Negation as Failure

Kyle Richardson

June 2022

A motivating example

Toy Student Database

```
Student(John)
Student(Mary)
Student(Sue)
Teacher(Julia)

Takes(John,C101)
Takes(Mary,C101)
Takes(Mary,C301)
Takes(Sue,C301)
Teaches(Julia,C101)

Math(C101)
Math(C301)
```

Example queries:

A motivating example

Toy Student Database

```
Student(John)
Student(Mary)
Student(Sue)
Teacher(Julia)

Takes(John,C101)
Takes(Mary,C101)
Takes(Mary,C301)
Takes(Sue,C301)
Teaches(Julia,C101)

Math(C101)
Math(C301)
```

Example queries: Is John taking C101? yes

A motivating example

Toy Student Database

```
Student(John)
Student(Mary)
Student(Sue)
Teacher(Julia)

Takes(John,C101)
Takes(Mary,C101)
Takes(Mary,C301)
Takes(Sue,C301)
Teaches(Julia,C101)

Math(C101)
Math(C301)
```

Example queries: What courses are taken by Mary? {C101, C301}

A motivating example

Toy Student Database

```
Student(John)
Student(Mary)
Student(Sue)
Teacher(Julia)

Takes(John,C101)
Takes(Mary,C101)
Takes(Mary,C301)
Takes(Sue,C301)
Teaches(Julia,C101)

Math(C101)
Math(C301)
```

Example queries: Is John taking C301? as far as we know, no

A motivating example

Toy Student Database

```
Student(John)
Student(Mary)
Student(Sue)
Teacher(Julia)

Takes(John,C101)
Takes(Mary,C101)
Takes(Mary,C301)
Takes(Sue,C301)
Teaches(Julia,C101)

Math(C101)
Math(C301)
```

Example queries: Is C505 not a math course? as far as we know, yes

A motivating example

Toy Student Database

```
Student(John)
Student(Mary)
Student(Sue)
Teacher(Julia)

Takes(John,C101)
Takes(Mary,C101)
Takes(Mary,C301)
Takes(Sue,C301)
Teaches(Julia,C101)

Math(C101)
Math(C301)
```

Example queries: Is Julia not a student? as far as we know, yes

A motivating example

Toy Student Database

```
Student(John)
Student(Mary)
Student(Sue)
Teacher(Julia)

Takes(John,C101)
Takes(Mary,C101)
Takes(Mary,C301)
Takes(Sue,C301)
Teaches(Julia,C101)

Math(C101)
Math(C301)
```

Operationalizing not and as far as we know in traditional databases (Codd, 1970): Assume that all tuples not in the database are false.

A motivating example

Toy Student Database

```
Student(John)
Student(Mary)
Student(Sue)
Teacher(Julia)

Takes(John,C101)
Takes(Mary,C101)
Takes(Mary,C301)
Takes(Sue,C301)
Teaches(Julia,C101)

Math(C101)
Math(C301)

Math-Teacher(X) ← Math(Y) & Teaches(X,Y)
```

Logic Programming: Not all information can not be read by direct inspection, *non-unit clauses*. (Lloyd, 2012)

A motivating example

Toy Student Database

```
Student(John)
Student(Mary)
Student(Sue)
Teacher(Julia)

Takes(John,C101)
Takes(Mary,C101)
Takes(Mary,C301)
Takes(Sue,C301)
Teaches(Julia,C101)

Math(C101)
Math(C301)

Math-Teacher(X) ← Math(Y) & Teaches(X,Y)
Non-Math-major(X) ← Math(Y) & not Takes(X,Y)
```

Logic Programming: Not all information can not be read by direct inspection, *non-unit clauses*. (Lloyd, 2012)

A motivating example

Toy Student Database

```
Student(John)
Student(Mary)
Student(Sue)
Teacher(Julia)

Takes(John,C101)
Takes(Mary,C101)
Takes(Mary,C301)
Takes(Sue,C301)
Teaches(Julia,C101)

Math(C101)
Math(C301)

Math-Teacher(X) ← Math(Y) & Teaches(X,Y)
Non-Math-major(X) ← Math(Y) & not Takes(X,Y)
```

Problem: not/as far as we know, not immediately amenable to classical logic.

A motivating example

Toy Student Database

```
Student(John)
Student(Mary)
Student(Sue)
Teacher(Julia)

Takes(John,C101)
Takes(Mary,C101)
Takes(Mary,C301)
Takes(Sue,C301)
Teaches(Julia,C101)

Math(C101)
Math(C301)

Math-Teacher(X) ← Math(Y) & Teaches(X,Y)
Non-Math-major(X) ← Math(Y) & not Takes(X,Y)
```

negation as failure: inference rule, *roughly*: derive inference $\neg A$ from failure to prove A.

A motivating example

Toy Student Database

```
Student(John)
Student(Mary)
Student(Sue)
Teacher(Julia)

Takes(John,C101)
Takes(Mary,C101)
Takes(Mary,C301)
Takes(Sue,C301)
Teaches(Julia,C101)

Math(C101)
Math(C301)

Math-Teacher(X) ← Math(Y) & Teaches(X,Y)
Non-Math-major(X) ← Math(Y) & not Takes(X,Y)
```

negation as failure: The inference rule that makes it possible to infer negative facts from our logic programs!

