VO Deductive Databases

WS 2014/2015

Stefan Woltran

Institut für Informationssysteme
Arbeitsbereich DBAI

Non-Ground Programs

- In the remainder of the lecture, we mostly consider programs with *variables*, so-called *non-ground* programs.
- Agenda:
  - Answer-Set semantics for non-ground programs;
  - equivalence between non-ground programs;
  - simplification of (non-ground) programs.

Non-Ground Programs—Introduction

- Recall Example:

$$\begin{array}{rcl} v(a). & & \\ e(a,b). & & \\ o(Y) & \leftarrow & v(X), e(X,Y). \\ o(Y) & \leftarrow & o(X), e(X,Y). \end{array}$$

  - While we consider this set as an ASP-program (and thus each subprogram is an ASP-program); only the final two rules are considered to form a datalog program.
  - Datalog programs are considered to be applicable to *any* database. The first two rules provide a concrete database instance.
  - Some equivalence notions provide a close link between these two different views.

Non-Ground Programs—Syntax

- We consider a language containing
  - a set $\mathcal{A}$ of *predicate symbols*;

    each $p \in \mathcal{A}$ has an associated arity $\alpha(p) \geq 0$
  - a set $\mathcal{V}$ of *variables*; and
  - a set $\mathcal{C}$ of *constants*; $\mathcal{C}$ is called the *domain*;

    unless stated otherwise we assume $\mathcal{C}$ to be countable infinite.
- An atom is an expression $p(t_1, \ldots, t_n)$, where
  - $p \in \mathcal{A}$;
  - $t_i \in \mathcal{V} \cup \mathcal{C}$, for each $1 \leq i \leq n$; and
  - $n = \alpha(p)$; if clear from context, we occasionally use n without explicity stating $n = \alpha(p)$.

Non-Ground Programs—Syntax (ctd.)

- An atom $p(t_1, \ldots, t_n)$ is called *ground* iff each argument t_i is a constant from $\mathcal{C}$.
- Let $A \subseteq \mathcal{A}$ and $C \subseteq \mathcal{C}$. Then, $B_{A,C}$ denotes the set of all ground atoms over predicates A with arguments from C, i.e.,

$$B_{A,C} = \{p(c_1, \ldots, c_{\alpha(p)}) \mid p \in A; c_1, \ldots, c_{\alpha(p)} \in C\}.$$

- Example. Let $A = \{e, v\}$ with $\alpha(e) = 2$; $\alpha(v) = 1$; and $C = \{a, b\}$. Then

$$\begin{aligned} B_{A,C} \quad = \{ \quad & e(a,a),\ e(a,b),\ e(b,a),\ e(b,b), \\ & v(a),\ v(b)\}. \end{aligned}$$

Non-Ground Programs—Syntax (ctd.)

- A rule r is an expression of the form

$$h_1 \ \vee \cdots \ \vee h_k \ \leftarrow b_1, \ldots, b_n, \ not \ b_{n+1}, \ldots, \ not \ b_m,$$

where $h_1, \ldots, h_k, b_1, \ldots, b_m$ are atoms, with $k \geq 0$, $m \geq n \geq 0$, and $k + m > 0$; and "not" is *default negation*.

- As for propositional programs, we call
  - $H(r) = \{h_1, \ldots, h_k\}$ the *head* of r;
  - $B(r) = \{b_1, \ldots, b_n, not \ b_{n+1}, \ldots, not \ b_m\}$ the *body* of r;
  - $B^+(r) = \{b_1, \ldots, b_n\}$ the *positive body* of r;
  - $B^-(r) = \{b_{n+1}, \ldots, b_m\}$ the *negative body* of r.

Non-Ground Programs—Syntax (ctd.)

- A rule r is *ground* iff each atom in r is ground;
- r is *safe* iff each variable occurring $H(r) \cup B^{-}(r)$ also occurs in $B^{+}(r)$.
- Examples:
  - $p(X) \leftarrow q(X,Y), not\ r(Y)$ is safe, while
  - $p(X) \leftarrow q(X,Y), not\ r(Z)$ or
  - $p(X) \leftarrow q(Y,Y), not\ r(Y)$ are not safe.

☞ Intuitively, safety guarantees that no additional constants come into play during the evaluation of a program.

