Was the Year 2000 a Leap Year? Step-wise Narrowing Theories with Metagol

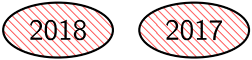
Michael Siebers and Ute Schmid
Cognitive Systems Group, University of Bamberg

28th International Conference on Inductive Logic Programming
2018-09-04

Was the Year 2000 a Leap Year?

Was the Year 2000 a Leap Year?

Was the Year 2000 a Leap Year?

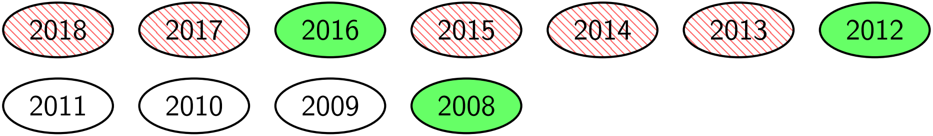


Was the Year 2000 a Leap Year?

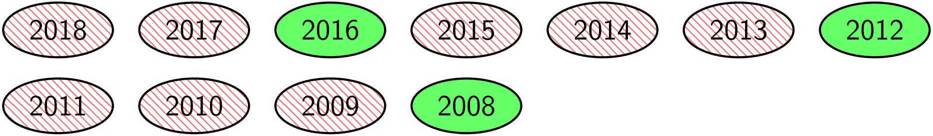
Was the Year 2000 a Leap Year?

Was the Year 2000 a Leap Year?

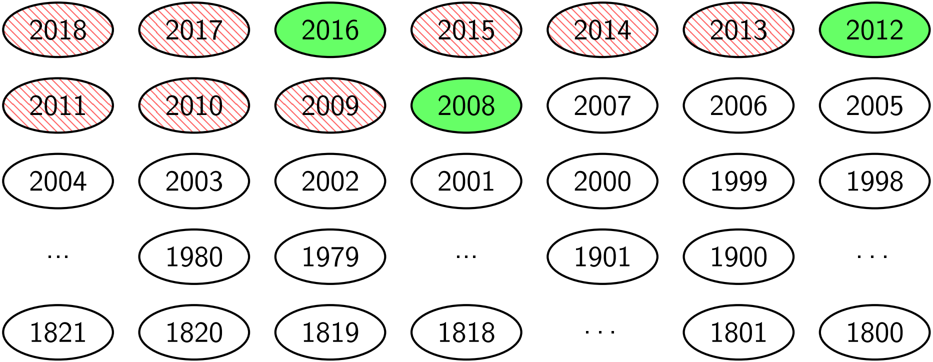
Was the Year 2000 a Leap Year?



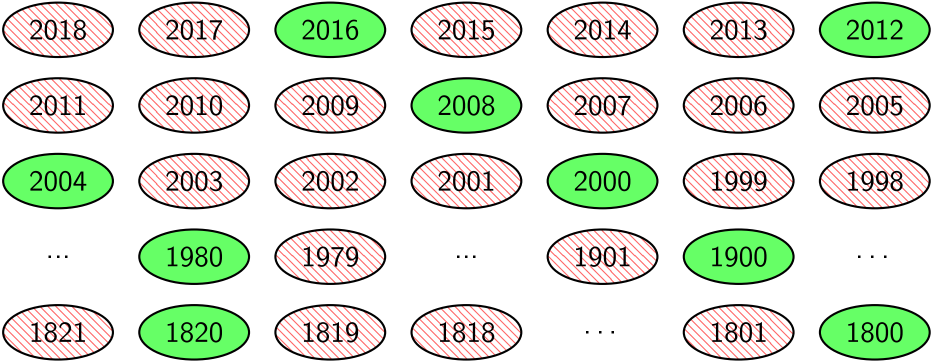
Was the Year 2000 a Leap Year?



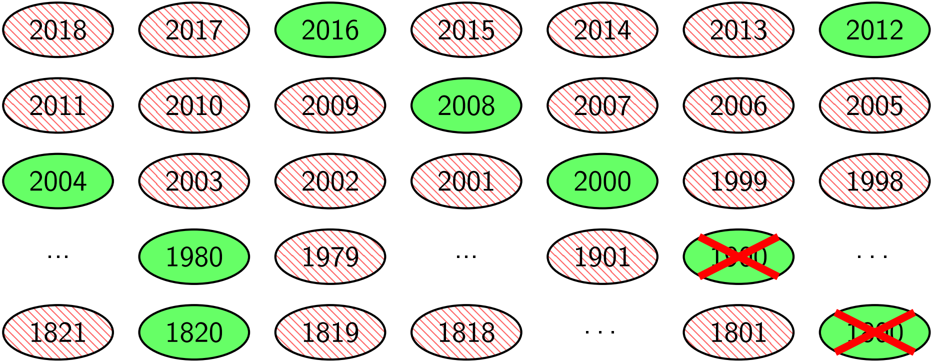
Was the Year 2000 a Leap Year?



