First-Order Resolution with Equality

Motivating example: Suppose that we are given the following clauses:

$\varphi_1 := P(a)$

$\varphi_2 := a=b$

It should be possible to derive the clause

$\varphi := P(b)$

via a simple substitution of b for a.

However, basic resolution provides no way to achieve this.

While the equality relation = is a simple binary relation:

- (a=b) might be written =(a,b) in a more formal way;

it does not enjoy any special status unless the proof system is augmented to provide this.

A simple example will illustrate this limitation.

Example:

Siblings have the same mother:
$(\forall x)(\forall y)(\forall u)(\forall w)$
$(\text{Sibling}(x,y) \wedge \text{Mother}(u,x) \wedge \text{Mother}(w,y)) \rightarrow (u=w)$

Normalized and in clausal form:
$\neg\text{Sibling}(x,y) \vee$
$\neg\text{Mother}(u,x) \vee \neg\text{Mother}(w,y) \vee (u=w)$

Everyone has a mother:
$(\forall x)(\exists y)\text{Mother}(y,x)$

In clausal form:
$\text{Mother}(m(x),x)$

Database of given facts:
Sibling(Tweety,Spike)
Mother(Olive,Tweety)

Goal:
Mother(Olive,Spike)

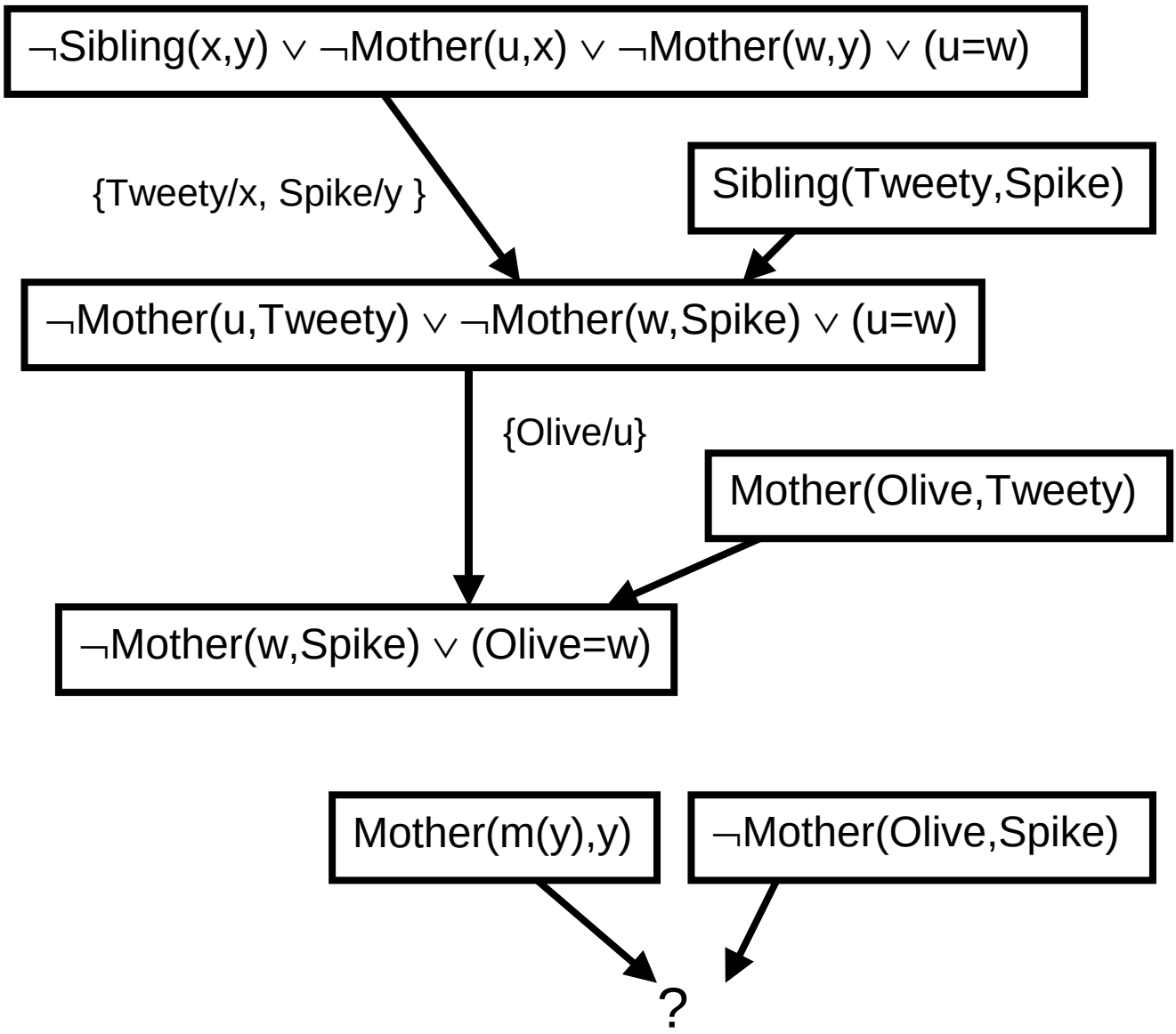
Negated goal:
$\neg$Mother(Olive,Spike)

Let us try to construct a proof of the goal from the hypotheses.

