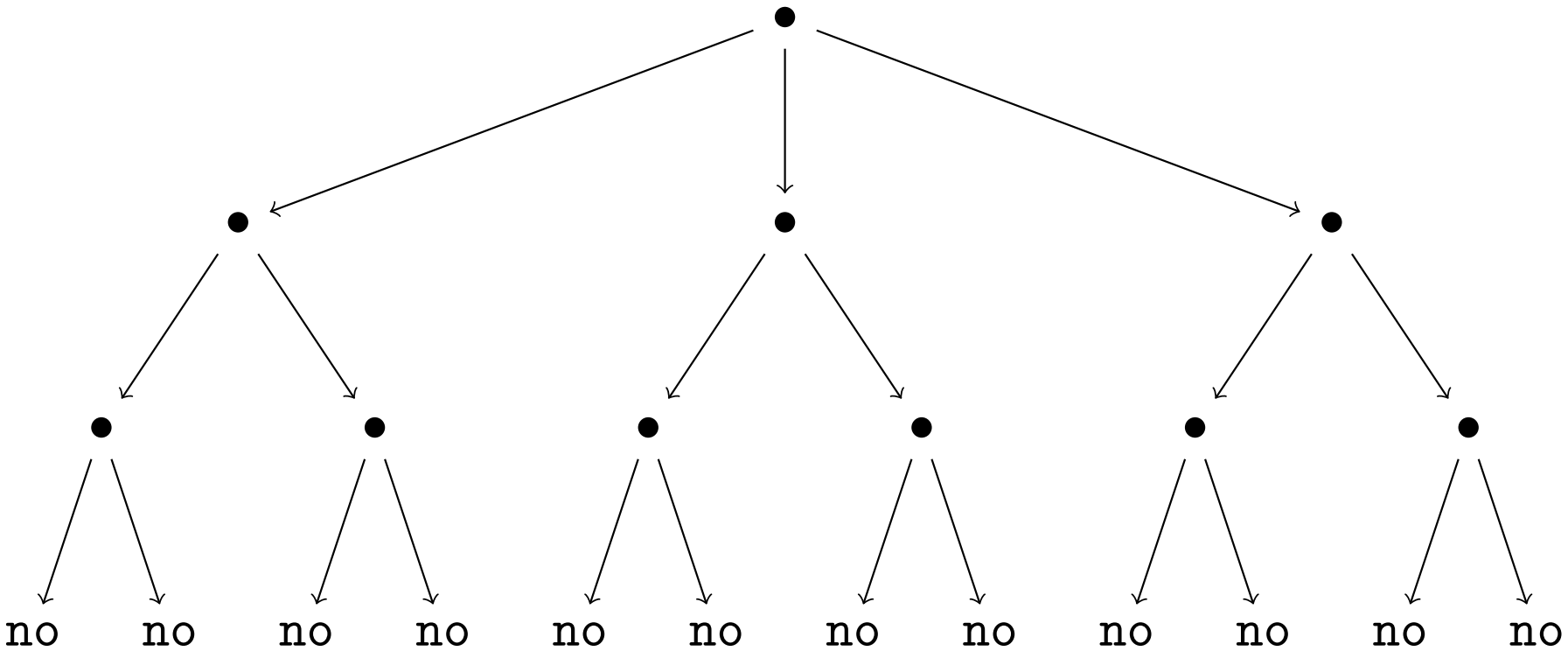
Nondeterministic Computation (Accept)



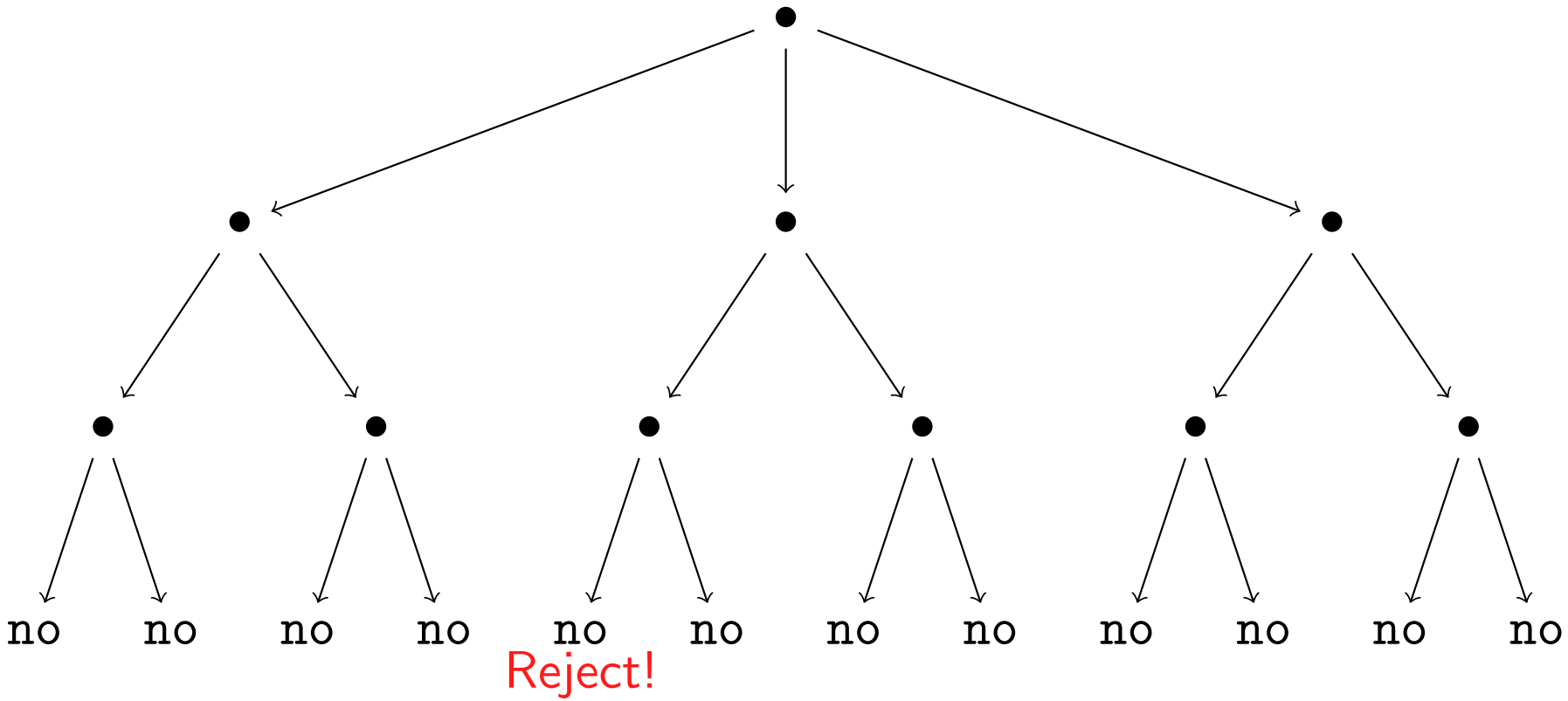
Nondeterministic Computation (Accept)



