EARIN
Jarosław Arabas
Inference in predicate logic

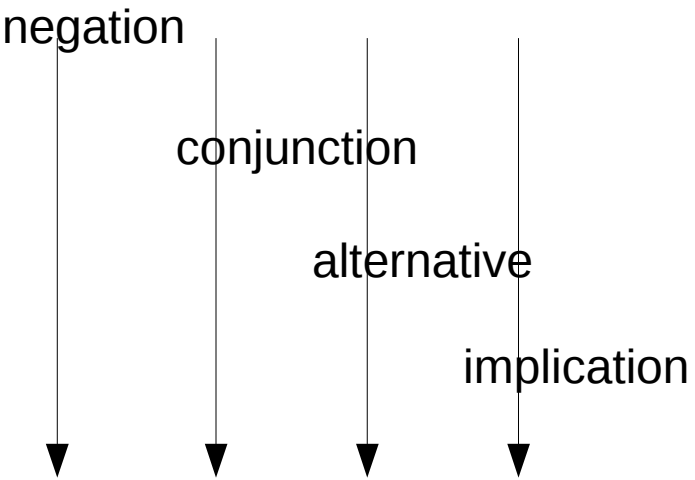
Propositional logic

- Constants and variables
- Truth value
- Connectives
- Axioms, theorems
- Closer world property (*tertium non datur*)
- Inference

Propositional calculus

- Two values: true and false {T,F} or {1,0}
- Propositional variables
  will be denoted X,Y,....
- Logical connectives (operators) of different arity
  arity 1: negation
  arity 2: alternative, conjunction, implication etc.
- Inference rules
- Axioms

Connectives

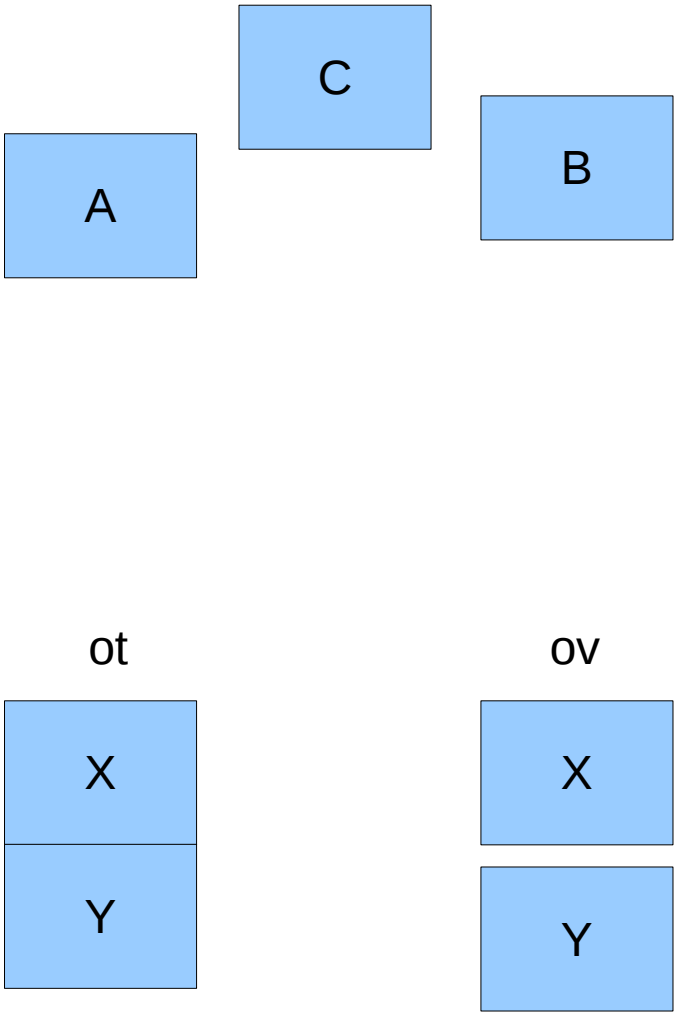


p	q	~p	p&q	p|q	p→q
0	0	1	0	0	1
0	1	1	0	1	1
1	0	0	0	1	0
1	1	0	1	1	1

Inference

1. ot(B,A) → ov(B,A)
2. ot(C,B) → ov(C,B)
3. ot(A,B) → ov(A,B)
4. ot(B,C) → ov(B,C)

5. ot(C,B)&ot(B,A) → ov(C,A)
6. ot(C,A)&ot(A,B) → ov(C,B)
7. ot(B,C)&ot(C,A) → ov(B,A)
8. ot(B,A)&ot(A,C) → ov(B,C)
9. ot(A,C)&ot(C,B) → ov(A,B)
10. ot(A,B)&ot(B,C) → ov(A,C)



Inference

- *Modus ponens:* IF(p → q AND p) THEN (q)
- *Conjunction:* IF(p1 AND p2) THEN (p1 & p2)
- *Modus tollens:* IF(p → q AND ~q) THEN (~p)
- *Alternative:* IF(~p1 AND ~p2)) THEN (~ (p1 | p2))

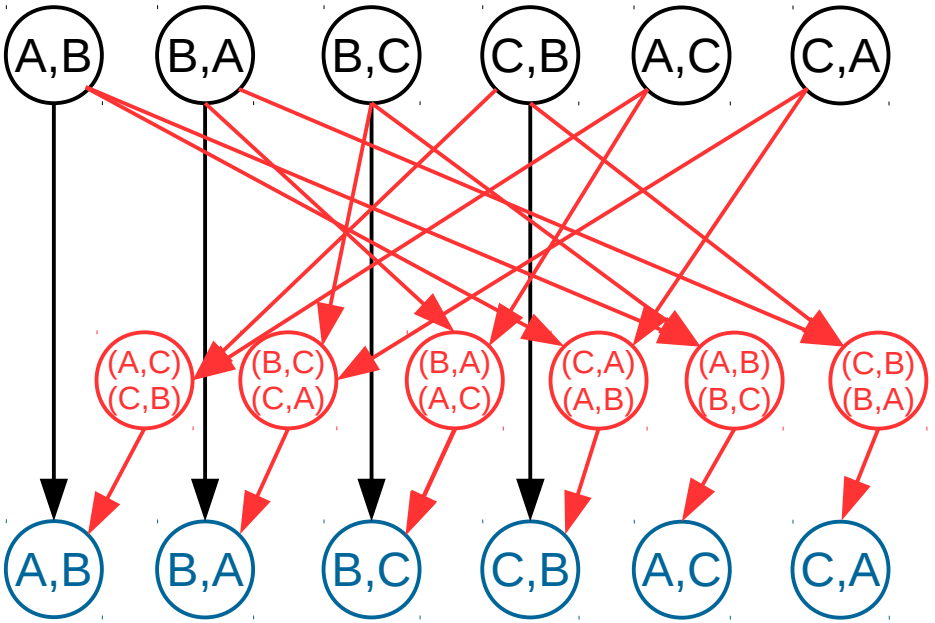
Inference

- *Modus ponens:* IF(p → q AND p) THEN (q)
- *Conjunction:* IF(p1 AND p2) THEN (p1 & p2)
- Forward chaining: *start with IF*
  goal driven inference
- Backward chaining: *start with THEN*
  data driven inference

Inference

1. ot(B,A) → ov(B,A)
2. ot(C,B) → ov(C,B)
3. ot(A,B) → ov(A,B)
4. ot(B,C) → ov(B,C)

5. ot(C,B)&ot(B,A) → ov(C,A)
6. ot(C,A)&ot(A,B) → ov(C,B)
7. ot(B,C)&ot(C,A) → ov(B,A)
8. ot(B,A)&ot(A,C) → ov(B,C)
9. ot(A,C)&ot(C,B) → ov(A,B)
10. ot(A,B)&ot(B,C) → ov(A,C)



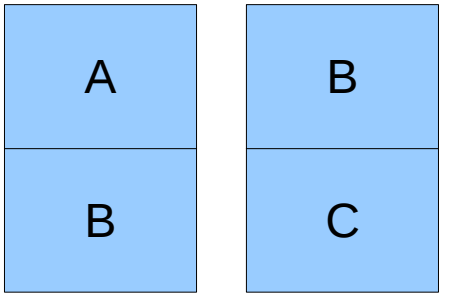
Which clauses are true?
Axioms needed

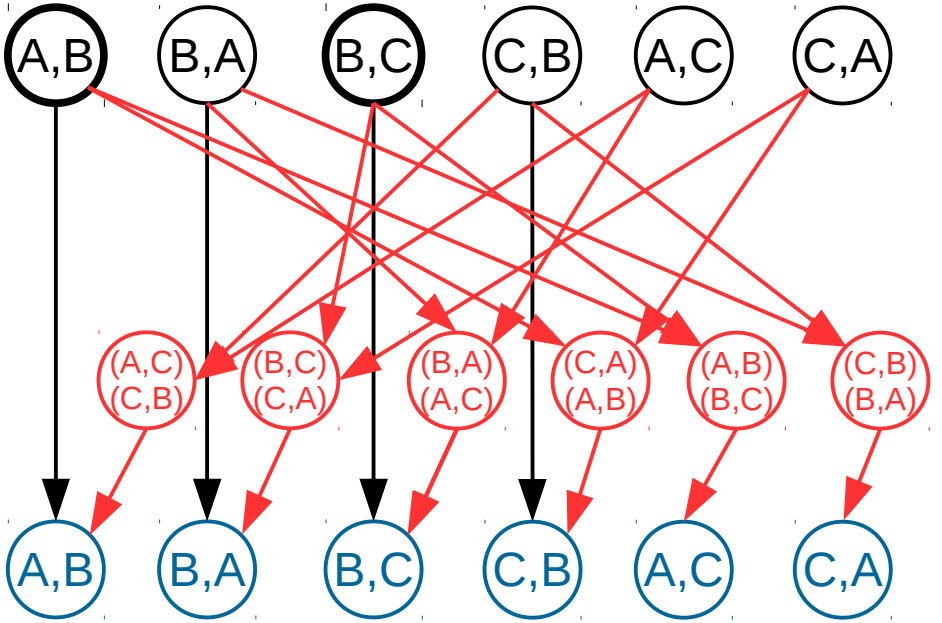
Inference

1. ot(B,A) → ov(B,A)
2. ot(C,B) → ov(C,B)
3. ot(A,B) → ov(A,B)
4. ot(B,C) → ov(B,C)

5. ot(C,B)&ot(B,A) → ov(C,A)
6. ot(C,A)&ot(A,B) → ov(C,B)
7. ot(B,C)&ot(C,A) → ov(B,A)
8. ot(B,A)&ot(A,C) → ov(B,C)
9. ot(A,C)&ot(C,B) → ov(A,B)
10. ot(A,B)&ot(B,C) → ov(A,C)

ot(A,B)
ot(B,C)



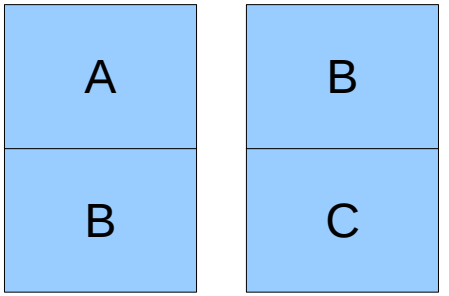


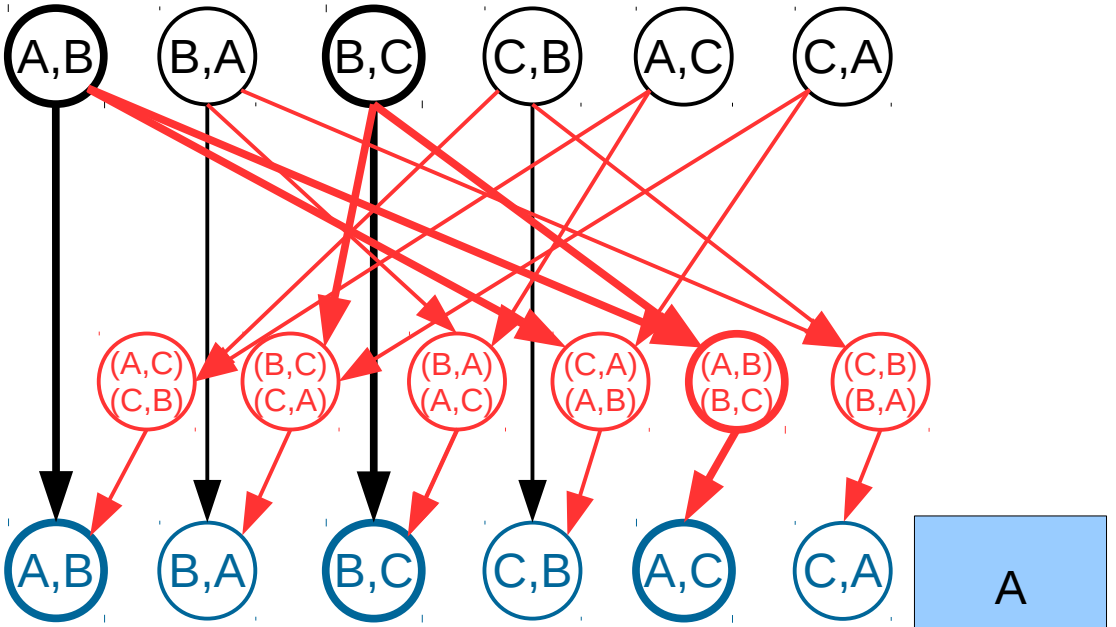
Inference

1. ot(B,A) → ov(B,A)
2. ot(C,B) → ov(C,B)
3. ot(A,B) → ov(A,B)
4. ot(B,C) → ov(B,C)

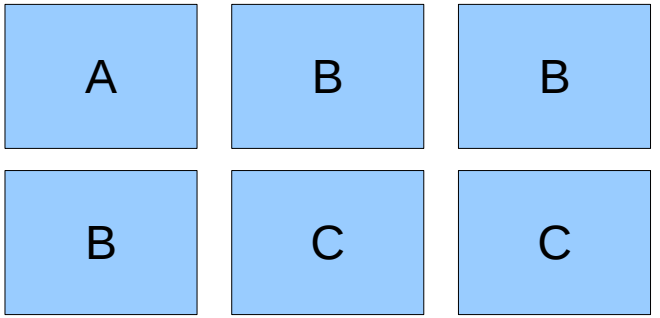
5. ot(C,B)&ot(B,A) → ov(C,A)
6. ot(C,A)&ot(A,B) → ov(C,B)
7. ot(B,C)&ot(C,A) → ov(B,A)
8. ot(B,A)&ot(A,C) → ov(B,C)
9. ot(A,C)&ot(C,B) → ov(A,B)
10. ot(A,B)&ot(B,C) → ov(A,C)

ot(A,B)
ot(B,C)





ov(A,B)
ov(B,C)
ov(A,C)

