

Specific Prolog Interpretation

- negation by failure
- procedural built-ins
 - Modification of the data base on the fly
 - Input/Output
 - Terminal
 - Files
 - Prolog Data bases

negation by failure

- negation by failure/negation as failure
versus
- ‚classical‘ negation

,classical‘ negation

Data basis:

highschool(peter).

highschool(irene).

\neg highschool(karl).

?- highschool(X). (for X=peter, karl, ute ?)

Resolution

refutation of negated proposition!

Proposition: $\text{highschool}(\text{peter})$

negated proposition: $\neg \text{highschool}(\text{peter})$

Data basis:

$\text{highschool}(\text{peter}).$

$\text{highschool}(\text{irene}).$

$\neg \text{highschool}(\text{karl}).$



F

Proposition is true

Resolution

refutation of negated proposition!

proposition: $\text{highschool}(\text{karl})$

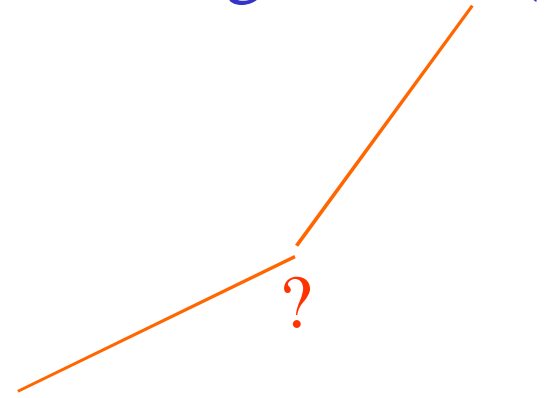
negated proposition: $\neg \text{highschool}(\text{karl})$

data basis:

$\text{highschool}(\text{peter}).$

$\text{highschool}(\text{irene}).$

$\neg \text{highschool}(\text{karl}).$



cannot be proved

Resolution

refutation of negated proposition!

proposition: \neg **highschool (karl)**

negated proposition: **highschool (karl)**

Datenbasis:

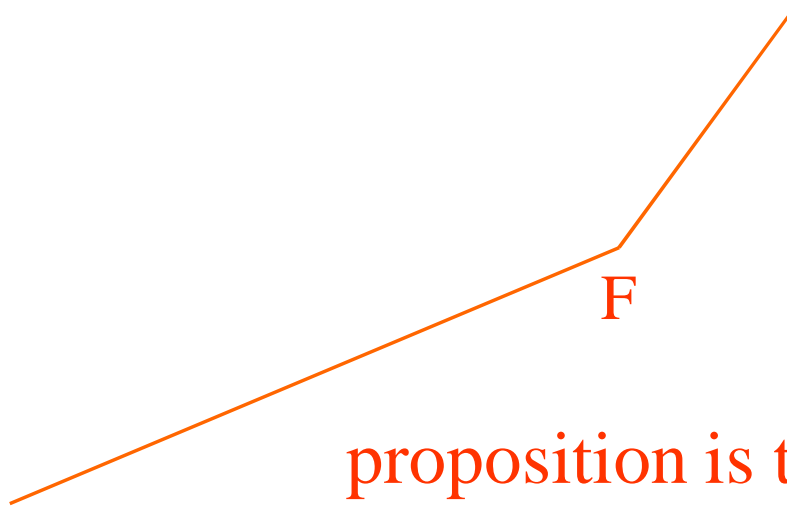
highschool(peter).

highschool(irene).

\neg highschool(karl).

F

proposition is true



Resolution

refutation of negated proposition!

proposition: $\text{highschool}(\text{ute})$

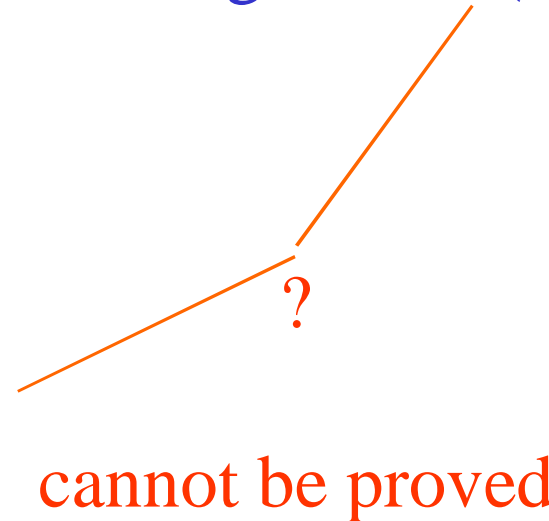
negated proposition: $\neg \text{highschool}(\text{ute})$

Datenbasis:

$\text{highschool}(\text{peter})$.

$\text{highschool}(\text{irene})$.