Nondeterministic Computation (Rejection)



Nondeterministic Computation (Rejection)



8.2 Propositional Logic Programming

Today we shall concentrate on

Query	Data Complexity	Program Complexity
Conjunctive query	AC_0	NP-complete
FO	AC_0	PSPACE-complete
Prop. LP		P-complete
Datalog	P-complete	EXPTIME-complete
Stratified Datalog	P-complete	EXPTIME-complete
Datalog(WFM)	P-complete	EXPTIME-complete
Datalog(INF)	P-complete	EXPTIME-complete
Datalog(Stable Model)	co-NP-complete	co-NEXPTIME-complete
Disjun. Datalog	Π_2^p-complete	co-NEXPTIME$^{\text{NP}}$-complete

Propositional LP

Theorem

Propositional logic programming is P*-complete.* → *8.2 p.110, Thm.224*

Proof: (Membership)

- The semantics of a given program P can be defined as the least fixpoint of the immediate consequence operator $\mathbf{T}_P$
- This least fixpoint $lfp(\mathbf{T}_P)$ can be computed in polynomial time even if the "naive" evaluation algorithm is applied.
- The number of iterations (i.e. applications of $\mathbf{T}_P$) is bounded by the number of rules plus 1.
- Each iteration step is clearly feasible in polynomial time.

Propositional LP P-hardness Proof

Proof: (Hardness)

- Encoding of a a deterministic Turing machine (DTM) T. Given a DTM T, an input string I and a number of steps N, where N is a polynomial of $|I|$, construct in logspace a program $P = P(T, I, N)$. An atom A such as $P \models A$ iff T accepts I in N steps.
- The transition function δ of a DTM with a single tape can be represented by a table whose rows are tuples $t = \langle s, \sigma, s', \sigma', d \rangle$. Such a tuple t expresses the following if-then-rule:

  if at some time instant τ the DTM is in state s, the cursor points to cell number π, and this cell contains symbol σ
  then at instant $\tau + 1$ the DTM is in state s', cell number π contains symbol σ', and the cursor points to cell number $\pi + d$.

Propositional LP P-hardness Proof

Proof: (Hardness)

- Encoding of a a deterministic Turing machine (DTM) T. Given a DTM T, an input string I and a number of steps N, where N is a polynomial of $|I|$, construct in logspace a program $P = P(T, I, N)$. An atom A such as $P \models A$ iff T accepts I in N steps.
- The transition function δ of a DTM with a single tape can be represented by a table whose rows are tuples $t = \langle s, \sigma, s', \sigma', d \rangle$. Such a tuple t expresses the following if-then-rule:

  if at some time instant τ the DTM is in state s, the cursor points to cell number π, and this cell contains symbol σ
  then at instant $\tau + 1$ the DTM is in state s', cell number π contains symbol σ', and the cursor points to cell number $\pi + d$.

Propositional LP P-hardness Proof

Proof: (Hardness)

- Encoding of a a deterministic Turing machine (DTM) T. Given a DTM T, an input string I and a number of steps N, where N is a polynomial of $|I|$, construct in logspace a program $P = P(T, I, N)$. An atom A such as $P \models A$ iff T accepts I in N steps.
- The transition function δ of a DTM with a single tape can be represented by a table whose rows are tuples $t = \langle s, \sigma, s', \sigma', d \rangle$. Such a tuple t expresses the following if-then-rule:

  if at some time instant τ the DTM is in state s, the cursor points to cell number π, and this cell contains symbol σ
  then at instant $\tau + 1$ the DTM is in state s', cell number π contains symbol σ', and the cursor points to cell number $\pi + d$.

Propositional LP P-hardness: the atoms

The propositional atoms in $P(T, I, N)$.
(there are many, but only polynomially many...)

symbol$_\alpha[\tau, \pi]$ for $0 \leq \tau \leq N$, $0 \leq \pi \leq N$ and $\alpha \in \Sigma$. Intuitive meaning: at instant τ of the computation, cell number π contains symbol α.

cursor$[\tau, \pi]$ for $0 \leq \tau \leq N$ and $0 \leq \pi \leq N$. Intuitive meaning: at instant τ, the cursor points to cell number π.

state$_s[\tau]$ for $0 \leq \tau \leq N$ and $s \in S$. Intuitive meaning: at instant τ, the DTM T is in state s.

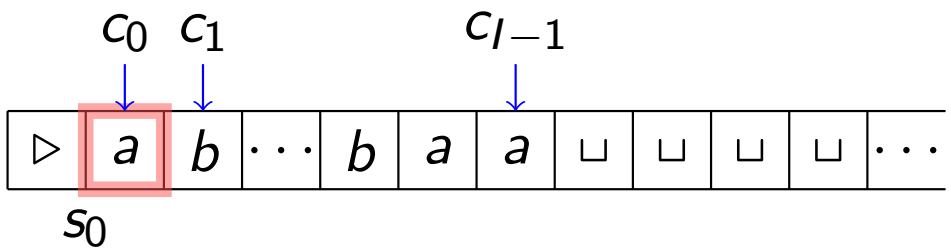
accept Intuitive meaning: T has reached state `yes`.

