4. Adding Explicit Negation

1. **Why two types of negation?**
   Explicit vs. classical negation
2. **Explicit negation for STABLE**
3. **Explicit negation for WFS**

Why two types of negation?

Normal logic programs answer YES or NOto every query.

Closed World Assumption applied to all predicates, i.e.,
each ground atom which is not provable is assumed false
no atom is left undecided

Adequate for domains where completeness of positive information can be assumed, e.g., database of flight connections

Inadequate in many other domains:

cross-tracks <- not train

may lead to disasterous results => use different negation not assumed to hold by default

cross-tracks <- ¬ train

Extending the language

$c \leftarrow a_1, ..., a_n, \text{not } b_1, ..., \text{not } b_m$

where a_i, bj, c are LITERALS, i.e. ¬ may appear in front of an atom "not" as before default negation. "¬" explicit negation

Note that "¬" does not behave like classical negation:

Law of Excluded Middle (A v ¬ A) does not hold

The program

hire <- qualified
reject <- ¬ qualified

does not derive hire v reject

Semantics

Let P be a logic program, S a set of literals. The S reduct of P, P^S, is the program obtained from P by

1. **deleting all rules with not L in the body, for some L ∈ S.**
2. **deleting all weakly negated literals from the remaining rules.**

The set of consequences Cn(P) of a program P without weak negation is the smallest set of literals which is

1. **closed under P, and**
2. **logically closed, i.e. consistent or equal to the set of all literals**

A set of atoms N is a <u>stable model</u> of a normal logic program P iff N is the minimal model of P^N.

A set of literals N is an <u>answer set</u> of an extended logic program P iff N is the set of consequences of P^N.

More concise characterization:

Define the following operator:

$$\boxed{\gamma_P(X) = Cn(P^X)}$$

S answer set of P iff:

$$\boxed{\gamma_P(S) = S}$$

Example 1

P:	**eligible**	**<-**	**highGPA**
	eligible	**<-**	**minority, fairGPA**
	¬ eligible	**<-**	**¬ fairGPA, ¬ highGPA**
	interview	**<-**	**not eligible, not ¬ eligible**
	fairGPA	**<-**	

single answer set of P: S = {fairGPA, interview}

S-reduct of P:	**eligible**	**<-**	**highGPA**
	eligible	**<-**	**minority, fairGPA**
	¬ eligible	**<-**	**¬ fairGPA, ¬ highGPA**
	interview	**<-**	
	fairGPA	**<-**	

Cn(P^S) = S

Example 2

P':	**eligible**	**<-**	**highGPA**
	eligible	**<-**	**minority, fairGPA**
	¬ eligible	**<-**	**¬ fairGPA, ¬ highGPA**
	interview	**<-**	**not eligible, not ¬ eligible**
	fairGPA	**<-**	
	minority	**<-**	

single answer set of P': S' = {fairGPA, minority, eligible}

S-reduct of P':	**eligible**	**<-**	**highGPA**
	eligible	**<-**	**minority, fairGPA**
	¬ eligible	**<-**	**¬ fairGPA, ¬ highGPA**
	~~**interview**~~	~~**<-**~~	~~**not eligible, not ¬ eligible**~~
	fairGPA	**<-**	
	minority	**<-**	

Cn(P'^S') = S'

Properties of answer sets

Let P be an extended logic program.

Exactly one of the following three statements is true:

P has no answer set,
the only answer set of P is the inconsistent set Lit,
P has an answer set, and all its answer sets are consistent.

If X and Y are answer sets of P and X $\subseteq$ Y then X = Y.

Each element of a consistent answer set of P is a head literal of P.

If the set of head literals of P is consistent then all answer sets are consistent.

Elimination of explicit negation

for each predicate symbol p of P introduce symbol non-p representing ¬ p

generate program P' by replacing each occurrence of ¬ p with non-p

consistent set of literals S is answer set of P iff S' is stable model of P', where S' is obtained from S by replacing ¬ p with non-p

```
f     <-  b, not ab1
¬ f   <-  p, not ab2
ab1   <-  p
b     <-  p
p     <-
```

```
f      <-  b, not ab1
non-f  <-  p, not ab2
ab1    <-  p
b      <-  p
p      <-
```

answer set: {p, b, ¬ f, ab1}

stable model: {p, b, non-f, ab1}

WFS for extended logic programs

WFS three-valued. For simplicity we are only interested in true literals here, false literals left implicit (can be obtained in canonical way)

Use characterization of WFS based on operator γ.

