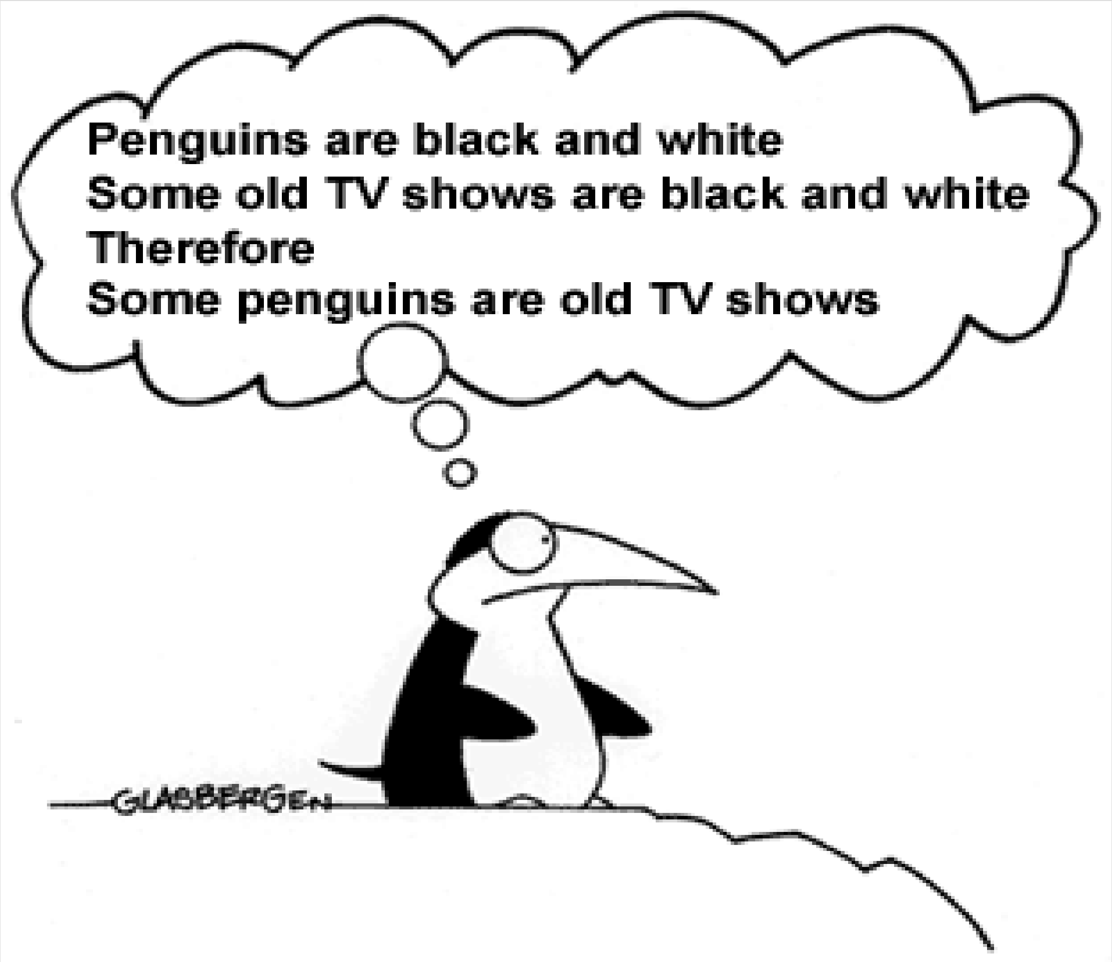
Using first order logic (Ch. 9)



Loglish

Consider the sentence:
"No one is sharing a seat"

Logic-English: "For all pairs of people if there is at least one chair they are both sitting on, then they must be the same person"

In logic:

$$\forall x_1, x_2 \exists y \; InSeat(x_1, y) \wedge InSeat(x_2, y) \Rightarrow x_1 = x_2$$

Loglish

Consider the sentence:
“No one is sharing a seat”

Logic-English: “For all pairs of people, on all the chairs both people are sitting in, the pair must be the same person”