Problem: Logical Consequence and Negation in Logic

- Any formula α is a **logical consequence** of (or *logically entailed by*) a set of formulas $\Gamma = \{F_1, F_2, ...\}$

$$\Gamma \models \alpha$$

iff there is no interpretation in which Γ is **true** and α is **false**.

Problem: Logical Consequence and Negation in Logic

- Any formula α is a **logical consequence** of (or *logically entailed by*) a set of formulas $\Gamma = \{F_1, F_2, ...\}$

$$\Gamma \models \alpha$$

iff there is no interpretation in which Γ is **true** and α is **false**.

E.g., in $\Gamma = \{\texttt{Student(Mary)}, \texttt{Teacher(Julia)}\}$, we have:
$\Gamma \not\models \neg\texttt{Student(Julia)}$ and $\Gamma \not\models \texttt{Student}(\textit{Julia})$

Problem: Logical Consequence and Negation in Logic

- Any formula α is a **logical consequence** of (or *logically entailed by*) a set of formulas $\Gamma = \{F_1, F_2, ...\}$

$$\Gamma \models \alpha$$

iff there is no interpretation in which Γ is **true** and α is **false**.

E.g., in $\Gamma = \{\texttt{Student(Mary)}, \texttt{Teacher(Julia)}\}$, we have:
$\Gamma \not\models \neg\texttt{Student(Julia)}$ and $\Gamma \not\models \texttt{Student}(\textit{Julia})$

From $\Gamma \not\models \alpha$ it **does not** **necessarily follow** that $\Gamma \models \neg\alpha$

Problem: Logical Consequence and Negation in Logic

- Any formula α is a **logical consequence** of (or *logically entailed by*) a set of formulas $\Gamma = \{F_1, F_2, ...\}$

$$\Gamma \models \alpha$$

iff there is no interpretation in which Γ is **true** and α is **false**.

E.g., in $\Gamma = \{\texttt{Student(Mary)}, \texttt{Teacher(Julia)}\}$, we have:
$\Gamma \not\models \neg\texttt{Student(Julia)}$ and $\Gamma \not\models \texttt{Student}(\textit{Julia})$

From $\Gamma \not\models \alpha$ it **does not necessarily follow** that $\Gamma \models \neg\alpha$

database context: We want to conclude $\neg\texttt{Student(Julia)}$, but this is not a **sound** inference w.r.t. classical logic.

Logical Consequence and Negation in Classical Logic

- To convince ourselves of this, we can recall the following correspondence between entailment and satisfiability (Davis et al., 1994)[p.235,Thrm2.1]:

Theorem : $\Gamma = \{F_1, ...F_n\} \models \alpha \Leftrightarrow F_1 \wedge .. \wedge F_n \wedge \neg\alpha$ is **unsatisfiable**

Logical Consequence and Negation in Classical Logic

- To convince ourselves of this, we can recall the following correspondence between entailment and satisfiability (Davis et al., 1994)[p.235,Thrm2.1]:

Theorem : $\Gamma = \{F_1, ...F_n\} \models \alpha \Leftrightarrow F_1 \wedge .. \wedge F_n \wedge \neg\alpha$ is **unsatisfiable**

```
## pip install python-sat
from pysat.solvers import Glucose3
### Gamma = [Student(Mary), Teacher(Julia)],

### Does ~Student(Julia) follow from Gamma?
try1 = Glucose3()
try1.add_clause([1])  ## 1=F_1=Student(Mary)
try1.add_clause([2])  ## 2=F_2=Teacher(Julia)
try1.add_clause([3])  ## 3=alpha=Student(Julia)
### conjunctive formula: 1 & 2 & 3
print(try1.solve())#=> 'sat', i.e., G={1,2} |/= not 3

### Does Student(Julia) follow from Gamma?
try2 = Glucose3()
try2.add_clause([1])  ## 1=F_1=Student(Mary)
try2.add_clause([2])  ## 2=F_1=Teacher(Julia)
try2.add_clause([-3]) ## -3=not Student(Julia)
### conjunctive formula: 1 & 2 & ~3
print(try2.solve())#=> 'sat', i.e., G={1,2} |/=  3
```

Logical Consequence and Negation in Classical Logic

- To convince ourselves of this, we can recall the following correspondence between entailment and satisfiability (Davis et al., 1994)[p.235,Thrm2.1]:

Theorem : $\Gamma = \{F_1, ...F_n\} \models \alpha \Leftrightarrow F_1 \wedge .. \wedge F_n \wedge \neg\alpha$ is **unsatisfiable**

```
## pip install python-sat
from pysat.solvers import Glucose3

try1 = Glucose3()
try1.add_clause([1])  ## 1=Student(Mary)
try1.add_clause([2])  ## 2=Teacher(Julia)
try1.add_clause([3,-3])  ## 3 can be  true or false

### enumerate all possible interpretations
for interpretation in try1.enum_models():
    print(interpretation)
# [1, 2, -3]
# [1, 2, 3]
# neither 3 nor -3 occurs in every model
```