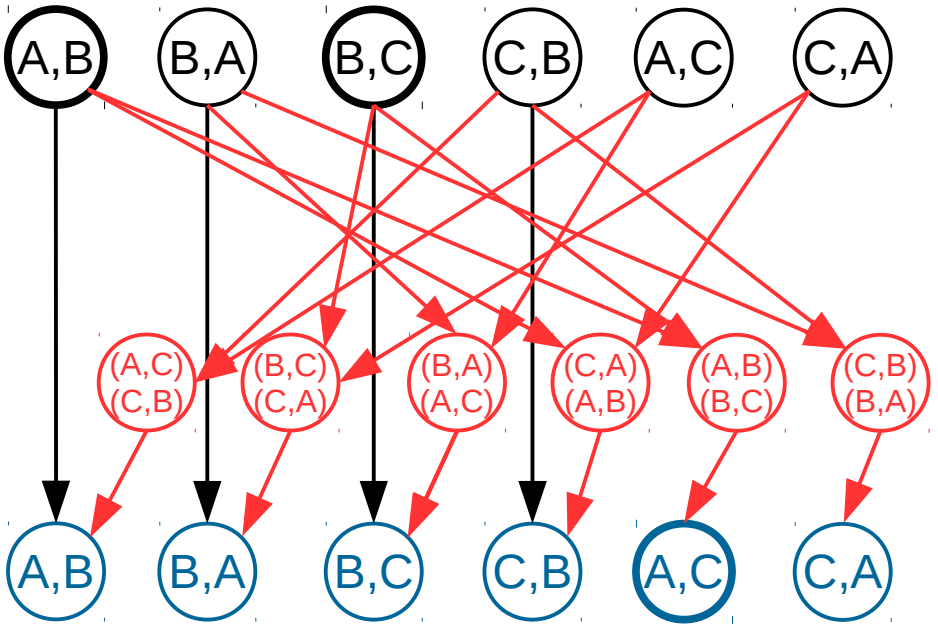


Modus ponens + conjunction

1. ot(B,A) → ov(B,A)
2. ot(C,B) → ov(C,B)
3. ot(A,B) → ov(A,B)
4. ot(B,C) → ov(B,C)

5. ot(C,B)&ot(B,A) → ov(C,A)
6. ot(C,A)&ot(A,B) → ov(C,B)
7. ot(B,C)&ot(C,A) → ov(B,A)
8. ot(B,A)&ot(A,C) → ov(B,C)
9. ot(A,C)&ot(C,B) → ov(A,B)
10. ot(A,B)&ot(B,C) → ov(A,C)

ot(A,B)
ot(B,C)



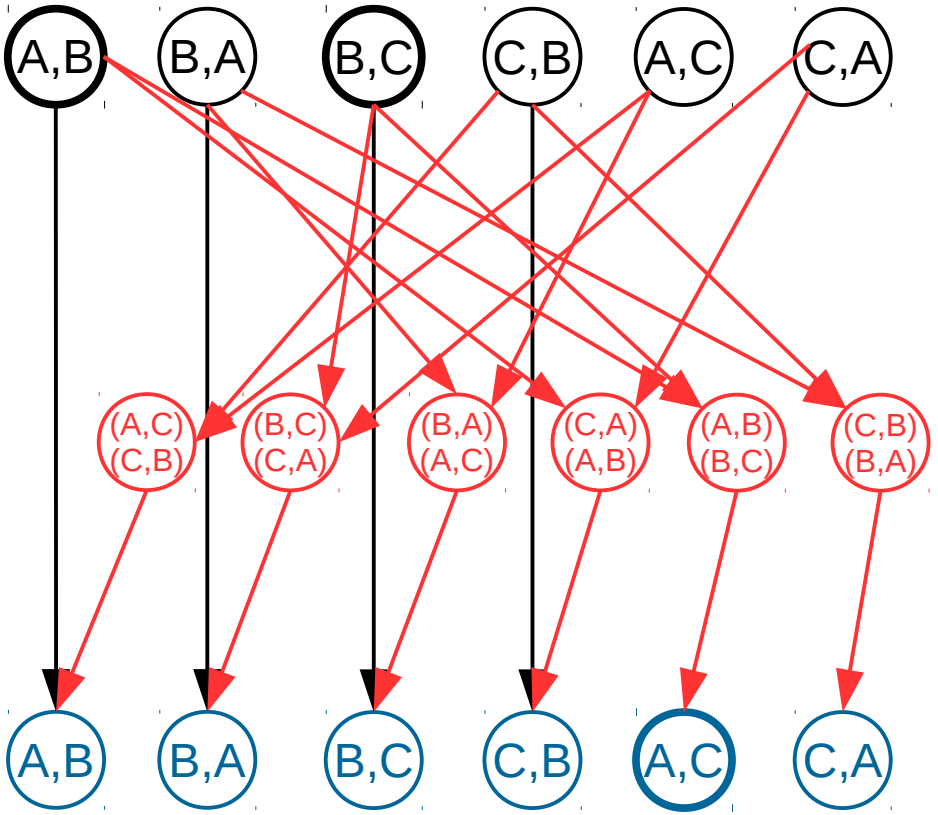
?ov(A,C)

Modus ponens + conjunction

ot(A,B)
ot(B,C)

?ov(A,C)

Find a tree
rooted in ov(AC)
s.t. all leaves are T



Modus ponens + conjunction backward chaining

?ov(A,C)

10.ot(A,B)&ot(B,C)→ov(A,C)

IF(p→q AND p) THEN (q)

Modus ponens + conjunction backward chaining

?ov(A,C)

10.ot(A,B)&ot(B,C)→ov(A,C)

IF(p→q AND p) THEN (q)

?ot(A,B) & ot(B,C)

IF(p1 AND p2) THEN (p1&p2)

Modus ponens + conjunction backward chaining

?ov(A,C)

10.ot(A,B)&ot(B,C)→ov(A,C)

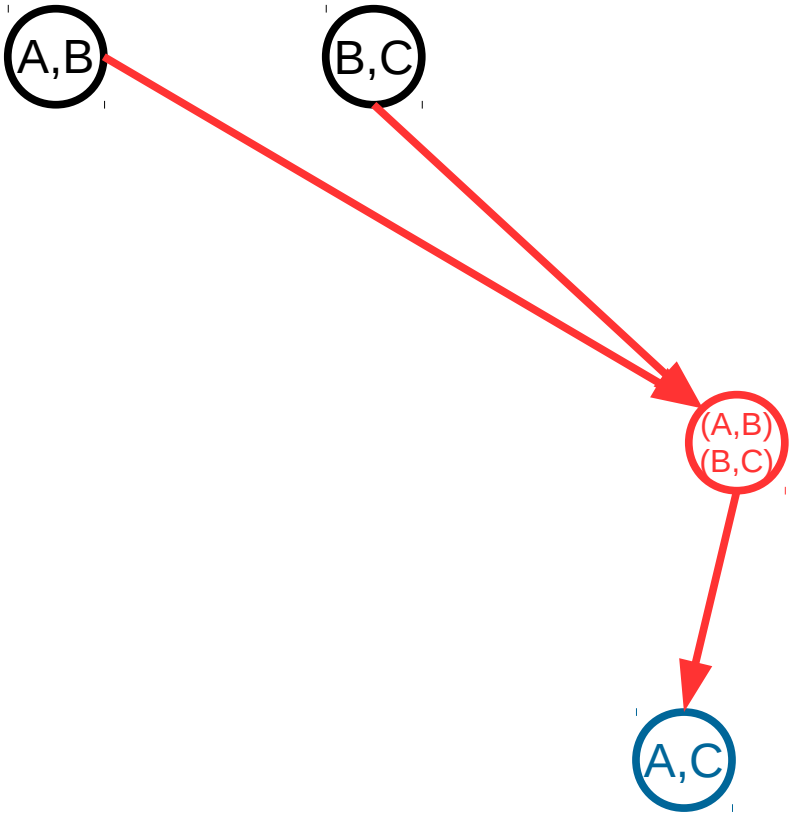
IF(p→q AND p) THEN (q)

?ot(A,B) & ot(B,C)

IF(p1 AND p2) THEN (p1&p2)

?ot(A,B) OK

?ot(B,C) OK



Modus ponens + conjunction forward chaining

ot(A,B)
ot(B,C)

IF(p1 AND p2) THEN (p1&p2)

Modus ponens + conjunction forward chaining

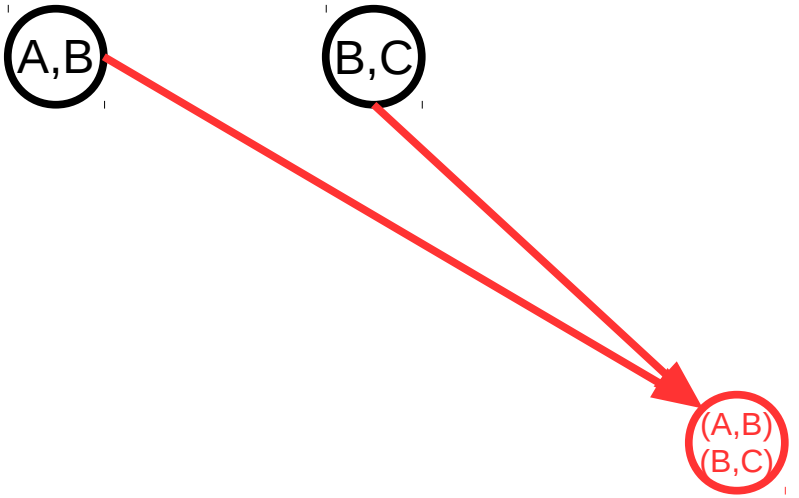
ot(A,B)
ot(B,C)

IF(p1 AND p2) THEN (p1&p2)

ot(A,B) & ot(B,C)

IF(p → q AND p) THEN (q)

10.ot(A,B)&ot(B,C) → ov(A,C)



Modus ponens + conjunction forward chaining

ot(A,B)
ot(B,C)

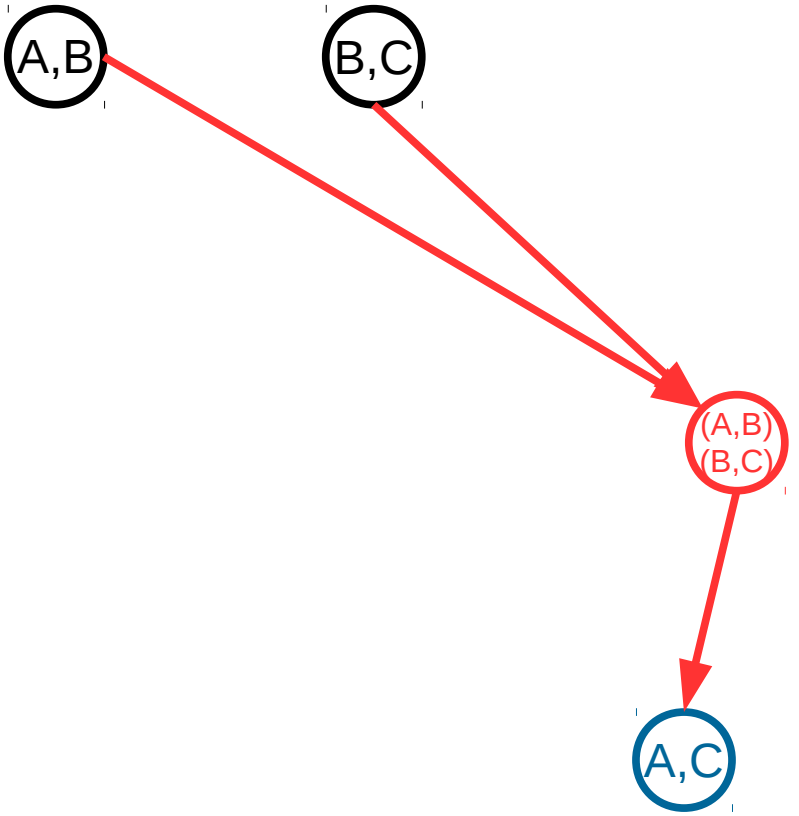
IF(p1 AND p2) THEN (p1&p2)

ot(A,B) & ot(B,C)

IF(p → q AND p) THEN (q)

10.ot(A,B)&ot(B,C) → ov(A,C)

ov(A,C)



Modus ponens + conjunction forward chaining

ot(A,B)
ot(B,C)

IF(p1 AND p2) THEN (p1&p2)

ot(A,B) & ot(B,C)

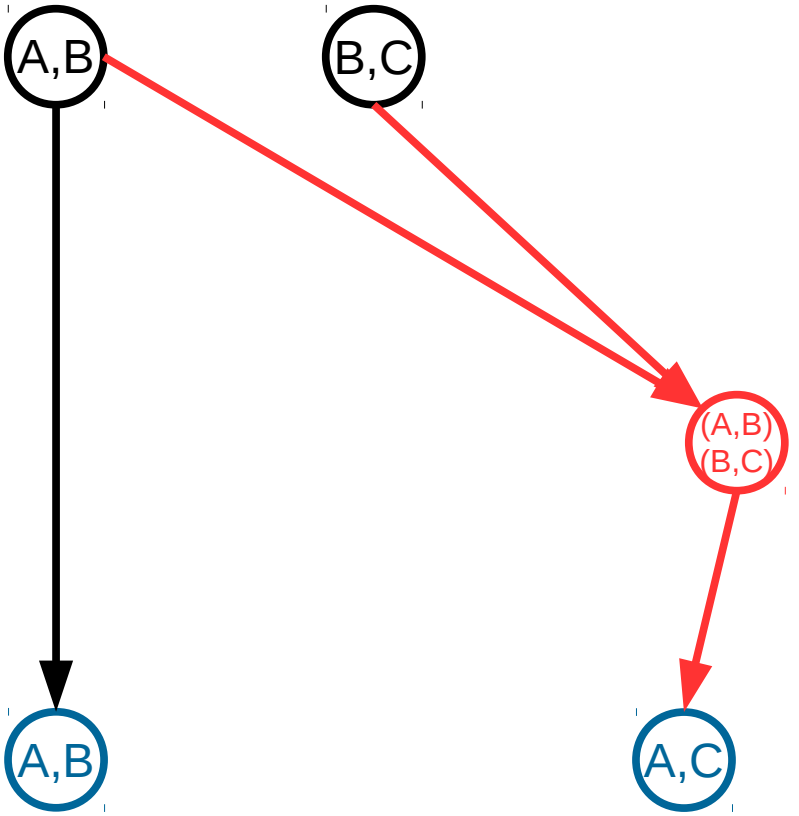
IF(p → q AND p) THEN (q)

10.ot(A,B)&ot(B,C) → ov(A,C)

ov(A,C)

IF(p → q AND p) THEN (q)

ov(A,B)



Modus ponens + conjunction forward chaining

ot(A,B)
ot(B,C)

IF(p1 AND p2) THEN (p1&p2)

ot(A,B) & ot(B,C)

IF(p → q AND p) THEN (q)