In logic:

$$\forall x_1, x_2, y \; InSeat(x_1, y) \wedge InSeat(x_2, y) \Rightarrow x_1 = x_2$$

Loglish

Consider the sentence:
“No one is sharing a seat”

Logic-English: “All chairs with someone in it, everyone else cannot be in it”

In logic:

$$\forall y \exists x_1 InSeat(x1, y) \Rightarrow [\forall x_2\ x_1 \neq x_2 \Rightarrow \neg InSeat(x_2, y)]$$

or

$$\forall y \exists x_1 \forall x_2 InSeat(x1, y) \Rightarrow [x_1 \neq x_2 \Rightarrow \neg InSeat(x_2, y)]$$

Unification

A <u>unification</u> is a substitution for variables that creates a valid sentence by specifying a map between variables and objects

For example, consider:

$$\text{Objects} = \{Sue,\ Alex,\ Devin\}$$
$$\forall x, y\ Sibling(x, y) \Rightarrow Sibling(y, x)$$
$$Sibling(Sue, Devin)$$
$$\neg Sibling(Devin, Alex)$$

What variables can we unify/substitute?

Unification

$$\text{Objects} = \{Sue,\ Alex,\ Devin\}$$
$$\forall x, y\ Sibling(x, y) \Rightarrow Sibling(y, x)$$
$$Sibling(Sue, Devin)$$
$$\neg Sibling(Devin, Alex)$$

First sentence is the only one with variables, there are 9 options (only 6 if $x \neq y$)

One unification is {x/Sue, y/Devin}
We cannot say {x/Devin, y/Alex}, as this is creates a contradiction

General modus ponens

We do not need to convert to propositional logic to use some rules of reasoning

Modus ponens can be applied even if there are variables, as long as we can unify them:

$$\forall x \; Large(x) \wedge Alive(x) \Rightarrow Dangerous(x)$$

$$\forall x \; Alive(x)$$

$$Large(Hippo)$$

We can unify the top sentence with {x/Hippo}, so we can conclude: $Dangerous(Hippo)$

General modus ponens

If you want to use this general modus ponens, finding the unification can be expensive

You basically need to try all substitutions, though you can store your data in smart ways to make look-up much more quickly

Using just general modus ponens, you can do basic inference with first order logic
(what is the problem??)

General modus ponens

Objects = {Cat, Dog, Frog, Rat, Sally, Jane}

$$\exists x \; Zodiac(x)$$
$$\forall x \; Alive(x) \Rightarrow Birthday(x)$$
$$\forall x \; Alive(x) \Rightarrow Eats(x)$$
$$\forall x, y \; Birthday(x) \Rightarrow Party(x, y)$$
$$\forall x \; Zodiac(x) \wedge Birthday(x) \Rightarrow Happy(x)$$
$$Alive(Sally)$$

Is Sally happy?

How about Party(Sally, Frog)?

General modus ponens

We can substitute {x/Sally} here with MP:

$\forall x \ Alive(x) \Rightarrow Birthday(x)$

To get: $Birthday(Sally)$

Then sub. {x/Sally, y/Frog} with MP here:

$\forall x, y \ Birthday(x) \Rightarrow Party(x, y)$

To get: $Party(Sally, Frog)$

However, we cannot tell if Sally is happy, as we cannot unify: $Zodiac(s1)$

$Birthday(Sally)$

General modus ponens

You try!

$$\forall x\ Meat(x) \wedge Make(Bread, x, Bread) \Rightarrow Sandwich(Bread)$$

$$\forall x, y\ OnGrill(x, y) \wedge Sandwich(y) \Rightarrow Grilled(y)$$

$$\forall x, y\ OnGrill(x, y) \wedge Meat(y) \Rightarrow Grilled(y)$$

$$\exists x\ Meat(x)$$

$$\forall x, y\ OnGrill(x, y)$$

$$\forall x, y, z\ Make(x, y, z)$$

$$Bread$$

Can you get Grilled(Bread)?

How about Grilled(Chicken)?

