Implementation Results in Classical Constructive Negation

Susana Muñoz-Hernández   Juan José Moreno-Navarro

Facultad de Informática

Universidad Politécnica de Madrid

28660 Madrid, Spain

susana@fi.upm.es

jjmoreno@fi.upm.es

Overview

- Motivation
- Constructive Negation
- Algorithm
- Implementation
- Examples
- Conclusions

Overview

- Motivation
- Constructive Negation
- Algorithm
- Implementation
- Examples
- Conclusions

Motivation

- The important role of Negation in Logic ($\neg$)
- A lack of Negation implementation at Prolog

Motivation

- The important role of Negation in Logic ($\neg$)
- A lack of Negation implementation at Prolog
- [REASON] Problems of the proposals:
  - Expressiveness
  - Semantics (soundness vs completeness)
  - Complexity

Motivation

- The important role of Negation in Logic ($\neg$)
- A lack of Negation implementation at Prolog
- [REASON] Problems of the proposals:
  - Expressiveness
  - Semantics (soundness vs completeness)
  - Complexity
- [CONSECUENCE] Limited implementations:
  - Negation as failure (naf)
  - Delay technique

Negation as Failure

- SLDNF resolution $\begin{cases} naf(G) :- & call(G), !, \\ & fail. \\ naf(G). & \end{cases}$

- Execution:

```
?- naf(even(s(s(0)))).
no

?- naf(even(s(0))).
yes
```

- *naf(G)* checks if *G* is true or false

Negation as Failure

- SLDNF resolution $\left\{ \begin{array}{ll} naf(G) :- & call(G), !, \\ & fail. \\ naf(G). & \end{array} \right.$

- Execution:

```
?- naf(even(s(s(0)))).
no

?- naf(even(s(0))).
yes

?- naf(even(X)).
no

?- naf(even(s(Y))).
no
```

- Problem: *naf* is incomplete

Interpretation of Quantifications

$$naf\,(p(\overline{X})) \;\equiv\; \neg\exists\overline{X}.\,p(\overline{X})$$

- $naf\,(p(\overline{X})$ checks if $p(\overline{X})$ is “true” or “false” $\Rightarrow$ **No variable instantiation**

Interpretation of Quantifications

$$naf(p(\overline{X})) \equiv \neg\exists\overline{X}.\, p(\overline{X})$$

- $naf(p(\overline{X})$ checks if $p(\overline{X})$ is "true" or "false" $\Rightarrow$ **No variable instantiation**

$$cneg(p(\overline{X})) \equiv \exists\overline{X}.\, \neg p(\overline{X})$$

- $cneg(p(\overline{X})$ provides the values of $\overline{X}$ that make false $p(\overline{X})$ $\Rightarrow$ **Constructive answer**

Constructive Answers

```
?- null(X).
X = 0 ?;
no
```

```
?- cneg(null(X)).
X = s(0) ?;
X = s(s(0)) ?;
X = s(s(s(0))) ?;
...
```

```
?- cneg(null(X)).
X =/= 0 ?;
no
```

Overview

- Motivation
- Constructive Negation
- Algorithm
- Implementation
- Examples
- Conclusions

Constructive Negation

- Papers about Semantical aspects
- Practical Chan's proposal (coroutining)
- Implementation problems (Eclipse)

Constructive Negation

- Papers about Semantical aspects
- Practical Chan's proposal (coroutining)
- Implementation problems (Eclipse)
- We provide:
  - A complete theoretical algorithm (refining and extending to the constructive negation method)
  - A discussion about implementation issues
  - A preliminary implementation

Semantics

Adequate for Prolog:

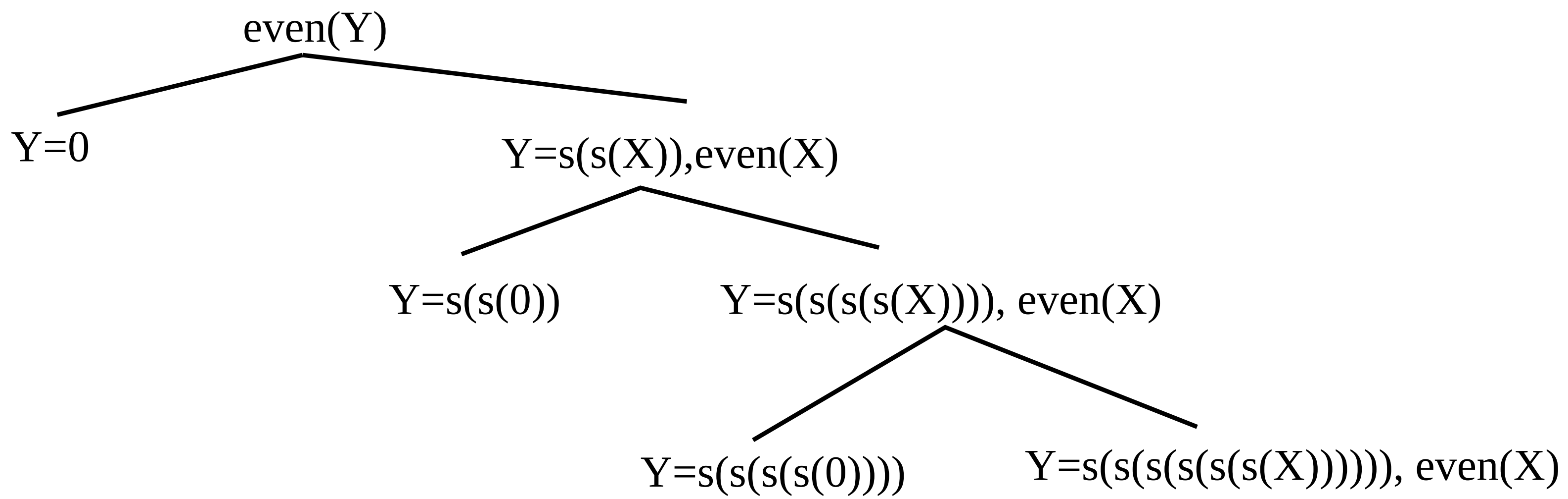
- **Declarative Semantics**: Clark's completion, CWA & CET [Clark78].
- **Denotational Semantics**: Kunen's 3-valued interpretation ($\{\underline{\mathrm{t}}, \underline{\mathrm{f}}, \underline{\mathrm{u}}\}$) [Kun87].
- **Procedural Semantics**: Stuckey's Immediate consequence ($\Phi_P^{\mathcal{A}}$) in an *admissible* constraint structure, $\mathcal{A}$ [Stu95].

Frontier

(From [Stuckey95])
A frontier of a goal G is the disjunction of a finite set of nodes in the derivation tree such that every derivation of G is either finitely failed or passes through exactly one frontier node.