Logical Consequence and Negation in Classical Logic

- To convince ourselves of this, we can recall the following correspondence between entailment and satisfiability (Davis et al., 1994)[p.235,Thrm2.1]:

Theorem : $\Gamma = \{F_1, ...F_n\} \models \alpha \Leftrightarrow F_1 \wedge .. \wedge F_n \wedge \neg\alpha$ is **unsatisfiable**

```
## pip install python-sat
from pysat.solvers import Glucose3

try1 = Glucose3()
try1.add_clause([1])  ## 1=Student(Mary)
try1.add_clause([2])  ## 2=Teacher(Julia)
try1.add_clause([3,-3])  ## 3 can be  true or false

### enumerate all possible interpretations
for interpretation in try1.enum_models():
    print(interpretation)
# [1, 2, -3]
# [1, 2, 3]
# neither 3 nor -3 occurs in every model
```

The point: Additional machinery is needed to operationalize not/as far as we know.

Logic Programming: Some Reminders

- **Declarative Semantics** (What?): Based on subsets of first-order logic (FOL), focus here: **Horn subset** and **definite clauses**:

$$\underbrace{A_0}_{\text{head}} \leftarrow \underbrace{A_1, ..., A_n}_{\text{body}} (n \geq 0) \quad \text{(clausal form } \forall \quad \underbrace{A_0}_{\text{pos. literal}} \quad \vee \quad \underbrace{\neg A_1}_{\text{neg. literal}} \quad \vee ... \vee \neg A_n)$$

consisting of a *single* positive literal, no negation. **Definite program** P: any set of definite clauses; **Normal programs**: allow negation in body.

Logic Programming: Some Reminders

- **Declarative Semantics** (What?): Based on subsets of first-order logic (FOL), focus here: **Horn subset** and **definite clauses**:

$$\underbrace{A_0}_{\text{head}} \leftarrow \underbrace{A_1, ..., A_n}_{\text{body}} (n \geq 0) \quad \text{(clausal form } \forall \quad \underbrace{A_0}_{\text{pos. literal}} \quad \vee \quad \underbrace{\neg A_1}_{\text{neg. literal}} \quad \vee ... \vee \neg A_n)$$

consisting of a *single* positive literal, no negation. **Definite program** P: any set of definite clauses; **Normal programs**: allow negation in body.

- **Procedural Semantics** (How?): Want to answer a **goal** query:

$$\leftarrow G_1, ... G_m \qquad \text{(clausal form } \forall \neg G_1 \vee ... \vee \neg G_m)$$

computed using special case of Resolution (Robinson, 1965), **SLD-resolution**. **aim** for program P:

$$P \models \exists (G_1 \wedge ... \wedge G_m) \underbrace{\theta_1 ... \theta_m}_{\text{answers}}$$

Closed World Assumption (CWA): Intuition

- The general form of the closed-world assumption (CWA, (Reiter, 1981)) for any FOL theory Γ:

$$\mathrm{CWA}(\Gamma) = \Gamma \cup \left\{ \neg\alpha \mid \ \alpha \text{ is ground formula s.t. } \Gamma \not\models \alpha \right\},$$

Closed World Assumption (CWA): Intuition

- The general form of the closed-world assumption (CWA, (Reiter, 1981)) for any FOL theory Γ:

$$\mathrm{CWA}(\Gamma) = \Gamma \cup \left\{ \neg\alpha \mid \ \alpha \text{ is ground formula s.t. } \Gamma \not\models \alpha \right\},$$

  in other words:

Closed World Assumption (CWA): Intuition

- The general form of the closed-world assumption (CWA, (Reiter, 1981)) for any FOL theory Γ:

$$\mathrm{CWA}(\Gamma) = \Gamma \cup \left\{ \neg\alpha \mid \ \alpha \text{ is ground formula s.t. } \Gamma \not\models \alpha \right\},$$

in other words:

From $\Gamma \not\models \alpha$ we **will conclude that** $\Gamma \models \neg\alpha$

Closed World Assumption (CWA): Intuition

- The general form of the closed-world assumption (CWA, (Reiter, 1981)) for any FOL theory Γ:

$$\mathrm{CWA}(\Gamma) = \Gamma \cup \left\{ \neg\alpha \mid \ \alpha \text{ is ground formula s.t. } \Gamma \not\models \alpha \right\},$$

in other words:

From $\Gamma \not\models \alpha$ we **will conclude that** $\Gamma \models \neg\alpha$

E.g., in $\Gamma = \{$`Student(Mary)`, `Teacher(Julia)`, `Teaches(Julia,C101)`, `Math(C101)`, `Math-teacher(`X`)` $\leftarrow$ `Math(Y)`, `Teaches(X,Y)`$\}$

Closed World Assumption (CWA): Intuition

- The general form of the closed-world assumption (CWA, (Reiter, 1981)) for any FOL theory Γ:

$$\mathrm{CWA}(\Gamma) = \Gamma \cup \left\{ \neg\alpha \mid \ \alpha \text{ is ground formula s.t. } \Gamma \not\models \alpha \right\},$$

in other words:

From $\Gamma \not\models \alpha$ we **will conclude that** $\Gamma \models \neg\alpha$

E.g., in $\Gamma = \{$`Student(Mary)`, `Teacher(Julia)`, `Teaches(Julia,C101)`, `Math(C101)`, `Math-teacher(`X`)` $\leftarrow$ `Math(Y)`, `Teaches(X,Y)`$\}$,

$$\mathrm{CWA}(\Gamma) = \Gamma \cup \{\neg\texttt{Teacher(Mary)}, \neg\texttt{Student(Julia)}, \neg\texttt{Teaches(Mary,C101)}, \neg\texttt{Math-teacher(Mary)}, \ldots\}$$

Closed World Assumption (CWA): Properties and Problems

- An important property of classical logic, **monotonicity**:

Given any theories Γ, Γ' s.t. $\Gamma \subseteq \Gamma'$, forall α : $\quad \Gamma \models \alpha$ implies $\Gamma' \models \alpha$

Closed World Assumption (CWA): Properties and Problems

- An important property of classical logic, **monotonicity**:

Given any theories Γ, Γ' s.t. $\Gamma \subseteq \Gamma'$, forall α : $\quad \Gamma \models \alpha$ implies $\Gamma' \models \alpha$

This is lost with CWA(**non-monotonic**), e.g.,
$\Gamma = \{\texttt{Student(Mary)}, \texttt{Teacher(Julia)}, ...\}$, $\Gamma' = \Gamma \cup \{\texttt{Teacher(Mary)}\}$
we have that

$$\mathrm{CWA}(\Gamma) \models \neg\texttt{Teacher(Mary)} \text{ yet } \mathrm{CWA}(\Gamma') \not\models \neg\texttt{Teacher(Mary)}$$

Closed World Assumption (CWA): Properties and Problems

- An important property of classical logic, **monotonicity**:

Given any theories Γ, Γ' s.t. $\Gamma \subseteq \Gamma'$, forall α : $\quad \Gamma \models \alpha$ implies $\Gamma' \models \alpha$

This is lost with CWA(**non-monotonic**), e.g.,
$\Gamma = \{\texttt{Student(Mary)}, \texttt{Teacher(Julia)}, ...\}$, $\Gamma' = \Gamma \cup \{\texttt{Teacher(Mary)}\}$
we have that

$$\mathrm{CWA}(\Gamma) \models \neg\texttt{Teacher(Mary)} \text{ yet } \mathrm{CWA}(\Gamma') \not\models \neg\texttt{Teacher(Mary)}$$

- Bigger Problem: FOL (even restricted to horn clauses) is undecidable (i.e., no general algorithm can show $\Gamma \not\models \alpha$ in finite-time), not possible to compute $\mathrm{CWA}(\Gamma)$ from Γ.

Negation as (finite) Failure

- A form of CWA with emphasis on *finite failure*:

$$\mathrm{CWA}_{\text{finite}} = \Gamma \cup \left\{ \neg\alpha \mid \text{ground } \alpha \text{ s.t. all attempts at } \Gamma \models \alpha \text{ (finitely) fail} \right\},$$

weaker than CWA, $\mathrm{CWA}_{\text{finite}}(\Gamma) \subseteq \mathrm{CWA}(\Gamma)$.

Negation as (finite) Failure

- A form of CWA with emphasis on *finite failure*:

$$\mathrm{CWA}_{\text{finite}} = \Gamma \cup \left\{ \neg\alpha \mid \text{ground } \alpha \text{ s.t. all attempts at } \Gamma \models \alpha \text{ (finitely) fail} \right\},$$

weaker than CWA, $\mathrm{CWA}_{\text{finite}}(\Gamma) \subseteq \mathrm{CWA}(\Gamma)$.

Operationally: *flip* notion of success and failure (Lloyd, 2012); e.g.,
$\Gamma = \{$`Student(Mary)`, `Teacher(Julia)`, `Teaches(Julia,C101)`, `Math(C101)`, `Math-teacher(`X`)` $\leftarrow$ `Math(Y)`, `Teaches(X,Y)`$\}$

