Systems and Solving Techniques for Knowledge Representation

– Datalog Part II –

Marco Maratea
University of Genoa, Italy

066 011 Double degree programme Computational Logic (Erasmus-Mundus)
066 931 Computational Intelligence
066 937 Software Engineering & Internet Computing
Institute of Information Systems

Syntax & Notation

Terms: Constants and Variables

Atoms: of the form *predicate*$(t_1, \dots, t_n)$

Literals: atoms a (pos.) and negated atoms $\text{not } a$ (neg.)

Rules: $h \text{ :- } p_1, \dots, p_n, \text{not } n_1, \dots \text{not } n_n.$

Head: $H(r) = h$

Body: $B(r) = \{p_1, \dots, p_n, \text{not } n_1, \dots \text{not } n_n.\}$

Positive Body: $B^+(r) = \{p_1, \dots, p_n\}$

Negative Body: $B^-(r) = \{\text{not } n_1, \dots \text{not } n_n\}$

Program: A set of rules

Safety: All variables occur in some positive body atom

Ground: no variable occurs in it

Positive Program: all rules are such that $B^-(r) = \emptyset$

Semantics Positive Programs

Interpretation: a set I of ground atoms

- atom a is true w.r.t. I if $a \in I$, it is false otherwise, and
- negative literal $\mathrm{not}\ a$ is true w.r.t. I if $a \notin I$, it is false otherwise.

Satisfaction: a rule r is satisfied w.r.t. I if $H(r) \in I$ whenever all literals $\ell \in B(r)$ are true w.r.t. I

Model: an interpretation I is a model for program P if all rules in P are satisfied by I

Least Model: an interpretation I is the least or minimal model for program P if every $I' \subset I$ is not a model for P

Example Models

Given:

$a :- b, c.$

$c :- d.$

$d.$

Interpretations and Models:

$I_1 = \{b, c, d\}$, $I_2 = \{a, b, c, d\}$ $I_3 = \{c, d\}$

$\rightarrow$ only I_2 and I_3 are models!

Example Models

Given:

$a :- b, c.$

$c :- d.$

$d.$

Interpretations and Models:

$I_1 = \{b, c, d\}$, $I_2 = \{a, b, c, d\}$ $I_3 = \{c, d\}$

$\rightarrow$ only I_2 and I_3 are models!

$\rightarrow$ I_3 is minimal!

Semantics Positive Programs

Rule Instantiation: given a rule r, $Inst(r)$ is the set of ground rules that can be obtained by replacing every variable in r by a constant occurring in a program P

Instantiation: given a program P, $G(P) = \cup_{r \in P} Inst(r)$

Model: an interpretation M is a model for program P if M is a model of $G(P)$

Least Model: an interpretation M is the least model of program P if M is the least model of $G(P)$

Operational Semantics for Positive Programs (Ground case)

Immediate Consequence Operator: Given a ground program P, and an interpretation I

$$T_p(I) = \{a | \exists r \in P \text{ s.t. } H(r) = a \wedge \forall l \in B(r) \text{ are true in } I\}$$

Example: $a \text{ :- } b.\ c \text{ :- } d.\ e \text{ :- } a.\ I = \{b\}$, $T_p(I) = \{a\}$.

Fixpoint procedure:

- Start with $I = \emptyset$.
- Repeatedly apply T_p until a fixpoint $T_p(I) = I$ is reached.

Least Model: The least fixpoint T_p.
Theorem: A positive Datalog program P has a unique least model, which is the minimal model corresponding to the intersection of all models of P.

Operational Semantics for Positive Programs (Ground case)

Immediate Consequence Operator: Given a ground program P, and an interpretation I

$T_p(I) = \{a | \exists r \in P$ s.t. $H(r) = a \wedge \forall l \in B(r)$ are true in $I\}$

Example: $a \,\text{:-}\, b.\ c \,\text{:-}\, d.\ e \,\text{:-}\, a.\ I = \{b\},\ T_p(I) = \{a\}$.

Fixpoint procedure:

- Start with $I = \emptyset$.
- Repeatedly apply T_p until a fixpoint $T_p(I) = I$ is reached.

Least Model: The least fixpoint T_p.
Theorem: A positive Datalog program P has a unique least model, which is the minimal model corresponding to the intersection of all models of P.

Operational Semantics for Positive Programs (Ground case)

Immediate Consequence Operator: Given a ground program P, and an interpretation I

$$T_p(I) = \{a | \exists r \in P \text{ s.t. } H(r) = a \ \wedge \forall l \in B(r) \text{ are true in } I\}$$

Example: $a \text{ :- } b.\ c \text{ :- } d.\ e \text{ :- } a.\ I = \{b\}$, $T_p(I) = \{a\}$.

Fixpoint procedure:

- Start with $I = \emptyset$.
- Repeatedly apply T_p until a fixpoint $T_p(I) = I$ is reached.

Least Model: The least fixpoint T_p.
Theorem: A positive Datalog program P has a unique least model, which is the minimal model corresponding to the intersection of all models of P.