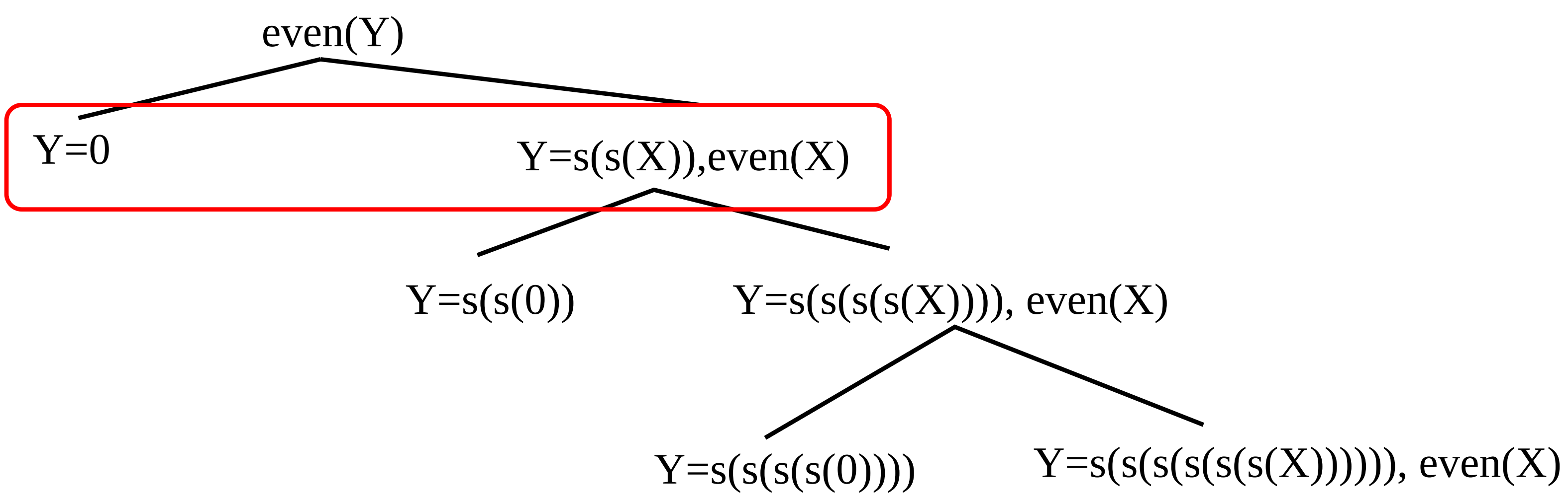
Frontier

```
even(0).
even(s(s(X))):- even(X).
```



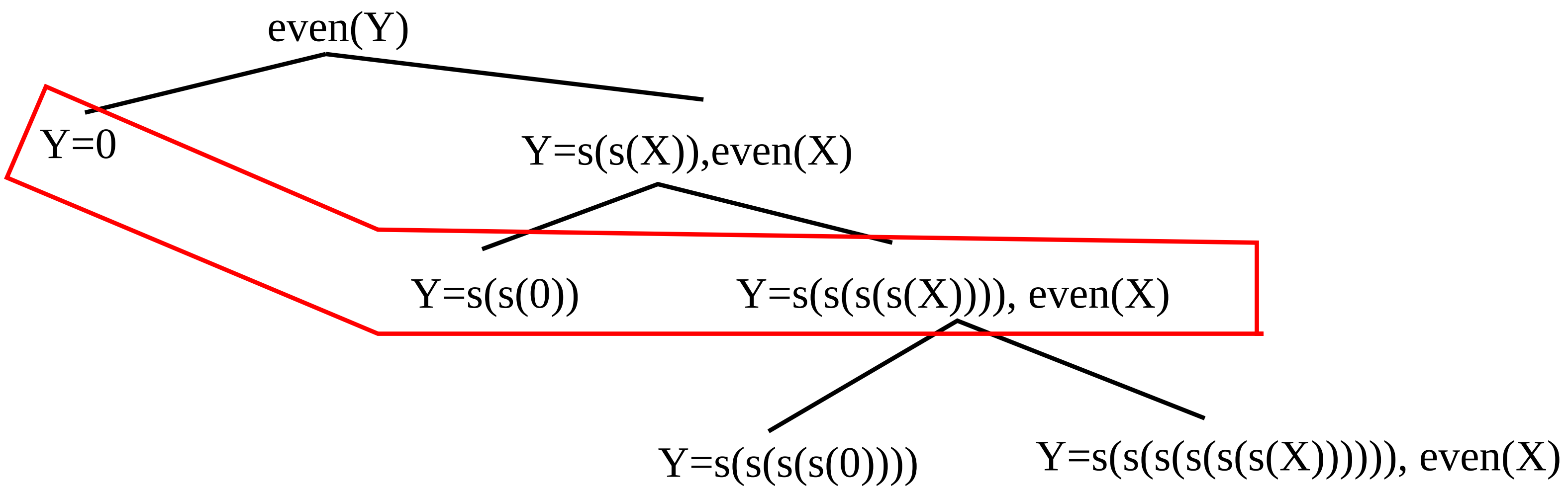
Frontier

```
even(0).
even(s(s(X))):- even(X).
```