Negation as (finite) Failure

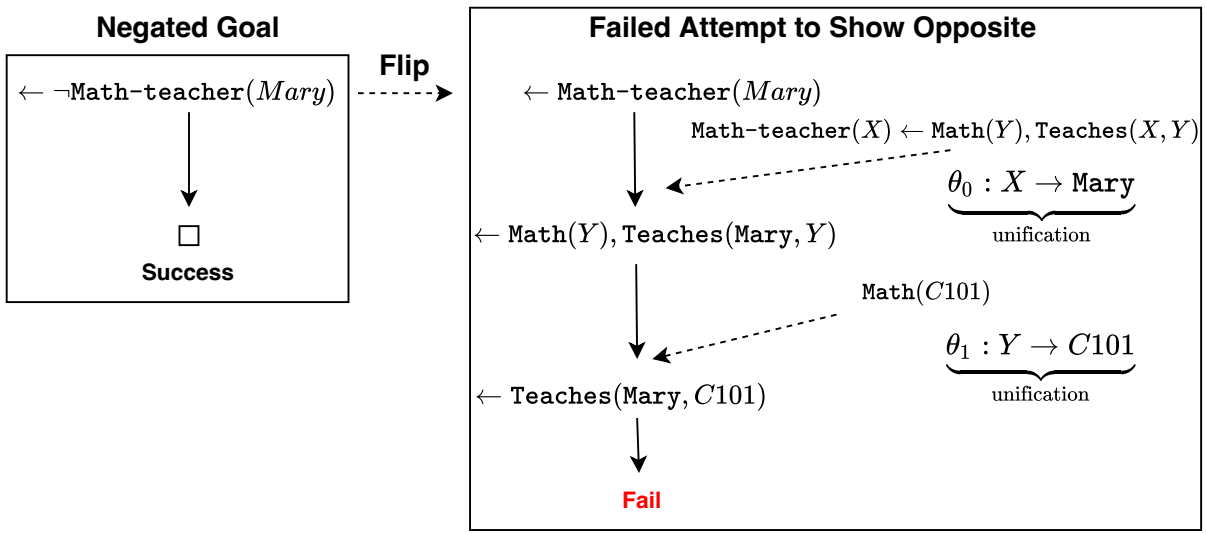
- A form of CWA with emphasis on *finite failure*:

$$\mathrm{CWA}_{\text{finite}} = \Gamma \cup \left\{ \neg\alpha \mid \text{ground } \alpha \text{ s.t. all attempts at } \Gamma \models \alpha \text{ (finitely) fail} \right\},$$

weaker than CWA, $\mathrm{CWA}_{\text{finite}}(\Gamma) \subseteq \mathrm{CWA}(\Gamma)$.

Operationally: *flip* notion of success and failure (Lloyd, 2012); e.g.,

$\Gamma = \{$`Student(Mary)`, `Teacher(Julia)`, `Teaches(Julia,C101)`, `Math(C101)`, `Math-teacher(`X`)` $\leftarrow$ `Math(Y)`, `Teaches(X,Y)`$\}$



Negation as (finite) Failure: How to operationalize

- **SLD-resolution**: resolution inference rule for *definite programs*:

$$\frac{\overbrace{\leftarrow G_1, \ldots, G_{i-1}, \underbrace{G_i}_{\text{selected}}, \ldots, G_m}^{\text{goal}}\ (m \geq 0) \qquad \overbrace{\underbrace{A_0}_{\text{selected}} \leftarrow A_1, \ldots, A_n}^{\text{selected clause}}\ (n \geq 0)}{\leftarrow G_1, \ldots, G_{i-1}, A_1, \ldots, A_n, \ldots G_m} \quad \underbrace{G_i.\theta = A_0.\theta}_{\text{unifyable}}$$

Negation as (finite) Failure: How to operationalize

- **SLD-resolution**: resolution inference rule for *definite programs*:

$$\frac{\overbrace{\leftarrow G_1, \ldots, G_{i-1}, \underbrace{G_i}_{\text{selected}}, \ldots, G_m}^{\text{goal}}\ (m \geq 0) \qquad \overbrace{\underbrace{A_0}_{\text{selected}} \leftarrow A_1, \ldots, A_n}^{\text{selected clause}}\ (n \geq 0)}{\leftarrow G_1, \ldots, G_{i-1}, A_1, \ldots, A_n, \ldots G_m} \quad \underbrace{G_i.\theta = A_0.\theta}_{\text{unifyable}}$$

coupled with selection and search method; permits inferring *positive information* when repeated application leads to the *empty goal*.

Negation as (finite) Failure: How to operationalize

- **SLD-resolution**: resolution inference rule for *definite programs*:

$$\frac{\overbrace{\leftarrow G_1, \ldots, G_{i-1}, \underbrace{G_i}_{\text{selected}}, \ldots, G_m}^{\text{goal}}\ (m \geq 0) \qquad \overbrace{\underbrace{A_0}_{\text{selected}} \leftarrow A_1, \ldots, A_n}^{\text{selected clause}}\ (n \geq 0)}{\leftarrow G_1, \ldots, G_{i-1}, A_1, \ldots, A_n, \ldots G_m} \underbrace{G_i.\theta = A_0.\theta}_{\text{unifyable}}$$

  coupled with selection and search method; permits inferring *positive information* when repeated application leads to the *empty goal*.

- **SLDNF-resolution**: SLD-resolution extended with additional rule for *negation as failure*, used in Prolog:

$$\frac{\leftarrow G_1, \ldots, G_{i-1}, \overbrace{\neg G_i}^{\text{negation}}, \ldots, G_m}{G_1, \ldots, G_{i-1}, \ldots, G_m}$$ all attempts at G_i (finitely) fail

Negation as (finite) Failure: How to operationalize

- **SLD-resolution**: resolution inference rule for *definite programs*:

$$\frac{\overbrace{\leftarrow G_1, \ldots, G_{i-1}, \underbrace{G_i}_{\text{selected}}, \ldots, G_m}^{\text{goal}}\ (m \geq 0) \qquad \overbrace{\underbrace{A_0}_{\text{selected}} \leftarrow A_1, \ldots, A_n}^{\text{selected clause}}\ (n \geq 0)}{\leftarrow G_1, \ldots, G_{i-1}, A_1, \ldots, A_n, \ldots G_m} \underbrace{G_i.\theta = A_0.\theta}_{\text{unifyable}}$$

  coupled with selection and search method; permits inferring *positive information* when repeated application leads to the *empty goal*.