General modus ponens

You try!

$$\forall x\ Meat(x) \wedge Make(Bread, x, Bread) \Rightarrow Sandwich(Bread)$$

$$\forall x, y\ OnGrill(x, y) \wedge Sandwich(y) \Rightarrow Grilled(y)$$

$$\forall x, y\ OnGrill(x, y) \wedge Meat(y) \Rightarrow Grilled(y)$$

$$\exists x\ Meat(x)$$

$$\forall x, y\ OnGrill(x, y)$$

$$\forall x, y, z\ Make(x, y, z)$$

$$Bread$$

Can you get Grilled(Bread)? Yes

How about Grilled(Chicken)? No

Forward chaining

You probably just reasoned out the way to think through this, but we will talk about two algorithms to do this in a procedural manner

The first we will look at is forward chaining, where you build up new sentences using modus ponens until you generate your goal

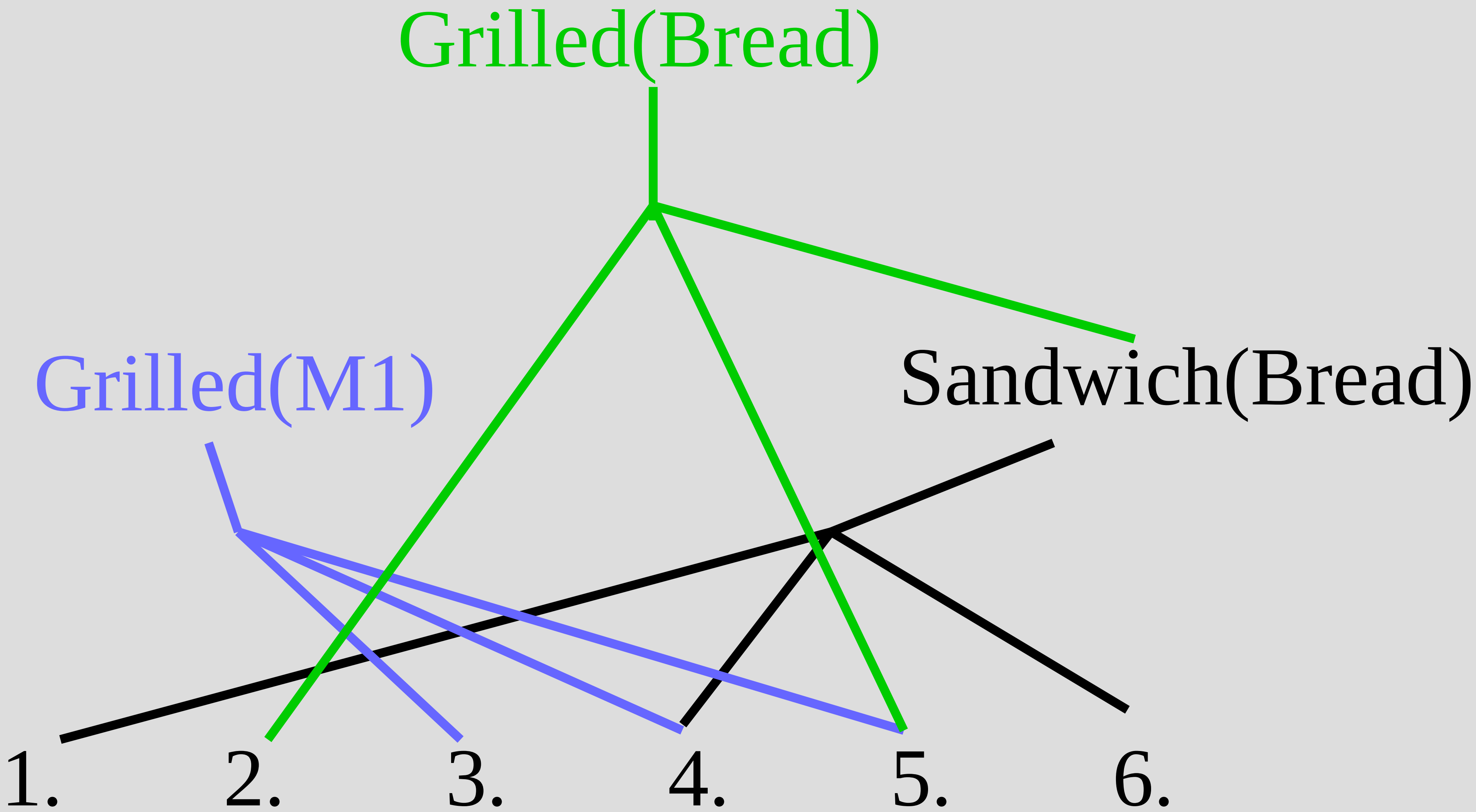
Then we will talk about improvements over this basic implementation

Forward chaining

Consider the following labeling...