Non-Ground Programs—Syntax (ctd.)

- A (non-ground) program is a finite set of safe rules.
- The classes of Horn, positive, and normal programs are defined analogously to the propositional setting.
- A program P is *ground* iff each rule in P is ground.
- A program P is *propositional* iff each predicate in P has arity 0.
  - ☞ It is convenient to assume that predicates of arity 0 include all ground atoms over $\mathcal{A}$ and $\mathcal{C}$. This allows to handle ground programs like propositional ones and vice versa.

Non-Ground Programs—Syntax (ctd.)

- Important distinction in (datalog) programs:
  - A predicate p (occurring in a program P) is called *extensional* (in P), if it is only used for atoms in bodies of rules (of P);
  - otherwise, p is called *intensional* (in P).
- Extensional predicates are identified as those which are specified by a database; intensional atoms are used to compute the query.
- The example program

$$o(Y) \leftarrow v(X), e(X, Y). \qquad o(Y) \leftarrow o(X), e(X, Y).$$

has extensional predicates v, e, and an intensional predicate o.

Non-Ground Programs—Syntax (ctd.)

- For simplicity, we shall consider a partition on the predicates $\mathcal{A} = (\mathcal{A}_I, \mathcal{A}_E)$, dedicating each predicate to be used as intensional or extensional in any program.

- In what follows, we assume that any program has its intensional predicates from $\mathcal{A}_I$ and its extensional predicates from $\mathcal{A}_E$.

- We call an atom $p(t_1, \ldots, t_n)$ intensional/extensional iff $p \in \mathcal{A}_I$ / $\mathcal{A}_E$.

Non-Ground Programs—Syntax (ctd.)

- For a rule r (resp. a program P), let
  - $\mathcal{A}_r$ (resp. $\mathcal{A}_P$) be the set of predicate symbols occurring in r (resp. P);
  - $\mathcal{V}_r$ (resp. $\mathcal{V}_P$) be the set of variables in r (resp. P); and
  - $\mathcal{C}_r$ (resp. $\mathcal{C}_P$) be the set of constants occurring in r (resp. P).
- We call
  - the set

$$U_P = \begin{cases} \mathcal{C}_P & \text{if } \mathcal{C}_P \neq \emptyset \\ \{c_1\} & \text{otherwise, with } c_1 \in \mathcal{C} \text{ arbitrary.} \end{cases}$$

    the *active domain* (or Herbrand universe) of P;
  - $B_P = B_{\mathcal{A}_P, U_P}$ the *Herbrand base* of P.

Non-Ground Programs—Syntax (ctd.)

- We finally need the concept of *substitutions*.
- A substitution $\theta : \mathcal{V} \to \mathcal{V} \cup \mathcal{C}$ is a (partial) function mapping variables to variables or constants.
  - If, for each $v \in \mathcal{V}$, $\theta(v) \in \mathcal{C}$, we call θ a grounding;
  - if, for each $v \in \mathcal{V}$, $\theta(v) \in \mathcal{V}$, we call θ a (variable) renaming.
- For an expression (i.e., an atom, sets of atoms, rule, program) e, denote by $e\theta$ the expression resulting from e by replacing each $v \in \mathcal{V}$ in e by $\theta(v)$.

Non-Ground Programs—Syntax (ctd.)

- For a rule r, and a set of constants C, we denote

$$Gr(r, C) = \{r\theta \mid \theta : \mathcal{V}_r \to C\},$$

and for a program P, $Gr(P, C) = \bigcup_{r \in P} Gr(r, C)$.

  - Note that $Gr(r, C)$, resp. $Gr(P, C)$, are ground programs.

- We define the *grounding of a program P* as $Gr(P) = Gr(P, U_P)$.

Non-Ground Programs—Syntax (ctd.)