Here is a first attempt at proving that Olive is the mother of Spike. Note that it fails, since we cannot unify c with Olive. The information that the two are the same is embodied in

$$\neg\text{Mother}(w,\text{Spike}) \vee (\text{Olive}=w)$$

but the proof process does not provide a means to extract it.



The idea behind paramodulation:

Paramodulation is a resolution-based technique which allows full use of the information represented by statements of equality.

A Simple example:

Let $\varphi_1 := P(a) \vee Q(c) \vee (b=a)$

$\varphi_2 := R(a)$

$\varphi := P(a) \vee Q(c) \vee R(b)$

Then $\{\varphi_1, \varphi_2\} \models \varphi$.

Indeed, let J be any valuation for which $J \models \{\varphi_1, \varphi_2\}$. There are two cases to consider, reflecting the two ways in which φ_1 may be true.

- If (b=a) is false, then since $J \models \{\varphi_1\}$,
  $J \models P(a) \vee Q(c)$, and so $J \models \varphi$.

- If (b=a) is true, then $R(b) \leftrightarrow R(a)$, so since $J \models \{\varphi_2\}$, $J \models R(b)$.

  Hence $J \models \varphi$ in either case.

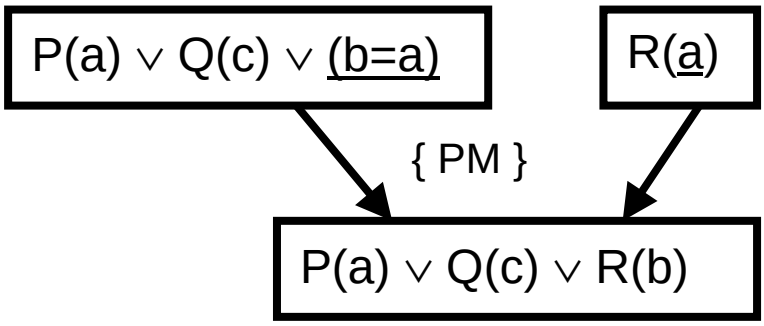
Thus, in paramodulation, instead of resolving on literals, we “resolve” (paramodulate) on an equality.

Note that it is *not* correct to substitute a for b in P(b). We cannot conclude that

$$\{\varphi_1, \varphi_2\} \models P(b) \vee Q(c) \vee R(b)$$

If (b=a) is false, this may not hold.

This is how it may be depicted graphically.



The equality relation which is used in the substitution, as well as the term(s) which are substituted for, are underlined.

A more complex example:

Unifiers may be applied to create matching components.

Let $\varphi_1 := P(g(f(x))) \vee Q(x)$
$\varphi_2 := (f(g(b))=a) \vee R(g(c))$
$\varphi := P(g(a)) \vee Q(g(b)) \vee R(g(c))$

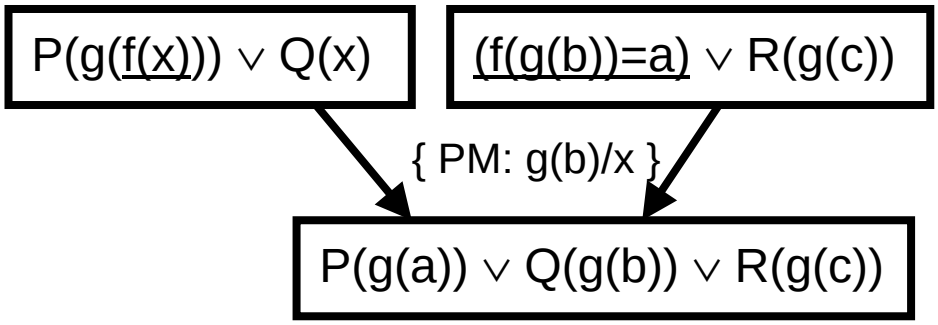
Then $\{\varphi_1, \varphi_2\} \models \varphi$.

To see this, apply the unifier {g(b)/x} to φ_1 and φ_2, yielding:

$P(g(f(g(b))) \vee Q(g(b))$
$(f(g(b))=a) \vee R(g(c))$

Reasoning as above, we may now deduce that

$\{\varphi_1, \varphi_2\} \models \varphi$.



The general technique of paramodulation:

Definition: Let

$$\varphi_1 := \ell[t] \vee \psi_1$$
$$\varphi_2 := (r=s) \vee \psi_2$$

where:

$\ell[t]$ is a literal containing term t;
r and s are any terms.

Suppose that t and r have an mgu σ.

The clause

$$\ell\sigma[s\sigma] \vee \psi_1\sigma \vee \psi_2\sigma$$

is called a *binary paramodulant* of φ_1 and φ_2.

Here $\ell\sigma[s\sigma]$ is obtained by

- Applying σ to all of ℓ;
- Replacing one occurrence of $t\sigma$ by $s\sigma$.