1. $\forall x\ Meat(x) \land Make(Bread, x, Bread) \Rightarrow Sandwich(Bread)$
2. $\forall x, y\ OnGrill(x, y) \land Sandwich(y) \Rightarrow Grilled(y)$
3. $\forall x, y\ OnGrill(x, y) \land Meat(y) \Rightarrow Grilled(y)$
4. $\exists x\ Meat(x)$
5. $\forall x, y\ OnGrill(x, y)$
6. $\forall x, y, z\ Make(x, y, z)$

Forward chaining



Forward chaining

Algorithm:

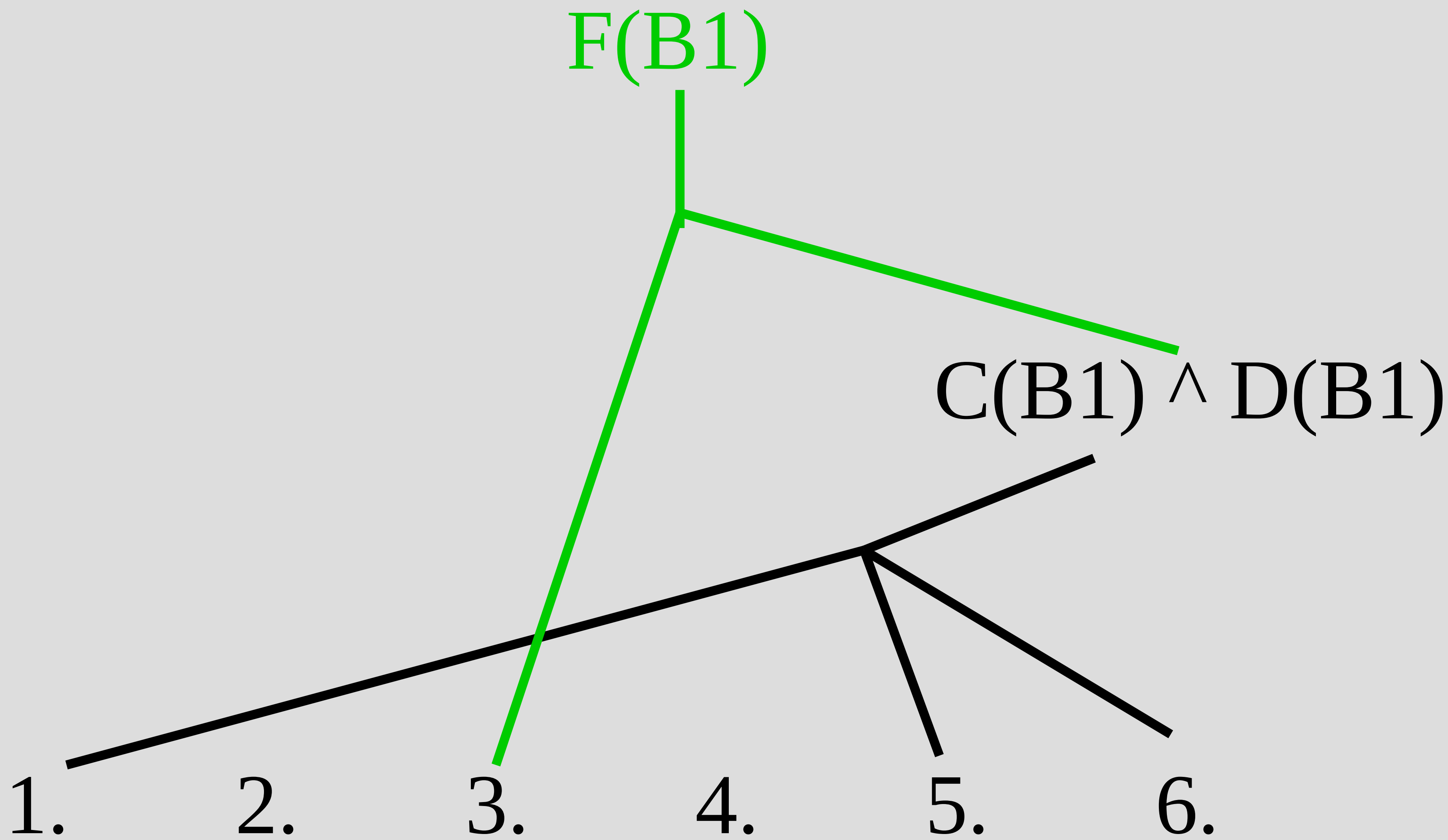
1. repeat until new empty
2. new ← {}
3. for each sentence in KB
4. for each substantiation for a modus ponens
5. q ← substitute RHS of modus ponens
6. if q does not unify/match sentence in KB
7. new ← new U q
8. if q satisfies query, return q
9. add new to KB
10. return false