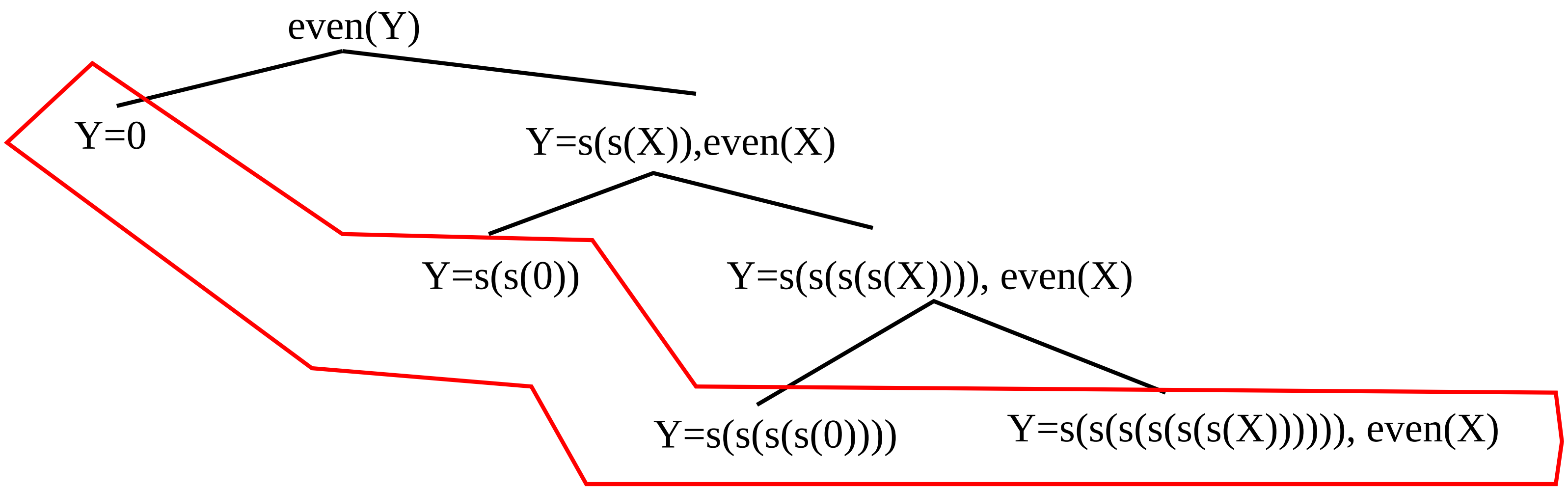
Frontier

```
even(0).
even(s(s(X))):- even(X).
```



Frontier

```
even(0).
even(s(s(X))):- even(X).
```



Frontier

```
even(0).
even(s(s(X))):- even(X).
```

Frontier$(even(Y)) = C_1 \vee C_2 =$

$$(Y = 0) \vee (\exists X.\, Y = s(s(X)) \wedge even(X))$$

Negation of a Frontier

```
even(0).
even(s(s(X))):- even(X).
```

```
?- cneg(even(Y)).
```

Frontier $(even(Y)) = C_1 \vee C_2 =$

$$(Y = 0) \vee (\exists X.\, Y = s(s(X)) \wedge even(X))$$

$\neg$ ***Frontier*** $(even(Y)) = \neg\, C_1 \wedge \neg\, C_2 = \ [Y \neq 0] \wedge$

$$[(\forall X1.\, Y \neq\ s(s(X1))) \vee ((\exists X2.\, Y = s(s(X2)) \wedge \neg\, even(X2))]$$

Overview

- Motivation
- Constructive Negation
- Algorithm
- Implementation
- Examples
- Conclusions

Preparation

- Simplification of the conjunction
- Organization of the conjunction
  (GoalVars :- RelVars (ImpVars + ExpVars))

$$C_i \equiv \overline{I} \wedge \overline{D} \wedge \overline{R} \equiv \overline{I} \wedge \overline{D}_{imp} \wedge \overline{D}_{exp} \wedge \overline{R}_{imp} \wedge \overline{R}_{exp}$$