- **SLDNF-resolution**: SLD-resolution extended with additional rule for *negation as failure*, used in Prolog:

$$\frac{\leftarrow G_1, \ldots, G_{i-1}, \overbrace{\neg G_i}^{\text{negation}}, \ldots, G_m}{G_1, \ldots, G_{i-1}, \ldots, G_m}\ \text{all attempts at } G_i \text{ (finitely) fail}$$

Question: Can we reconcile this with logic, does it make sense?

Negation as (finite) Failure: Semantics

Clark (1978) gave one of the earliest declarative semantics for negation as failure, based on notion of **program completion**.

Keith L. Clark

Department of Computer Science & Statistics

Queen Mary College, London, England

ABSTRACT

A query evaluation process for a logic data base comprising a set of clauses is described. It is essentially a Horn clause theorem prover augmented with a special inference rule for dealing with negation. This is the negation as failure inference rule whereby ~ P can be inferred if every possible proof of P fails. The chief advantage of the query evaluator described is the effeciency with which it can be implemented. Moreover, we show that the negation as failure rule only allows us to conclude negated facts that could be inferred from the axioms of the completed data base, a data base of relation definitions and equality schemas that we consider is implicitly given by the data base of clauses. We also show that when the clause data base and the queries satisfy certain constraints, which still leaves us with a data base more general than a conventional relational data base, the query evaluation process will find every answer that is a logical consequence of the completed data base.

Negation as (finite) Failure: Semantics

- **Intuition**: When using theory/program Γ with CWA, we *actually* consider the **completion** of Γ, $\textsc{Comp}(\Gamma)$.

Negation as (finite) Failure: Semantics

- **Intuition**: When using theory/program Γ with CWA, we *actually* consider the **completion** of Γ, $\textsc{Comp}(\Gamma)$.

  E.g., $\Gamma = \{\texttt{Student}(\mathit{Mary}), \texttt{Teacher}(\mathit{Julia})\}$

  $\textsc{Comp}(\Gamma) = \Gamma \cup \{\texttt{Student}(X) \leftrightarrow \underbrace{X = \mathit{Mary}}_{\text{equality pred.}}, \texttt{Teacher}(X) \leftrightarrow \underbrace{\texttt{X = Julia}}_{\text{equality}}\}$

Negation as (finite) Failure: Semantics

- **Intuition**: When using theory/program Γ with CWA, we *actually* consider the **completion** of Γ, $\textsc{Comp}(\Gamma)$.

E.g., $\Gamma = \{\texttt{Student}(\mathit{Mary}), \texttt{Teacher}(\mathit{Julia})\}$

$$\textsc{Comp}(\Gamma) = \Gamma \cup \left\{\texttt{Student}(X) \leftrightarrow \underbrace{X = \mathit{Mary}}_{\text{equality pred.}}, \texttt{Teacher}(X) \leftrightarrow \underbrace{\texttt{X = Julia}}_{\text{equality}}\right\}$$

```
compl_G.add_clause([1])  ## Gamma = [ 1=Student(Mary),
compl_G.add_clause([2])  ## 2=Teacher(Julia) ]
   ## queries = [ 3=Student(Julia),  4=Teacher(Mary) ]
## 5 = (Mary = Julia) = (Julia = Mary) = False
compl_G.add_clause([-5])
### Student(Julia) <-> Julia=Mary
compl_G.add_clause([-3,5]) # 3 -> 5
compl_G.add_clause([-5,3]) # 5 -> 3
### Teacher(Mary) <-> Mary = Julia
compl_G.add_clause([-4,5]) # 4 -> 5
compl_G.add_clause([-5,4]) # 5 -> 4
for interp in compl_G.enum_models(): print(interp)
# [1, 2, -3, -4, -5]
```

Negation as (finite) Failure: Semantics

- **Intuition**: When using theory/program Γ with CWA, we *actually* consider the **completion** of Γ, $\textsc{Comp}(\Gamma)$.

E.g., $\Gamma = \{\texttt{Student}(\mathit{Mary}), \texttt{Teacher}(\mathit{Julia})\}$

$$\textsc{Comp}(\Gamma) = \Gamma \cup \Big\{\texttt{Student}(X) \leftrightarrow \underbrace{X = \mathit{Mary}}_{\text{equality pred.}}, \texttt{Teacher}(X) \leftrightarrow \underbrace{\texttt{X = Julia}}_{\text{equality}}\Big\}$$

```
compl_G.add_clause([1])  ## Gamma = [ 1=Student(Mary),
compl_G.add_clause([2])  ## 2=Teacher(Julia) ]
   ## queries = [ 3=Student(Julia),  4=Teacher(Mary) ]
## 5 = (Mary = Julia) = (Julia = Mary) = False
compl_G.add_clause([-5])
### Student(Julia) <-> Julia=Mary
compl_G.add_clause([-3,5]) # 3 -> 5
compl_G.add_clause([-5,3]) # 5 -> 3
### Teacher(Mary) <-> Mary = Julia
compl_G.add_clause([-4,5]) # 4 -> 5
compl_G.add_clause([-5,4]) # 5 -> 4
for interp in compl_G.enum_models(): print(interp)
# [1, 2, -3, -4, -5]
```