Forward chaining

Build the whole forward chaining KB for:

1. $\forall x \;\; A(x) \wedge B(x) \Rightarrow C(x) \wedge D(x)$
2. $\exists x \;\; C(x) \Rightarrow E(x)$
3. $\forall x \;\; D(x) \Rightarrow F(x)$
4. $\forall x \;\; E(x) \wedge F(x) \Rightarrow G(x)$
5. $\forall x \;\; A(x)$
6. $\exists x \;\; B(x)$

Forward chaining



Forward chaining

This basic approach is redundant and can be improved in three major ways:

1. Improve the efficiency of unification
   - Allows for faster modus ponens
2. Incremental forward chaining
   - Reduces redundant computations
3. Eliminate irrelevant facts
   - Prunes KB

Unification efficiency

It is efficient to unify/substitute for the variable with the least possibilities on the left hand side (LHS) of modus ponens

This is basically the same arguments are the "minimum remaining value" for CSPs

The look-up of values for a single variable is constant time, but then we need to compare against all other in sentence (NP-hard problem)

Unification efficiency

In the example, we only have B(B1) true for some variable B1, which is probably smaller than all possible A(x)

So here it would make sense to substitute B1 first into the first sentence, then try to find a matching A value (which is easy as A(x) is valid for any x)

Incremental chaining

All novel sentences build off the “new” set (except for building the first level)

The computer re-searches all the old sentences every time and regenerates the same sentences

By requiring the “new” set to be involved, we can greatly cut down computation of the depth of chain tree is fairly deep

Incremental chaining

In the example, the first loop of chaining finds:

$C(B1) \wedge D(B1)$

When starting the second loop, all possible combinations of the original KB will be searched again, and generate the above again

Instead, we can limit our search to just $C(B1) \wedge D(B1)$ combined with any of the original KB sentences

Eliminate irrelevancy

There are two primary ways to do this:

1. Start from the goal and work backwards
2. Restrict KB to help guide search

The first way works backwards keeping track of any possible useful sentences

Any sentences not found on the backtrack can be discarded without effecting this query

Eliminate irrelevancy

You can add more restrictions to existing sentences to focus the search early on

This combined with the unification efficiency can greatly speed up search

For example, if we queried: $F(Cat)$, we could modify the first sentence:

$$\forall x \; Elim(x) \wedge A(x) \wedge B(x) \Rightarrow C(x) \wedge D(x)$$

and add Elim(Cat) to cause conflict early

Forward chaining

Forward chaining is <u>sound</u> (will not create invalid sentences)

If all the sentences in the KB are <u>definite</u> then it is <u>complete</u> (can find all entailed sentences)

<u>Definite</u> means that there can only be one positive literal in CNF form (anything with an implies has only one relationship on the RHS)

Backward chaining

Backward chaining is almost the opposite of forward chaining (and eliminating irrelevancy)