10.ot(A,B)&ot(B,C) → ov(A,C)

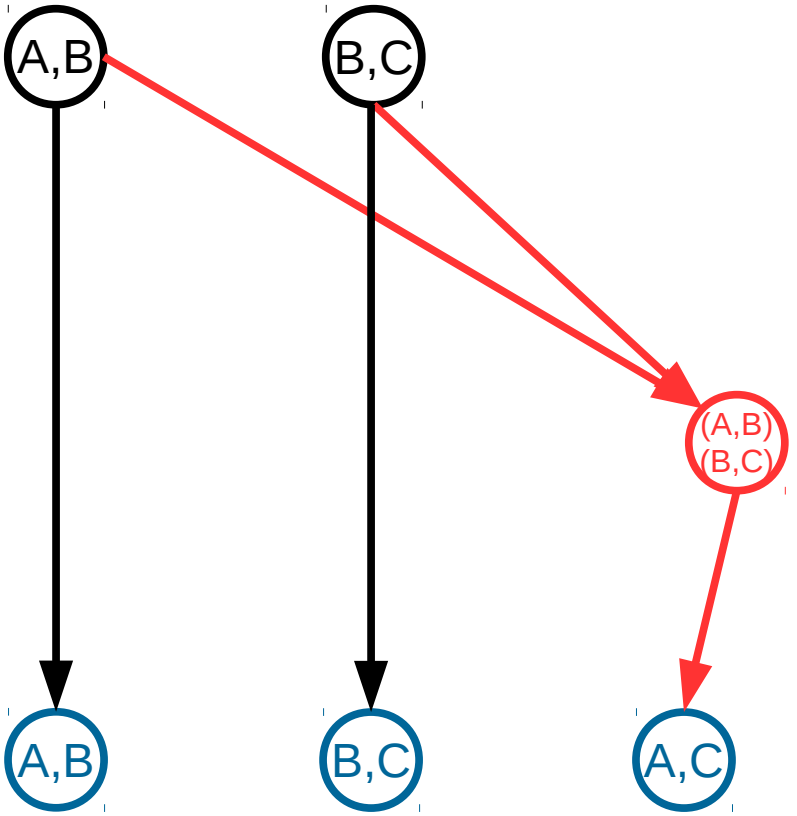
ov(A,C)

IF(p → q AND p) THEN (q)

ov(A,B)

IF(p → q AND p) THEN (q)

ov(B,C)

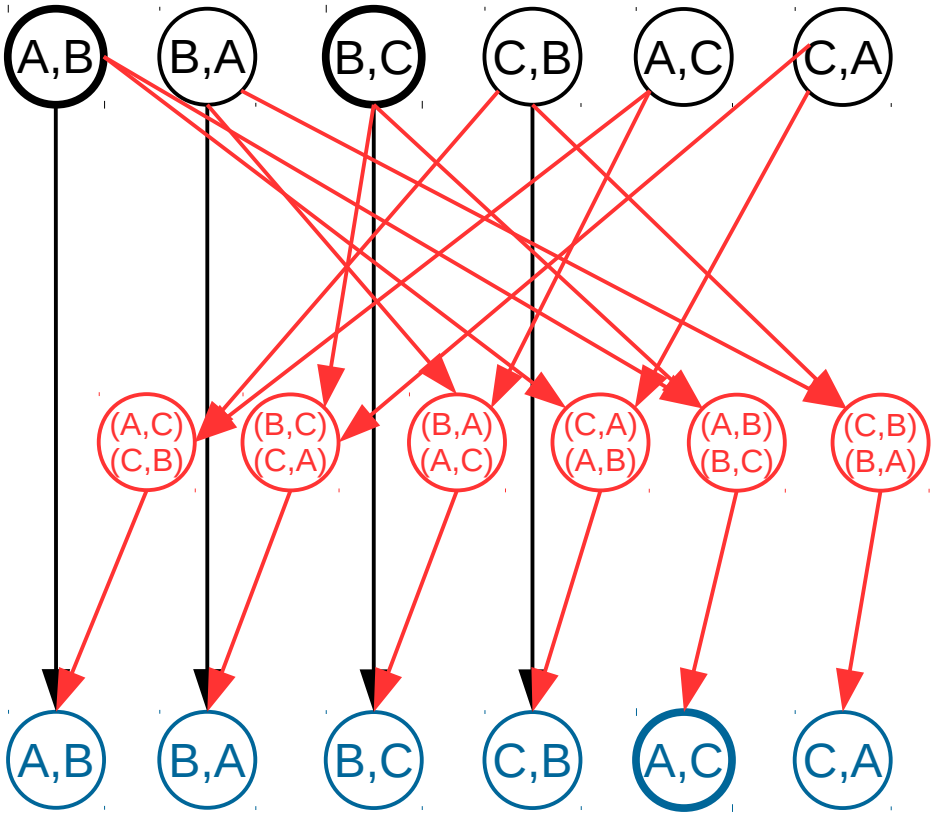


Inference as a search task

ot(A,B)
ot(B,C)

?ov(A,C)

Find a tree
rooted in ov(AC)
s.t. all leaves are T



Inference as a search task

- How to represent a tree in a graph of clauses?
- What elements contains the search space?
- How to transform the search space elements?
- How to evaluate search space elements?
- Which search method should be used?

Inference as a search task

?ov(A,C)

10.ot(A,B)&ot(B,C)→ov(A,C)

IF(p→q AND p) THEN (q)

?ot(A,B) & ot(B,C)

IF(p1 AND p2) THEN (p1 & p2)

?ot(A,B) OK

?ot(B,C) OK

ot(A,B)

ot(B,C)

IF(p1 AND p2) THEN (p1 & p2)

ot(A,B) & ot(B,C)

IF(p→q AND p) THEN (q)

10.ot(A,B)&ot(B,C)→ov(A,C)

ov(A,C)

Search space

- Example proof (forward chaining):

1. conjunction
   IF(p1 AND p2) THEN (p1 & p2)
   p1=ot(A,B);p2=ot(B,C)
2. modus ponens
   IF(p → q AND p) THEN (q)
   p=ot(A,B) & ot(B,C) ; q=ov(A,C)

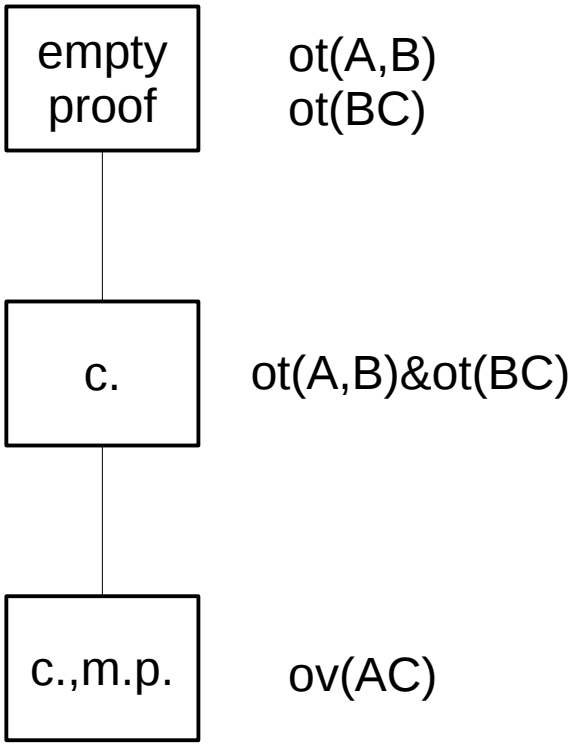
- Example proof (backward chaining):

1. modus ponens
   IF(p → q AND p) THEN (q)
   p=ot(A,B) & ot(B,C) ; q=ov(A,C)
2. conjunction
   IF(p1 AND p2) THEN (p1 & p2)
   p1=ot(A,B);p2=ot(B,C)

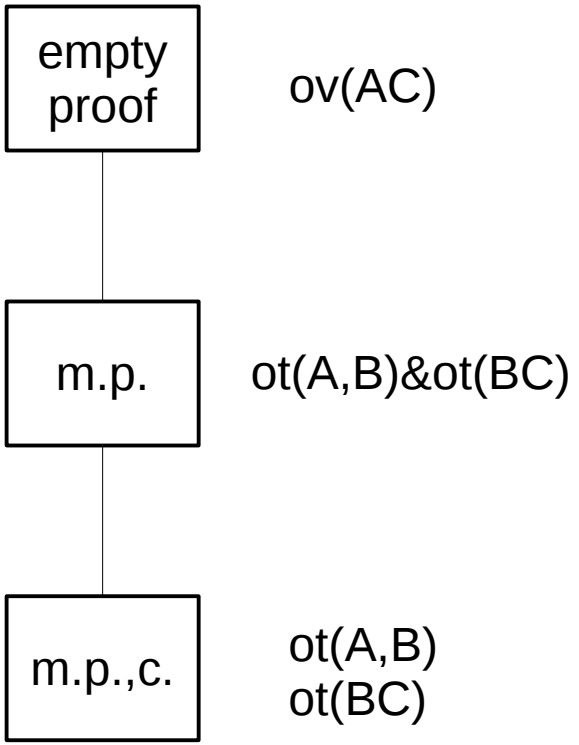
Search space

- Proof – a sequence of inference rules with instantiation of their variables
- All priors of inference rules must be:
  - Connected with the goal (backward chaining)
  - Connected with axioms (forward chaining)
- A proof is complete if:
  - All clauses with no prior are axioms (backward chaining)
  - Goal has been achieved (forward chaining)

Two different search spaces



Forward chaining



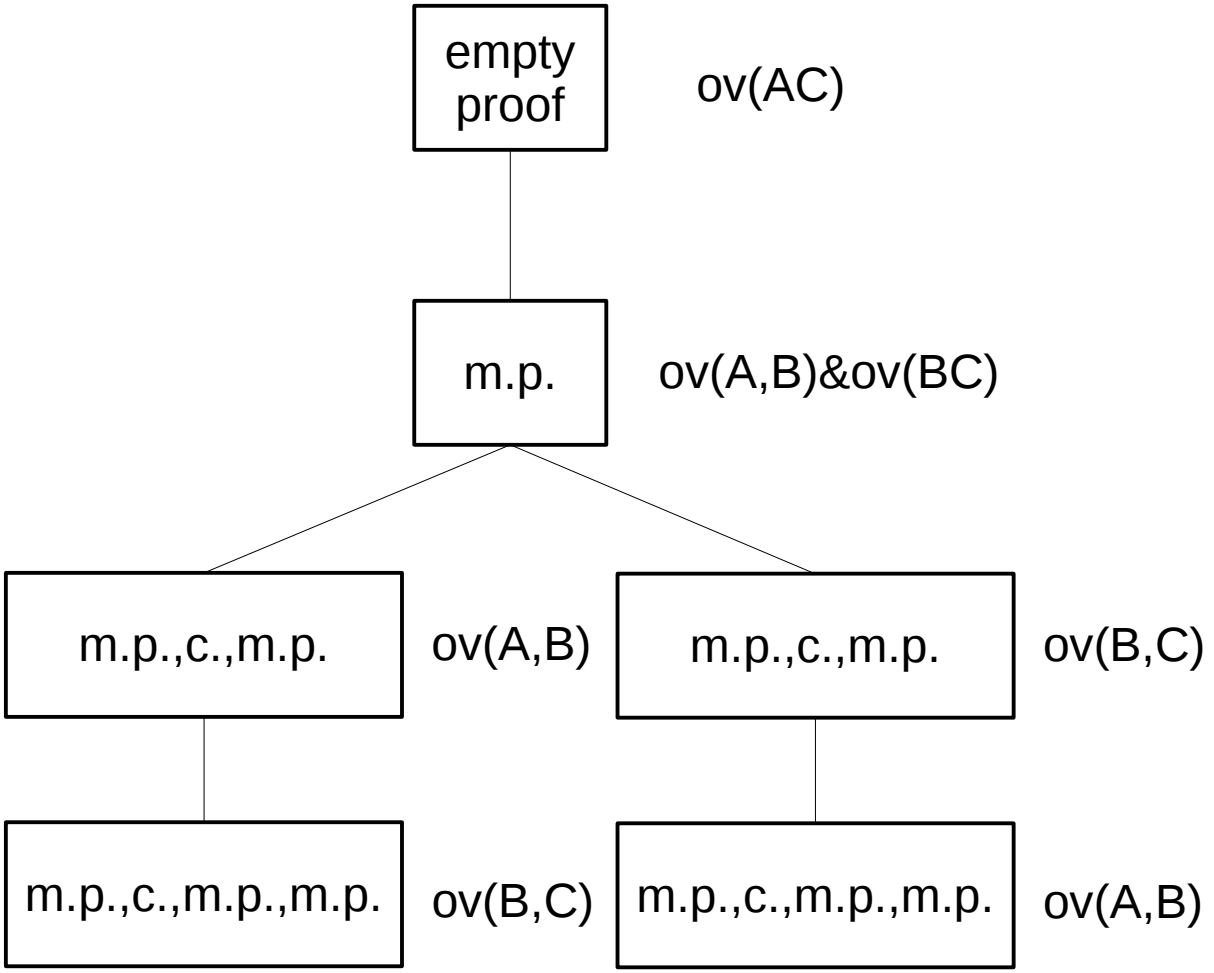
Backward chaining

A different set of rules

1. ot(B,A) → ov(B,A)
2. ot(C,B) → ov(C,B)