Observation: γ is anti-monotone

Therefore: double application of γ defines monotone operator

Monotone operators have unique least fixpoint

Theorem:
for a general logic program P the literals contained in the least fixpoint of $\gamma\gamma$ are exactly those true in WFS(P)

γ already defined for extended logic programs

lfp can be reached by iterating on the empty set

WFS based on γ

$X \xrightarrow{\gamma} \gamma(X) \xrightarrow{\gamma} X' = \gamma\gamma(X)$

X
literals known
to be derivable

γ(X)
literals
potentially derivable
(not known
to be underivable)

X' = γγ(X)
literals known
to be derivable,
possibly larger

a <- b, c, not x1
d <- e, not y2
f <- h, not y9
....
r <- s, not x23

P

~~a <- b, c, not x1~~
d <- e, ~~not y2~~
f <- h, ~~not y9~~
....
~~r <- s, not x23~~

P^X

~~a <- b, c, not x1~~
d <- e, ~~not y2~~
~~f <- h, not y9~~
....
~~r <- s, not x23~~

P^γ(X)

Example

b <- not a
c <- not b
e <- not d
d <- not e

P

b <- ~~not a~~
c <- ~~not b~~
e <- ~~not d~~
d <- ~~not e~~

P^{}

b <- ~~not a~~
~~c <- not b~~
~~e <- not d~~
~~d <- not e~~

P^{b, c, d, e}

b <- ~~not a~~
~~c <- not b~~
e <- ~~not d~~
d <- ~~not e~~

P^{b}

b <- ~~not a~~
~~c <- not b~~
~~e <- not d~~
~~d <- not e~~

P^{b, d, e}

X	$\gamma_P(X)$	$\gamma_P^2(X)$
S0 = {}	{b, c, d, e}	{b}
S1 = {b}	{b, d, e}	{b}
S2 = S1		

Example 2 in WFS

P:			
	eligible	<-	highGPA
	eligible	<-	minority, fairGPA
	¬ eligible	<-	¬ fairGPA, ¬ highGPA
	interview	<-	not eligible, not ¬ eligible
	fairGPA	<-	
	minority	<-	

γ ({}) = {fairGPA, minority, interview, eligible}

lfp → γ^2({}) = {fairGPA, minority, eligible}

γ^3({}) = {fairGPA, minority, eligible}

γ^4({}) = {fairGPA, minority, eligible}

in this case lfp and single answer set coincide

Variants:

WFS sometimes very weak:

b <- not ¬ b
a <- not ¬ a
¬ a <- not a

WFS empty since γ({}) = Lit and γ(Lit) = {}
due to logical closedness of Cn

Define operator $\gamma*$(X) = Cl(P^X) where Cl is like Cn without logical closedness and consider monotone operator $\gamma\gamma*$.

We have $\gamma*$({}) = {a, ¬ a, b} and $\gamma\gamma*$({}) = {b}.

Variants 2:

Coherence principle violated: explicit negation does not imply default neg.

P:

b <- not a
a <- not b
¬ a <-

b not derived!

WFSX (Pereira et al.) transforms program P to seminormal form Sem(P) by adding to each rule body the default negation of the head complement and considers

$$\boxed{\gamma_P \gamma_{Sem(P)}{}^*}$$

Sem(P):

b <- not a, not ¬ b
a <- not b, not ¬ a
¬ a <- not a

$$\gamma_P \gamma_{Sem(P)}{}^*(\{\}) = \{\neg a, b\}$$

5. Logic Programs with Priorities

1. **Introduction: why preferences matter**
2. **Preferred Answer Sets**
3. **Dynamic Preferences: A Legal Reasoning Example**
4. **WFS with Dynamic Preferences**
5. **Conclusions**

5.1 Why preferences matter

Specificity:
in case of a conflict prefer the more specific rule

Model Based Diagnosis:
generate only the more likely diagnoses
technically: introduce normalcy assumptions with different priorities

Configuration:
usually not all constraints satisfiable
distinguish important ones from less important ones, i.e. introduce priorities

Legal Reasoning:
laws may be in conflict: introduce priorities

Example

P:
1) fly <- bird, not ¬fly
2) ¬fly <- peng, not fly
3) bird <- peng
4) peng

Answer sets: S1 = {peng, bird, fly}
S2 = {peng, bird, ¬ fly}

P^S1:
fly <- bird
bird <- peng
peng

Cn(P^S1) = S1

P^S2:
¬fly <- peng
bird <- peng
peng

Cn(P^S2) = S2

We would like to tell the system that rule 2 should win over rule 1 in case of a conflict: 2 < 1

5.2 Prioritized Logic Programs

A PLP is a pair (P,<), where P is an extended logic program, and < a strict partial order on P. r < r' means "r is preferred over r'.