Operational Semantics (Non-ground case)

Ground + Fixpoint:

Given P, build $G(P)$, apply operator to compute fixpoint until $T_{G(P)}(M) = M$.

Consider:

$a(X) :- b(X), c(X).$

$b(a).\ b(b).\ c(a).\ c(c).$

Instantiation:

$a(a) :- b(a), c(a).$

$a(b) :- b(b), c(b).$

$a(c) :- b(c), c(c).$

$\ldots$

Operational Semantics (Non-ground case)

Ground + Fixpoint:

Given P, build $G(P)$, apply operator to compute fixpoint until $T_{G(P)}(M) = M$.

Consider:

$a(X) :- b(X), c(X).$

$b(a).\ b(b).\ c(a).\ c(c).$

Instantiation:

$a(a) :- b(a), c(a).$

$a(b) :- b(b), c(b).$

$a(c) :- b(c), c(c).$

$\ldots$

Operational Semantics (Non-ground case)

Ground + Fixpoint:

Given P, build $G(P)$, apply operator to compute fixpoint until $T_{G(P)}(M) = M$.

Consider:

$a(X) : -b(X), c(X).$

$b(a).\ b(b).\ c(a).\ c(c).$

Instantiation:

$a(a) : -b(a), c(a).$

$a(b) : -b(b), c(b).$

$a(c) : -b(c), c(c).$

… **Do we need all these ground rules?**

Operational Semantics (Non-ground case)

Ground + Fixpoint:

Given P, build $G(P)$, apply operator to compute fixpoint until $T_{G(P)}(M) = M$.

Consider:

$a(X) :- b(X), c(X).$

$b(a).\ b(b).\ c(a).\ c(c).$

Instantiation:

$a(a) :- b(a), c(a).$

$a(b) :- b(b), c(b).$

$a(c) :- b(c), c(c).$

... **Do they have any chance to be satisfied?**

Operational Semantics (Non-ground case)

Ground + Fixpoint:

Given P, build $G(P)$, apply operator to compute fixpoint until $T_{G(P)}(M) = M$.

Consider:

$a(X) : -b(X), c(X).$

$b(a). \ b(b). \ c(a). \ c(c).$

Instantiation:

$a(a) : -b(a), c(a).$

$a(b) : -b(b), c(b).$

$a(c) : -b(c), c(c).$

... Start from facts, match bodies, apply ... fixpoint!

Example Semantics

Consider:

$grandParent(X, Y) \text{ :- } parent(X, Z), parent(Z, Y).$

$parent(a, b).\ parent(b, c).$

Evaluation:

1. $I = \emptyset$
2. $I = \{parent(a, b), parent(b, c)\}$
3. body can be instantiated $(parent(a, b), parent(b, c))$
   Apply T_P: $I := I \cup \{grandParent(a, c)\}$
4. no body can be matched with atoms in I ... STOP!

Results: $\{parent(a, b), parent(b, c), grandParent(a, c)\}$ is the least model

Example Semantics

Consider:

$grandParent(X, Y) :- parent(X, Z), parent(Z, Y).$

$parent(a, b).\ parent(b, c).$

Evaluation:

1. $I = \emptyset$
2. $I = \{parent(a, b), parent(b, c)\}$
3. body can be instantiated $(parent(a, b), parent(b, c))$
   Apply T_P: $I := I \cup \{grandParent(a, c)\}$
4. no body can be matched with atoms in I ... STOP!

Results: $\{parent(a, b), parent(b, c), grandParent(a, c)\}$ is the least model

Example Semantics

Consider:

$grandParent(X, Y) :- parent(X, Z), parent(Z, Y).$

$parent(a, b).\ parent(b, c).$

Evaluation:

1. $I = \emptyset$
2. $I = \{parent(a, b), parent(b, c)\}$
3. body can be instantiated $(parent(a, b), parent(b, c))$
   Apply T_P: $I := I \cup \{grandParent(a, c)\}$
4. no body can be matched with atoms in I ... STOP!

Results: $\{parent(a, b), parent(b, c), grandParent(a, c)\}$ is the least model

Example Semantics

Consider:

$grandParent(X, Y) :- parent(X, Z), parent(Z, Y).$

$parent(a, b).\ parent(b, c).$

Evaluation:

1. $I = \emptyset$
2. $I = \{parent(a, b), parent(b, c)\}$
3. body can be instantiated $(parent(a, b), parent(b, c))$
   Apply T_P: $I := I \cup \{grandParent(a, c)\}$
4. no body can be matched with atoms in I ... STOP!

Results: $\{parent(a, b), parent(b, c), grandParent(a, c)\}$ is the least model

Semantics c.t.d.

Immediate Consequence Operator:
Given a non-ground program P, and an interpretation I

$$T_p(I) = \{H(r_g) | \exists r_g \text{ instantiating } r \in P \text{ s.t. the body of } r_g \text{ is true w.r.t. } I\}$$

Operational Semantics:

Compute $M = T_p(M)$ by repeatedly applying T_p starting from EDB.