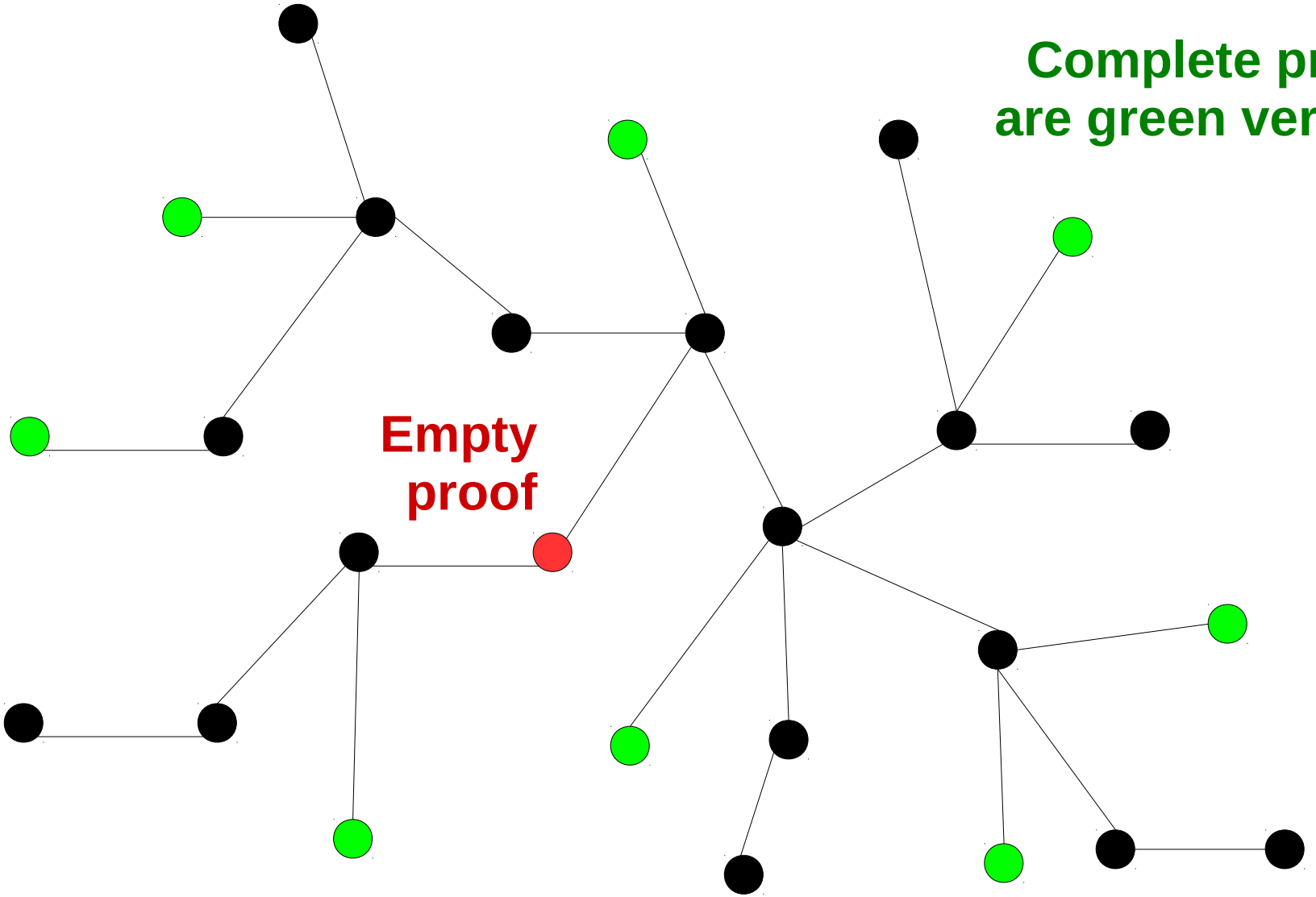
3. ot(A,B) → ov(A,B)
4. ot(B,C) → ov(B,C)

5. ov(C,B)&ov(B,A) → ov(C,A)
6. ov(C,A)&ov(A,B) → ov(C,B)
7. ov(B,C)&ov(C,A) → ov(B,A)
8. ov(B,A)&ov(A,C) → ov(B,C)
9. ov(A,C)&ov(C,B) → ov(A,B)
10. ov(A,B)&ov(B,C) → ov(A,C)

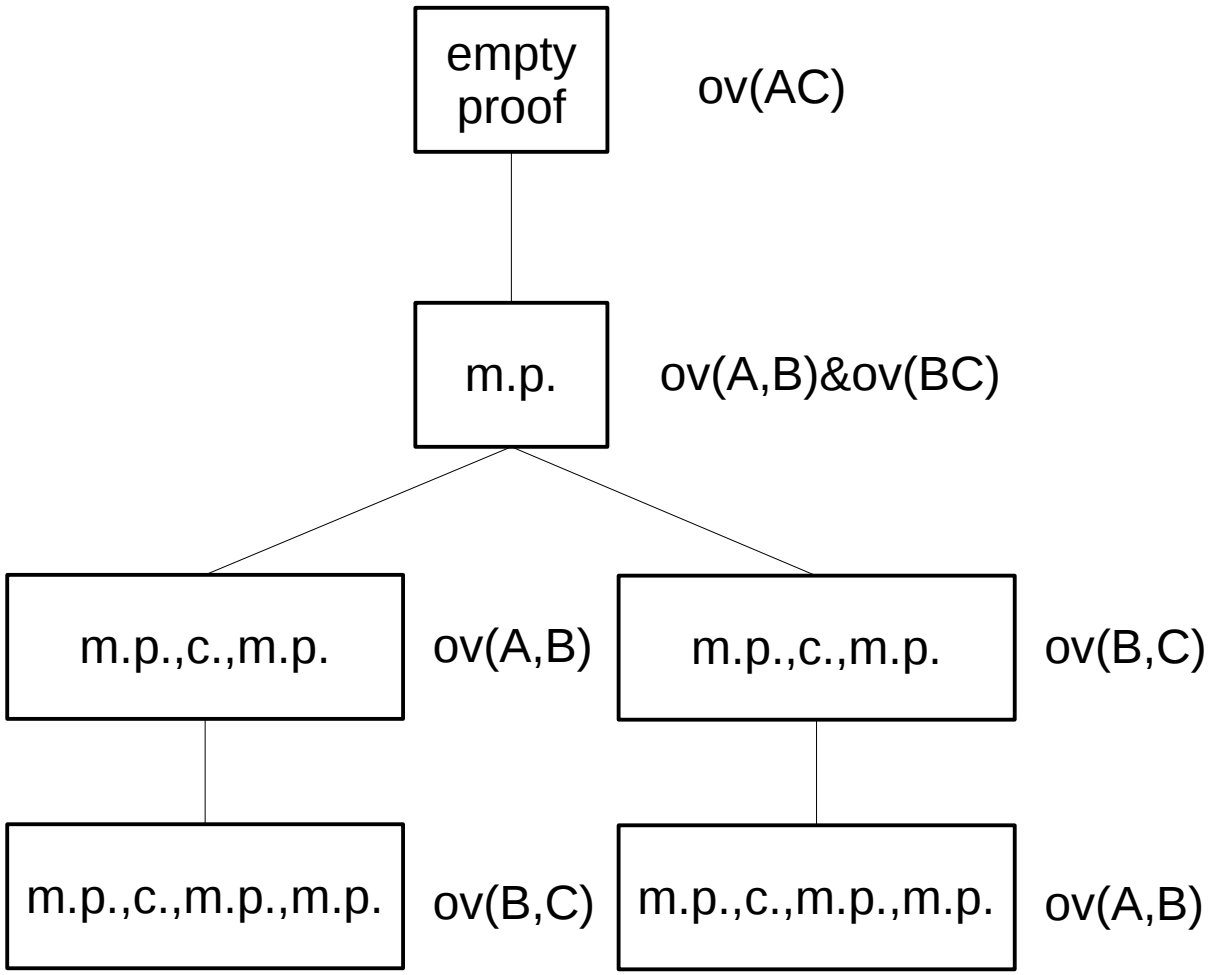


Backward chaining

Structure of the search space



Search method in space of proofs



Backward chaining

Uninformed search

Breadth-first
Shortest proof
Sensitive to branching

Depth-first
Possibly faster proof
Sensitive to looping
and infinite proofs

Predicate calculus

- Variables
  - A variable represents a set of values – its domain
- Predicates
  - q(X) – a logical function of a variable
- Example predicates:

ot(X,Y)

		X		
		A	B	C
Y	A			
	B	1		
	C		1	

ov(X,Y)

		X		
		A	B	C
Y	A			
	B	1		
	C	1	1	

eq(X,Y)

		X		
		A	B	C
Y	A	1		
	B		1	
	C			1

Predicate calculus

- Quantifiers – relations between predicates
  - Existential quantifier $\exists$
    there exists a value of X s.t. q(x) is true
  - Universal quantifier $\forall$
    for all values of X it holds q(x) is true

$$\forall X \forall Y\, ot(X,Y) \rightarrow ov(X,Y)$$
$$\exists X \exists Y\, ov(Y,Y) \wedge \sim ot(X,Y)$$

		X		
		A	B	C
Y	A			
	B	1		
	C		1	

ot(X,Y)

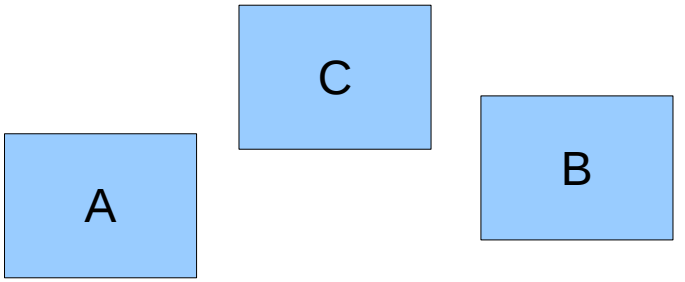
		X		
		A	B	C
Y	A			
	B	1		
	C	1	1	

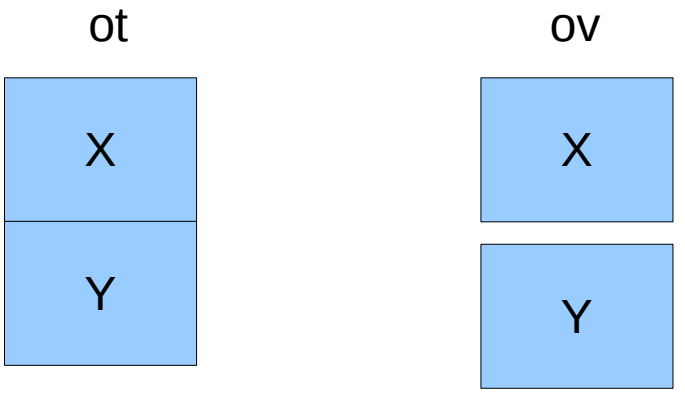
ov(X,Y)

Predicate calculus

1. ot(X,Y) → ov(X,Y)
2. ot(X,Y)&ot(Y,Z) → ov(X,Z)

X,Y are variables
from the domain {A,B,C}





Predicate calculus

- Skolem normal form
  - All universal quantifiers
  - Alternative of rules
  - Conjunctions in premise parts of rules

Conjunctive normal form (CNF)

- Normalization rules

$p \leftrightarrow q$	==	$p \rightarrow q \;\&\; q \rightarrow p$
$p \rightarrow q$	==	$\sim p \;\&\; q$
$\sim(\sim p)$	==	p
$\sim(p \;\&\; q)$	==	$\sim p \mid \sim q$
$\sim(p \mid q)$	==	$\sim p \;\&\; \sim q$
$p \mid (q \;\&\; r)$	==	$(p \mid q) \;\&\; (p \mid r)$
$p \;\&\; (q \mid r)$	==	$(p \& q) \mid (p \& r)$

Skolem normal form (SNF)

- Quantifier elimination rules

$\sim(\forall X)\ p(X) \quad == \quad \exists X\ \sim p(X)$

$\sim(\exists X)\ p(X) \quad == \quad \forall X\ \sim p(X)$

$((\forall X)\ p(X)\ )\ \#\ q \quad == \quad (\forall X)\ (p(X)\ \#\ q)$

$((\exists X)\ p(X)\ )\ \#\ q \quad == \quad (\exists X)\ (p(X)\ \#\ q)$

where # stands for & , |, →, ↔

Skolem normal form (SNF)

- Skolemization
  - Apply quantifier elimination rules and CNF
  - Substitute existential quantifier variables with
    - Constants, when they are not preceded with universal quantifier
    - Functions when they are predeced with universal quantifier

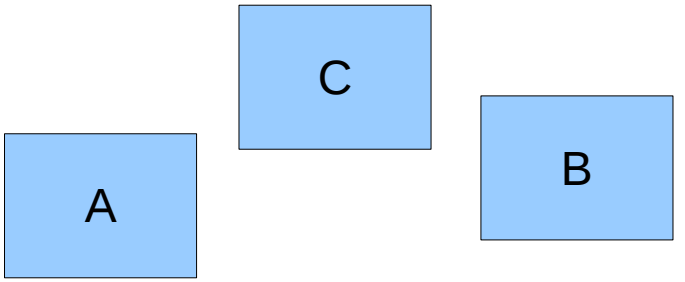
PROLOG

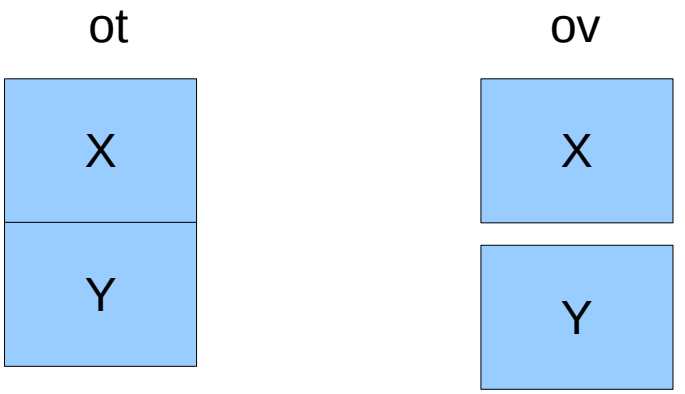
- PROgramming in LOGic
- A language to define rules and axioms
  - A declarative language
  - Predicates in the Skolem normal form
- Inference machine
  - Modus tollens
  - Backward chaining (goal driven)
  - Depth first

Rules and facts in PROLOG

1. ot(X,Y) → ov(X,Y)
2. ot(X,Y)&ot(Y,Z) → ov(X,Z)

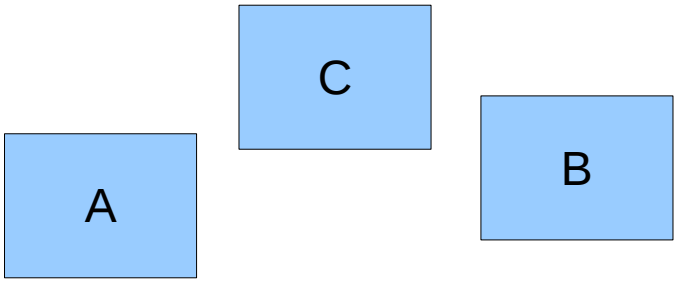
X,Y are variables
from the domain {A,B,C}



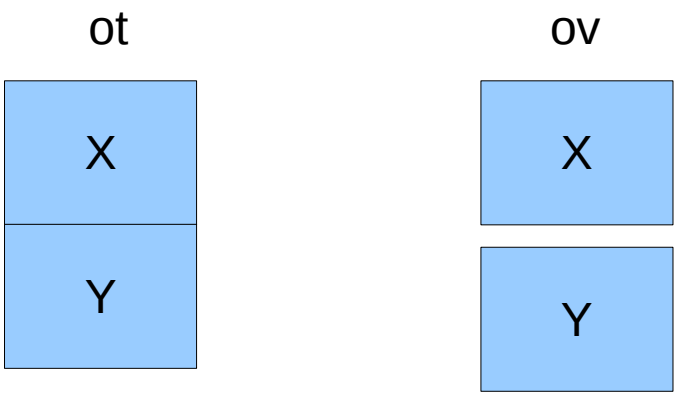


Rules and facts in PROLOG

1. ot(X,Y) → ov(X,Y)
2. ot(X,Y)&ot(Y,Z) → ov(X,Z)




```
over(X,Y) :- ontop(X,Y).
over(X,Y) :- ontop(X,Z),ontop(Z,Y).

ontop(brickA,brickB).
ontop(brickB,brickC).

?over(brickA,brickC).
'yes'
```