$\neg \text{highschool}(\text{karl})$.



cannot be proved

Resolution

refutation of negated proposition!

proposition: \neg **highschool (ute)**

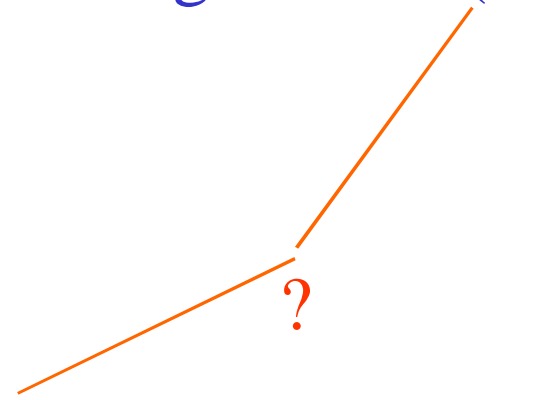
negated proposition: **highschool (ute)**

Datenbasis:

highschool(peter).

highschool(irene).

\neg highschool (karl).



cannot be proved

,classical‘ negation

Data basis:

highschool(peter).

highschool(irene).

\neg highschool(karl).

?- highschool (karl) : no!

?- \neg highschool(karl) : yes!

?- highschool(ute) : don‘t know!

?- \neg highschool(ute) : don‘t know!

Prolog-Negation

Data basis:

highschool(peter).

highschool(irene).

~~\neg highschool(karl).~~

cannot be represented as a fact in
Prolog!

Prolog negation

Data basis:

highschool(peter).

highschool(irene).

?- highschool (peter) : yes!

?- highschool (karl) : no!

?- highschool (ute) : no!

?- \neg highschool(X) : question cannot be represented in
Prolog!

Prolog negation

- The fact that an information is missing
- and that the truth of an information is explicitly negated

are not distinguished in prolog!

A statement is considered to be negated, if it cannot be proved!

= negation by failure / negation as failure

In logical terms: Prolog bases on the Closed world assumption

What does this mean for the negation sign?

- \neg : Negation in prolog mostly expressed by `\+`
- `\+(statement)` : valid,
if `statement` cannot be proved!

This can be defined in prolog by the built-in predicate `call`:

```
\+(Statement) :- call(Statement), !, fail.
```

```
\+(Statement).
```

Example of negation by failure

Preferences:

```
enjoys(vincent,X) :- big_kahuna_burger(X),!,fail.  
enjoys(vincent,X) :- burger(X).
```

Data basis:

```
burger(X) :- big_mac(X).  
burger(X) :- big_kahuna_burger(X).  
burger(X) :- whopper(X).
```

```
big_mac(a).
```

```
big_kahuna_burger(b).
```

```
big_mac(c).
```

```
whopper(d).
```

Example of negation by failure

Preferences:

```
enjoys(vincent,X) :- big_kahuna_burger(X),!,fail.  
enjoys(vincent,X) :- burger(X).
```

Data basis:

```
burger(X) :- big_mac(X).  
burger(X) :- big_kahuna_burger(X).  
burger(X) :- whopper(X).
```

big_mac(a).	?_enjoys(vincent,a): yes
big_kahuna_burger(b).	?_enjoys(vincent,b): no
big_mac(c).	?_enjoys(vincent,c): yes
whopper(d).	?_enjoys(vincent,d): yes

Example of negation by failure

simpler representation with negation:

```
enjoys(vincent,X) :- burger(X),  
    \+(big_kahuna_burger(X)).
```

Data basis:

```
burger(X) :- big_mac(X).
```

```
burger(X) :- big_kahuna_burger(X).
```

```
burger(X) :- whopper(X).
```

```
big_mac(a).
```

```
big_kahuna_burger(b).
```

```
big_mac(c).
```

```
whopper(d).
```

```
?_enjoys(vincent,a): yes
```

```
?_enjoys(vincent,b): no
```

```
?_enjoys(vincent,c): yes
```

```
?_enjoys(vincent,d): yes
```


Example of negation by failure

however, this is no classical logic!

```
enjoys(vincent,X) :- burger(X),  
  \+(big_kahuna_burger(X)).
```

Data basis:

```
burger(X) :- big_mac(X).
```

```
burger(X) :- big_kahuna_burger(X).
```

```
burger(X) :- whopper(X).
```

```
big_mac(a).
```

```
big_kahuna_burger(b).
```

```
big_mac(c).
```

```
whopper(d).
```

```
?_enjoys(vincent,X) :
```

```
X=a ;
```

```
X=c ;
```

```
X=d ;
```

```
no
```

Example of negation by failure

versus

```
enjoys(vincent,X) :- \+(big_kahuna_burger(X)), burger(X).
```

Datenbasis:

```
burger(X) :- big_mac(X).
```

```
burger(X) :- big_kahuna_burger(X).
```

```
burger(X) :- whopper(X).
```

?_enjoys(vincent,X) :

```
big_mac(a).
```

```
big_kahuna_burger(b).
```

no

```
big_mac(c).
```

```
whopper(d).
```

Be cautious when using nbf !

but very useful in many cases, also for reasons of efficiency!
... but not always:

p true, if a and b hold or if c holds but not a!

$p :- a, b.$

$p :- \backslash + a, c.$

$p :- a, !, b.$

$p :- c.$

equivalent?

What happens if $a = m(X)$, $b = n(X)$?