Was the Year 2000 a Leap Year?



Was the Year 2000 a Leap Year?



Was the Year 2000 a Leap Year?

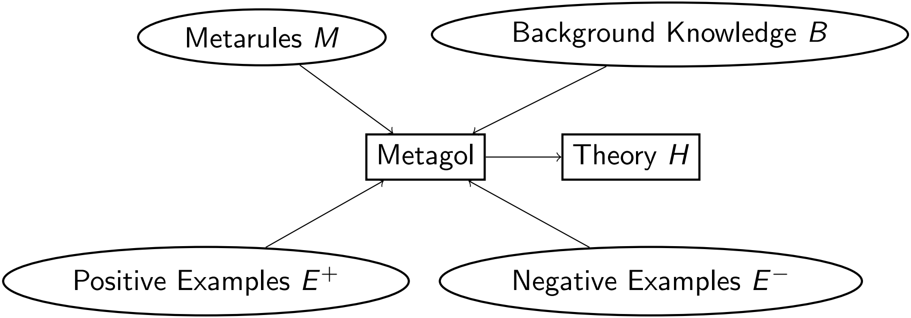
Definition [Richards, 2013 (p. 599)]

Every year that is exactly divisible by 4 is a leap year, *except* for years that are exactly divisible by 100, *but* these centurial years are leap years if they are exactly divisible by 400.

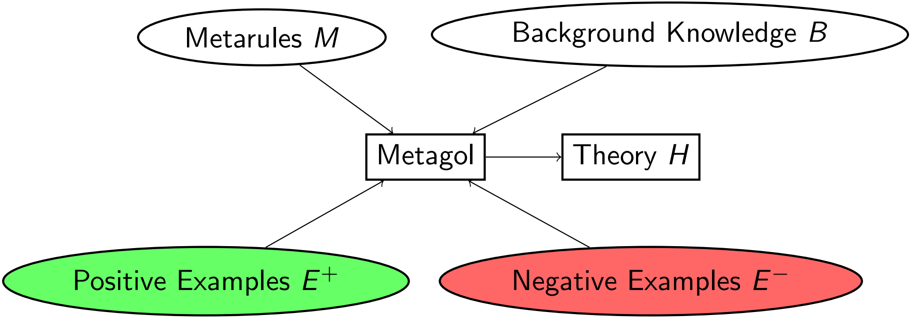
Logic Program

leapyear(X) $\leftarrow$ divisible(X,4), not divisible(X,100).
leapyear(X) $\leftarrow$ divisible(X,400).

Metagol

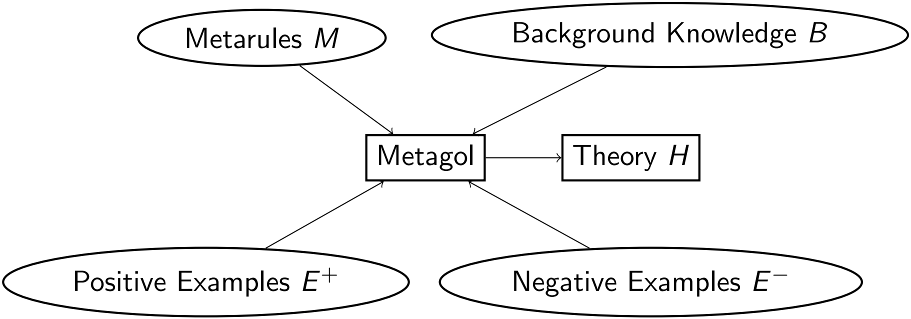

Metarules M
Background Knowledge B
Metagol
Theory H
Positive Examples E^+
Negative Examples E^-