You try all sentences that are of the form: $P1 \wedge P2 \wedge ... \Rightarrow Goal$, and try to find a way to satisfy P1, P2, ... recursively

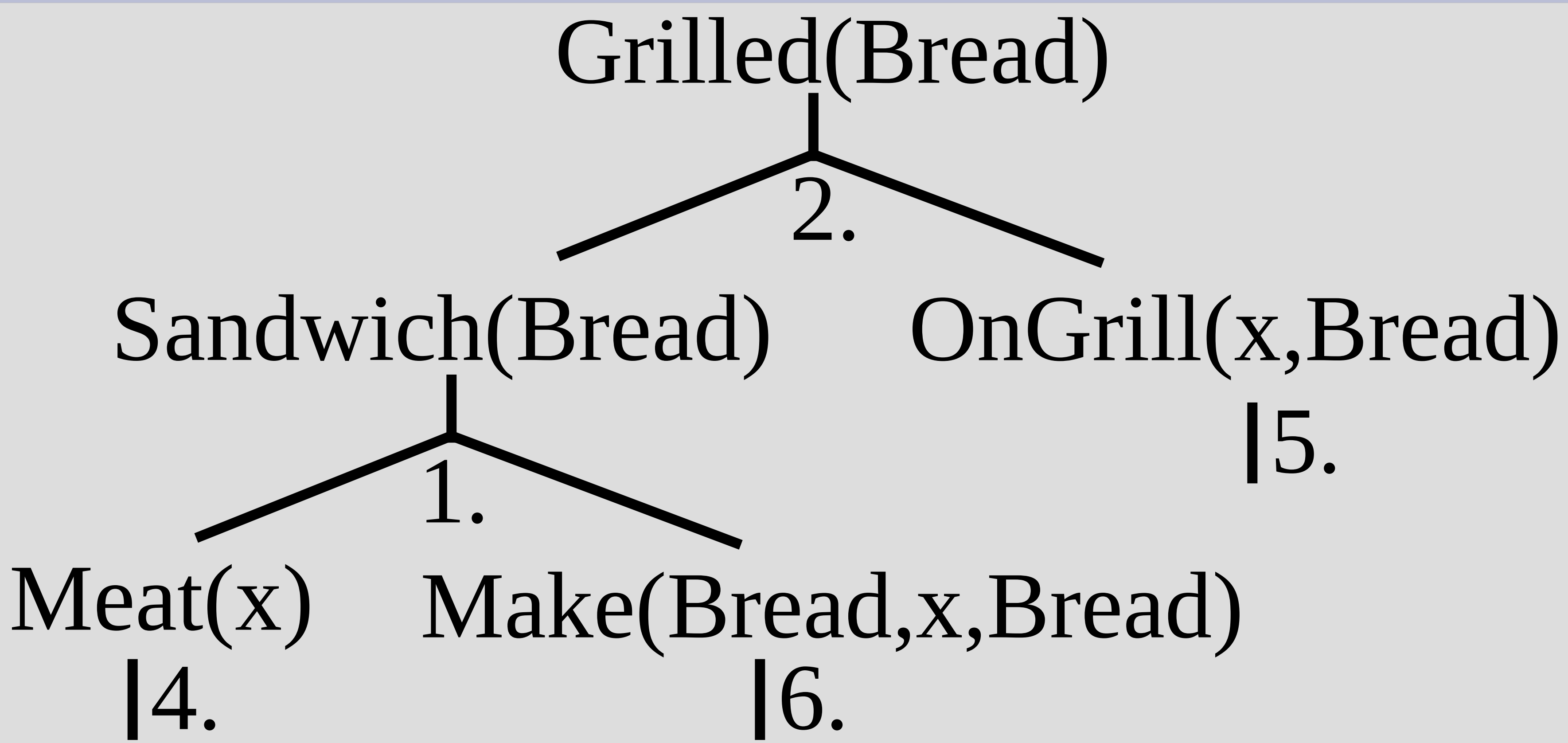
This is similar to a depth first search AND/OR trees (OR are possible substitutions while AND nodes are the sentence conjunctions)