- Example program P:

$$
\begin{array}{ll}
v(a). & e(a,b). \\
o(Y) \leftarrow v(X), e(X,Y). & o(Y) \leftarrow o(X), e(X,Y).
\end{array}
$$

- We have $Gr(P, \{a\})$ given as

$$
\begin{array}{ll}
v(a). & e(a,b). \\
o(a) \leftarrow v(a), e(a,a). & o(a) \leftarrow o(a), e(a,a).
\end{array}
$$

- $Gr(P, \{a,b\}) = Gr(P, U_P) = Gr(P)$ is given by

$$
\begin{array}{ll}
v(a). & e(a,b). \\
o(a) \leftarrow v(a), e(a,a). & o(a) \leftarrow o(a), e(a,a). \\
o(a) \leftarrow v(b), e(b,a). & o(a) \leftarrow o(b), e(b,a). \\
o(b) \leftarrow v(a), e(a,b). & o(b) \leftarrow o(a), e(a,b). \\
o(b) \leftarrow v(b), e(b,b). & o(b) \leftarrow o(b), e(b,b).
\end{array}
$$

Non-Ground Programs—Semantics

- An interpretation I is a set of ground atoms, i.e., $I \subseteq B_{\mathcal{A},\mathcal{C}}$.
- We rephrase concepts from propositional programs:
  - An interpretation I is a model of a ground rule r of form

    $$h_1 \vee \cdots \vee h_k \leftarrow b_1, \ldots, b_n, not\ b_{n+1}, \ldots, not\ b_m$$

    iff the following holds:

    If $b_1, \ldots, b_n$ are all in I, and none of $b_{n+1}, \ldots, b_m$ are in I then at least one out of $h_1, \ldots, h_k$ is in I.
  - An interpretation I is a model of a ground program P if I is a model of any rule $r \in P$.
- An interpretation I is a model of a non-ground program P if I is a model of the grounding of P, $Gr(P)$.

Non-Ground Programs—Semantics (ctd.)

- The notion of a *reduct* is defined only for ground programs, and thus analogously to the propositional case. Recall: Let I be an interpretation and P a ground program. Then,

$$P^I = \{H(r) \leftarrow B^+(r) \mid r \in P,\ I \cap B^-(r) = \emptyset\}.$$

- An interpretation I is an **answer set** of program P iff I is a minimal model of $Gr(P)^I$, i.e., iff I is answer set of $Gr(P)$.
- An answer set I of a program P is always a subset of the Herbrand-base of P, i.e., $I \subseteq B_P = B_{\mathcal{A}_P, U_P}$.
- Hence, the computation of answer sets works as in the propositional case, with an additional "pre-processing" step of grounding.
- But: Groundings of a program P are in general of exponential size compared to P.

Non-Ground Programs—Semantics (ctd.)

- Example: We already have obtained from P

$$
\begin{array}{ll}
v(a). & e(a,b). \\
o(Y) \leftarrow v(X), e(X,Y). & o(Y) \leftarrow o(X), e(X,Y).
\end{array}
$$

its grounding $Gr(P)$:

$$
\begin{array}{ll}
v(a). & e(a,b). \\
o(a) \leftarrow v(a), e(a,a). & o(a) \leftarrow o(a), e(a,a). \\
o(a) \leftarrow v(b), e(b,a). & o(a) \leftarrow o(b), e(b,a). \\
o(b) \leftarrow v(a), e(a,b). & o(b) \leftarrow o(a), e(a,b). \\
o(b) \leftarrow v(b), e(b,b). & o(b) \leftarrow o(b), e(b,b).
\end{array}
$$

- Since there is no "*not*" operator, we just have to compute the minimal models of $Gr(P)$. The only answer set of P is given by $\{v(a), e(a,b), o(b)\}$.

Non-Ground Programs—Complexity

- We briefly mention complexity issues:
  Deciding whether a Horn (normal, disjunctive) program has at least one answer set is $\mathrm{EXPTIME}$ ($\mathrm{NEXPTIME}$, $\mathrm{NEXPTIME}^{\mathrm{NP}}$) -complete.
  - E. Dantsin, T. Eiter, G. Gottlob, and A. Voronkov: Complexity and Expressive Power of Logic Programming. *ACM Computing Surveys* 33(3):374–425. 2001.
- If the arity of each predicate in a program is bound by a fixed constant then the problem becomes easier:
  Deciding whether an arity-bound Horn (normal, disjunctive) program has at least one answer set is complete for NP (Π_2^P, Π_3^P).
  - T. Eiter, W. Faber, M. Fink, G. Pfeifer, and S. Woltran: Complexity Results for Answer Set Programming with Bounded Predicate Arities and Implications. *Ann. Math. Artif. Intell.* 51(2-4): 123-165. 2007.