Metagol

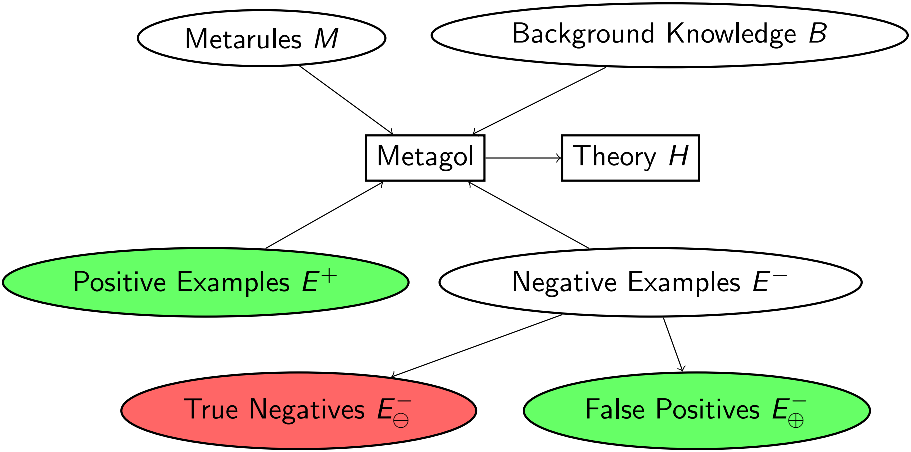


Metarules M
Background Knowledge B
Metagol
Theory H
Positive Examples E^+
Negative Examples E^-

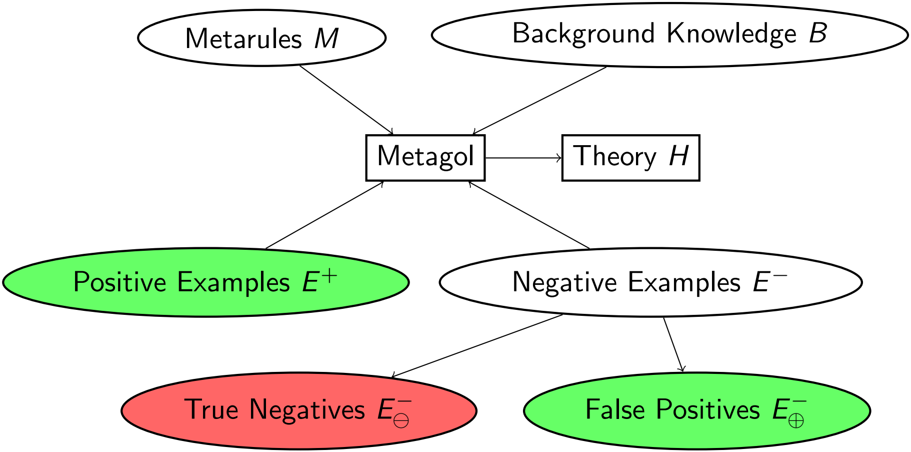
Relaxed Metagol



Relaxed Metagol



Relaxed Metagol



$\left|E^-_\oplus\right| \leq l$

Leapyear?

Logic Program

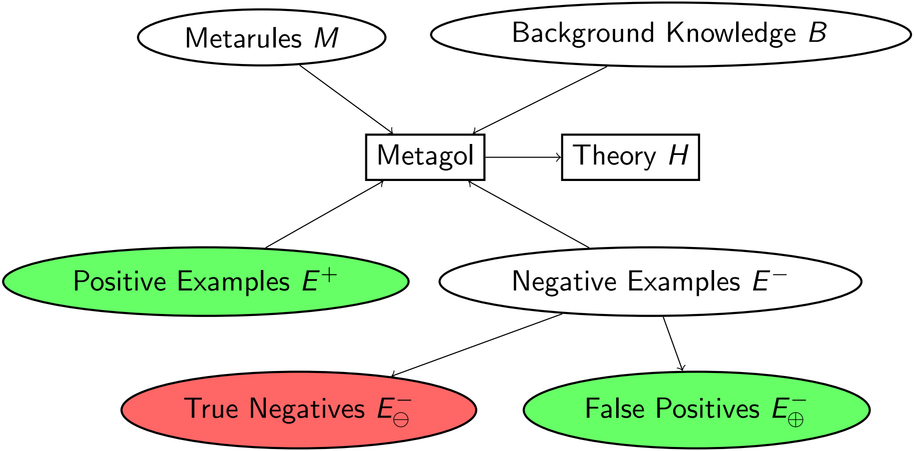
leapyear(X) $\leftarrow$ divisible(X,4).