Substitution of variables

- A variable is substituted with a term
- *Similar* but *not identical* to assignment of a value
- Actual value of the variable will equal the actual value of the term used for substitution

```
over(X,Y) :- ontop(X,Y).
over(X,Y) :- ontop(X,Z),ontop(Z,Y).

ontop(brickA,brickB).
ontop(brickB,brickC).

?over(brickA,brickC).
```

```
over(X,Y) :- ontop(X,Y).
?over(brickA,brickC)
```

X/brickA, Y/brickC

Substitution of variables

- A variable is substituted with a term
- *Similar* but *not identical* to assignment of a value
- Actual value of the variable will equal the actual value of the term used for substitution

```
over(X,Y) :- ontop(X,Y).
over(X,Y) :- ontop(X,Z),ontop(Z,Y).

ontop(brickA,brickB).
ontop(brickB,brickC).

?over(brickA,brickC).
```

```
over(X,Y) :-
   ontop(X,Z),ontop(Z,Y).
?over(brickA,brickC)
```

X/brickA, Y/brickC, Z/domain(Z)

Substitution of variables

- A variable is substituted with a term
- *Similar* but *not identical* to assignment of a value
- Actual value of the variable will equal the actual value of the term used for substitution

```
over(X,Y) :- ontop(X,Y).
over(X,Y) :- ontop(X,Z),ontop(Z,Y).

ontop(brickA,brickB).
ontop(brickB,brickC).

?over(X,brickC).
```

```
over(X,Y) :- ontop(X,Y).
?over(X,brickC)
```

X/domain(X), Y/brickC

Unification of variables

- Unification – agreement of substitutions

```
over(X,Y) :- ontop(X,Y).
over(X,Y) :- ontop(X,Z),ontop(Z,Y).

ontop(brickA,brickB).
ontop(brickB,brickC).

?over(brickA,brickC).
```

```
over(X,Y) :-
   ontop(X,Z),ontop(Z,Y).
?over(brickA,brickC)
```

X/brickA, Y/brickC, Z/domain(Z)

ontop(X/brickA,Z/brickB)

ontop(Z/brickB,Y/brickC)

SLD inference

- Horn clauses
  - Implication: p1 & p2 → q
  - Horn clause: ~p1 | ~p2 | q
- Selective Linear Definite clause resolution
  - a clause: ~q1 |...|~qi | … ~qn
  - a Horn clause: ~p1 | ~p2 | qi
  - *modus tollens*: IF ~q and p → q THEN ~p
  - substitute: ~q1 |...|~p1|~p2 | … ~qn

SLD inference

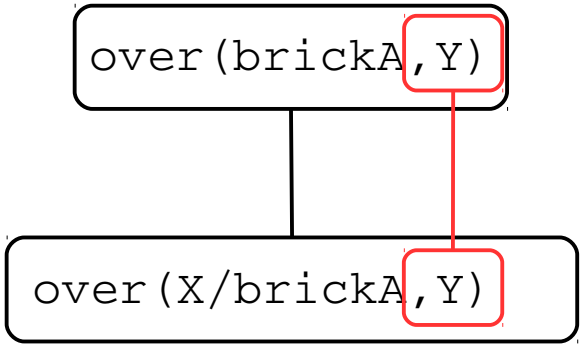
- Selective Linear Definite clause resolution
  - negate the goal A={~q}
  - apply the SLD resolution for rules and A
  - given
    - a goal clause: A = ~q1 |... ~qi | … ~qn
    - a Horn clause: ~p1 | ~p2 | qi
  - define
    - a new goal clause A = ~q1 |... ~p1| ~p2 | … ~qn
  - If the goal clause contains all negated axioms then it becomes false, ~q is false and q is true

Inference in PROLOG

$procedure\ \mathbf{inference}\left(\mathbf{q_0}\right)$
$A \leftarrow \{\sim q_0\}$
repeat
  $\mathbf{q} \leftarrow popLIFO\left(A\right)$
  $P \leftarrow premises\ of\ rules\ that\ match\ \mathbf{q}$
  $P \leftarrow unify\ variables\left(P, \mathbf{q}\right)$
  $pushLIFO\left(A, P\right)$
until $P \neq \emptyset$
if $\left(\forall\, \mathbf{q} \in A\right) \mathbf{q} = F$ ***then***
  $\mathbf{q_0} = T$
else
  $\mathbf{q_0} = F$

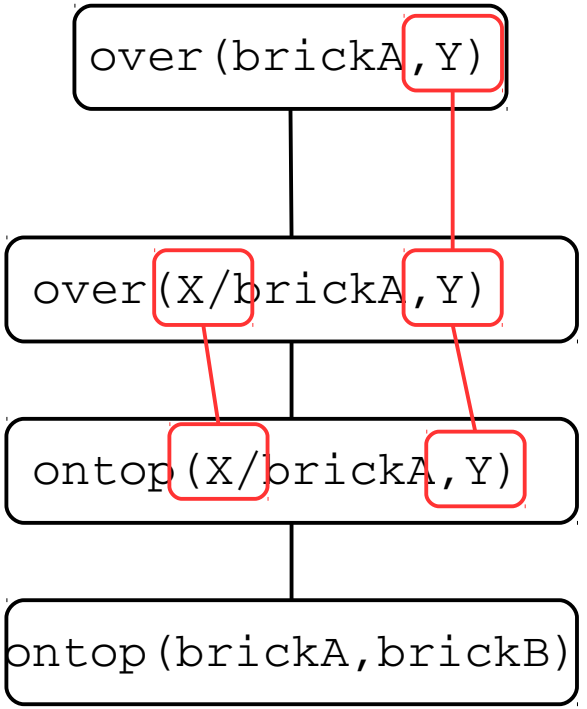
Inference in PROLOG

```
over(X,Y) :- ontop(X,Y).
over(X,Y) :- ontop(X,Z),ontop(Z,Y).

ontop(brickA,brickB).
ontop(brickB,brickC).

?over(brickA,Y).
```



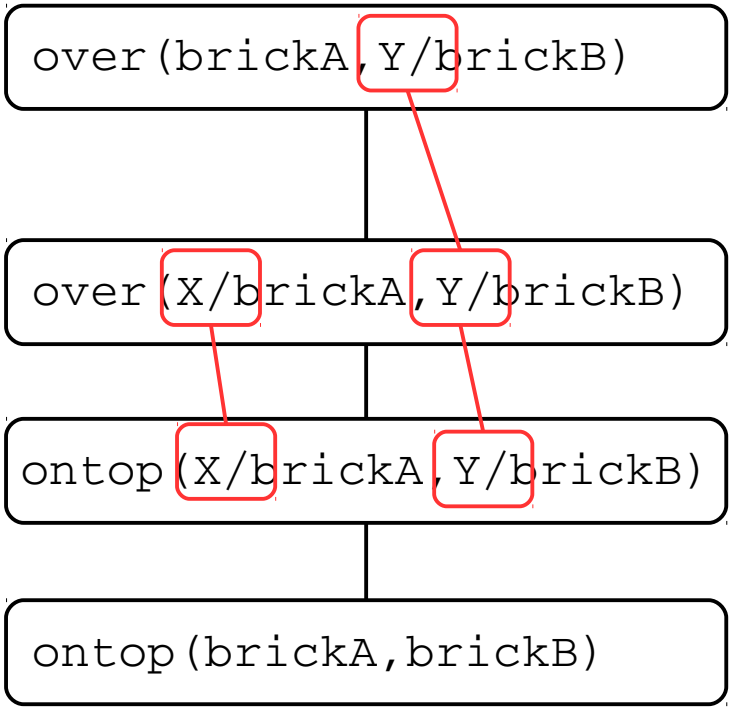
Inference in PROLOG

```
over(X,Y) :- ontop(X,Y).
over(X,Y) :- ontop(X,Z),ontop(Z,Y).

ontop(brickA,brickB).
ontop(brickB,brickC).

?over(brickA,Y).
```



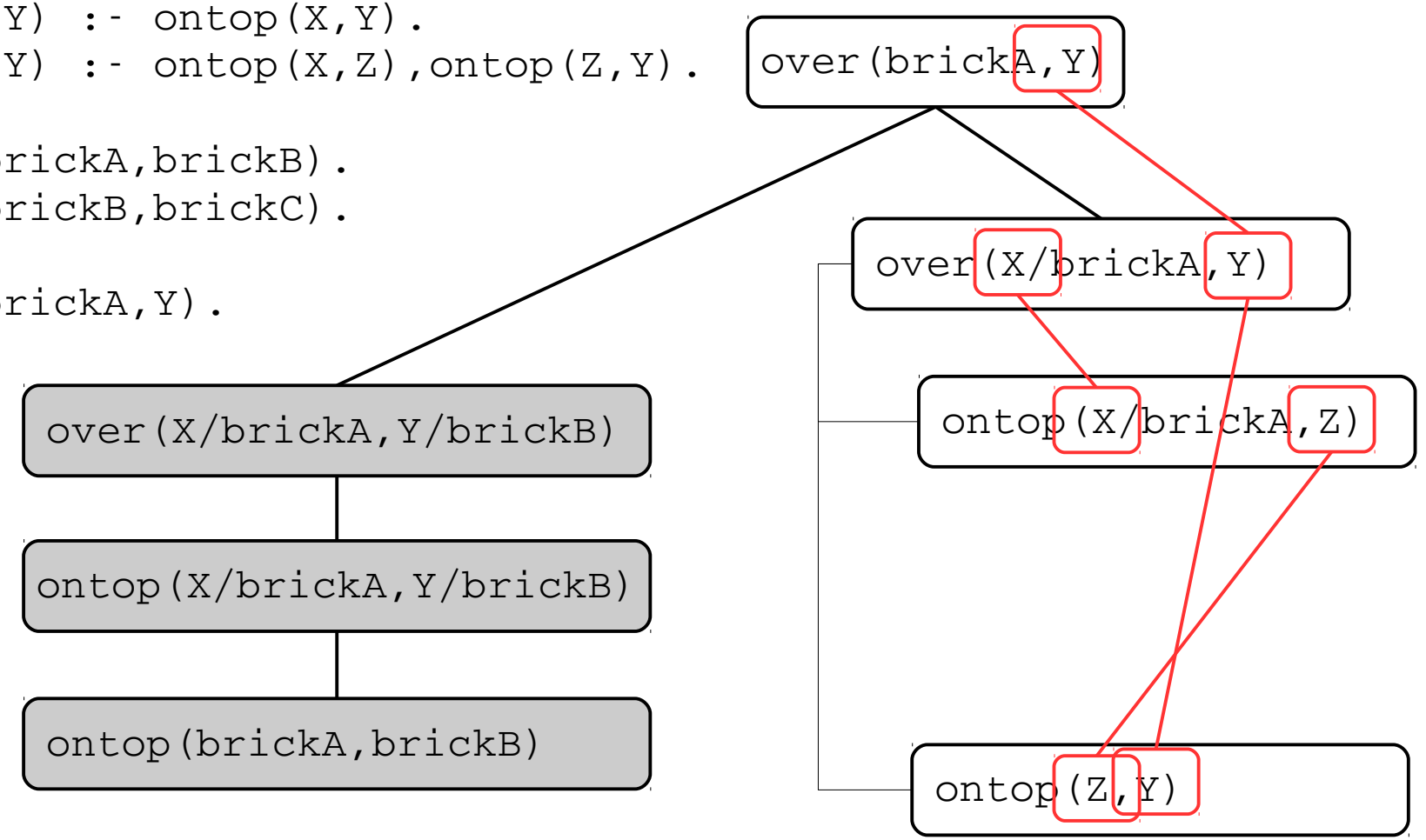
Inference in PROLOG

```
over(X,Y) :- ontop(X,Y).
over(X,Y) :- ontop(X,Z),ontop(Z,Y).

ontop(brickA,brickB).
ontop(brickB,brickC).

?over(brickA,Y).
```



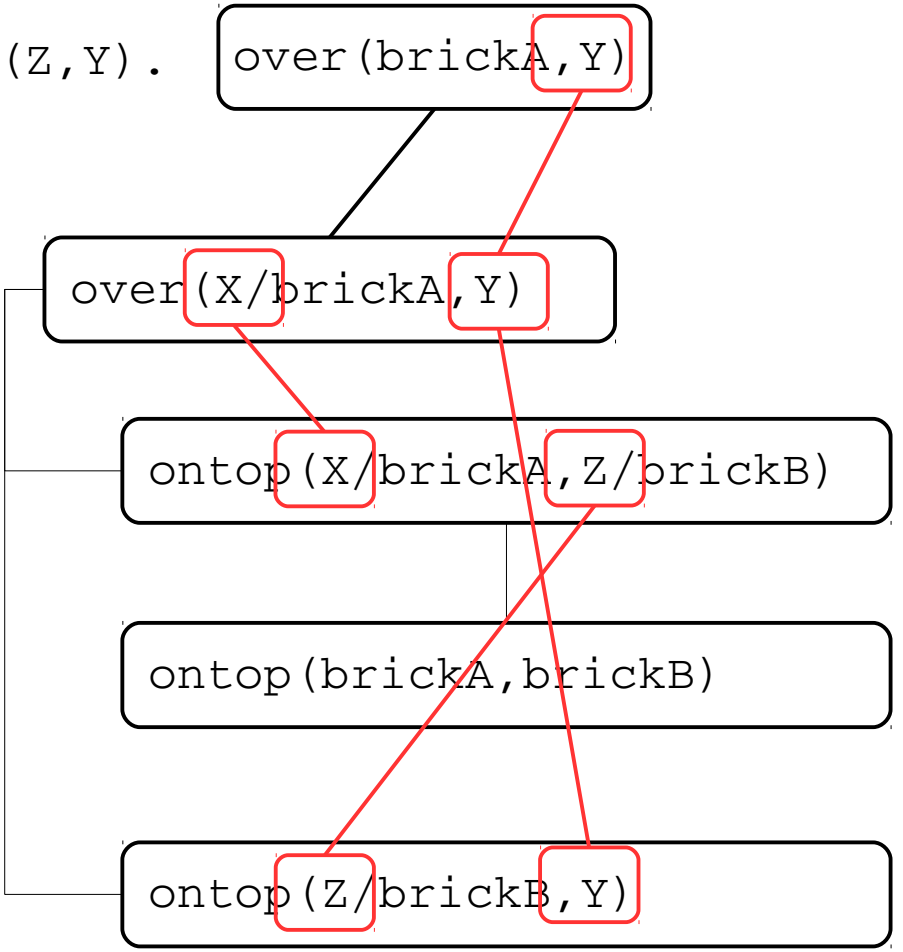
Inference in PROLOG

```
over(X,Y)  :-  ontop(X,Y).
over(X,Y)  :-  ontop(X,Z),ontop(Z,Y).

ontop(brickA,brickB).
ontop(brickB,brickC).

?over(brickA,Y).
```