Propositional LP P-hardness: the rules

initialization facts: in $P(T, I, N)$:

$$
\begin{aligned}
symbol_{\sigma}[0, \pi] &\leftarrow && \text{for } 0 \leq \pi < |I|, \text{ where } I_{\pi} = \sigma \\
symbol_{\sqcup}[0, \pi] &\leftarrow && \text{for } |I| \leq \pi \leq N \\
cursor[0, 0] &\leftarrow \\
state_{s_0}[0] &\leftarrow
\end{aligned}
$$

The tape of the TM



Propositional LP P-hardness: the rules

- transition rules: for each entry $\langle s, \sigma, s', \sigma', d \rangle$, $0 \leq \tau < N$, $0 \leq \pi < N$, and $0 \leq \pi + d$.

$$\begin{aligned} \mathit{symbol}_{\sigma'}[\tau+1, \pi] &\leftarrow \mathit{state}_s[\tau], \mathit{symbol}_\sigma[\tau, \pi], \mathit{cursor}[\tau, \pi] \\ \mathit{cursor}[\tau+1, \pi+d] &\leftarrow \mathit{state}_s[\tau], \mathit{symbol}_\sigma[\tau, \pi], \mathit{cursor}[\tau, \pi] \\ \mathit{state}_{s'}[\tau+1] &\leftarrow \mathit{state}_s[\tau], \mathit{symbol}_\sigma[\tau, \pi], \mathit{cursor}[\tau, \pi] \end{aligned}$$

- inertia rules: where $0 \leq \tau < N$, $0 \leq \pi < \pi' \leq N$

$$\begin{aligned} \mathit{symbol}_\sigma[\tau+1, \pi] &\leftarrow \mathit{symbol}_\sigma[\tau, \pi], \mathit{cursor}[\tau, \pi'] \\ \mathit{symbol}_\sigma[\tau+1, \pi'] &\leftarrow \mathit{symbol}_\sigma[\tau, \pi'], \mathit{cursor}[\tau, \pi] \end{aligned}$$

- accept rules: for $0 \leq \tau \leq N$

$$\mathit{accept} \leftarrow \mathit{state}_{\mathtt{yes}}[\tau]$$

Propositional LP P-hardness

- The encoding precisely simulates the behaviour machine T on input I up to N steps. (This can be formally shown by induction on the time steps.)
- $P(T, I, N) \models \mathit{accept}$ iff the DTM T accepts the input string I within N steps.
- The construction is feasible in Logspace

Horn clause inference is P-complete

8.3 Datalog Complexity

Today we shall concentrate on

Query	Data Complexity	Program Complexity
Conjunctive query	AC_0	NP-complete
FO	AC_0	PSPACE-complete
Prop. LP		P-complete
Datalog	P-complete	EXPTIME-complete
Stratified Datalog	P-complete	EXPTIME-complete
Datalog(WFM)	P-complete	EXPTIME-complete
Datalog(INF)	P-complete	EXPTIME-complete
Datalog(Stable Model)	co-NP-complete	co-NEXPTIME-complete
Disjun. Datalog	Π_2^p-complete	co-NEXPTIME$^{\text{NP}}$-complete

Complexity of Datalog Programs – Data complexity

Theorem

Datalog is data complete for P. → *8.6 p.116, Thm.228*

Proof: (Membership)

Effective reduction to Propositional Logic Programming is possible. Given P, D, A:

- Generate $ground(P, D)$
- Decide whether $ground(P, D) \models A$

Grounding of Datalog Rules

- Let U_D be the universe of D (usually the active universe (domain), i.e., the set of all domain elements present in D).
- The grounding of a rule r, denoted $ground(r, D)$, is the set of all rules obtained from r by all possible uniform substitutions of elements of U_D for the variables in r.

For any datalog program P and database D,

$$ground(P, D) = \bigcup_{r \in P} ground(r, D).$$

Grounding example

P and D:

$parent(X, Y) \leftarrow father(X, Y) \quad parent(X, Y) \leftarrow mother(X, Y)$

$ancestor(X, Y) \leftarrow parent(X, Y)$

$ancestor(X, Y) \leftarrow parent(X, Z), ancestor(Z, Y)$

$father(john, mary), father(joe, kurt), mother(mary, joe), mother(tina, kurt)$

$ground(P, D)$:

$parent(john, john) \leftarrow father(john, john)$

$parent(john, john) \leftarrow father(john, marry)$

$\ldots$

$parent(john, john) \leftarrow mother(john, john)$

$parent(john, marry) \leftarrow mother(john, marry)$

$\ldots$

$ancestor(john, john) \leftarrow parent(john, john)$

$\ldots$

Grounding complexity

Given P, D, the number of rules in $ground(P, D)$ is bounded by

$$|P| * \#consts(D)^{vmax}$$

- $vmax(\geq 1)$ is the maximum number of different variables in any rule $r \in P$
- $\#consts(D) = |U_D|$ is the number of constants in D (ass.: $|U_D| > 0$).
- $ground(P \cup D)$ can be exponential in the size of P.
- $ground(P \cup D)$ is polynomial in the size of D.

hence, the complexity of propositional logic programming is an upper bound for the data complexity.

Datalog data complexity: hardness

Proof: Hardness The P-hardness can be shown by writing a simple datalog *meta-interpreter* for propositional $\mathrm{LP}(k)$, where k is a constant.

- Represent rules $A_0 \leftarrow A_1, \ldots, A_i$, where $0 \leq i \leq k$, by tuples $\langle A_0, \ldots, A_i \rangle$ in an $(i+1)$-ary relation R_i on the propositional atoms.
- Then, a program P in $\mathrm{LP}(k)$ which is stored this way in a database $D(P)$ can be evaluated by a fixed datalog program $P_{MI}(k)$ which contains for each relation R_i, $0 \leq i \leq k$, a rule

$$T(X_0) \leftarrow T(X_1), \ldots, T(X_i), R_i(X_0, \ldots, X_i).$$

- $T(x)$ intuitively means that atom x is true. Then, $P \models A$ just if $P_{MI} \cup P(D) \models T(A)$. P-hardness of the data complexity of datalog is then immediately obtained.