Backward chaining

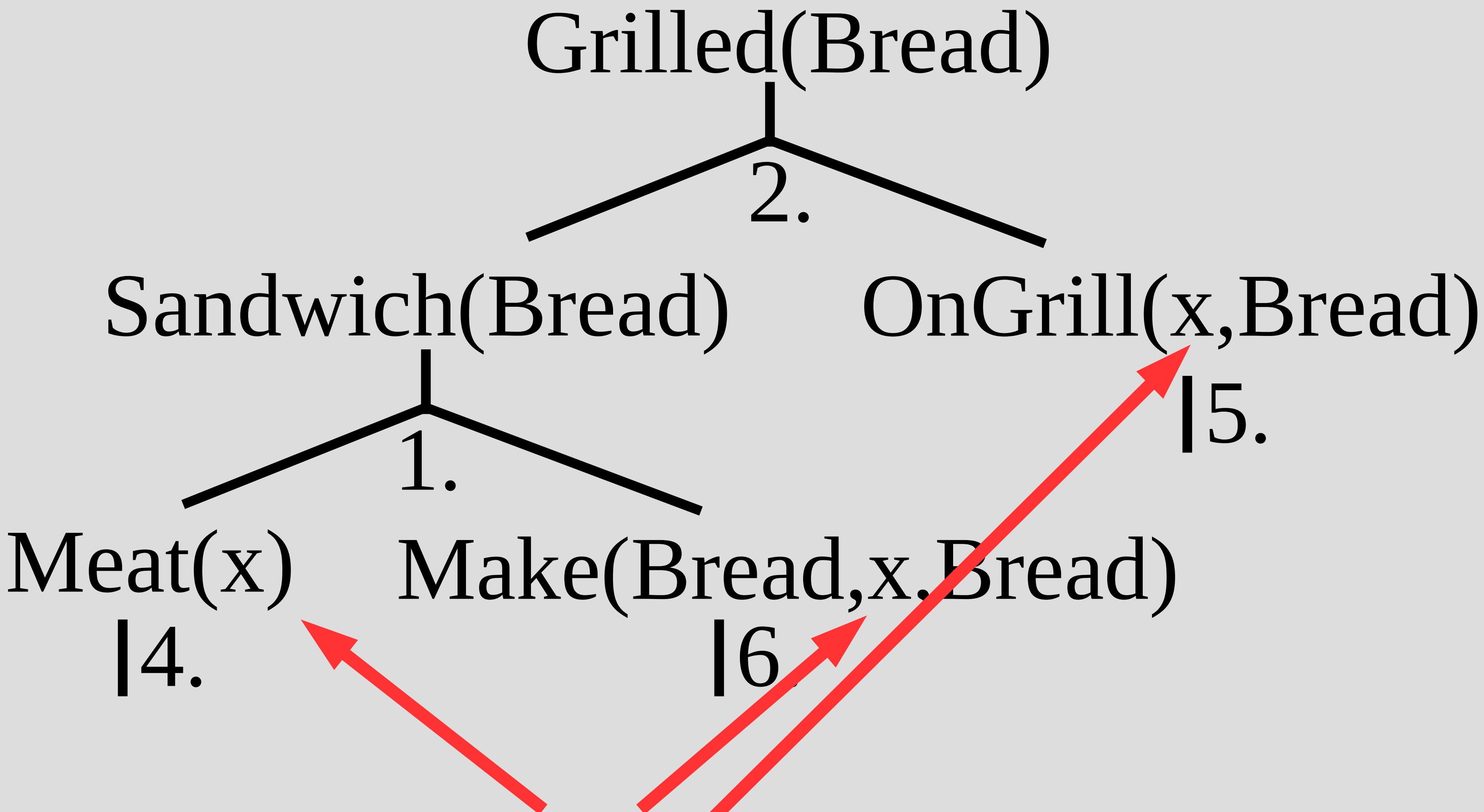
Let's go back to this and backward chain Grilled(Bread)