Stratified Programs

Dependency Graph: Given a program P, the graph $DG(P) := (V, E)$ is defined as follows:

- a node p in V for each predicate p occurring in P
- positive edge $p \leftarrow q$ in E if there is rule r s.t. p occurs in $H(r)$ and q occurs in $B^+(r)$
- negative edge $p \leftarrow_n q$ in E if there is rule r s.t. p occurs in $H(r)$ and q occurs in $B^-(r)$.

Recursive Program: P is recursive if $DG(P)$ is cyclic.

Stratified Program: P is stratified if no cycle in $DG(P)$ contains a negative edge.

Negation and Recursion

Consider:

$p(X) :- q(X), \text{not } p(X).$

$q(1).\ q(2).$

Evaluation:

1. q(1). q(2).
2. q(1). q(2). p(1). p(2).
3. ...

Stratified Program

Consider:

$r_1 : reach(X) :- source(X).$

$r_2 : reach(X) :- reach(Y), arc(Y, X).$

$r_3 : noReach(X) :- target(X), not\ reach(X).$

Dependency Graph:

- V = {reach,source,target,noReach,arc}
- E = {(reach,source), (reach,reach), (reach,arc), (noReach,target), (noReach,reach)$_n$}
- cyclic, but stratified!

Stratified Program

Consider:

$r_1 : reach(X) :- source(X).$

$r_2 : reach(X) :- reach(Y), arc(Y, X).$

$r_3 : noReach(X) :- target(X), not\ reach(X).$

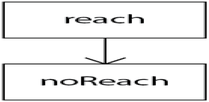
Dependency Graph:

- V = {reach,source,target,noReach,arc}
- E = {(reach,source), (reach,reach), (reach,arc), (noReach,target), (noReach,reach)$_n$}
- cyclic, but stratified!

Stratified Program - components and modules

Components and Subprograms:

- Let Comp(DG) be the set of the strongly connected components of DG
- Given $C \in Comp(DG)$ the subprogram associated to C is $Sub(P, C) = \{r \in P \text{ s.t. } H(r) \in C\}$
- Given C' depends on C'' if there is some (negative) arc in DG from a node in C'' to a node in C'



Example ctd:

- $Comp(DG) = \{\{reach\}, \{noReach\}\}$
- $Sub(P, \{reach\}) = \{r_1, r_2\}$
- $Sub(P, \{noReach\}) = \{r_3\}$

Stratified Program - Evaluation

Evaluation:

1. Start from the components that do not depend on other components
2. Evaluate subprograms associated to components as for positive programs
3. Remove evaluated components
4. Go to step 2. if still components are to be evaluated

Example ctd:

1. Evaluate $\{\{reach\}\}$
2. Evaluate $\{\{noReach\}\}$

Example Stratified Program

Consider:

$r_1 : reach(X) :- source(X).$

$r_2 : reach(X) :- reach(Y), arc(Y, X).$

$r_3 : noReach(X) :- target(X), not\ reach(X).$

EDB: $node(1).node(2).node(3).node(4).arc(1,2).$
$arc(3,4).arc(4,3).source(1), target(2).target(3).$

Evaluate $Sub(P, \{reach\}) = \{r_1, r_2\}$**:**

1. $I = \{source(1), target(2), target(3), ...\}$
2. $I := I \cup \{reach(1)\}$
3. $I := I \cup \{reach(2)\}$...STOP!

Evaluate $Sub(P, \{noReach\}) = \{r_3\}$**:**

1. $I := I \cup \{noReach(3)\}$...STOP!

Example Stratified Program

Consider:

$r_1 : reach(X) :- source(X).$

$r_2 : reach(X) :- reach(Y), arc(Y, X).$

$r_3 : noReach(X) :- target(X), not\ reach(X).$

EDB: $node(1).node(2).node(3).node(4).arc(1, 2).$
$arc(3, 4).arc(4, 3).source(1), target(2).target(3).$

Evaluate $Sub(P, \{reach\}) = \{r_1, r_2\}$**:**

1. $I = \{source(1), target(2), target(3), ...\}$
2. $I := I \cup \{reach(1)\}$
3. $I := I \cup \{reach(2)\}$...STOP!

Evaluate $Sub(P, \{noReach\}) = \{r_3\}$**:**

1. $I := I \cup \{noReach(3)\}$...STOP!

Example Stratified Program

Consider:

$r_1 : reach(X) :- source(X).$

$r_2 : reach(X) :- reach(Y), arc(Y, X).$

$r_3 : noReach(X) :- target(X), not\ reach(X).$

EDB: $node(1).node(2).node(3).node(4).arc(1, 2).$
$arc(3, 4).arc(4, 3).source(1), target(2).target(3).$

Evaluate $Sub(P, \{reach\}) = \{r_1, r_2\}$**:**

1. $I = \{source(1), target(2), target(3), ...\}$
2. $I := I \cup \{reach(1)\}$
3. $I := I \cup \{reach(2)\}$...STOP!

Evaluate $Sub(P, \{noReach\}) = \{r_3\}$**:**

1. $I := I \cup \{noReach(3)\}$...STOP!

Thanks to Francesco Ricca for a preliminary version of these slides