Program Complexity Datalog

Theorem

Datalog is program complete for EXPTIME. → *8.6 p.117, Thm.229*

- **Membership**. Grounding P on D leads to a propositional program $\mathit{grounding}(P, D)$ whose size is exponential in the size of the fixed input database D. Hence, the program complexity is in EXPTIME.
- **Hardness**.
  - Adapt the propositional program $P(T, I, N)$ deciding acceptance of input I for T within N steps, where $N = 2^m$, $m = n^k (n = |I|)$ to a datalog program $P_{dat}(T, I, N)$
  - Note: We can not simply generate $P(T, I, N)$, since this program is exponentially large (and thus the reduction would not be polynomial!)

Datalog Program Complexity: Hardness

Main ideas for lifting $P(T, I, N)$ to $P_{dat}(T, I, N)$:

- use the predicates $symbol_\sigma(X, Y)$, $cursor(X, Y)$ and $state_s(X)$ instead of the propositional letters $symbol_\sigma[X, Y]$, $cursor[X, Y]$ and $state_s[X]$ respectively.
- The time points τ and tape positions π from 0 to $N - 1$ are encoded in binary, i.e. by m-ary tuples $t_\tau = \langle c_1, \ldots, c_m \rangle$, $c_i \in \{0, 1\}$, $i = 1, \ldots, m$, such that $0 = \langle 0, \ldots, 0 \rangle$, $1 = \langle 0, \ldots, 1 \rangle$, $N - 1 = \langle 1, \ldots, 1 \rangle$.
- The functions $\tau + 1$ and $\pi + d$ are realized by means of the successor $Succ^m$ from a linear order $\leq^m$ on U^m.

Datalog Program Complexity: Hardness

The predicates $Succ^m$, $First^m$, and $Last^m$ are provided.

- The initialization facts $symbol_\sigma[0, \pi]$ are readily translated into the datalog rules

$$symbol_\sigma(\mathbf{X}, \mathbf{t}) \leftarrow First^m(\mathbf{X}),$$

  where $\mathbf{t}$ represents the position π,
- Similarly the facts $cursor[0, 0]$ and $state_{s_0}[0]$.
- Initialization facts $symbol_\sqcup[0, \pi]$, where $|I| \leq \pi \leq N$, are translated to the rule

$$symbol_\sqcup(\mathbf{X}, \mathbf{Y}) \leftarrow First^m(\mathbf{X}),\ \leq^m(\mathbf{t}, \mathbf{Y})$$

  where $\mathbf{t}$ represents the number $|I|$.

Datalog Program Complexity: Hardness

- Transition and inertia rules: for realizing $\tau + 1$ and $\pi + d$, use in the body atoms $Succ^m(\mathbf{X}, \mathbf{X}')$. For example, the clause

$$symbol_{\sigma'}[\tau + 1, \pi] \leftarrow state_s[\tau], symbol_\sigma[\tau, \pi], cursor[\tau, \pi]$$

  is translated into

$$symbol_{\sigma'}(\mathbf{X}', \mathbf{Y}) \leftarrow state_s(\mathbf{X}), symbol_\sigma(\mathbf{X}, \mathbf{Y}), cursor(\mathbf{X}, \mathbf{Y}), Succ^m(\mathbf{X}, \mathbf{X}')$$

- The translation of the accept rules is straightforward.

Defining $Succ^m(\mathbf{X}, \mathbf{X}')$ and $\leq^m$

- The ground facts $Succ^1(0,1)$, $First^1(0)$, and $Last^1(1)$ are provided.
- For an inductive definition, suppose $Succ^i(\mathbf{X}, \mathbf{Y})$, $First^i(\mathbf{X})$, and $Last^i(\mathbf{X})$ tell the successor, the first, and the last element from a linear order $\leq^i$ on U^i, where $\mathbf{X}$ and $\mathbf{Y}$ have arity i. Then, use rules

$$
\begin{aligned}
Succ^{i+1}(Z, \mathbf{X}, Z, \mathbf{Y}) &\leftarrow Succ^i(\mathbf{X}, \mathbf{Y}) \\
Succ^{i+1}(Z, \mathbf{X}, Z', \mathbf{Y}) &\leftarrow Succ^1(Z, Z'), Last^i(\mathbf{X}), First^i(\mathbf{Y}) \\
First^{i+1}(Z, \mathbf{X}) &\leftarrow First^1(Z), First^i(\mathbf{X}) \\
Last^{i+1}(Z, \mathbf{X}) &\leftarrow Last^1(Z), Last^i(\mathbf{X})
\end{aligned}
$$

Defining $Succ^m(\mathbf{X}, \mathbf{X}')$ and $\leq^m$

- The ground facts $Succ^1(0,1)$, $First^1(0)$, and $Last^1(1)$ are provided.
- For an inductive definition, suppose $Succ^i(\mathbf{X}, \mathbf{Y})$, $First^i(\mathbf{X})$, and $Last^i(\mathbf{X})$ tell the successor, the first, and the last element from a linear order $\leq^i$ on U^i, where $\mathbf{X}$ and $\mathbf{Y}$ have arity i. Then, use rules

$$
\begin{aligned}
Succ^{i+1}(0, \mathbf{X}, 0, \mathbf{Y}) &\leftarrow Succ^i(\mathbf{X}, \mathbf{Y}) \\
Succ^{i+1}(1, \mathbf{X}, 1, \mathbf{Y}) &\leftarrow Succ^i(\mathbf{X}, \mathbf{Y}) \\
Succ^{i+1}(0, \mathbf{X}, 1, \mathbf{Y}) &\leftarrow Last^i(\mathbf{X}), First^i(\mathbf{Y}) \\
First^{i+1}(0, \mathbf{X}) &\leftarrow First^i(\mathbf{X}) \\
Last^{i+1}(1, \mathbf{X}) &\leftarrow Last^i(\mathbf{X})
\end{aligned}
$$