In the previous example:

r := f(g(b))	ψ_1 := Q(x)
s := a	ψ_2 := R(g(c))
t := f(x)	σ := {g(b)/x}
$\ell[t]$:= P(g(f(x)))	$\ell\sigma$:= P(g(f(g(b))))
	$\ell\sigma[s\sigma]$:= P(g(a))

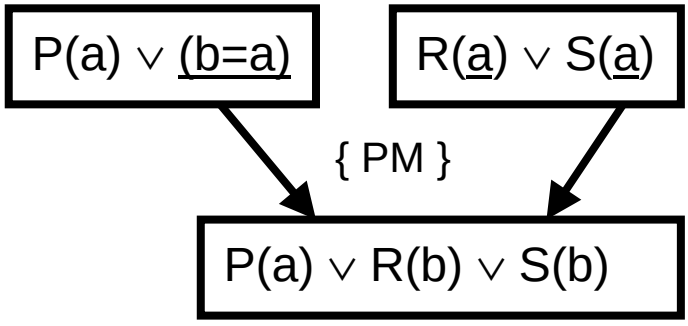
Fact: Binary paramodulation, together with resolution, forms a sound and complete refutation mechanism for clauses with equality. □

Note: Some technical details are omitted.

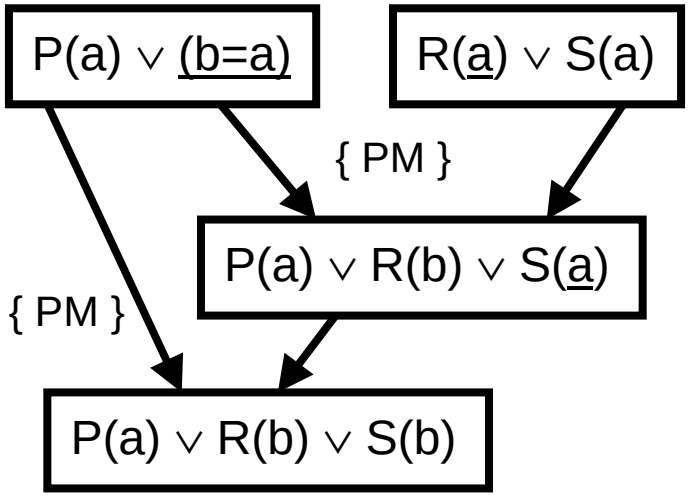
It is not necessary to limit the substitution to just one literal.

Example: $\varphi_1 := P(a) \vee (b=a)$
$\varphi_2 := R(a) \vee S(a)$
$\varphi := P(a) \vee R(b) \vee S(b)$

Then $\{\varphi_1, \varphi_2\} \models \varphi$ may be established using paramodulation. The constant b may be substituted for a in φ_2 in one step.