How to use preference information:

- **more conclusions**
- **same conclusions as before whenever < is empty**
- **only conclusions which are in at least one answer set (in coherent case)**

==> extend < to a preference relation on answer sets

Remark: partial case can be reduced to total orders in canonical way: A is preferred answer set of (P,<) if A is preferred answer set of (P,<') for some strict total order <' containing <.

Principles for treating preferences

Principle I:

Let E_1 and E_2 be two belief sets (answer sets, extensions,...) generated by the rules $R \cup \{d_1\}$, respectively $R \cup \{d_2\}$. If d_1 is preferred over d_2 then E_2 is not maximally preferred.

Principle II:

Let E be a preferred belief set of a theory (T,<) such that E does not contain all prerequisites of a rule r. Then E is a preferred belief set of ($T \cup \{r\}$, <') whenever < and <' agree on rules in T.

Help from prioritized DL?

**Answer sets in 1-to-1-correspondence with default logic extensions.
Prioritized versions of DL exist: does this solve the problem?**

Two major approaches:

Type 1: **controlling generation of extensions
(Marek/Trusczcynski, Baader/Hollunder, Brewka)**

Type 2: **looking for best applied default rule
(Rintanen)**

Type 1

underlying idea: in each step of the construction of extensions apply best default rule which is applicable and so far unapplied.

1) A: B / B 2) true: ¬B / ¬B 3) true: A / A
1 < 2 < 3

Extensions: **E1 = Th({A, ¬B})
E2 = Th({A, B})**

Type 1-approaches prefer E1 violating Principle I!

Type 2

underlying idea: pick the extension in which the rule with highest priority was applied.

1) A: B / B **2) true: ¬A / ¬A** **3) true: A / A**
1 < 2 < 3

Extensions: E1 = Th({¬A})
E2 = Th({A, B})

Type 2-approach prefers E2 violating Principle II!

Preferred answer sets

Observation:

To check whether A is an answer set
- **use A to reduce the program to easy subcase (rules without not)**
- **check whether A is the set of consequences of the reduced program.**

Idea:

To check whether A is a preferred answer set
- **use A to reduce the program to easy subcase (rules without prerequisites)**
- **check whether A is preferred answer set of the reduced program.**

Checking priorities - the easy case

1) A <- not B 2) B <- not A 3) C <- not B, not D
1 < 2 < 3

Idea: collect heads of rules, starting with most preferred rule.
In each step add head of current rule r to collection if r is not "defeated" by literals collected so far.
Make sure defeated rules do not increase priority of rules with same head

Let < be a total order on prerequisite-free P. We assume $r_i < r_j$ whenever $i < j$.

$C_<(S)$ is the smallest logically closed superset of $\bigcup S_i$ where

$S_0 = \{\}$,
$S_i = S_{i-1}$ if S_{i-1} defeats r_i or S defeats r_i and head(r_i) in S,
S_{i-1} + head(r_i) otherwise

A is preferred answer set of P if A is answer set of P and $C_<(A) = A$.

Alternative characterization

Let (P,<) be a fully prioritized prerequisite-free logic program,
A an answer set of P.

A is preferred iff
head(r) $\notin$ A implies r is defeated by a rule r' < r applied in A.

Preferred answer sets, general case

"Dual" Gelfond-Lifschitz transformation:

P logic program, S set of literals. ^{S}P is the program obtained from P by
1. **deleting all rules with prerequisite L such that $L \notin S$,**
2. **deleting all prerequisites from the remaining rules.**

Reduced program obviously prerequisite-free

A is preferred answer set of P iff A is preferred answer set of ^{A}P.

Example 1:

1) B <- not ¬B, A
2) ¬B <- not B
3) A <- not ¬A
1 < 2 < 3

Answer sets: $A_1 = \{A, \neg B\}$
$A_2 = \{A, B\}$

Reduced program P' (in both cases)

1) B <- not ¬B
2) ¬B <- not B
3) A <- not ¬A

$C_<(A_2) = A_2$, A_2 is single preferred answer set

Example 2:

1) B <- not ¬B, A 2) ¬A <- not A 3) A <- not ¬A
1 < 2 < 3

Answer sets: $A_1 = \{\neg A\}$
$A_2 = \{A, B\}$

$A_1{}_P$:

2) ¬A <- not A
3) A <- not ¬A

$A_2{}_P$:

1) B <- not ¬B
2) ¬A <- not A
3) A <- not ¬A

$C_<(A_1) = A_1$

$C_<(A_2) = \{B, \neg A\} \neq A_2$

Weakly preferred answer sets (finite case)

1) A <- not B
2) B <- not A
3) C <- A, not C

No preferred answer set exists.