Defining $Succ^m(\mathbf{X}, \mathbf{X}')$ and $\leq^m$

- The ground facts $Succ^1(0,1)$, $First^1(0)$, and $Last^1(1)$ are provided.
- For an inductive definition, suppose $Succ^i(\mathbf{X}, \mathbf{Y})$, $First^i(\mathbf{X})$, and $Last^i(\mathbf{X})$ tell the successor, the first, and the last element from a linear order $\leq^i$ on U^i, where $\mathbf{X}$ and $\mathbf{Y}$ have arity i. Then, use rules

$$\begin{aligned} Succ^{i+1}(0,\mathbf{X},0,\mathbf{Y}) &\leftarrow Succ^i(\mathbf{X},\mathbf{Y}) \\ Succ^{i+1}(1,\mathbf{X},1,\mathbf{Y}) &\leftarrow Succ^i(\mathbf{X},\mathbf{Y}) \\ Succ^{i+1}(0,\mathbf{X},1,\mathbf{Y}) &\leftarrow Last^i(\mathbf{X}), First^i(\mathbf{Y}) \\ First^{i+1}(0,\mathbf{X}) &\leftarrow First^i(\mathbf{X}) \\ Last^{i+1}(1,\mathbf{X}) &\leftarrow Last^i(\mathbf{X}) \end{aligned}$$

- The order $\leq^m$ is easily defined from $Succ^m$ by two clauses

$$\begin{aligned} \leq^m(\mathbf{X},\mathbf{X}) &\leftarrow \\ \leq^m(\mathbf{X},\mathbf{Y}) &\leftarrow Succ^m(\mathbf{X},\mathbf{Z}),\ \leq^m(\mathbf{Z},\mathbf{Y}) \end{aligned}$$

Datalog Program Complexity Conclusion

- Let $P_{dat}(T, I, N)$ denote the datalog program with empty *edb* described for T, I, and $N = 2^m$, $m = n^k$ (where $n = |I|$)
- $P_{dat}(T, I, N)$ is constructible from T and I in polynomial time (in fact, careful analysis shows feasibility in logarithmic space).
- $P_{dat}(T, I, N)$ has *accept* in its least model $\Leftrightarrow$ T accepts input I within N steps.
- Thus, the decision problem for any language in EXPTIME is reducible to deciding $P \models A$ for datalog program P and fact A.
- Consequently, deciding $P \models A$ for a given datalog program P and fact A is EXPTIME-hard.

Complexity of Datalog with Stratified Negation

Theorem

Stratified propositional logic programming with negation is P*-complete. Stratified datalog with negation is data complete for* P *and program complete for* EXPTIME. → *8.7 p.118, Thm.230*

- *stratified* P can be partitioned into disjoint sets $S_1, \ldots, S_n$ s.t. the semantics of P is computed by successively computing fixpoints of the immediate consequence operators $\mathbf{T}_{S_1}, \ldots, \mathbf{T}_{S_n}$.
- Let $\mathbf{I}_0$ be the initial instance over the extensional predicate symbols of P and let $\mathbf{I}_i$ (with $1 \leq i \leq n$) be defined as follows:

$$\mathbf{I}_1 := \mathbf{T}^{\omega}_{S_1}(\mathbf{I}_0), \quad \mathbf{I}_2 := \mathbf{T}^{\omega}_{S_2}(\mathbf{I}_1), \quad \ldots, \quad \mathbf{I}_n := \mathbf{T}^{\omega}_{S_n}(\mathbf{I}_{n-1})$$

  Then the semantics of program P is given through the set $\mathbf{I}_n$.
- In the propositional case, $\mathbf{I}_n$ is clearly polynomially computable. Hence, stratified negation does not increase the complexity.

8.4 Complexity Stable Model

Today we shall concentrate on

Query	Data Complexity	Program Complexity
Conjunctive query	AC_0	NP-complete
FO	AC_0	PSPACE-complete
Prop. LP		P-complete
Datalog	P-complete	EXPTIME-complete
Stratified Datalog	P-complete	EXPTIME-complete
Datalog(WFM)	P-complete	EXPTIME-complete
Datalog(INF)	P-complete	EXPTIME-complete
Datalog(Stable Model)	co-NP-complete	co-NEXPTIME-complete
Disjun. Datalog	Π_2^p-complete	co-NEXPTIME$^{\text{NP}}$-complete

Recall Stable Model Semantics

Let S be a (possibly infinite) set of ground normal clauses, i.e., of formulas of the form $A \leftarrow L_1 \wedge \ldots \wedge L_n$ where $n \geq 0$ and A is a ground atom and the L_i for $1 \leq i \leq n$ are ground literals.

Gelfond-Lifschitz Transformation → 5.3.2 p.59, Def.165

Let $B \subseteq HB$. The Gelfond-Lifschitz transform $GL_B(S)$ of S with respect to B is obtained from S as follows:

1. remove each clause whose antecedent contains a literal $\neg A$ with $A \in B$.
2. remove from the antecedents of the remaining clauses all negative literals.

Stable Model → 5.3.2 p.60, Def.166

An Herbrand interpretation $HI(B)$ is a stable model of S iff it is the unique minimal Herbrand model of $GL_B(S)$.

Complexity Prop. LP Stable model

Theorem

Given a propositional normal logic program P, deciding whether P has a stable model is NP*-complete.* → *8.9 p.119, Thm.234*

Membership. Clearly, P^I is polynomial time computable from P and I. Hence, a stable model M of P can be guessed and checked in polynomial time.

Stable Model Prop. LP - Hardness

Proof hardness

- Encoding of a non-deterministic Turing machine (NDTM) T. Given a NDTM T, an input string I and a number of steps N, where N is a polynomial of $|I|$, construct in logspace a program $P = P(T, I, N)$. P has a stable model iff T accepts I in non-deterministically N steps.
- Much similar to the encoding of DTM with propositional LP. Modification on deterministic property.

Stable Model Prop. LP - Hardness

Proof hardness

- Encoding of a non-deterministic Turing machine (NDTM) T. Given a NDTM T, an input string I and a number of steps N, where N is a polynomial of $|I|$, construct in logspace a program $P = P(T, I, N)$. P has a stable model iff T accepts I in non-deterministically N steps.
- Much similar to the encoding of DTM with propositional LP. Modification on deterministic property.