Result: $\textsc{Comp}(\Gamma) \models \neg \mathit{Student}(\mathit{Julia})$ and $\textsc{Comp}(\Gamma) \models \neg \mathit{Teacher}(\mathit{Mary})$

Negation as (finite) Failure: Semantics

- **Intuition**: When using theory/program Γ with CWA, we *actually* consider the **completion** of Γ, $\textsc{Comp}(\Gamma)$.

E.g., $\Gamma = \{\texttt{Student}(\mathit{Mary}), \texttt{Teacher}(\mathit{Julia})\}$

$$\textsc{Comp}(\Gamma) = \Gamma \cup \{\texttt{Student}(X) \leftrightarrow \underbrace{X = \mathit{Mary}}_{\text{equality pred.}}, \texttt{Teacher}(X) \leftrightarrow \underbrace{\texttt{X = Julia}}_{\text{equality}}\}$$

```
compl_G.add_clause([1])  ## Gamma = [ 1=Student(Mary),
compl_G.add_clause([2])  ## 2=Teacher(Julia) ]
   ## queries = [ 3=Student(Julia),  4=Teacher(Mary) ]
## 5 = (Mary = Julia) = (Julia = Mary) = False
compl_G.add_clause([-5])
### Student(Julia) <-> Julia=Mary
compl_G.add_clause([-3,5]) # 3 -> 5
compl_G.add_clause([-5,3]) # 5 -> 3
### Teacher(Mary) <-> Mary = Julia
compl_G.add_clause([-4,5]) # 4 -> 5
compl_G.add_clause([-5,4]) # 5 -> 4
for interp in compl_G.enum_models(): print(interp)
# [1, 2, -3, -4, -5]
```

Possible to reconcile with classical logic and $\neg$.

Clark Completion Rules

- **Clark Completion** (Clark, 1978): standard set of completion rules coupled with *equality axioms* (special cases excluded).

Clark Completion Rules

- **Clark Completion** (Clark, 1978): standard set of completion rules coupled with *equality axioms* (special cases excluded).

  - **1. Variable Restrictions**: For all **facts** $A(t_1, ..., t_o) \leftarrow \underline{\text{add}}$

$$A(x_1, ..., x_o) \leftarrow x_1 = t_2 \wedge ... \wedge x_o = t_o$$

Clark Completion Rules

- **Clark Completion** (Clark, 1978): standard set of completion rules coupled with *equality axioms* (special cases excluded).

  - **1. Variable Restrictions**: For all **facts** $A(t_1, ..., t_o) \leftarrow$ <u>add</u>

  $$A(x_1, ..., x_o) \leftarrow x_1 = t_2 \wedge ... \wedge x_o = t_o$$

  - **2. Material Equivalences**: for all **rules** $A(X) \leftarrow A'_i$ $(\forall i \leq k)$ <u>add</u>:

  $$A(X) \leftrightarrow A'_1 \vee A'_2 \vee .. \vee A'_k :$$

Clark Completion Rules

- **Clark Completion** (Clark, 1978): standard set of completion rules coupled with *equality axioms* (special cases excluded).

- **1. Variable Restrictions**: For all **facts** $A(t_1, ..., t_o) \leftarrow$ add

$$A(x_1, ..., x_o) \leftarrow x_1 = t_2 \wedge ... \wedge x_o = t_o$$

- **2. Material Equivalences**: for all **rules** $A(X) \leftarrow A'_i$ $(\forall i \leq k)$ add:

$$A(X) \leftrightarrow A'_1 \vee A'_2 \vee .. \vee A'_k :$$

- **3. Equality**: add $\neg(t_i = t_j)$ for all non-unifiable terms t_i, t_j. **Undefined predicates**: negate any k-ary predicate A in body not included as head:

$$\neg A(X_1, .., X_k)$$

Clark Completion Rules: Example

E.g., $\Gamma = \{\texttt{Student(Mary)}, \texttt{Teacher(Julia)}, \texttt{Teaches(Julia,C301)}, \texttt{Math(C101)}, \texttt{Math-teacher}(X) \leftarrow \texttt{Math}(Y), \texttt{Teaches}(X, Y)\}$

$$\textsc{Comp}(\Gamma) = \Gamma \cup \Big\{$$
$$\texttt{Student}(X) \leftrightarrow \texttt{X=Mary},$$
$$\texttt{Teacher}(X) \leftrightarrow \texttt{X=Julia},$$
$$\texttt{Math}(X) \leftrightarrow \texttt{X=C101},$$
$$\texttt{Math-teacher}(X) \leftrightarrow \texttt{Math}(Y), \texttt{Teaches}(X, Y),$$
$$\texttt{Teaches(X,Y)} \leftrightarrow \texttt{X=Julia}, \texttt{Y=C301},$$
$$\neg(\texttt{Mary} = \texttt{Julia}), \neg(\texttt{Mary} = \texttt{C101}), \neg(\texttt{Julia} = \texttt{C101})$$
$$\Big\}$$