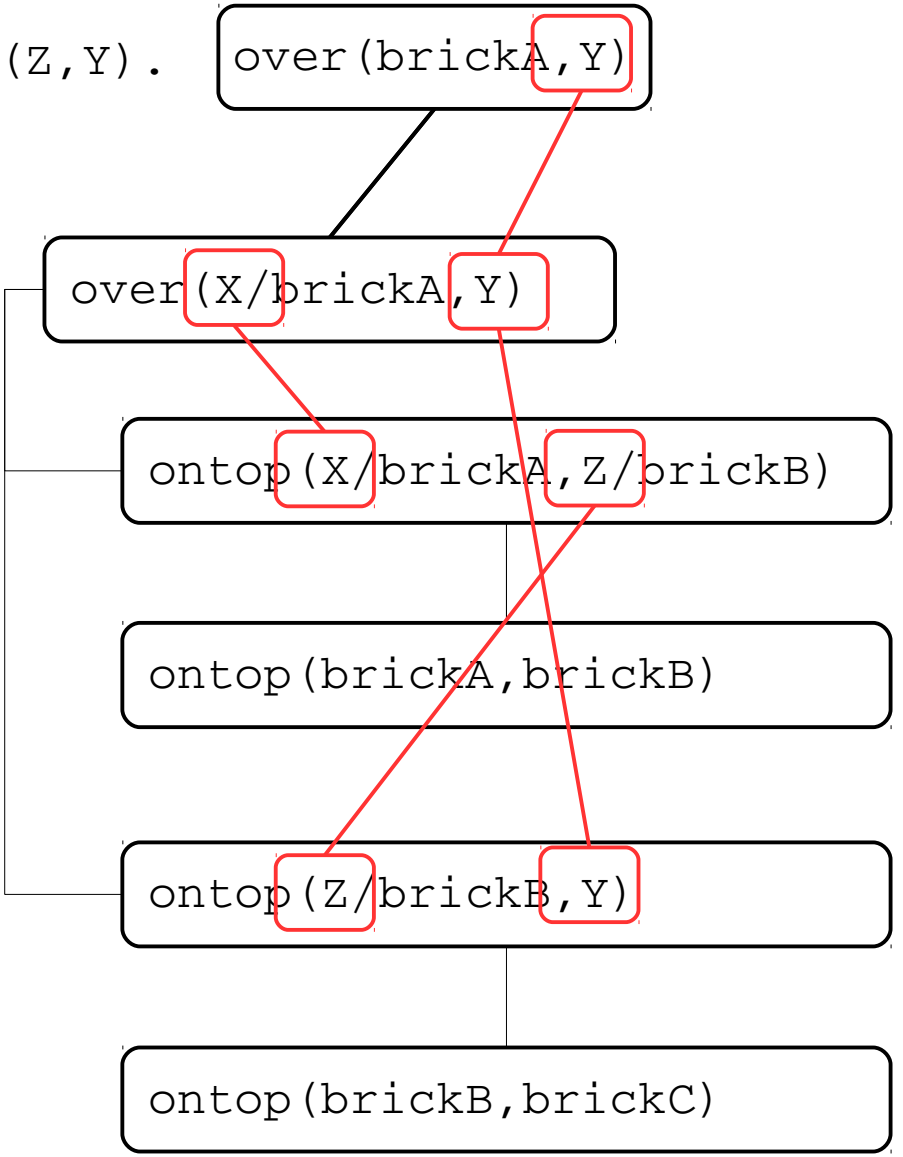
Inference in PROLOG

```
over(X,Y)  :-  ontop(X,Y).
over(X,Y)  :-  ontop(X,Z),ontop(Z,Y).

ontop(brickA,brickB).
ontop(brickB,brickC).

?over(brickA,Y).
```



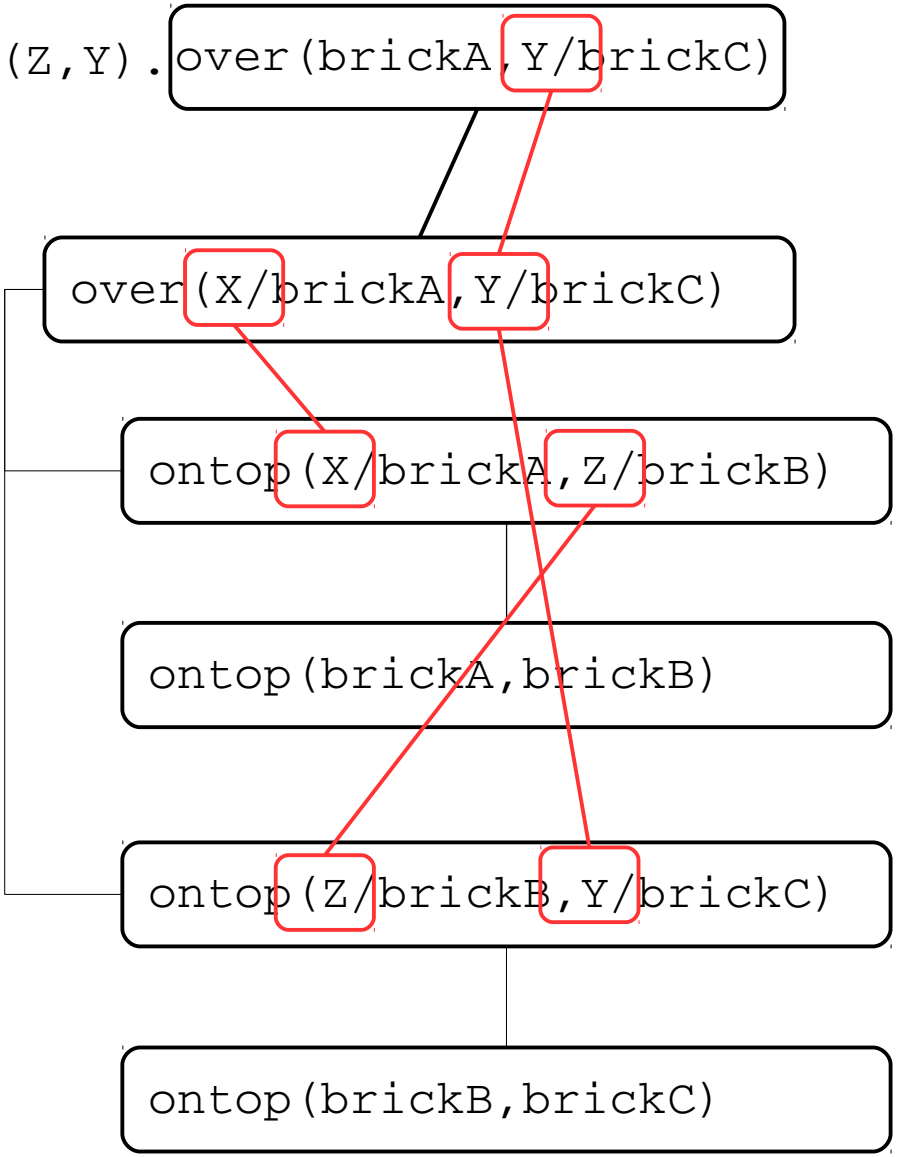
Inference in PROLOG

```
over(X,Y)  :-  ontop(X,Y).
over(X,Y)  :-  ontop(X,Z),ontop(Z,Y).

ontop(brickA,brickB).
ontop(brickB,brickC).

?over(brickA,Y).
```



Inference in PROLOG

```
over(X,Y)  :-  ontop(X,Y).
over(X,Y)  :-  ontop(X,Z),ontop(Z,Y).

ontop(brickA,brickB).
ontop(brickB,brickC).

?over(brickA,Y).
```



More on unification

- Unification – nonconflicting substitution of variables
- Substitution – a set of values than a variable can take
- Most General Unification – largest set of values that allows for a nonconflicting substitution

More on unification

- Unification rules:
  - term and term2 constants ↔ they are the same atom, or the same number

```
brickA ↔ brickA

5 ↔ 5
```

More on unification

- Unification rules:
  - term1 a variable, term2 a constant ↔ term1 is instantiated to term2

```
X ↔ brickA

X|brickA
```

More on unification

- Unification rules:
  - term1 and term2 variables ↔ share values

```
X ↔ Y

X|brickA ↔ Y|brickA
X|brickB ↔ Y|brickB
```

More on unification

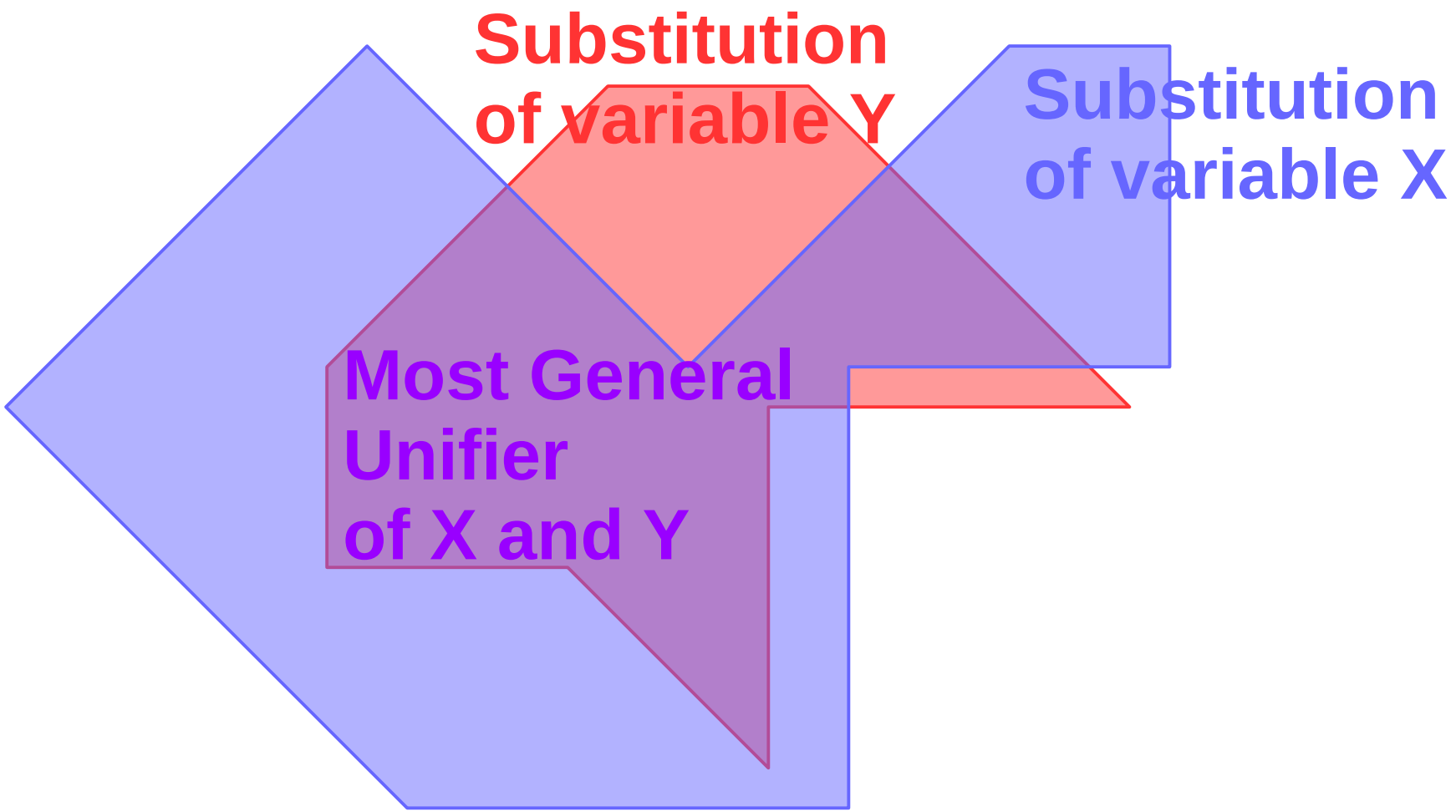
- Unification rules:
  - term1 and term2 are complex ↔ unify when:
    - same functor and arity
    - all their corresponding arguments unify
    - variable instantiations are compatible

```
ov(X,brickC) ↔ ov(brickA,Y)

X|brickA, Y|brickC
```

More on unification

- Unification rules:
  - term and term2 constants ↔ they are the same atom, or the same number
  - term1 a variable, term2 any type ↔ term1 is instantiated to term2
  - term1 and term2 variables ↔ share values
  - term1 and term2 are complex ↔ unify when:
    - same functor and arity
    - all their corresponding arguments unify
    - variable instantiations are compatible

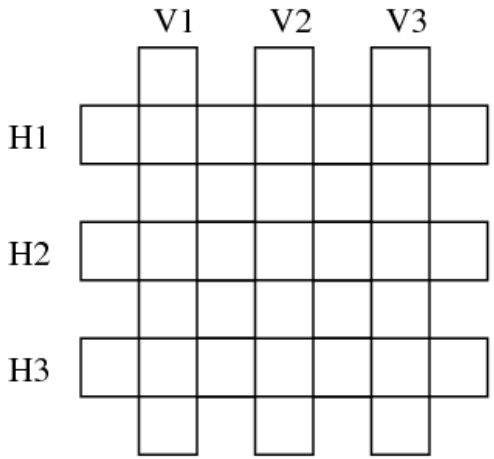
More on unification



Example predicates in PROLOG

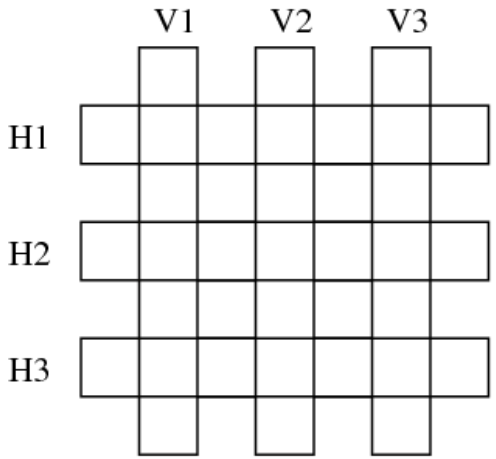
```
vertical(point(X,Y),point(X,Z)).
horizontal(point(X,Y),point(Z,Y)).


?- vertical(point(1,1),point(1,2)).
true.

?- vertical(point(1,1),point(1,2)).
true.

?- vertical(point(1,1),point(X,2)).
X = 1.

?- vertical(point(1,1),point(X,Y)).
X = 1.
```