Stable Model Prop. LP - Hardness

Example: $\langle s, \sigma, s_1, \sigma'_1, d_1 \rangle$, $\langle s, \sigma, s_2, \sigma'_2, d_2 \rangle$
Transition rules $0 \leq \tau < N$, $0 \leq \pi < N$, and $0 \leq \pi + d$.

$$
\begin{aligned}
symbol_{\sigma'_1}[\tau+1, \pi] &\leftarrow state_s[\tau], symbol_\sigma[\tau, \pi], cursor[\tau, \pi] \\
cursor[\tau+1, \pi+d_1] &\leftarrow state_s[\tau], symbol_\sigma[\tau, \pi], cursor[\tau, \pi] \\
state_{s_1}[\tau+1] &\leftarrow state_s[\tau], symbol_\sigma[\tau, \pi], cursor[\tau, \pi]
\end{aligned}
$$

$$
\begin{aligned}
symbol_{\sigma'_2}[\tau+1, \pi] &\leftarrow state_s[\tau], symbol_\sigma[\tau, \pi], cursor[\tau, \pi] \\
cursor[\tau+1, \pi+d_2] &\leftarrow state_s[\tau], symbol_\sigma[\tau, \pi], cursor[\tau, \pi] \\
state_{s_2}[\tau+1] &\leftarrow state_s[\tau], symbol_\sigma[\tau, \pi], cursor[\tau, \pi]
\end{aligned}
$$

What is wrong here? Enforcement violated: At any time instance τ, there is exactly one cursor; each cell of the tape contains exactly one element; in exactly one state.

Stable Model Prop. LP - Hardness

Example: $\langle s, \sigma, s_1, \sigma'_1, d_1 \rangle$, $\langle s, \sigma, s_2, \sigma'_2, d_2 \rangle$
Transition rules $0 \leq \tau < N$, $0 \leq \pi < N$, and $0 \leq \pi + d$.

$$
\begin{aligned}
symbol_{\sigma'_1}[\tau+1, \pi] &\leftarrow state_s[\tau], symbol_\sigma[\tau, \pi], cursor[\tau, \pi] \\
cursor[\tau+1, \pi+d_1] &\leftarrow state_s[\tau], symbol_\sigma[\tau, \pi], cursor[\tau, \pi] \\
state_{s_1}[\tau+1] &\leftarrow state_s[\tau], symbol_\sigma[\tau, \pi], cursor[\tau, \pi]
\end{aligned}
$$

$$
\begin{aligned}
symbol_{\sigma'_2}[\tau+1, \pi] &\leftarrow state_s[\tau], symbol_\sigma[\tau, \pi], cursor[\tau, \pi] \\
cursor[\tau+1, \pi+d_2] &\leftarrow state_s[\tau], symbol_\sigma[\tau, \pi], cursor[\tau, \pi] \\
state_{s_2}[\tau+1] &\leftarrow state_s[\tau], symbol_\sigma[\tau, \pi], cursor[\tau, \pi]
\end{aligned}
$$

What is wrong here? Enforcement violated: At any time instance τ, there is exactly one cursor; each cell of the tape contains exactly one element; in exactly one state.

Stable Model Prop. LP - Hardness

- For each state s and symbol σ, introduce atoms $B_{s,\sigma,1}[\tau],\ldots,$ $B_{s,\sigma,k}[\tau]$ for all $1 \leq \tau < N$ and for all transitions $\langle s, \sigma, s_i, \sigma'_i, d_i \rangle$, where $1 \leq i \leq k$.
- Add $B_{s,\sigma,i}[\tau]$ in the bodies of the transition rules for $\langle s, \sigma, s_i, \sigma'_i, d_i \rangle$.
- Add the rule

$$B_{s,\sigma,i}[\tau] \leftarrow \neg B_{s,\sigma,1}[\tau], \ldots, \neg B_{s,\sigma,i-1}[\tau], \neg B_{s,\sigma,i+1}[\tau], \ldots, \neg B_{s,\sigma,k}[\tau].$$

  Intuitively, these rules non-deterministically select precisely one of the possible transitions for s and σ at time instant τ, whose transition rules are enabled via $B_{s,\sigma,i}[\tau]$.

- Finally, add a rule

$$\mathit{accept} \leftarrow \neg \mathit{accept}.$$

  It ensures that *accept* is true in every stable model.

Stable Model Prop. LP - Hardness

Example: $\langle s, \sigma, s_1, \sigma'_1, d_1 \rangle$, $\langle s, \sigma, s_2, \sigma'_2, d_2 \rangle$

$$
\begin{aligned}
symbol_{\sigma'_1}[\tau+1, \pi] &\leftarrow state_s[\tau], symbol_\sigma[\tau, \pi], cursor[\tau, \pi], B_{s,\sigma,1}[\tau] \\
cursor[\tau+1, \pi+d_1] &\leftarrow state_s[\tau], symbol_\sigma[\tau, \pi], cursor[\tau, \pi], B_{s,\sigma,1}[\tau] \\
state_{s_1}[\tau+1] &\leftarrow state_s[\tau], symbol_\sigma[\tau, \pi], cursor[\tau, \pi], B_{s,\sigma,1}[\tau] \\
symbol_{\sigma'_2}[\tau+1, \pi] &\leftarrow state_s[\tau], symbol_\sigma[\tau, \pi], cursor[\tau, \pi], B_{s,\sigma,2}[\tau] \\
cursor[\tau+1, \pi+d_2] &\leftarrow state_s[\tau], symbol_\sigma[\tau, \pi], cursor[\tau, \pi], B_{s,\sigma,2}[\tau] \\
state_{s_2}[\tau+1] &\leftarrow state_s[\tau], symbol_\sigma[\tau, \pi], cursor[\tau, \pi], B_{s,\sigma,2}[\tau]
\end{aligned}
$$

$$
\begin{aligned}
B_{s,\sigma,1}[\tau] &\leftarrow \neg B_{s,\sigma,2}[\tau] \\
B_{s,\sigma,2}[\tau] &\leftarrow \neg B_{s,\sigma,1}[\tau]
\end{aligned}
$$

One and only one atom from $B_{s,\sigma,1}[\tau]$ and $B_{s,\sigma,2}[\tau]$ is true. Which one?