False Positives

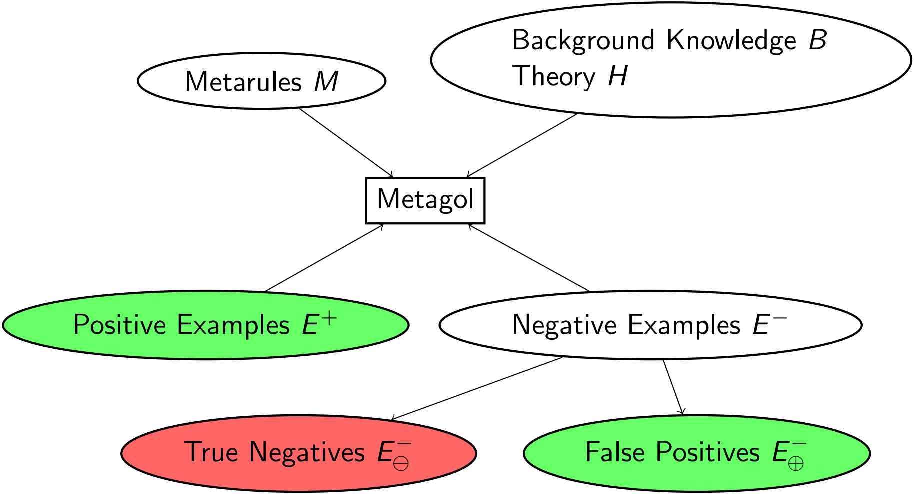
1900, 1800, 1700, 1500, 1400, ...

$\Rightarrow$ 99.25% accuracy

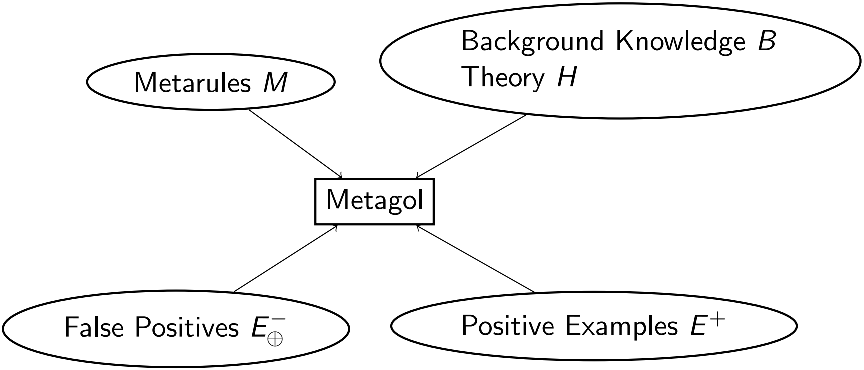
Utilizing False Positives



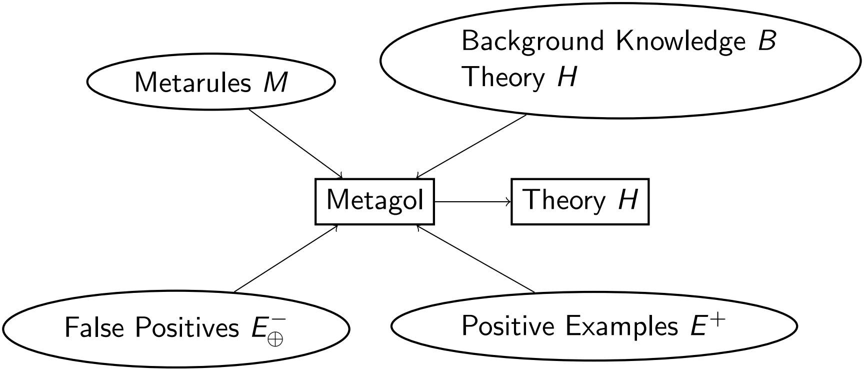
Utilizing False Positives



Utilizing False Positives



Utilizing False Positives



Step-wise Narrowed Theory

Definition (Step-wise Narrowed Theory)

Every higher-order datalog program is a step-wise narrowed theory. Let H be a higher-order datalog program, Φ a mapping between predicate symbols and S a step-wise narrowed theory. Then, $\langle H, \Phi, S \rangle$ is also a step-wise narrowed theory.

Leapyear?