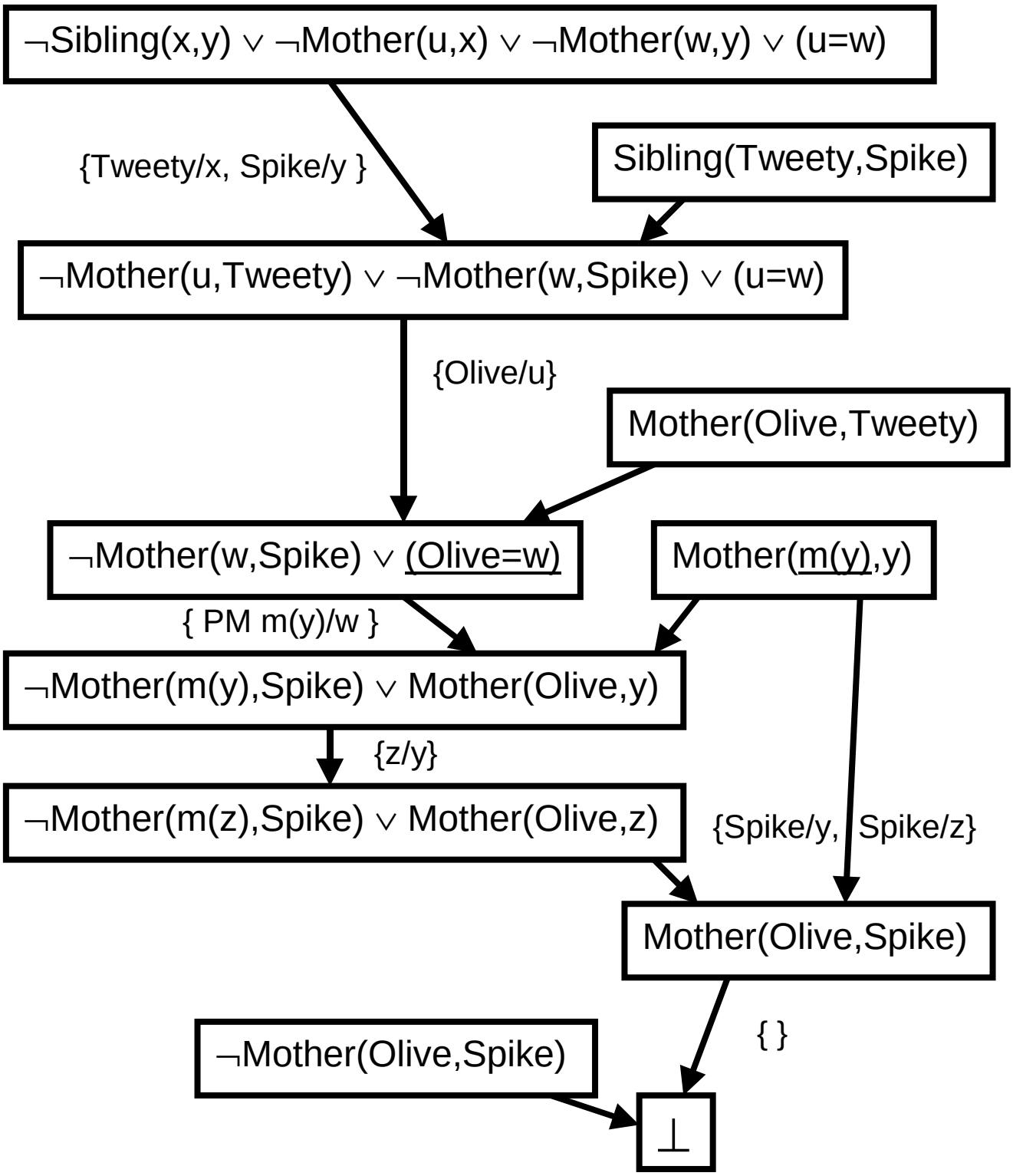


It is easy to justify this sort of operation, which will be called *full paramodulation*:



Return to the running example:

Now we can prove that Olive is the mother of Spike.



Answer extraction and paramodulation:

To see how the answer extraction process behaves in the running example, let us replace the goal

Mother(Olive,Spike)

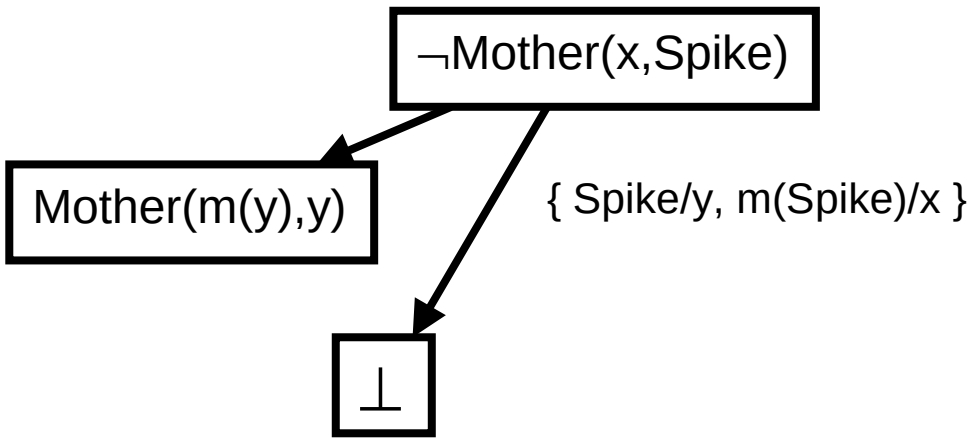
with

$(\exists x)$Mother(x,Spike)

The negate goal becomes:

$\neg$Mother(x,Spike)

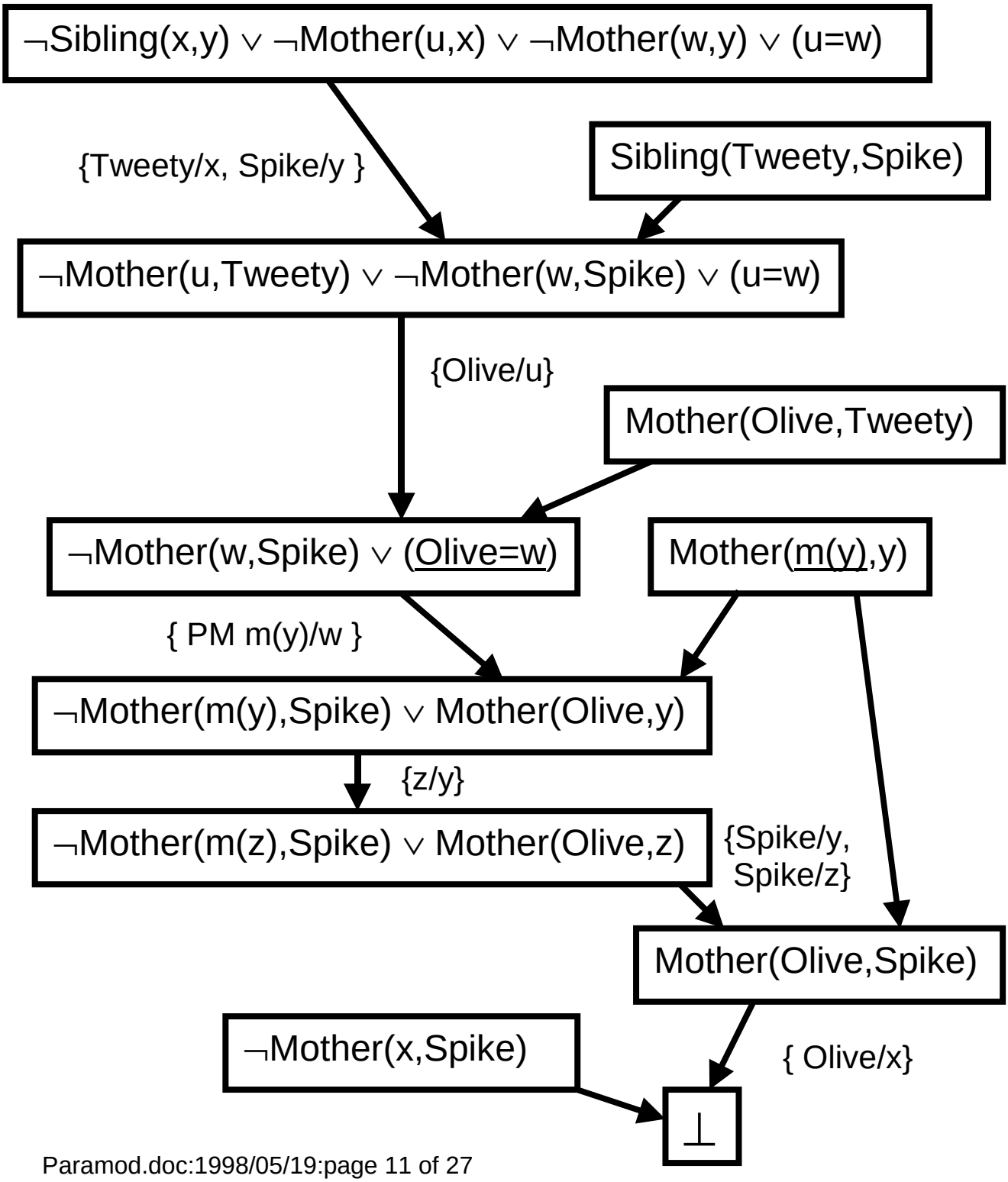
Satisfaction of the goal is now trivial.



