Open vs. Closed World Assumption

Closed World Assumption (CWA)
What is not implied by a logic program is false.

Open World Assumption (OWA)
What is not implied by a logic program is unknown.

Scope
- traditional database applications: CWA
- information integration: OWA or CWA

Can negation be allowed inside Datalog rules?
Datalog

Syntax

Rules with negated goals in the body:

\[ A_0 : -A_1, \ldots, A_k, \text{not } B_1, \ldots, \text{not } B_m. \]

Example

\[ \text{forebear}(X,Y) : \neg \text{anc}(X,Y), \text{not } \text{parent}(X,Y). \]

Generalizing \( T_P \)

\[ T_P(I) = \{ A \mid \exists r \in \text{ground}(P). \ r = A : -A_1, \ldots, A_n, \text{not } B_1, \ldots, \text{not } B_m \]
\[ \land A_1 \in I \land \cdots \land A_n \in I \land B_1 \notin I \land \cdots \land B_m \notin I \}. \]

Datalog\text{not}: semantics

Semantics

- minimal (Herbrand) models:
  - one or more
  - the right one?
- minimal fixpoints of \( T_P \):
  - none, one, or more than one
  - the right one?
- bottom-up evaluation

Solutions

- restrict programs syntactically: \textit{stratified,}\ ...
- consider multiple logical meanings: \textit{stable models,}\ ...
**Stratification [ABW88]**

**Dependency graph \( pdg(P) \)**
- **vertices:** predicates of a \( \text{Datalog}^{\text{not}} \) program \( P \)
- **edges:**
  - a **positive** edge \( (p, q) \) if there is a clause in \( P \) in which \( q \) appears in a positive goal in the body and \( p \) appears in the head
  - a **negative** edge \( (p, q) \) if there is a clause in \( P \) in which \( q \) appears in a negative goal in the body and \( p \) appears in the head

**Stratified \( P \)**

No cycle in \( pdg(P) \) contains a negative edge.

**Stratification**

Mapping \( s \) from the set of predicates in \( P \) to nonnegative integers such that:
- if a positive edge \( (p, q) \) is in \( pdg(P) \), then \( s(p) \geq s(q) \)
- if a negative edge \( (p, q) \) is in \( pdg(P) \), then \( s(p) > s(q) \)

There is a **polynomial-time** algorithm to determine whether a program is stratified, and if it is, to find a stratification for it.

---

**Stratified Datalog\( ^{\text{not}} \): query evaluation**

**Bottom-up evaluation**

1. compute a stratification of a program \( P \)
2. partition \( P \) into \( P_1, \ldots, P_n \) such that
   - each \( P_i \) consisting of all and only rules whose head belongs to a single stratum
   - \( P_1 \) is the lowest stratum
3. evaluate bottom-up \( P_1, \ldots, P_n \) (in that order).

**Result**

- does not depend on the stratification
- can be semantically characterized in various ways: minimal, perfect...
- is used to compute query results (like \( M_P \))
Expressiveness

Query equivalence
Two queries are equivalent if their semantics defines the same mapping from input databases to output results.

Query language containment
$L_1 \subseteq L_2$ if for every query $Q_1 \in L_1$, there is an equivalent query $Q_2$ in $L_2$.

Expressiveness
- Relational Algebra $\subseteq$ Stratified Datalog
- Datalog $\not\subseteq$ Relational Algebra
  - transitive closure
- Relational Algebra $\not\subseteq$ Datalog
  - set difference

Computational complexity

Decision problem
Is a tuple $t$ in the result $Q(D)$ of a query $Q$ applied to a database $D$?

Data complexity [Var82]
Complexity as a function of the cardinality of the database $D$:
- fixed: database schema, query $Q$
- input: database $D$

Combined complexity
Nothing considered fixed.

Theorem
Data complexity of Stratified Datalog queries is in PTIME.
Stable model semantics [GL88]

$M$ a subset of the Herbrand base of a Datalog

Reduct $P^M_g$

Obtained from $\text{ground}(P)$ by the Gelfond-Lifschitz transform:
- for every $A \in M$: remove every clause that contains $\text{not } A$ in the body
- for every $A \notin M$: remove $\text{not } A$ from the body of every clause in which it appears.

Stable model

$M$ is a stable model of $P$ if $M$ is the least (Herbrand) model of the reduct $P^M_g$.

Properties of stable models

- a program can have zero, one, or more stable models
- a stratified program has a single stable model computed by bottom-up evaluation.

Encoding propositional satisfiability [MT99]

Given a CNF formula $\phi$ with the set of clauses $C$ and the set of propositional variables $V$.

Set of facts $E_\phi$

- $\text{var}(a)$ for every $a \in V$
- $\text{clause}(c)$ for every $c \in C$
- $\text{pos}(c, v)$ if $v$ occurs positively in $c$
- $\text{neg}(c, v)$ if $v$ occurs negatively in $c$

Generating all possible truth assignments

(SAT1) $\text{true}(X):- \text{var}(X), \text{not } \text{false}(X)$.
(SAT2) $\text{false}(X):- \text{var}(X), \text{not } \text{true}(X)$.

Clause satisfaction

(SAT3) $\text{sat}(C):- \text{var}(X), \text{clause}(C), \text{true}(X), \text{pos}(C, X)$.
(SAT4) $\text{sat}(C):- \text{var}(X), \text{clause}(C), \text{false}(X), \text{neg}(C, X)$.
(SAT5) $f:- \text{clause}(C), \text{not } \text{sat}(C), \text{not } f$. 
Fact

M is a stable model of the program consisting of $E_{\phi}$ and (SAT1)–(SAT5) iff M contains exactly the following facts for some $U \subseteq V$:

- $E_{\phi}$
- $\text{sat}(c)$ for every $c \in C$
- $\text{true}(v)$ for every $v \in U$
- $\text{false}(v)$ for every $v \in V - U$.

Corollary

Data complexity of checking the existence of a stable model of a Datalog$^{\text{not}}$ program is NP-complete.

Querying using stable models

Query answer

A tuple $t$ is a cautious query answer if $\text{query}(t)$ belongs to every stable model of $P$.

Theorem

Data complexity of computing cautious answers to Datalog$^{\text{not}}$ queries is co-NP-complete.
Towards a Theory of Declarative Knowledge. 

M. Gelfond and V. Lifschitz. 
The Stable Model Semantics for Logic Programming. 

V. W. Marek and M. Truszczynski. 
Stable logic programming – an alternative logic programming paradigm. 
Also: CoRR cs.LO/9809032.

M. Y. Vardi. 
The Complexity of Relational Query Languages. 