Step-wise Narrowed Theory

$\langle$

leapyear(A) $\leftarrow$ divisible(A,4).,

leapyear $\mapsto$ _leapyear,

$\langle$

_leapyear(A) $\leftarrow$ divisible(A,100).,

_leapyear $\mapsto$ __leapyear,

__leapyear(A) $\leftarrow$ divisible(A,400).

$\rangle$

$\rangle$

Leapyear?

Flattened Theory

$\langle$

leapyear(A) $\leftarrow$ divisible(A,4).,

leapyear $\mapsto$ _leapyear,

$\langle$

_leapyear(A) $\leftarrow$ divisible(A,100).,

_leapyear $\mapsto$ __leapyear,

__leapyear(A) $\leftarrow$ divisible(A,400).

$\rangle$

$\rangle$

Leapyear?

Flattened Theory

$\langle$

leapyear(A) $\leftarrow$ divisible(A,4).,

leapyear $\mapsto$ _leapyear,

$\langle$

_leapyear(A) $\leftarrow$ divisible(A,100), not(__leapyear(A)).

__leapyear(A) $\leftarrow$ divisible(A,400).

$\rangle$

$\rangle$

Leapyear?

Flattened Theory

⟨

leapyear(A) ← divisible(A,4).,
leapyear ↦ _leapyear,
_leapyear(A) ← divisible(A,100), not(__leapyear(A)).
__leapyear(A) ← divisible(A,400).

⟩

Leapyear?

Flattened Theory

leapyear(A) $\leftarrow$ divisible(A,4), not(_leapyear(A)).
_leapyear(A) $\leftarrow$ divisible(A,100), not(__leapyear(A)).
__leapyear(A) $\leftarrow$ divisible(A,400).

Leapyear?

Flattened Theory

leapyear(A) ← divisible(A,4), not(_leapyear(A)).
_leapyear(A) ← divisible(A,100), not(__leapyear(A)).
__leapyear(A) ← divisible(A,400).

Caution

Only for non-recursive theories!

Compare Metagol and Metagol$_{SN}$

Experimental Setup

Domain leapyear

Training data

- years 1582–2018
- randomly samples 20%, 40%, . . . , 100%

Test data years 2019–3018

Repititions 10x

Learning time out 30 min

Compare Metagol and Metagol$_{SN}$

Background Knowledge

divisible/2 where *divisible(X,Y)* holds if and only if the integer Y divides the integer X exactly and

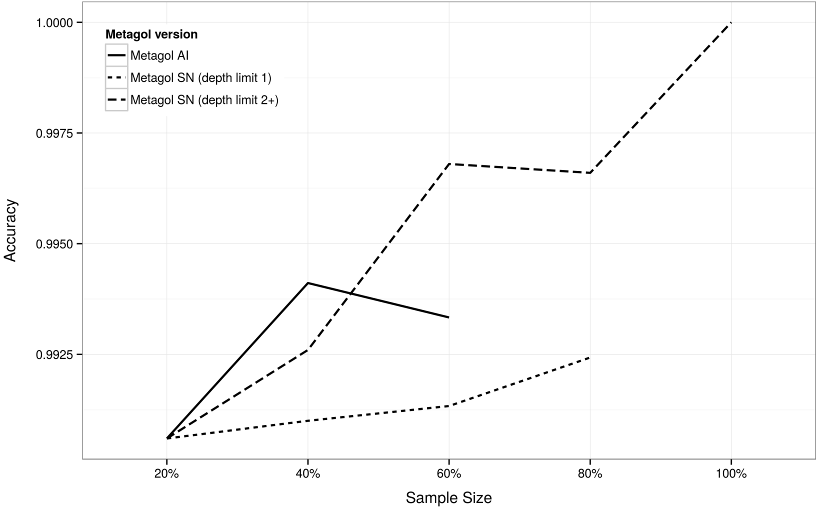
not_divisible/2 where *not_divisible(X,Y)* holds if and only if X and Y are integers and *divisible(X,Y)* does not hold.

Y's could be choosen from all divisors of all examples.