However, it provides little information. By generating a longer and more informative proof, we may determine the identity of the mother of Spike.

Here is a more elaborate proof, providing the answer that Olive is the mother of Spike. It is the same as the previous proof of Mother(Olive,Spike), except for the last step.

Thus, it may be necessary to direct the search to extract informative answers.

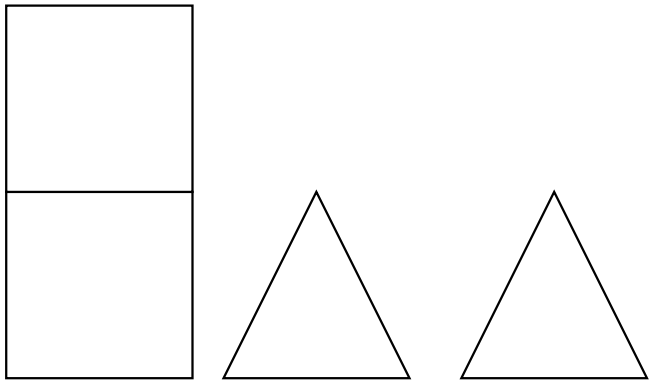


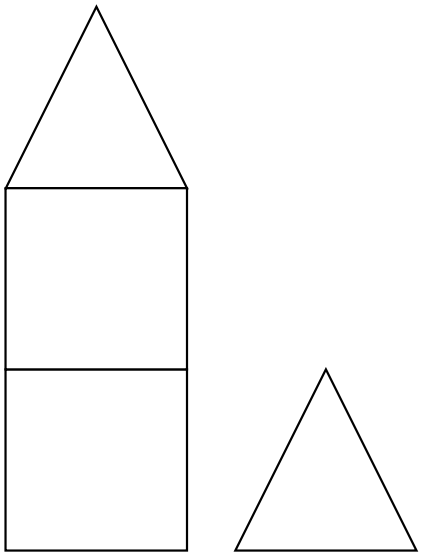
Solution of a blocks-world problem:

In the slides on propositional resolution, a proof was given that if B2 is atop B1, then either P1 or else P2 must be on the table.

Here we will give a proof of a more general result:

The goal is to establish that if one cube is stack atop the other, then at least one pyramid must be on the table. The possibilities for such a state are depicted below. There are two arrangements, with a total of six possible states.





Thus, this problem is somewhat more general than the one solved earlier. Here is an expression of the goal, in first-order logic.

$$((\exists x)(\exists y)(\text{Is\_cube}(x) \wedge \text{Is\_cube}(y) \wedge \text{On}(x,y)) \rightarrow (\exists z)(\text{Is\_pyramid}(z) \wedge \text{On\_table}(z)))$$

The first step is to express this constraint as a set of normalized clauses.

As a resolvent in a resolution proof, the goal is negated.

$$\neg((\exists x)(\exists y)(\text{Is\_cube}(x) \wedge \text{Is\_cube}(y) \wedge \text{On}(x,y))) \rightarrow (\exists z)(\text{Is\_pyramid}(z) \wedge \text{On\_table}(z)))$$

The first step is to remove the implication:

$$\neg(\neg(\exists x)(\exists y)((\text{Is\_cube}(x) \wedge \text{Is\_cube}(y) \wedge \text{On}(x,y))) \vee (\exists z)(\text{Is\_pyramid}(z) \wedge \text{On\_table}(z)))$$

Move the negations in to the atoms:

$$(\exists x)(\exists y)(\text{Is\_cube}(x) \wedge \text{Is\_cube}(y) \wedge \text{On}(x,y))) \wedge (\forall z)(\neg\text{Is\_pyramid}(z) \vee \neg\text{On\_table}(z))$$

Bring the quantifiers out to the front in optimal order:

$$(\exists x)(\exists y)(\forall z)((\text{Is\_cube}(x) \wedge \text{Is\_cube}(y) \wedge \text{On}(x,y))) \wedge (\neg\text{Is\_pyramid}(z) \vee \neg\text{On\_table}(z)))$$