- Normalization of the conjunction
  - Elimination of redundant variables and equalities
  - Elimination of irrelevant disequalities

Negation of Subformulas

$$C_i \;\equiv\; \overline{I} \wedge \overline{D}_{imp} \wedge \overline{R}_{imp} \wedge \overline{D}_{exp} \wedge \overline{R}_{exp}$$

$$\neg\, C_i \equiv \neg\, \overline{I} \quad \vee$$

$$\overline{I} \wedge \neg\, \overline{D}_{imp} \quad \vee$$

$$\overline{I} \wedge \overline{D} \wedge \neg\, \overline{R}_{imp} \quad \vee$$

$$\overline{I} \wedge \overline{D} \wedge \overline{R}_{imp} \wedge \neg\, (\overline{D}_{exp} \wedge \overline{R}_{exp})$$

Negation of Subformulas(I)

- Negation of $\overline{I}$

$$\overline{I} \equiv I_1 \wedge \ldots \wedge I_{NI} \equiv$$

$$\underbrace{\exists\, \overline{Z}_1.\; X_1 = t_1}_{I_1} \wedge \ldots \wedge \underbrace{\exists\, \overline{Z}_{NI}.\; X_{NI} = t_{NI}}_{I_{NI}}$$

$$\neg\, C_i \equiv \neg\, \overline{I} \equiv \bigvee_{i=1}^{NI} \forall\, \overline{Z}_i.\; X_i \neq t_i \equiv$$

$$\underbrace{\forall\, \overline{Z}_1.\; X_1 \neq t_1}_{\neg\, I_1} \vee \ldots \vee \underbrace{\forall\, \overline{Z}_{NI}.\; X_{NI} \neq t_{NI}}_{\neg\, I_{NI}}$$

Negation of Subformulas(II)

- Negation of $\overline{D}_{imp}$

$$\overline{D}_{imp} \equiv D_1 \wedge \ldots \wedge D_{N_{D_{imp}}}$$

$$D_i \equiv \forall\, \overline{W}_i.\ \exists\, \overline{Z}_i.\ Y_i \neq s_i$$

$$\neg\ D_i \equiv \exists\, \overline{W}_i.\ Y_i = s_i$$

$$\begin{aligned}\neg\ C_i \equiv\ & \overline{I} \wedge \neg\ D_1 \vee \\ & \overline{I} \wedge D_1 \wedge \neg\ D_2 \vee \\ & \ldots \\ & \overline{I} \wedge D_1 \wedge \ldots \wedge D_{N_{D_{imp}}-1} \wedge \neg\ D_{N_{D_{imp}}}\end{aligned}$$

Negation of Subformulas(III)

- Negation of $\overline{R}_{imp}$

$$\overline{R}_{imp} \equiv R_1 \wedge \ldots \wedge R_{N_{R_{imp}}}$$

$$\begin{aligned} \neg\, C_i \equiv\ & \overline{I} \wedge \overline{D}_{imp} \wedge \neg\, R_1 \vee \\ & \overline{I} \wedge \overline{D}_{imp} \wedge R_1 \wedge \neg\, R_2 \vee \\ & \ldots \\ & \overline{I} \wedge \overline{D}_{imp} \wedge R_1 \wedge \ldots \wedge R_{N_{R_{imp}}-1} \wedge \neg\, R_{N_{R_{imp}}} \end{aligned}$$

Negation of Subformulas(IV)

- Negation of $\overline{D}_{exp} \wedge \overline{R}_{exp}$

$$\neg\,(\exists\,\overline{V}_{exp}.\overline{D}_{exp} \wedge \overline{R}_{exp}) \equiv \forall\,\overline{V}_{exp}.\neg\,(\overline{D}_{exp} \wedge \overline{R}_{exp})$$

$$\neg\, C_i \equiv \overline{I} \wedge \overline{D}_{imp} \wedge \overline{R}_{imp} \wedge \forall\,\overline{V}_{exp}.\,\neg\,(\overline{D}_{exp} \wedge \overline{R}_{exp})$$