Non-Ground Programs—Equivalences

- As in the propositional case, we can define different notions of equivalence.
- Straight-forward is *ordinary* equivalence: Two programs P, Q are ordinary equivalent iff P and Q have the same answer sets.
- Complexity results are at the same level as checking whether answer sets exist.
- For the other equivalence notions things become more complicated;
  - ➥ we have to face the problem that the potential program extension may extend the active domain.

Non-Ground Programs—Equivalences (ctd.)

- Consider

$$P = \{o(X) \leftarrow v(X), e(X,X)\} \quad \text{and} \quad Q = \{o(Y) \leftarrow v(X), e(X,Y)\}.$$

  Both programs have active domain $\{c_1\}$ and the empty set as their only answer sets.

- Consider $a, b \in \mathcal{C}$ from our language. Adding $R = \{v(a), e(a,b)\}$ yields different answer sets for the two programs.

- This phenomenon makes some equivalence problems *undecidable* for infinite $\mathcal{C}$.

- Is an infinite domain $\mathcal{C}$ relevant for practice?
  - Yes. For example, we want to compare queries/programs over any graph (without any restriction on the names of the vertices).

Non-Ground Programs—Equivalences (ctd.)

- We define strong equivalence as expected:

  Two programs P and Q are strongly equivalent, in symbols $P \equiv_s Q$, iff, for each program R, $AS(P \cup R) = AS(Q \cup R)$.

- It turns out that the concept of SE-models characterizes strong equivalence, also in the non-ground case.

Non-Ground Programs—Equivalences (ctd.)

- We need a further technical concept, which can be understood as a form of an extended Herbrand universe.
- Let P a program and let n be the maximal number of different variables in a rule of P. Then, U_P^+ is defined as $\mathcal{C}_P \cup \{c_1, \ldots, c_n\}$, where the c_i's are constants different from $\mathcal{C}_P$.
- Note that the cardinality of U_P^+ is polynomial in size of P.

Non-Ground Programs—Equivalences (ctd.)

➤ **Theorem.** The following propositions are equivalent:

1. $P \equiv_s Q$;
2. for each $C \subseteq \mathcal{C}$, $SE(Gr(P, C)) = SE(Gr(Q, C))$;
3. for $D = U^{+}_{P \cup Q}$, $SE(Gr(P, D)) = SE(Gr(Q, D))$.

(Proof in the additional material).

➥ Property 3. shows that SE is decidable, also for infinite $\mathcal{C}$; it is also the basis to show that SE is $\mathrm{coNEXPTIME}$-complete.

☞ T. Eiter, M. Fink, H. Tompits, and S. Woltran: Strong and Uniform Equivalence in Answer-Set Programming: Characterizations and Complexity Results for the Non-Ground Case. *Proceedings AAAI-05*, pages 695–700. AAAI Press, 2005.

Non-Ground Programs—Equivalences (ctd.)

- **Example:** Consider programs

$$P = \{ \; p(X,Y) \leftarrow e(X,Y) \quad o(Y) \leftarrow v(X), p(X,Y) \quad o(Y) \leftarrow o(X), p(X,Y) \}$$

$$Q = \{ \; p(X,Y) \leftarrow e(X,Y) \quad o(Y) \leftarrow v(X), p(X,Y) \quad p(X,Z) \leftarrow p(X,Y), p(Y,Z) \}$$

- The two programs are not strongly equivalent.

  Take $D = U^{+}_{P \cup Q} = \{a, b, c\}$. Then, for

$$I = \{p(a,b), p(b,c)\}$$

  $(I, I) \in SE(Gr(P, D))$ but $(I, I) \notin SE(Gr(Q, D))$.

- However, understood as datalog queries over predicate o, P and Q are equivalent over all input graphs (details later!)

Non-Ground Programs—Equivalences (ctd.)

- We next consider uniform equivalence. The definition is as follows.
- Two programs are uniformly equivalent, $P \equiv_u Q$, if for any finite set F of facts, $AS(P \cup F) = AS(Q \cup F)$.
- By safety, any fact has to be ground, thus F is ground.
- We first show a positive result, viz. that uniform equivalence between positive programs is decidable.