Skolemize and drop universal quantifiers:

$$((\text{Is\_cube}(c_x) \wedge \text{Is\_cube}(c_y) \wedge \text{On}(c_x,c_y))) \wedge (\neg\text{Is\_pyramid}(z) \vee \neg\text{On\_table}(z)))$$

Break into clauses:

(1) $\text{Is\_cube}(c_x)$
(2) $\text{Is\_cube}(c_y)$
(3) $\text{On}(c_x,c_y)$
(4) $\neg\text{Is\_pyramid}(z) \vee \neg\text{On\_table}(z)$

The next step is to write down the axioms of the blocks world in clausal form. For exercise, we will do all of them, but will show only the final results. Also, we will defer the renaming of variables until later. First, the original formula will be given, and then the clausal form, with a number in front.

- Everything is either a block or a pyramid:
  $(\forall x)(\text{Is\_cube}(x) \vee \text{Is\_pyramid}(x))$

(5) $\text{Is\_cube}(x) \vee \text{Is\_pyramid}(x)$

- Nothing is both a block and a pyramid:
  $(\forall x)(\neg(\text{Is\_cube}(x) \wedge \text{Is\_pyramid}(x)))$

(6) $\neg\text{Is\_cube}(x) \vee \neg\text{Is\_pyramid}(x)$

- Domain closure; the only objects are those which are identified explicitly:

$(\forall x)(\ Is\_cube(x) \leftrightarrow (\ x=B1 \vee x=B2))$
$(\forall x)(\ Is\_pyramid(x) \leftrightarrow (\ x=P1 \vee x=P2))$

(7) $\neg Is\_cube(x) \vee (x=B1) \vee (x=B2)$
(8) $Is\_cube(B1)$
(9) $Is\_cube(B2)$
(10) $\neg Is\_pyramid(x) \vee (x=P1) \vee (x=P2)$
(11) $Is\_pyramid(P1)$
(12) $Is\_pyramid(P2)$

- Objects are distinct:

$(B1 \neq B2) \wedge (P1 \neq P2) \wedge (B1 \neq P1) \wedge (B1 \neq P2)$
$\wedge (B2 \neq P1) \wedge (B2 \neq P2)$

(13) $\neg(B1 = B2)$
(14) $\neg(P1 = P2)$
(15) $\neg(B1 = P1)$
(16) $\neg(B1 = P2)$
(17) $\neg(B2 = P1)$
(18) $\neg(B2 = P2)$

- No object can rest atop a pyramid.

$(\forall x)(\forall y)(\neg(Is\_pyramid(x) \wedge On(y,x)))$

(19) $\neg Is\_pyramid(x) \vee \neg On(y,x)$

- No object can rest atop another object and lie on the table at the same time.
  $(\forall x)(\forall y)(\neg(On\_table(x) \wedge On(x,y)))$

(20) $\neg On\_table(x) \vee \neg On(x,y)$

- Every object is either on the table or else atop another object.
  $(\forall x)(\exists y)(On\_table(x) \vee On(x,y))$

(21) $On\_table(x) \vee On(x,base(x))$

Here base is a Skolem function. We have given it a "meaningful" name.

- No object can rest atop itself.
  $(\forall x)(\neg On(x,x))$

To facilitate the proof, this will be written in a different but equivalent way, as the compound negation:
  $(\forall x)(\forall y)((On(x,y) \wedge (x=y)) \rightarrow \perp)$

(22) $\neg On(x,y) \vee \neg(x=y)$

- An object can rest atop at most one object.
  $(\forall x)(\forall y)(\forall z)\ ((On(x,y) \wedge On(x,z)) \rightarrow y=z))$

(23) $\neg On(x,y) \vee \neg On(x,z) \vee (y=z)$

- At most one object can rest atop another object.

$(\forall x)(\forall y)(\forall z)\ ((On(y,x) \wedge On(z,x)) \rightarrow y=z))$

(24) $\neg On(y,x) \vee \neg On(z,x) \vee (y=z)$

The proof will be written in linear fashion, with citations to clause numbers.

1. Resolve
   (22) $\neg On(x,y) \vee \neg(x=y)$
with
   (3) $On(c_x,c_y)$
using { c_x/x, c_y/y}
to obtain

(25) $\neg(c_x=c_y)$

2. Resolve
   (7) $\neg Is\_cube(x) \vee (x=B1) \vee (x=B2)$
with
   (1) $Is\_cube(c_x)$
using { c_x/x }
to obtain

(26) $(c_x=B1) \vee (c_x=B2)$

3. Similarly, resolve

(7) $\neg$Is_cube(x) $\vee$ (x=B1) $\vee$ (x=B2)

with

(2) Is_cube(c_y)

using { c_y/x }

to obtain

(27) (c_y=B1) $\vee$ (c_y=B2)

4. Use paramodulation on

(26) (c_x=B1) $\vee$ <u>(c_x=B2)</u>

and

(25) $\neg$(<u>c_x</u>=c_y)

with {} and paramodulating objects underlined

to obtain

(28) (c_x=B1) $\vee$ $\neg$(B2=c_y)

5. Resolve (28) with (27) using {} to obtain

(29) (c_x=B1) $\vee$ (c_y=B1)