Solution: define distance D(<, <') between orderings (number of switches needed to transform < into <')

Let A be an answer set of (P,<).
the preference violation degree of A, pvd(A), is the minimal distance between < and some <' such that A is a preferred a.s. of (P,<').

A is a weakly preferred answer set of (P,<) iff for no answer set B of (P,<) we have pvd(B) < pvd(A).

Preferred and weakly preferred a.s. coincide whenever the former exist.
Weakly preferred a.s. exist whenever a.s. exist.

Complexity Results (Th. Eiter)

Let Π = (P,<) be a finite propositional PLP, L a literal, A a set of literals

- **deciding whether A is preferred answer set of Π is polynomial**
- **deciding whether $\Pi \models$ L is co-NP-complete**
- **deciding whether $\Pi \models_w$ L is Δ_2^p[O(log n)]-complete**

Conclusions 5.2

1. **Preferences are relevant also for logic programming**
2. **Prioritized DL approaches have some problems**
3. **Standard semantics for extended LPs can be modified adequately**
4. **Preferred answer sets take preferences very seriously**
5. **Weakly preferred answer sets always exist for finite coherent programs**

5.3. Adding Dynamic Preferences to WFS

1. Why dynamic preferences are a useful thing to have
2. Adding Preferences I: syntactic extension
3. Adding Preferences II: the new fixpoint operator
4. Legal reasoning example

Dynamic Preferences

Our extension of answer sets used explicit preference order <.

Similar treatment as add-on in prioritized nonmonmotonic logics (Prioritized Circumscription., Prioritized Default Logic, HAEL, ...)

unsatisfactory since

* **priorities treated in an extra-logical manner.**
* **priorities must be completely prespecified.**
* **not possible to reason and argue about priorities in the logic.**
* **we want to derive priorities dynamically from facts and defaults.**

WANTED: a representation of preferences IN the logical language

A Legal Reasoning Example

According to Uniform Commercial Code (UCC) a security interest in goods is perfected by taking possession of the collateral.

According to Ship Mortgage Act (SMA) security interest in a ship may only be perfected by filing a financing statement.

UCC is state law, SMA federal law. UCC is more recent than SMA

The principle Lex Posterior gives precedence to newer laws.

The principle Lex Superior gives federal law precedence to state law.

Miller has possession of a certain ship but did not file a financing statement.

Is Miller´s security interest perfected?

Why extended logic programs under WFS?

- **Extended logic programs**
  expressive enough to represent many examples in a natural way
- **Well-founded semantics**
  efficient enough to actually compute results (quadratic in size of rule base)
- **Monotone operators**
  not difficult to modify accordingly

Well-Founded Semantics

γ γ

X
literals known
to be derivable

γ(X)
literals
potentially derivable
(not known
to be underivable)

X' = γγ(X)
literals known
to be derivable,
possibly larger

a <- b, c, not x1
d <- e, not y2
f <- h, not y9
....
r <- s, not x23

P

~~a <- b, c, not x1~~
d <- e, ~~not y2~~
f <- h, ~~not y9~~
....
~~r <- s, not x23~~

P^X

~~a <- b, c, not x1~~
d <- e, ~~not y2~~
~~f <- h, not y9~~
....
~~r <- s, not x23~~

P^γ(X)

Formal Definitions

Let P be a logic program, X a set of literals. The X-reduct of P, P^X, is the program obtained from P by

1. **deleting each rule containing a literal not L such that L in X, and**
2. **deleting all weakly negated literals from each remaining rule.**

Define anti-monotone operator

$$\boxed{\gamma_P(X) = Cn(P\text{^}X)}$$

Well-founded conclusions are obtained by iterating $\gamma_P{}^2$ on the empty set.

Reformulation: $\boxed{\gamma_P^2\,(X) = Cn^*(SAFE(X))}$

where Cn* is like Cn but neglects weakly negated literals,
and SAFE(X) are the rules not defeated by $\gamma_P(X)$.