Non-Ground Programs—Equivalences (ctd.)

- We show a more general result.

- For any positive programs P, Q, $P \equiv_s Q$ iff $P \equiv_u Q$.
  - The only-if direction is by definition.
  - For the if-direction, suppose $P \not\equiv_s Q$, i.e., we have a pair $(J, I) \in SE(Gr(P, C))$ but $(J, I) \notin SE(Gr(Q, C))$; by our theorem on strong equivalence, we can assume C to be finite.
    By $Gr(P,C)^I = Gr(P,C) = Gr(P,C)^J$ and $Gr(Q,C)^I = Gr(Q,C)^J$, we have $(J, J) \in SE(Gr(P, C))$ and $(J, J) \notin SE(Gr(Q, C))$.
    But then J is answer set of $P \cup J$ but not of $Q \cup J$, and since J is finite (because C is finite), $P \not\equiv_u Q$, by definition.

Non-Ground Programs—Equivalences (ctd.)

- Programs P, Q are *query equivalent* (with respect to a predicate p), iff, for each set F of ground extensional atoms, p evaluates the same in $AS(P \cup F)$ and $AS(Q \cup F)$ ...

  ... formally, for each set F of ground extensional atoms, $(B_{\{p\},\mathcal{C}} \cap AS(P \cup F)) = (B_{\{p\},\mathcal{C}} \cap AS(Q \cup F))$ has to hold.

- Seminal result from database theory: Query-equivalence is undecidable, even for Horn programs.

  ☞ O. Shmueli: Decidability and Expressiveness Aspects of Logic Queries. *Proceedings PODS'87*, ACM Press.

- The proof maps an undecidability problem from grammars to query equivalence.

Non-Ground Programs—Equivalences (ctd.)

- Excurs: A **context-free grammar** (CFG) G is a tuple (N, Σ, s, P), where
  - N is a finite set of *nonterminal* symbols;
  - Σ is a finite alphabet of *terminal* symbols, disjoint from N;
  - $s \in N$ is the *start* symbol;
  - P is a finite set of *productions* $n \rightarrow w$ with $n \in N$ and w a word over $N \cup \Sigma$.
- A CFG G defines a language $L(G)$ consisting of all words over Σ^* that can be derived from s by repeated application of the productions.

Non-Ground Programs—Equivalences (ctd.)

- Example: Let $G = (\{s,t\},\{a,b,c\},s,P)$ with

$$P = \{s \rightarrow t;\ t \rightarrow atc;\ t \rightarrow b\}.$$

Then, $L(G) = \{a^n b\, c^n \mid n \geq 0\}$.

- **Result**: Given CFG grammars G_1, G_2, it is undecidable whether $L(G_1) = L(G_2)$.
- Undecidability, holds already for CFGs which are ϵ-free and do not have start symbol s in any rhs of P.

Non-Ground Programs—Equivalences (ctd.)

- Let $G = (N, \Sigma, s, P)$ be CFG with above restrictions. We construct a program P_G as follows, assuming
  - N as a set of predicates from $\mathcal{A}_I$ of arity 2;
  - $\Sigma \subseteq \mathcal{C}$ as part of our domain;
  - a predicate r of arity 3 from $\mathcal{A}_E$.
- To each production $n \to s_1 \dots s_m$ we associate a rule

$$n(X_1, X_{m+1}) \leftarrow a_1, \dots, a_m;$$

  where
  - if s_i is nonterminal $n' \in N$, $a_i = n'(X_i, X_{i+1})$; and
  - if s_i is terminal v, then $a_i = r(X_i, v, X_{i+1})$.

Non-Ground Programs—Equivalences (ctd.)

- We consider that words $w = v_1 \ldots v_m$ over Σ are encoded by the set $S_w = \{r(1, v_1, 2), r(2, v_2, 3), \ldots, r(m, v_m, m+1)\}$.
- Our example $G = (\{s, t\}, \{a, b, c\}, s, \{s \to t; t \to atc; t \to b\}$ yields

$$
\begin{aligned}
&s(X, Y) \leftarrow t(X, Y);\\
&t(V, Z) \leftarrow r(V, a, X), t(X, Y), r(Y, c, Z);\\
&t(X, Y) \leftarrow r(X, b, Y).
\end{aligned}
$$

  Consider this program together with S_w for $w = b$, $w = abc$, and $w = ac$. Then,

  - $S_w = \{r(1, b, 2)\}$: we derive $s(1, 2)$;
  - $S_w = \{r(1, a, 2), r(2, b, 3), r(3, c, 4)\}$: we derive $s(1, 4)$;
  - $S_w = \{r(1, a, 2), r(2, c, 3)\}$: we do not derive $s(1, 3)$.