Clark Completion Rules: Example

E.g., $\Gamma = \{\texttt{Student(Mary)}, \texttt{Teacher(Julia)}, \texttt{Teaches(Julia,C301)}, \texttt{Math(C101)}, \texttt{Math-teacher}(X) \leftarrow \texttt{Math}(Y), \texttt{Teaches}(X, Y)\}$

$$\textsc{Comp}(\Gamma) = \Gamma \cup \Big\{ \begin{array}{l} \texttt{Student}(X) \leftrightarrow \texttt{X=Mary}, \\ \texttt{Teacher}(X) \leftrightarrow \texttt{X=Julia}, \\ \texttt{Math}(X) \leftrightarrow \texttt{X=C101}, \\ \texttt{Math-teacher}(X) \leftrightarrow \texttt{Math}(Y), \texttt{Teaches}(X, Y), \\ \texttt{Teaches(X,Y)} \leftrightarrow \texttt{X=Julia}, \texttt{Y=C301}, \\ \neg(\texttt{Mary} = \texttt{Julia}), \neg(\texttt{Mary} = \texttt{C101}), \neg(\texttt{Julia} = \texttt{C101}) \end{array} \Big\}$$

Note: used as a declarative semantics for negation as failure, can be used directly (**computable**) with general FOL/SAT solvers.

Properties of Clark Completion and Negation as Failure

Some important properties of completion, stated for *definite programs*.

- **Completion extends** Γ, i.e., $\Gamma \models \alpha$ implies $\textsc{Comp}(\Gamma) \models \alpha$ for any α.
- **Adds no new positive information**, i.e., If $\textsc{Comp}(\Gamma) \models \alpha$ holds for an atomic formula α, then $\Gamma \models \alpha$.

Properties of Clark Completion and Negation as Failure

Some important properties of completion, stated for *definite programs*.

- **Completion extends** Γ, i.e., $\Gamma \models \alpha$ implies $\text{COMP}(\Gamma) \models \alpha$ for any α.
- **Adds no new positive information**, i.e., If $\text{COMP}(\Gamma) \models \alpha$ holds for an atomic formula α, then $\Gamma \models \alpha$.

Relating our procedural approach (*SLDNF*-resolution) to Clark completion, sound for COMP w.r.t success and failure (Clark, 1978).

- If α *succeeds* (*precise definition omitted*) from program/theory Γ via SLDNF with answer θ, then $\text{COMP}(\Gamma) \models \alpha\theta$.
- If α *fails* from program/theory Γ via SLDNF, then $\text{COMP}(\Gamma) \models \neg\alpha$.

Conclusion and Summary

- **Problem**: Deriving negative information from logic programs/databases.
  negation as failure: negate things we can't prove, general **closed-world assumption** (CWA), non-monotonic inference rule.

  Negation as (finite) failure: weaker form of CWA, **procedurally**: extension of SLD-resolution, prove negation by exhaustively showing finite failure to prove opposite.

  declarative semantics in terms of **Clark completion**.

Credits and Additional Reading

- Many ideas and examples taken from the following (see for more details):

  Lecture notes: Richard Mayr: https://www.inf.ed.ac.uk/teaching/courses/lp/2014/slides/lpTheory7.pdf, Marin Mircea: https://staff.fmi.uvt.ro/~mircea.marin/lectures/LFP/FoundationsOfLP.pdf, Vladimir Lifshitz https://www.cs.utexas.edu/users/vl/teaching/lbai/completion.pdf, Marek Sergot: https://www.doc.ic.ac.uk/~mjs/teaching/KnowledgeRep491/NBF_491-2x1.pdf

  General references *general logic programming* (Lloyd, 2012), *SLDNF-resolution* (Nienhuys-Cheng and Wolf, 1997), *NAF* (Clark, 1978; Shepherdson, 1998).

Thank you.

References I

Clark, K. L. (1978). Negation as failure. In *Logic and data bases*, pages 293–322. Springer.

Codd, E. F. (1970). A relational model of data for large shared data banks. In *Communications of ACM*.

Davis, M., Sigal, R., and Weyuker, E. J. (1994). *Computability, complexity, and languages: fundamentals of theoretical computer science*. Elsevier.

Lloyd, J. W. (2012). *Foundations of logic programming*. Springer Science & Business Media.

Nienhuys-Cheng, S.-H. and Wolf, R. (1997). *SLDNF-resolution*, pages 127–159. Springer Berlin Heidelberg, Berlin, Heidelberg.

Reiter, R. (1981). On closed world data bases. In *Readings in artificial intelligence*, pages 119–140. Elsevier.

Robinson, J. A. (1965). A machine-oriented logic based on the resolution principle. *Journal of the ACM (JACM)*, 12(1):23–41.

Shepherdson, J. (1998). Negation as failure, completion and stratification in D. Gabbay et al. *Handbook of Logic in AI and Logic Programming*, pages 356–409.