Non-deterministic

Stable Model Prop. LP - Hardness

Example: $\langle s, \sigma, s_1, \sigma'_1, d_1 \rangle$, $\langle s, \sigma, s_2, \sigma'_2, d_2 \rangle$

$$
\begin{aligned}
symbol_{\sigma'_1}[\tau+1, \pi] &\leftarrow state_s[\tau], symbol_\sigma[\tau, \pi], cursor[\tau, \pi], B_{s,\sigma,1}[\tau] \\
cursor[\tau+1, \pi+d_1] &\leftarrow state_s[\tau], symbol_\sigma[\tau, \pi], cursor[\tau, \pi], B_{s,\sigma,1}[\tau] \\
state_{s_1}[\tau+1] &\leftarrow state_s[\tau], symbol_\sigma[\tau, \pi], cursor[\tau, \pi], B_{s,\sigma,1}[\tau] \\
symbol_{\sigma'_2}[\tau+1, \pi] &\leftarrow state_s[\tau], symbol_\sigma[\tau, \pi], cursor[\tau, \pi], B_{s,\sigma,2}[\tau] \\
cursor[\tau+1, \pi+d_2] &\leftarrow state_s[\tau], symbol_\sigma[\tau, \pi], cursor[\tau, \pi], B_{s,\sigma,2}[\tau] \\
state_{s_2}[\tau+1] &\leftarrow state_s[\tau], symbol_\sigma[\tau, \pi], cursor[\tau, \pi], B_{s,\sigma,2}[\tau]
\end{aligned}
$$

$$
\begin{aligned}
B_{s,\sigma,1}[\tau] &\leftarrow \neg B_{s,\sigma,2}[\tau] \\
B_{s,\sigma,2}[\tau] &\leftarrow \neg B_{s,\sigma,1}[\tau]
\end{aligned}
$$

One and only one atom from $B_{s,\sigma,1}[\tau]$ and $B_{s,\sigma,2}[\tau]$ is true. Which one?
Non-deterministic

Stable Model Prop. LP - Hardness

Example: $\langle s, \sigma, s_1, \sigma'_1, d_1 \rangle$, $\langle s, \sigma, s_2, \sigma'_2, d_2 \rangle$

$$
\begin{aligned}
symbol_{\sigma'_1}[\tau+1, \pi] &\leftarrow state_s[\tau], symbol_\sigma[\tau, \pi], cursor[\tau, \pi], B_{s,\sigma,1}[\tau] \\
cursor[\tau+1, \pi+d_1] &\leftarrow state_s[\tau], symbol_\sigma[\tau, \pi], cursor[\tau, \pi], B_{s,\sigma,1}[\tau] \\
state_{s_1}[\tau+1] &\leftarrow state_s[\tau], symbol_\sigma[\tau, \pi], cursor[\tau, \pi], B_{s,\sigma,1}[\tau] \\
symbol_{\sigma'_2}[\tau+1, \pi] &\leftarrow state_s[\tau], symbol_\sigma[\tau, \pi], cursor[\tau, \pi], B_{s,\sigma,2}[\tau] \\
cursor[\tau+1, \pi+d_2] &\leftarrow state_s[\tau], symbol_\sigma[\tau, \pi], cursor[\tau, \pi], B_{s,\sigma,2}[\tau] \\
state_{s_2}[\tau+1] &\leftarrow state_s[\tau], symbol_\sigma[\tau, \pi], cursor[\tau, \pi], B_{s,\sigma,2}[\tau]
\end{aligned}
$$

$$
\begin{aligned}
B_{s,\sigma,1}[\tau] &\leftarrow \neg B_{s,\sigma,2}[\tau] \\
B_{s,\sigma,2}[\tau] &\leftarrow \neg B_{s,\sigma,1}[\tau]
\end{aligned}
$$

One and only one atom from $B_{s,\sigma,1}[\tau]$ and $B_{s,\sigma,2}[\tau]$ is true. Which one?
Non-deterministic

Stable Model Prop. LP - Hardness

Proof.

- Assume there is a sequence of choices leading to the state yes, Let I be the set of the propositional atoms along the computation path reaching the state *accept*. $accept \in I$ due to the rule:

$$accept \leftarrow state_{\mathtt{yes}}[\tau]$$

  Clearly I is a stable model of P.

- On the contrary, if there does not exist a sequence of choices leading to the state yes in the computation tree. Assume I is a stable model, $accept \in I$ must hold. Since $(accept \leftarrow \neg accept) \notin P^I$, $P^I \not\models accept$ holds. Then I is not a least Herbrand model of P^I. Contradiction.

Further Complexity Results

Theorem

Propositional logic programming with negation under well-founded semantics is P*-complete. Datalog with negation under well-founded semantics is data complete for* P *and program complete for* EXPTIME.

→ *8.8 p.119, Thm.232*

Theorem

Propositional logic programming with negation under inflationary semantics is P*-complete. Datalog with negation under inflationary semantics is data complete for* P *and program complete for* EXPTIME.

→ *8.8 p.119, Thm.233*

Further Complexity Results

Theorem

Propositional logic programming with negation under well-founded semantics is P*-complete. Datalog with negation under well-founded semantics is data complete for* P *and program complete for* EXPTIME.

→ *8.8 p.119, Thm.232*

Theorem

Propositional logic programming with negation under inflationary semantics is P*-complete. Datalog with negation under inflationary semantics is data complete for* P *and program complete for* EXPTIME.

→ *8.8 p.119, Thm.233*

Further Complexity Results

Theorem

Propositional logic programming with negation under stable model semantics is co-NP*-complete. Datalog with negation under stable model semantics is data complete for* co-NP *and program complete for* co-NEXPTIME*.*

→ *8.9 p.120, Thm.235*

Note that the decision problem here is whether an atom is true in all stable models.