Adding Preferences I: Extending the language

need to be able to talk about rules: introduce rule names N
need to be able to talk about preferences: introduce reserved symbol <

< two-place predicate symbol applied to names,
denotes transitive anti-symmetric relation
corresponding rule schemata are assumed to be contained in each program

A prioritized logic program is a pair (R, name) where

* R is a set of rules,
* name is a partial injective naming function assigning a unique name nj $\in$ N to rules in R.

Explicit vs. implicit priorities

Example 1:

n1: b <- not c
n2: ¬ b <- not b
n3: n2 < n1

We want to derive b, i.e. implicit preferences have priority
explicit preferences only used to handle conflicts not handled implicitly

Example 2:

n1: b <- not c, not ¬ b
n2: ¬b <- not b
n3: n2 < n1

Now ¬ b is intended conclusion.

Adding Preferences II: the monotonic operator

Basic intuitions:

original approach:
r considered X-safe iff not defeatedby consequences of the X-undefeated rules

new approach:
r considered X-safe iff not defeated by consequences of a certain <u>subset</u> of the X-undefeated rules, the rules not "dominated" by r.

r' is dominated by r iff r' is

*** known to be less preferred than r, and**
*** defeated when r is applied together with rules already known to be X-safe**

A simple example

n1: b <- not c, not ¬ b
n2: ¬ b <- not b
n3: n1 < n2
(x < z <- x < y, y < z)
(¬ y < z <- x < y)

	WFS(P)	WFSpr(P)
S0	{}	{}
S1	{n1 < n2, ¬ n2 < n1}	{n1 < n2, ¬ n2 < n1}
S2	S1	{n1 < n2, ¬ n2 < n1, b}
S3		S2

n1 is S1-safe because:

1) **n1 dominates n2 (n1 < n2 ∈ S1 and n1 defeats n2)**
2) **n1 is not defeated by the rules without n2**

Formal Definitions

Let P = (R,name) be a prioritized LP, X a set of literals, Y a set of rules, $r \in R$.
Let R_X be the set of rules in R not defeated by X.

The set of rules dominated by r wrt X and Y, denoted $Dom_{X,Y}(r)$, is the set

$$\{r' \in R \mid name(r) < name(r') \in X \text{ and } Cn^*(Y + r) \text{ defeats } r'\}$$

The set of X-safe rules, SAFE(X), is the union of all Rj, where

R0 = {}, and for i > 0
$Ri = \{r \in R \mid r \text{ not defeated by } Cn^*(R\_X \setminus Dom_{X,Ri-1}(r))\}$

monotonic operator:

$$\Gamma_P(X) = Cn^*(SAFE(X))$$

Legal Reasoning Example

Notation: c <= a1, ..., an, not b1, ..., not bm
stands for: c <- a1, ..., an, not b1, ..., not bm, not -c

defaults:		facts H:
UC: perfected	<= possession	possession
SM: ¬ perfected	<= ship, ¬ financial-statement	ship
LP(di, dj): di < dj	<= more-recent(di, dj)	¬financial-statement
LS(di, dj): di < dj	<= federal-law(di), state-law(dj)	more-recent(UC, SM)
		federal-law(SM)
		state-law(UC)
		LS(SM,UC) < LP(UC,SM)

Iteration yields:
S0 = {}
$S1 = H \cup \{\neg LP(UC,SM) < LS(SM,UC)\}$
$S2 = S1 \cup \{SM < UC, \neg UC < SM\}$
$S3 = S2 \cup \{\neg perfected\}$
S4 = S3

$O(n^3)$ Algorithm

Procedure WFS+
Input: A prioritized logic program $P = (R,name)$ with $|R| = n$
Output: least fixed-point of Γ

$S_0 := \{\}$;
$R_0 := \{\}$;
for $i = 1$ to n do
  if there is a rule $r \in R\_S_{i-1} \setminus R_{i-1}$ such that
  $Cn^*(R\_S_{i-1} \setminus Dom_{S_{i-1},R_{i-1}}(r))$ does not defeat r
  then $R_i := R_{i-1} + r$; $S_i := Cn^*(R_i)$
  else return S_{i-1}
endfor
end WFS+

Conclusions

- **Dynamic preference information useful for many applications**
- **Extended logic programs under the WFS provide basis for dynamic priorities**
- **Our approach uses explicit preferences only for conflicts not solved by WFS**
- **Rule names and reserved predicate symbol used to represent preferences**
- **Modification of monotone operator takes preferences into account**
- **Computation remains tractable**