6. Using steps similar to those of (4), derive

(30) (c_x=B2) $\vee$ (c_y=B2)

7. Use generalized paramodulation on

(7) $\neg$Is_cube(x) $\vee$ <u>(x=B1)</u> $\vee$ (x=B2)

and

(29) (c_x=<u>B1</u>) $\vee$ (c_y=<u>B1</u>)

with {} to obtain

(31) $\neg$Is_cube(x) $\vee$ (x=B2) $\vee$ (c_x=x) $\vee$ (c_y=x)

8. Use generalized paramodulation on

(31) $\neg$Is_cube(x) $\vee$ <u>(x=B2)</u> $\vee$ (c_x=x) $\vee$ (c_y=x)

and

(30) (c_x=<u>B2</u>) $\vee$ (c_y=<u>B2</u>)

with {} to obtain

(32) $\neg$Is_cube(x) $\vee$ (c_x=x) $\vee$ (c_y=x)

9. Resolve

(4) $\neg$Is_pyramid(z) $\vee$ $\neg$On_table(z)

with

(11) Is_pyramid(P1)

with {P1/z} to obtain

(33) $\neg$On_table(P1)

10. Derive similarly

(34) $\neg$On_table(P2)

11. Resolve
(21) On_table(x) $\vee$ On(x,base(x))
with
(33) $\neg$On_table(P1)
using {P1/x} to obtain

(35) On(P1,base(P1))

12. Similarly, derive

(36) On(P2,base(P2))

12. Resolve
(19) $\neg$Is_pyramid(x) $\vee$ $\neg$On(y,x)
with (35) using {P1/y, base(P1)/x} to obtain

(37) $\neg$Is_pyramid(base(P1))

13. Resolve
(13) Is_cube(x) $\vee$ Is_pyramid(x)
with (37) using {base(P1)/x} to obtain

(38) Is_cube(base(P1))

14. Derive similarly

(39) Is_cube(base(P2))

15. Resolve

(32) $\neg$Is_cube(x) $\vee$ (c_x=x) $\vee$ (c_y=x)

with

(38) Is_cube(base(P1))

using {base(P1)/x} to obtain

(40) (c_x=base(P1)) $\vee$ (c_y=base(P1))

16. Derive similarly

(41) (c_x=base(P2)) $\vee$ (c_y=base(P2))

17. Paramodulate

(40) (c_x=base(P1)) $\vee$ (c_y=base(P1))

with

(35) On(P1,base(P1))

using {} to obtain

(42) On(P1,c_x) $\vee$ (c_y=base(P1))

18. Paramodulate

(42) On(P1,c_x) $\vee$ (c_y=base(P1))

with

(35) On(P1,base(P1))

using {} to obtain

(43) On(P1,c_x) $\vee$ On(P1,c_y)

19. Derive similarly

(44) $On(P2,c_x) \vee On(P2,c_y)$

20. Resolve
    (24) $\neg On(y,x) \vee \neg On(z,x)) \vee (y=z)$
with
    (3) $On(c_x,c_y)$
using { c_y/x, c_x/y } to obtain

(45) $\neg On(z, c_y) \vee (c_x=z)$

21. Resolve (45) with
    (43) $On(P1,c_x) \vee On(P1,c_y)$
using {P1/z} to obtain

(46) $(c_x=P1) \vee On(P1,c_x)$

22. Paramodulate
    (1) $Is\_cube(\underline{c_x})$
with
    (46) $\underline{(c_x=P1)} \vee On(P1,c_x)$
using {} to obtain

(47) $Is\_cube(P1) \vee On(P1,c_x)$

23. Resolve
(47) Is_cube(P1) $\vee$ On(P1,c_x)
with
(6) $\neg$Is_cube(x) $\vee$ $\neg$Is_pyramid(x)
using {P1/x} to obtain

(48) $\neg$Is_pyramid(P1) $\vee$ On(P1,c_x)

24. Resolve (48) with
(11) Is_pyramid(P1)
Using {} to obtain

(49) On(P1,c_x)

25. Derive similarly

(50) On(P2,c_x)

26. Resolve (50) with
(24) $\neg$On(y,x) $\vee$ $\neg$On(z,x)) $\vee$ (y=z)
using {P2/y, c_x/x} to obtain

(51) $\neg$On(z, c_x)) $\vee$ (P2=z)

27. Resolve (51) with (49) using {P1/z} to obtain

(51) (P2=P1)

28. Resolve

(51) (P2=P1)

with

(15) ¬(P1 = P2)

using {} to obtain

(52) ⊥

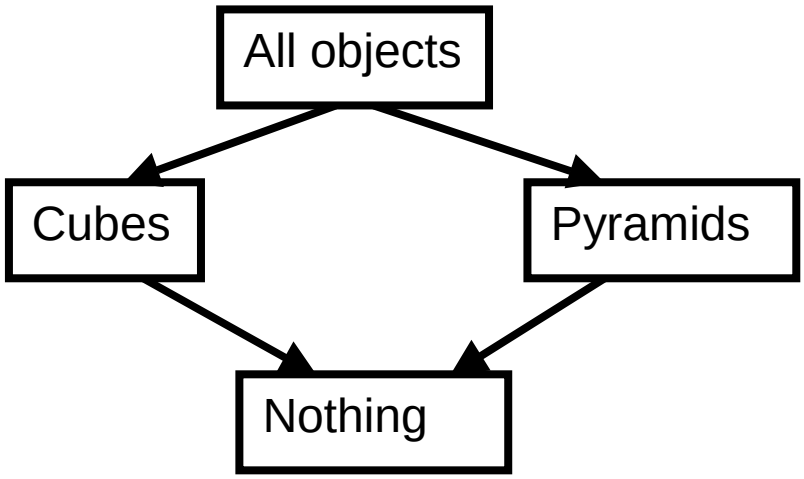
Remarks on complexity:

One cannot help but notice that much of the preceding proof is consumed by inferences about objects being cubes or pyramids.