Further Complexity Results

Theorem

The program complexity of conjunctive queries is NP*-complete*

→ *8.3 p.113, Thm.225*

Theorem

First-order queries are program-complete for PSPACE*. Their data complexity is in the class* AC^0*, which contains the languages recognized by unbounded fan-in circuits of polynomial size and constant depth*

→ *8.4 p.113, Thm.226*

Further Complexity Results

Theorem

The program complexity of conjunctive queries is NP*-complete*

→ *8.3 p.113, Thm.225*

Theorem

First-order queries are program-complete for PSPACE*. Their data complexity is in the class* AC^0*, which contains the languages recognized by unbounded fan-in circuits of polynomial size and constant depth*

→ *8.4 p.113, Thm.226*

Further Complexity Results

Theorem

Logic programming is r.e.-complete. → *8.11 p.121, Thm.238*

Theorem

Nonrecursive logic programming is NEXPTIME*-complete.* → *8.11 p.121, Thm.239*

Further Complexity Results

Theorem

Logic programming is r.e.-complete. → *8.11 p.121, Thm.238*

Theorem

Nonrecursive logic programming is NEXPTIME*-complete.*

→ *8.11 p.121, Thm.239*

8.5 Expressive Power

- A *query* q defines a mapping $\mathcal{M}_q$ that assigns to each suitable input database D_{in} (over a fixed input schema) a result database $D_{out} = \mathcal{M}_q(D_{in})$ (over a fixed output schema)
- Formally, the *expressive power* of a query language Q is the set of mappings $\mathcal{M}_q$ for all queries q expressible in the language Q by some *query expression* (program) E
- Research tasks concerning expressive power
  - comparing two query languages Q_1 and Q_2 in their expressive power (e.g. FO vs. SQL vs. Datalog), which is important for designing and analysing a query language
  - determining the absolute expressive power of a query language, e.g. proving that a given query language Q is able to express exactly all queries whose evaluation complexity is in a complexity class $\mathcal{C}$. We say Q captures $\mathcal{C}$ and write simply $Q = \mathcal{C}$.

Expressive Power

There is a substantial difference between showing that the query evaluation problem for a certain query language Q is $\mathcal{C}$-complete and showing that Q captures $\mathcal{C}$.

- If the evaluation problem for Q is $\mathcal{C}$-complete, then at least one $\mathcal{C}$-hard query is expressible in Q.
- If Q captures $\mathcal{C}$, then Q expresses all queries evaluable in $\mathcal{C}$ (including, of course, all $\mathcal{C}$-hard queries).
- Example: Evaluating Datalog is P hard (data complexity), but positive Datalog can only express monotone properties, however, there are of course problems in P which are non-monotonic.

Expressive Power: Ordered Structures

- To prove that a query language Q captures a machine-based complexity class $\mathcal{C}$, one usually shows that each $\mathcal{C}$-machine with (encodings of) finite structures as inputs that computes a generic query can be represented by an expression in language Q.
- A Turing machine works on a string encoding of the input database D. Such an encoding provides an implicit *linear order* on D, in particular, on all elements of the universe U_D
- Therefore, one often assumes that a linear ordering of the universe elements is predefined
- We consider here ordered databases whose schemas contain special relation symbols *Succ*, *First*, and *Last*

Expressive Power: Datalog

Theorem

$datalog^+ \subsetneq \mathrm{P}$. → *8.12 p.123, Thm.240*

Show that there exists no datalog$^+$ program P that can tell whether the universe U of the input database has an even number of elements.

Theorem

On ordered databases, datalog$^+$ captures P. → *8.12 p.123, Thm.241*

Expressive Power: Datalog

Theorem

$datalog^+ \subsetneq \mathrm{P}$. → *8.12 p.123, Thm.240*

Show that there exists no datalog$^+$ program P that can tell whether the universe U of the input database has an even number of elements.

Theorem

On ordered databases, datalog$^+$ captures P. → *8.12 p.123, Thm.241*

Expressive Power: More Results

Theorem

Nonrecursive range-restricted datalog with negation =
relational algebra =
domain-independent relational calculus. =
first-order logic (without function symbols).

→ *8.12 p.125, Thm.243*

Theorem

On ordered databases, the following query languages capture P*:*

- *stratified datalog,*
- *datalog under well-founded semantics,*
- *datalog under inflationary semantics.*

→ *8.12 p.125, Thm.244*

Theorem

Datalog under stable model semantics captures co-NP*.*

→ *8.12 p.126, Thm.246*

Expressive Power: More Results

Theorem

Nonrecursive range-restricted datalog with negation =
relational algebra =
domain-independent relational calculus. =
first-order logic (without function symbols).

→ *8.12 p.125, Thm.243*

Theorem

On ordered databases, the following query languages capture P*:*

- *stratified datalog,*
- *datalog under well-founded semantics,*
- *datalog under inflationary semantics.*

→ *8.12 p.125, Thm.244*

Theorem

Datalog under stable model semantics captures co-NP*.*

→ *8.12 p.126, Thm.246*

Expressive Power: More Results

Theorem

Nonrecursive range-restricted datalog with negation =
relational algebra =
domain-independent relational calculus. =
first-order logic (without function symbols).

→ *8.12 p.125, Thm.243*

Theorem

On ordered databases, the following query languages capture P*:*

- *stratified datalog,*
- *datalog under well-founded semantics,*
- *datalog under inflationary semantics.*

→ *8.12 p.125, Thm.244*

Theorem

Datalog under stable model semantics captures co-NP*.*

→ *8.12 p.126, Thm.246*

Monday, 3rd September, 2007

Session 8:30 – 10:30

- 1 Introduction
- 3 Syntax
- 4 Declarative Semantics: Fundamentals

Session 11:00 – 13:00

- 5 Declarative Semantics: Adaptations

Session 14:30 – 16:00

- 6 Operational Semantics: Positive
- 7 Operational Semantics: Negative

Session 16:30 – 18:00

- 8 Complexity and Expressive Power

Essential concepts and methods of rule-based query languages

Acknowledgments

This research has been funded by the European Commission and by the Swiss Federal Office for Education and Science within the 6th Framework Programme project REWERSE number 506779 (cf. `http://rewerse.net`).

This work is based on:

François Bry, Norbert Eisinger, Thomas Eiter, Tim Furche, Georg Gottlob, Clemens Ley, Benedikt Linse, Reinhard Pichler, and Fang Wei:
Foundations of Rule-Based Query Answering.
Reasoning Web, Third International Summer School 2007,
G. Antoniou et al. (eds.), LNCS 4636, 2007. ©Springer-Verlag

This is version 1.0.r404, September 18, 2007.