Example predicates in PROLOG



A proper crossword for the words
astante
astoria
baratto
cobalto
pistola
statale

Example predicates in PROLOG




```
word(astante, a,s,t,a,n,t,e).
word(astoria, a,s,t,o,r,i,a).
word(baratto, b,a,r,a,t,t,o).
word(cobalto, c,o,b,a,l,t,o).
word(pistola, p,i,s,t,o,l,a).
word(statale, s,t,a,t,a,l,e).

letter(L,2,W):-word(W,_,L,_,_,_,_,_).
letter(L,4,W):-word(W,_,_,_,L,_,_,_).
letter(L,6,W):-word(W,_,_,_,_,_,L,_).

crossed(R,C,W1,W2):-
   letter(L,R,W1),letter(L,C,W2).

crossword(H1,H2,H3,V1,V2,V3):-
   crossed(2,2,H1,V1),crossed(4,2,H1,V2),
   crossed(6,2,H1,V3),crossed(2,4,H2,V1),
   crossed(4,4,H2,V2),crossed(6,4,H2,V3),
   crossed(2,6,H3,V1),crossed(4,6,H3,V2),
   crossed(6,6,H3,V3).
```

Example predicates in PROLOG

a	b		s
a s t	a	n	t e
t	r		a
b a r	a	t	t o
n	t		a
s t a	t	a	l e
e	o		e

```
?- crossword(H1,H2,H3,V1,V2,V3).
H1 = V1, V1 = astante,
H2 = V2, V2 = baratto,
H3 = V3, V3 = statale ;
H1 = astante,
H2 = cobalto,
H3 = pistola,
V1 = astoria,
V2 = baratto,
V3 = statale ;
H1 = astoria,
H2 = baratto,
H3 = statale,
V1 = astante,
V2 = cobalto,
V3 = pistola ;
H1 = V1, V1 = astoria,
H2 = V2, V2 = cobalto,
H3 = V3, V3 = pistola ;
H1 = H2, H2 = V1, V1 = V2, V2 = baratto,
H3 = V3, V3 = statale ;
H1 = V1, V1 = cobalto,
H2 = V2, V2 = baratto,
H3 = V3, V3 = statale
```

Example predicates in PROLOG

a	b		s
a s t	a	n	t e
t	r		a
c o b	a	l	t o
r	t		a
p i s	t	o	l a
a	o		e

```
?- crossword(H1,H2,H3,V1,V2,V3).
H1 = V1, V1 = astante,
H2 = V2, V2 = baratto,
H3 = V3, V3 = statale ;
H1 = astante,
H2 = cobalto,
H3 = pistola,
V1 = astoria,
V2 = baratto,
V3 = statale ;
H1 = astoria,
H2 = baratto,
H3 = statale,
V1 = astante,
V2 = cobalto,
V3 = pistola ;
H1 = V1, V1 = astoria,
H2 = V2, V2 = cobalto,
H3 = V3, V3 = pistola ;
H1 = H2, H2 = V1, V1 = V2, V2 = baratto,
H3 = V3, V3 = statale ;
H1 = V1, V1 = cobalto,
H2 = V2, V2 = baratto,
H3 = V3, V3 = statale
```

Example predicates in PROLOG

Representation of a list

```
[A,B,C]
[Head|Tail]
[A,B|L]
```

```
ad([],L,L).
ad(L,[],L).
ad([H1|T],[H2|L],[H1|TL]):-
    ad(T,[H2|L],TL).
```

Example predicates in PROLOG

```
ad([],L,L).
ad(L,[],L).
ad([H1|T],[H2|L],[H1|TL]):-
    ad(T,[H2|L],TL).
```

```
?- ad([1],[2,3],[1,2,3]).
true

?- ad([1],[2,3],C).
C = [1, 2, 3]

?- ad([1],L,[1,2,3]).
L = [2, 3]

?- ad([1],[2,3],[1,X,3]).
X = 2
```

```
?- ad(A,B,[1,2,3]).
A = [],
B = [1, 2, 3] ;
A = [1, 2, 3],
B = [] ;
A = [1],
B = [2, 3] ;
A = [1, 2],
B = [3] ;
```

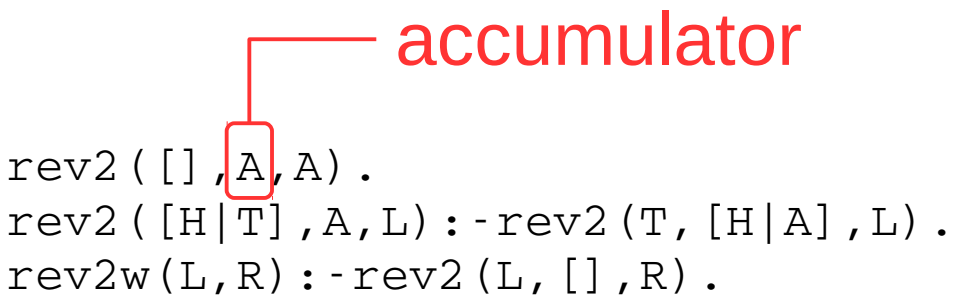
Example predicates in PROLOG

```
ad([],L,L).
ad(L,[],L).
ad([H1|T],[H2|L],[H1|TL]):-
    ad(T,[H2|L],TL).

rev([],[]).
rev([H|T],L):-
    ad(RT,[H],L),rev(T,RT).

nl(1,[H|T],H).
nl(N,[H|T],X):-
    nl(NN,T,X),N is NN+1.
```

Example predicates in PROLOG

```
ad([],L,L).
ad(L,[],L).
ad([H1|T],[H2|L],[H1|TL]):-
    ad(T,[H2|L],TL).

rev([],[]).
rev([H|T],L):-
    ad(RT,[H],L),rev(T,RT).

nl(1,[H|T],H).
nl(N,[H|T],X):-
    nl(NN,T,X),N is NN+1.
```

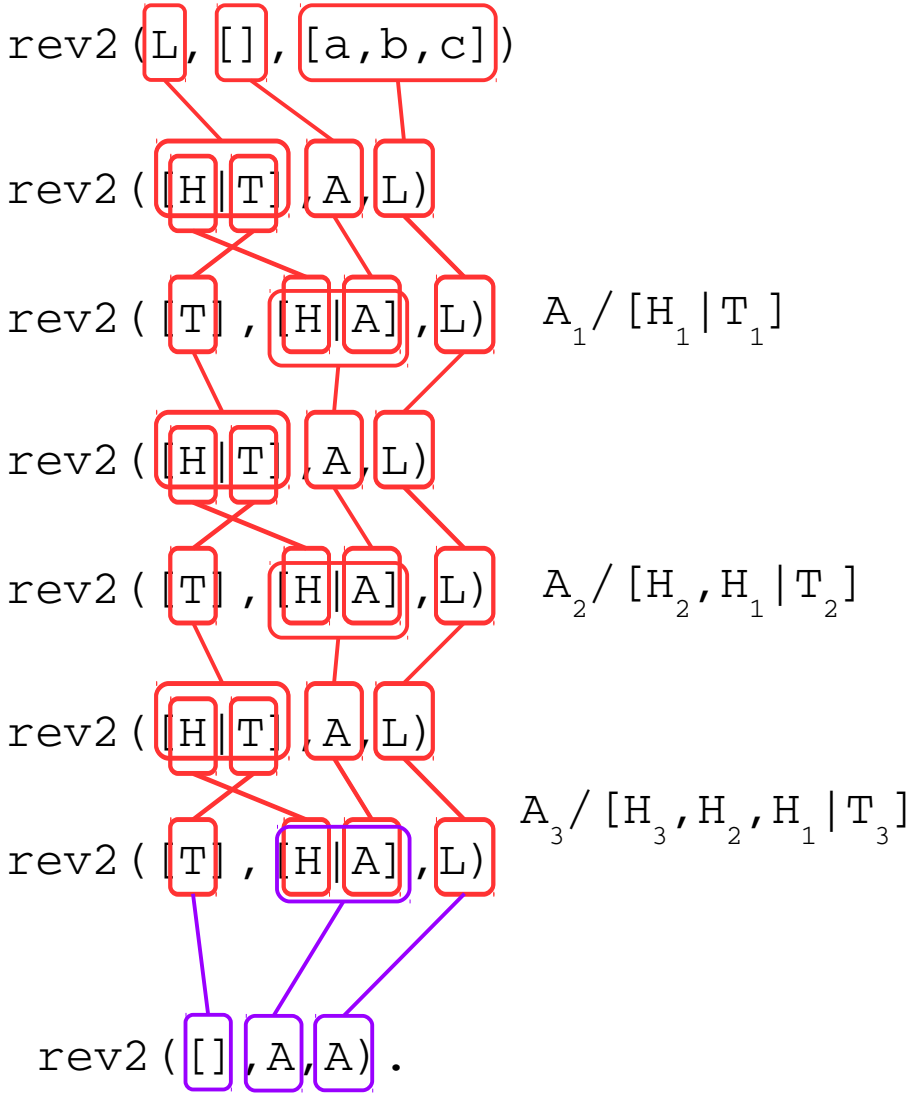
Example predicates in PROLOG accumulator technique




```
rev2([],A,A).
rev2([H|T],A,L):-rev2(T,[H|A],L).
rev2w(L,R):-rev2(L,[],R).
```

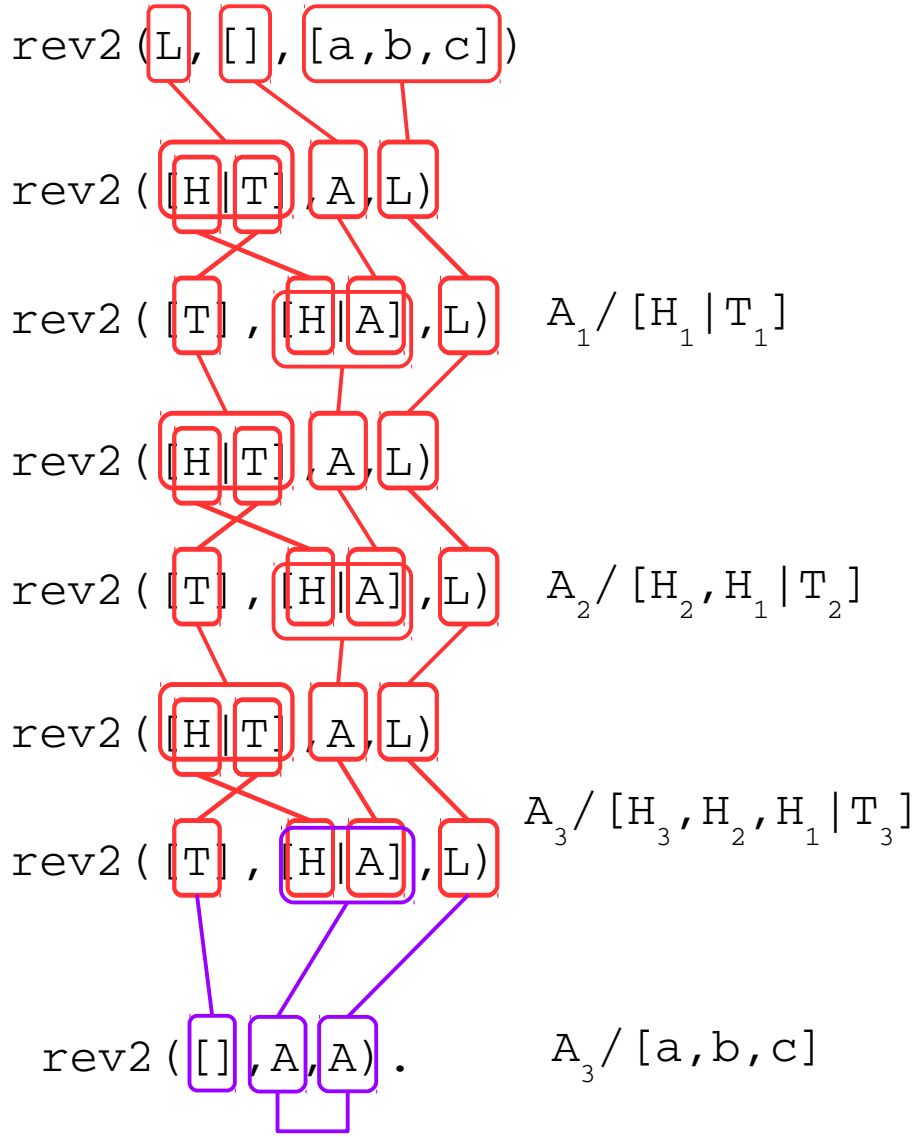
Example predicates in PROLOG accumulator technique

```
rev2([],A,A).
rev2([H|T],A,L):-
    rev2(T,[H|A],L).


?-rev2(L,[],[a,b,c])
```

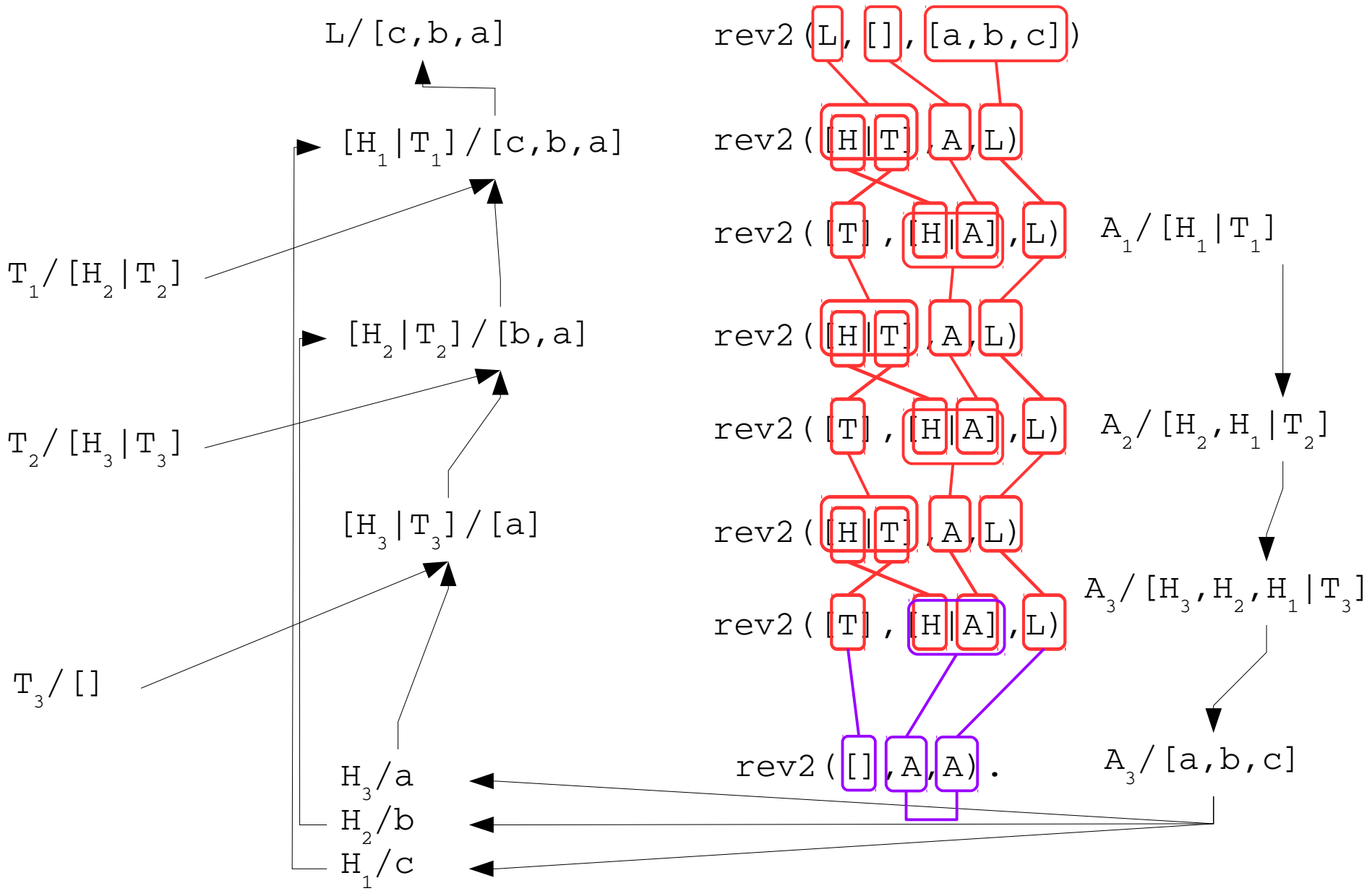


Example predicates in PROLOG accumulator technique

```
rev2([],A,A).
rev2([H|T],A,L):-
    rev2(T,[H|A],L).

?-rev2(L,[],[a,b,c])
```