Metarules

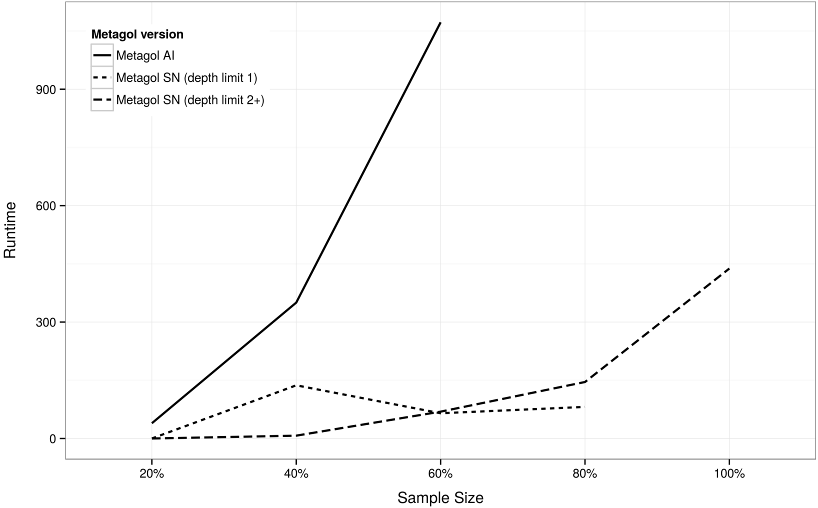
Const	$P(A,B) \leftarrow$
And	$P(A) \leftarrow Q(A), R(A)$
Chain	$P(A,B) \leftarrow Q(A,C), R(C,B)$
Curry	$P(A) \leftarrow Q(A,B)$

Results



Metagol version
Metagol AI
Metagol SN (depth limit 1)
Metagol SN (depth limit 2+)
1.0000
0.9975
0.9950
0.9925
Accuracy
20%
40%
60%
80%
100%
Sample Size

Results


Metagol version
Metagol AI
Metagol SN (depth limit 1)
Metagol SN (depth limit 2+)
Runtime
900
600
300
0
20%
40%
60%
80%
100%
Sample Size

Conclusion

Summary

- Relaxed Metagol
- Allowed (some) negation in Metagol
- Metagol_{SN} is faster
- Metagol_{SN} might have better results

Further Research

- Evaluate Metagol_{SN} on more domains
- Use relaxed Metagol for pre-tests
- Use relaxed Metagol for anytime-algorithm

Try it!

Available on GitHub

https://github.com/michael-siebers/metagol/tree/ilp2018

Try it!

Available on GitHub

https://github.com/michael-siebers/metagol/tree/ilp2018

Questions / Comments

Metagol Code

```
metagol(Pos,Neg,Prog) :-
  prove_all(Pos,[],Prog),
  prove_none(Neg,Prog).

prove_all([],Prog,Prog).
prove_all([Atom|Atoms],Prog1,Prog2) :-
  prove_one(Atom,Prog1,Prog3),
  prove_all(Atoms,Prog3,Prog2).

prove_one(Atom,Prog,Prog) :- call(Atom).
prove_one(Atom,Prog1,Prog2) :-
  metarule(Name,MetaSub,(Atom :- Body)),
  store(sub(Name,MetaSub),Prog1,Prog3),
  prove_all(Body,Prog3,Prog2).
```

Metagol Code (Cont'd)

```
metagol(Pos,Neg,Prog) :-
  prove_all(Pos,[],Prog),
  prove_none(Neg,Prog).

prove_none([],Prog).
prove_none([Atom|Atoms],Prog) :-
  not(prove_one(Atom,Prog,Prog)),
  prove_none(Atoms,Prog).
```

Metagol Relaxed

```
metagol_relaxed(Pos,Neg,Prog,FalsePos) :-
  prove_all(Pos,[],Prog),
  prove_some(Neg,Prog,FalsePos).

prove_some([],Prog,[]).
prove_some([Atom|Atoms],Prog,[Atom|Proven]) :-
  prove_some(Atoms,Prog,Proven),
  prove_one(Atom,Prog,Prog).
prove_some([Atom|Atoms],Prog,Proven) :-
  prove_some(Atoms,Prog,Proven),
  not(prove_one(Atom,Prog,Prog)).
```

Metagol$_{SN}$

```
metagol_sn(Pos,Neg,SNT) :-
  let P be the predicate symbol used in Pos,
  metagol_relaxed(Pos,Neg,Prog1,FalsePos),
  if FalsePos=[]
    SNT = Prog1
  else
    let PPrime be P prefixed with '_',
    let PosNext be FalsePos with P renamed to PPrime,
    let NegNext be Pos with P renamed to PPrime,
    assert_prog(Prog1),
    metagol_sn(PosNext,NegNext,Prog2),
    SNT=snt(Prog1,PPrime,Prog2).
```