Non-Ground Programs—Equivalences (ctd.)

- Proof Sketch for undecidability for query equivalence.
  - One can show that, for a word of length m, $s(1, m+1)$ can be derived from $P_G \cup S_w$, only if $w \in L(G)$.
  - Hence, given two grammars G, H (over same Σ and same start symbol s), we have that for any word w over Σ, the predicate s provides the same output on $P_G \cup S_w$ and $P_H \cup S_w$.
  - This correspondence extends to any set of extensional atoms.
  - Hence, $L(G) = L(H)$ iff P_G and P_H are query equivalent with respect to predicate s.

Non-Ground Programs—Equivalences (ctd.)

- P and Q are *program equivalent* iff, for any set F of extensional atoms, $AS(P \cup F) = AS(Q \cup F)$.

- We map query equivalence to program equivalence as follows: Define, for programs P, Q, and a predicate p,

$$P^* = P \cup Q' \cup \{p^*(X_1, \ldots, X_n) \leftarrow p(X_1, \ldots, X_n)\} \quad \text{and}$$

$$Q^* = P \cup Q' \cup \{p^*(X_1, \ldots, X_n) \leftarrow p'(X_1, \ldots, X_n)\},$$

  where Q' results from Q by replacing each *intensional* predicate symbol q by q', and p^* is a fresh predicate which refers to the query predicate p.

- P^* and Q^* are program equivalent iff P and Q are query equivalent with respect to p.
  - Program equivalence between Horn programs is undecidable.

Non-Ground Programs—Equivalences (ctd.)

- One can map program equivalence between Horn programs to uniform equivalence between disjunctive programs.
  - ☞ T. Eiter, M. Fink, H. Tompits, and S. Woltran: Strong and Uniform Equivalence in Answer-Set Programming: Characterizations and Complexity Results for the Non-Ground Case. *Proceedings AAAI-05*, pages 695–700. AAAI Press, 2005.
- **Theorem.** Deciding uniform equivalence between disjunctive non-ground programs is undecidable.
- Later this result was strengthened to normal programs.
  - ☞ T. Eiter, M. Fink, H. Tompits, and S. Woltran: Complexity Results for Checking Equivalence of Stratified Logic Programs. *Proceedings IJCAI'07*, pages 330–335.

Non-Ground Programs—Equivalences (ctd.)

- Remark: Uniform equivalence was originally introduced as decidable (but incomplete) test for query equivalence between Horn programs.
  - Y. Sagiv: Optimizing Datalog Programs. In J. Minker (ed.): *Foundations of Deductive Databases and Logic Programming*. Morgan Kaufmann, 1988.
- More on the decidability/undecidability frontier for query equivalence between Horn programs (wrt several extensions/restrictions).
  - A. Halevy, I. Mumick, Y. Sagiv, O. Shmueli: Static Analysis in Datalog Extensions. J. ACM 48(5): 971-1012 (2001).
- Brief survey on different equivalence notions:
  - S. Woltran: Equivalence between Extended Datalog Programs - A Brief Survey. Datalog 2010: 106-119, Springer.

Exercise

- Formulate a non-ground program which computes the vertex cover for undirected graphs (V, E). Graphs are given as input over predicates $v(\cdot)$ and $e(\cdot, \cdot)$. A vertex cover is a set $S \subseteq V$, such that, for each $(a, b) \in E$, $\{a, b\} \cap S \neq \emptyset$.
- Consider the program

$$P = \{s(X) \leftarrow t(X);\ t(Y) \leftarrow u(Y);\ u(Z) \leftarrow s(Z)\}$$

and the rule $r = s(X) \leftarrow u(X)$. Is P strongly equivalent to $P \cup \{r\}$? Why (not)?