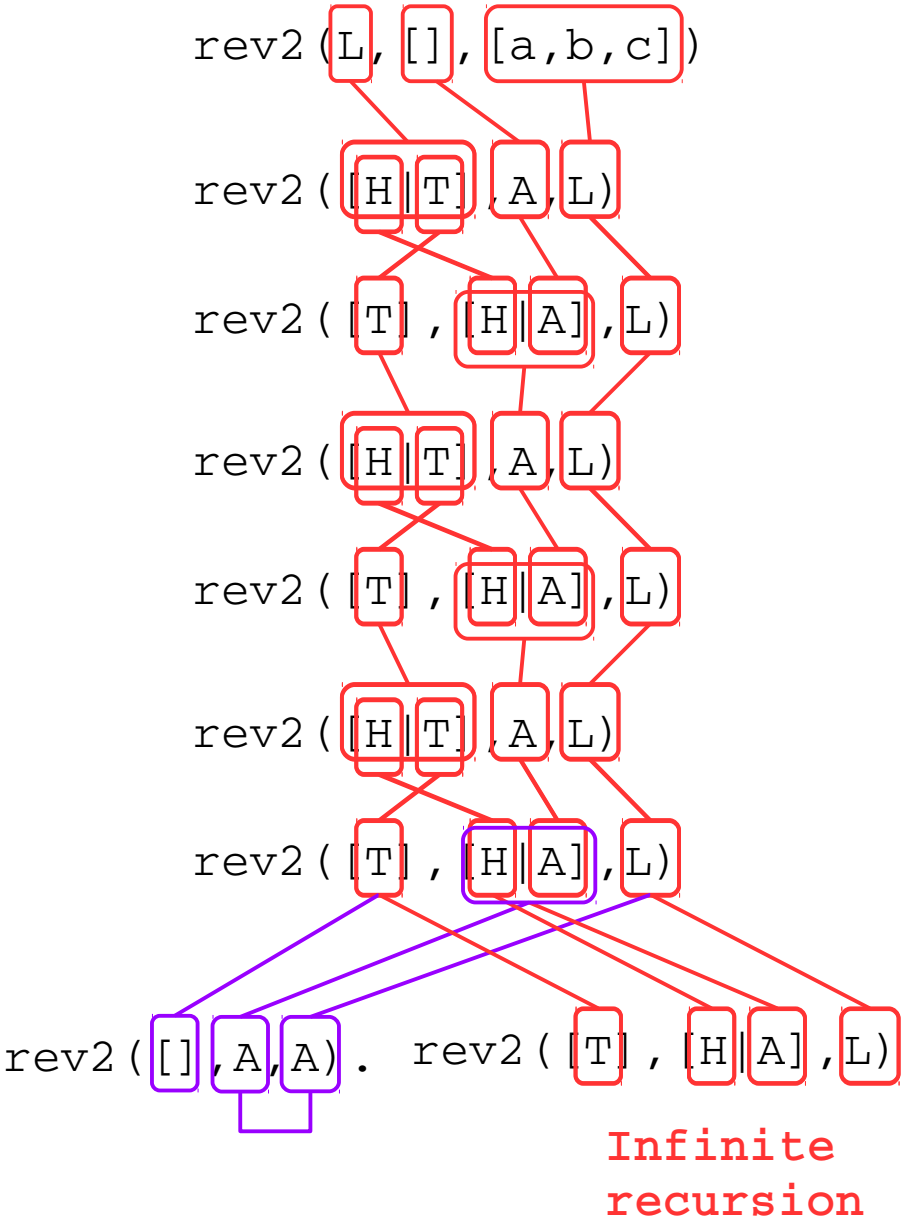


Example predicates in PROLOG accumulator technique



Example predicates in PROLOG accumulator technique

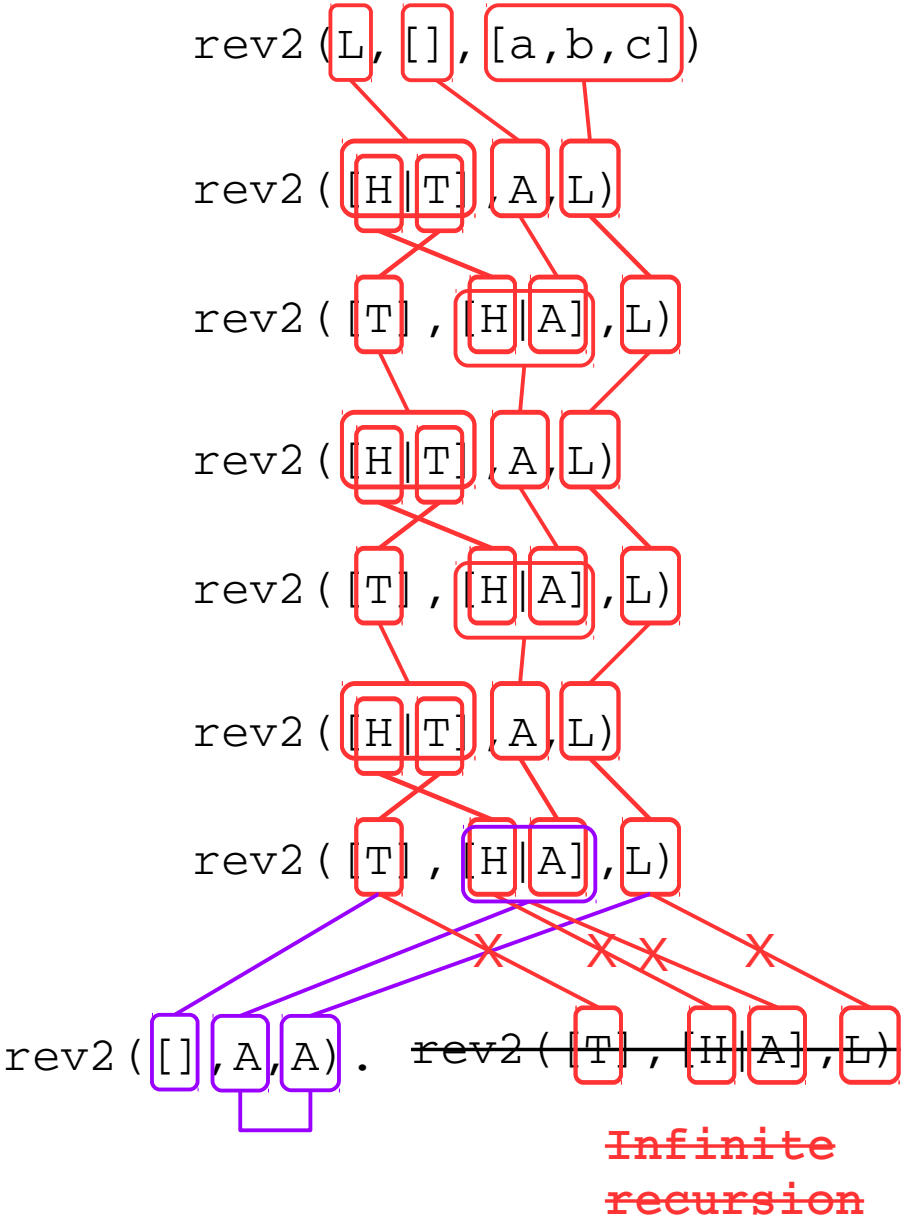
```
rev2([],A,A).
rev2([H|T],A,L):-
    rev2(T,[H|A],L).


?-rev2(L,[],[a,b,c])
```



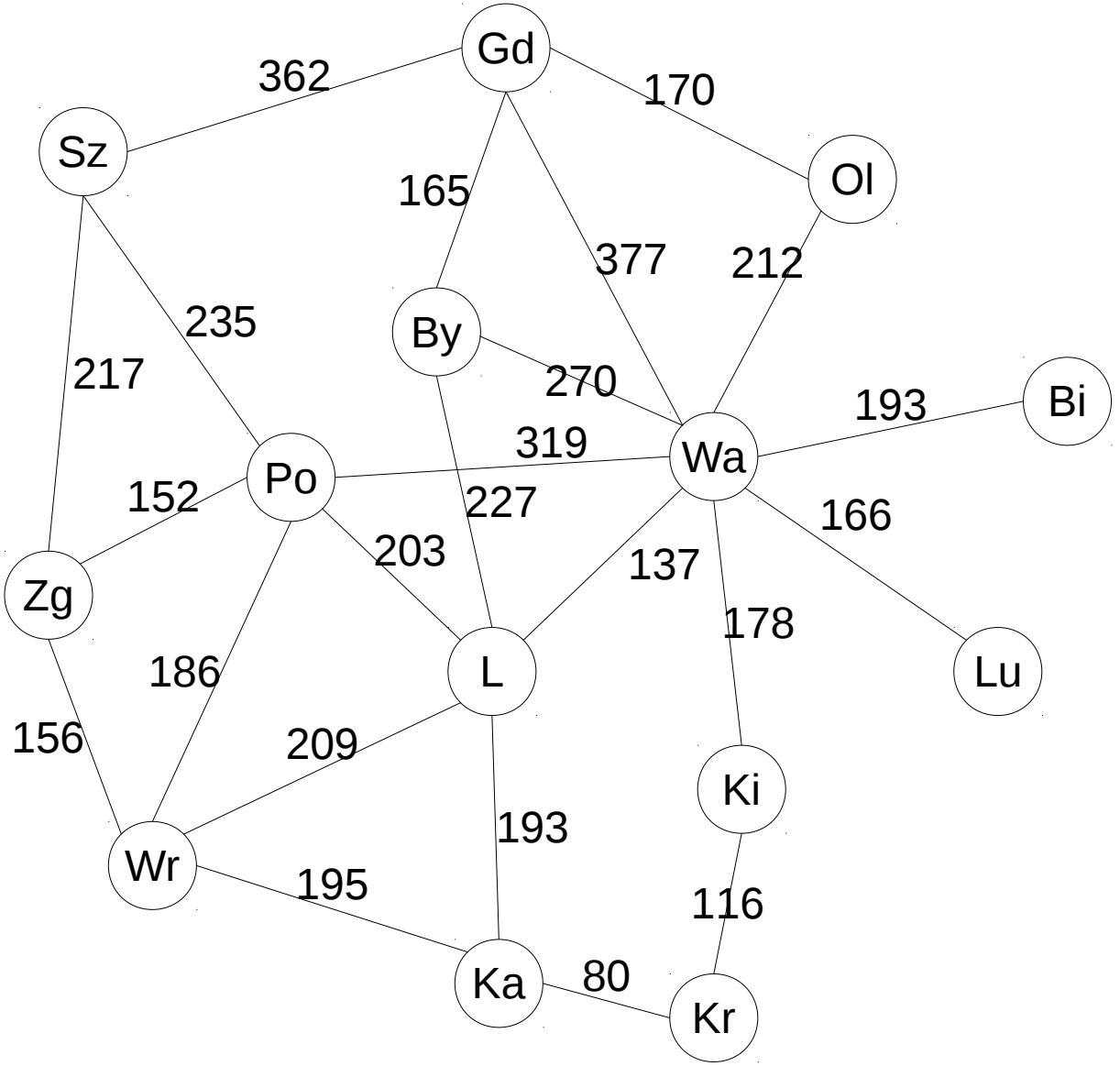
Example predicates in PROLOG cut

```
rev2([],A,A):-!.
rev2([H|T],A,L):-
    rev2(T,[H|A],L).


?-rev2(L,[],[a,b,c])
```

cut



Example predicates in PROLOG representing a graph




```
direct(mGd,mOl,170).
direct(mGd,mSz,362).
direct(mGd,mBy,165).
direct(mGd,mWa,270).
direct(mSz,mPo,235).
direct(mSz,mZg,217).
direct(mZg,mWr,156).
direct(mPo,mWr,186).
direct(mPo,mZg,152).
direct(mPo,mL,203).
direct(mBy,mWa,270).
direct(mBy,mL,227).
direct(mWa,mPo,319).
direct(mWa,mL,137).
direct(mWa,mKi,178).
direct(mWa,mOl,212).
direct(mWa,mBi,193).
direct(mWa,mLu,166).
direct(mL,mWr,209).
direct(mL,mKa,193).
direct(mKi,mKr,116).
direct(mKa,mKr,80).
direct(mKa,mWr,195).
```

Example predicates in PROLOG finding a least weighted path

```
connection(X,Y,W):-direct(X,Y,W).
connection(X,Y,W):-direct(Y,X,W).

notIn(A,[]).
notIn(A,[H|T]):-A\==H,notIn(A,T).

path(From,To,Visited,VL,[To|Visited],RL,Lim):-
   connection(From,To,W),!,
   RL is VL+W, RL<Lim.

path(From,To,Visited,VL,Result,RL,Lim):-
   connection(From,Next,W),
   notIn(Next,Visited), VL+W<Lim,
   path(Next,To,[Next|Visited],VL+W,Result,RL,Lim).

shortest(From,To,Path,Weight,Lim,ResPath,ResWeight):-
   path(From,To,[From],0,R,W,Lim), !,
   shortest(From,To,R,W,W,ResPath,ResWeight).
shortest(From,To,Path,Weight,Lim,Path,Weight):-Weight>0.

shortestPath(From,To,Limit,Path,Weight):-
   shortest(From,To,[From],0,Limit,Path,Weight),.
```