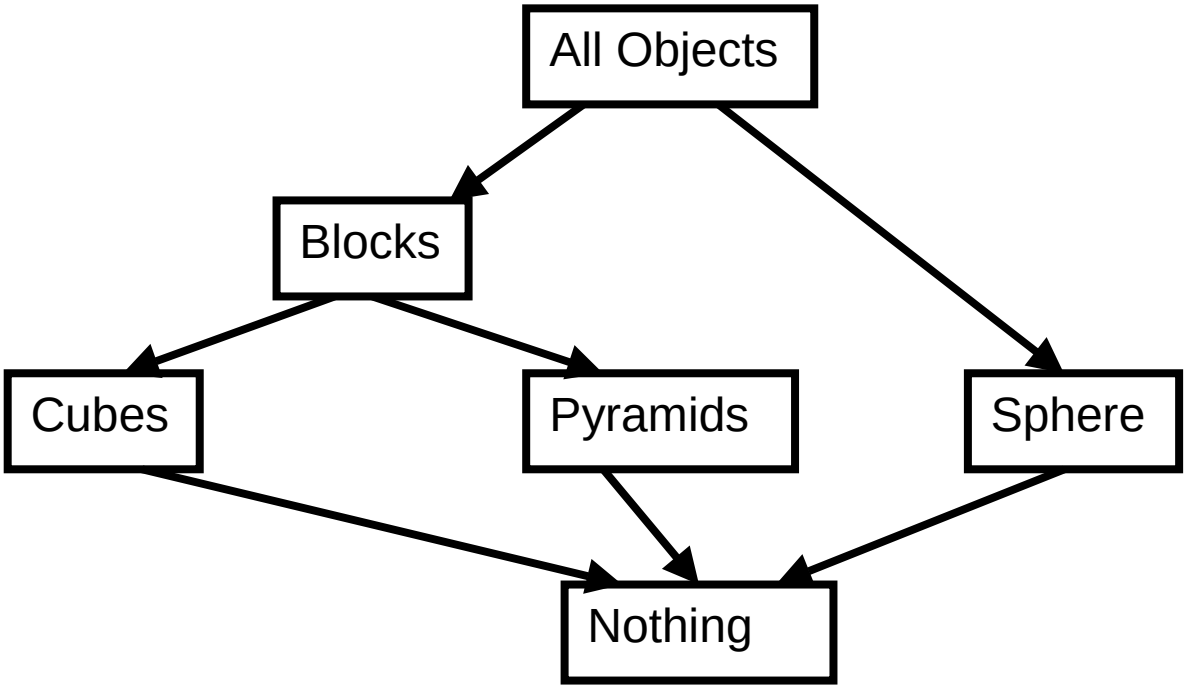
It is natural to ask whether the process could be improved by incorporating this information in some other fashion.

The answer is in the affirmative. By incorporating so-called *type hierarchies* into the proof process in an integral fashion, the efficiency of many inferences may be improved immensely.

Here is the type hierarchy for this simple example.



A slightly more complex example is the following:



A system with typing would relegate the axioms concerning such a hierarchy to a “higher level.”

This means that they would a different, built-in inference mechanism.

The axioms for the above schema would include:

$(\forall x)(\text{Is\_pyramid}(x) \rightarrow \text{Is\_block}(x))$
$(\forall x)(\text{Is\_cube}(x) \rightarrow \text{Is\_block}(x))$
$(\forall x)((\text{Is\_block}(x) \wedge \text{Is\_sphere}(x)) \rightarrow \perp)$
$(\forall x)((\text{Is\_cube}(x) \wedge \text{Is\_pyramid}(x)) \rightarrow \perp)$
$(\forall x)(\text{Is\_block}(x) \vee \text{Is\_sphere}(x))$

Further information:

Elementary presentations of paramodulation are not easy to find.

Probably the best overall reference is the classical text of Chang and Lee.

Chang, Chin-Liang, and Lee, Richard Char-Tung, *Symbolic Logic and Mechanical Theorem Proving,* Academic Press, 1973.

Most modern textbooks on artificial intelligence will give basic information on the use of type hierarchies in knowledge representation. Several special-purpose programming languages have built-in inheritance.

Aït-Kaci, H, and Nasr, R., "LOGIN: A logic programming language with built-in inheritance," *J. Logic Programming,* **3**(1986), pp. 185-215.

Dörre, J., and Dorna, M., "CUF – a formalism for linguistic knowledge representation," in Dörre, J., editor, *Computational Aspects of Constraint-Based Linguistic Descriptions, DYANA-2 Deliverable R.1.2.A,* pp 3-22, ESPRIT, 1993.