Overview

- Motivation
- Constructive Negation
- Algorithm
- Implementation
- Examples
- Conclusions

Implementation Issues (I)

- Code expansion

```
:- module(mod1,[even/1,p/1],[cneg]).

even(0).
even(s(s(X))) :- even(X).

p(X) :- ..., cneg(even(X)), ... .

stored_clause(even(0),[]).
stored_clause(even(s(s(X))),[even(X)]).
```

Implementation Issues (II)

- Disequality constraints (Attributed variables)
  Constraints Normal Form

$$\underbrace{\bigwedge_i (X_i = t_i)}_{\text{positive information}} \qquad \wedge$$

$$\underbrace{(\bigwedge_j \forall\, \overline{Z}^1_j .\ (Y^1_j \neq s^1_j) \vee \ldots \vee \bigwedge_l \forall\, \overline{Z}^n_l .\ (Y^n_l \neq s^n_l))}_{\text{negative information}}$$

Optimizations

- Compact information (disjunction of conjunction of disequalities)
- Pruning subgoals (equivalent to *true* / *false*)
- Constraint simplification

$$F \equiv \bigvee_i \bigwedge_j \forall\, \overline{Z}_j^i\ (Y_j^i \neq s_j^i)$$

- Finite variant, $cnegf$, to negate goals that have a finite number of solutions. The last frontier is negated. $\neg G \equiv \neg(S_1 \vee S_2 \vee \ldots \vee S_n)$

Overview

- Motivation
- Constructive Negation
- Algorithm
- Implementation
- Examples
- Conclusions

Examples (I)

```
boole(0).
boole(1).
```

```
?- cneg(boole(X)).
X=/=1,X=/=0 ? ;
no
```

```
positive(0).
positive(s(X)):-
        positive(X).
```

```
?- cneg(positive(X)).
X=/=s(fA(_A)),X=/=0 ? ;
X = s(_A),
_A=/=s(fA(_B)),_A=/=0 ? ;
X = s(s(_A)),
_A=/=s(fA(_B)),_A=/=0 ?
yes
```

Examples (II)

```
number(0).
number(s(X)):-
        number(X).

greater(s(X),0):-
        number(X).
greater(s(X),s(Y)):-
        greater(X,Y).
```

```
?- cneg(greater(X,Y)).
Y=/=0,Y=/=s(fA(_A)) ?;

Y=/=s(fA(_A)),
X=/=s(fA(_B)) ?;

X = s(_A),Y = 0,
_A=/=s(fA(_B)),_A=/=0 ?;

X = s(s(_A)),Y = 0,
_A=/=s(fA(_B)),_A=/=0 ?
yes
```

Overview

- Motivation
- Constructive Negation
- Algorithm
- Implementation
- Examples
- Conclusions

Experimental Results

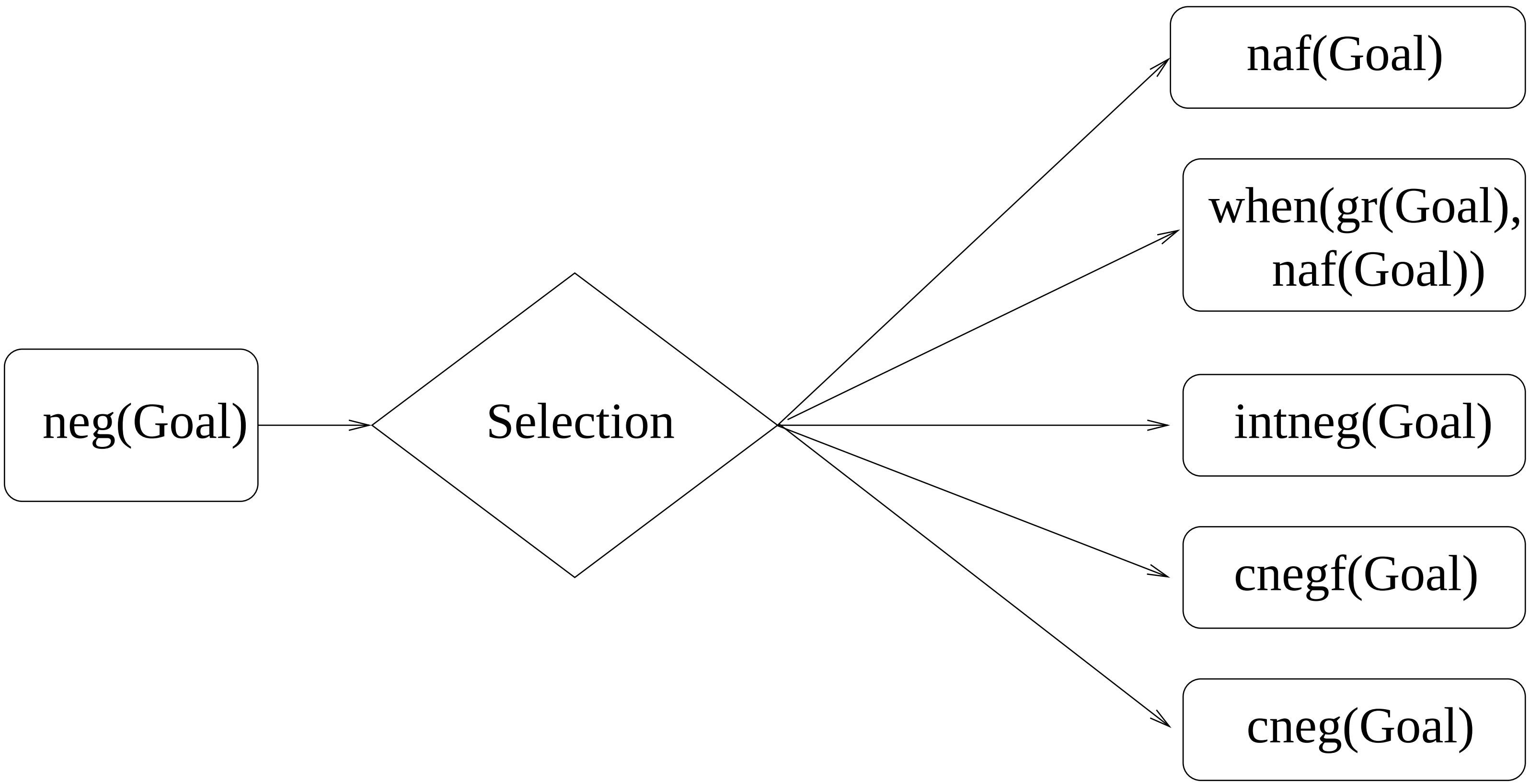
goals	Goal	naf(Goal)	cneg(Goal)	ratio
boole(1)	2049	2099	2069	0.98
positive(s(s(s(s(s(s(0))))))	2079	1600	2159	1.3
greater(s(s(s(0))),s(0))	2110	2099	2100	1.00
average				1.06
positive(s^{500000}(0))	2930	2949	41929	14.21
positive($s^{1000000}$(0))	3820	3689	81840	22.18
greater(s^{500000}(0),s^{500000}(0))	3200	3339	22370	7.70
average				14.69
positive(X)	2020	-	7189	
greater(s(s(s(0))),X)	2099	-	6990	
queens(s(s(0)),Qs)	6939	-	9119	

Conclusion and Future Work

- Detailed description of the modified algorithm
- Complete and consistent implementation
- Experimental results
- Extensible to other LP systems (future work)

- Efficiency problem
  - Statical Analysis to improve the frontier
  - WAM level (future work)
  - Negation System for Prolog

Negation System for Prolog



- Static phase + Dynamic phase

Implementation Results in Classical Constructive Negation

Susana Muñoz-Hernández Juan José Moreno-Navarro

Facultad de Informática

Universidad Politécnica de Madrid

28660 Madrid, Spain

susana@fi.upm.es

jjmoreno@fi.upm.es