1. $\forall x\ Meat(x) \wedge Make(Bread, x, Bread) \Rightarrow Sandwich(Bread)$
2. $\forall x, y\ OnGrill(x, y) \wedge Sandwich(y) \Rightarrow Grilled(y)$
3. $\forall x, y\ OnGrill(x, y) \wedge Meat(y) \Rightarrow Grilled(y)$
4. $\exists x\ Meat(x)$
5. $\forall x, y\ OnGrill(x, y)$
6. $\forall x, y, z\ Make(x, y, z)$

Backward chaining

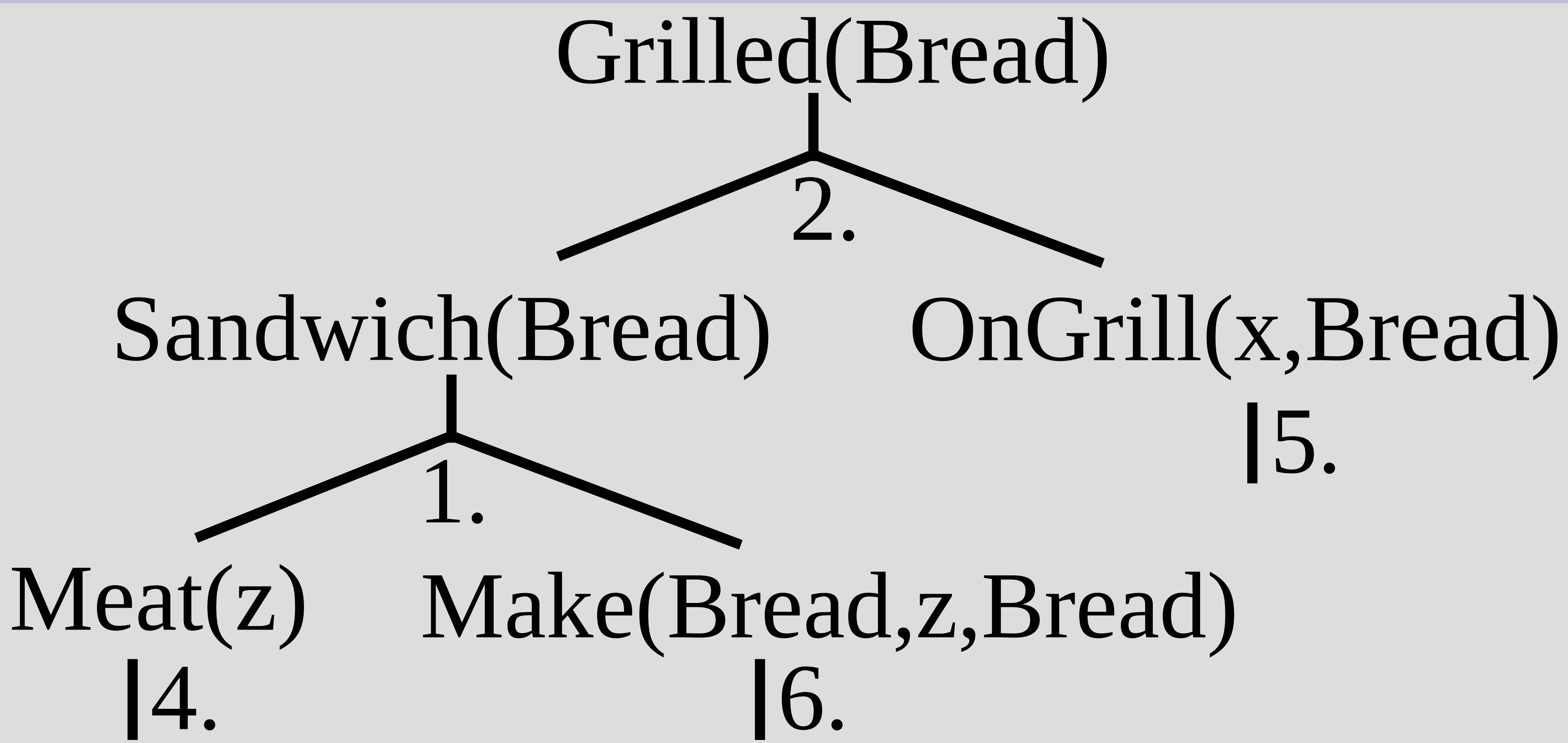


Backward chaining



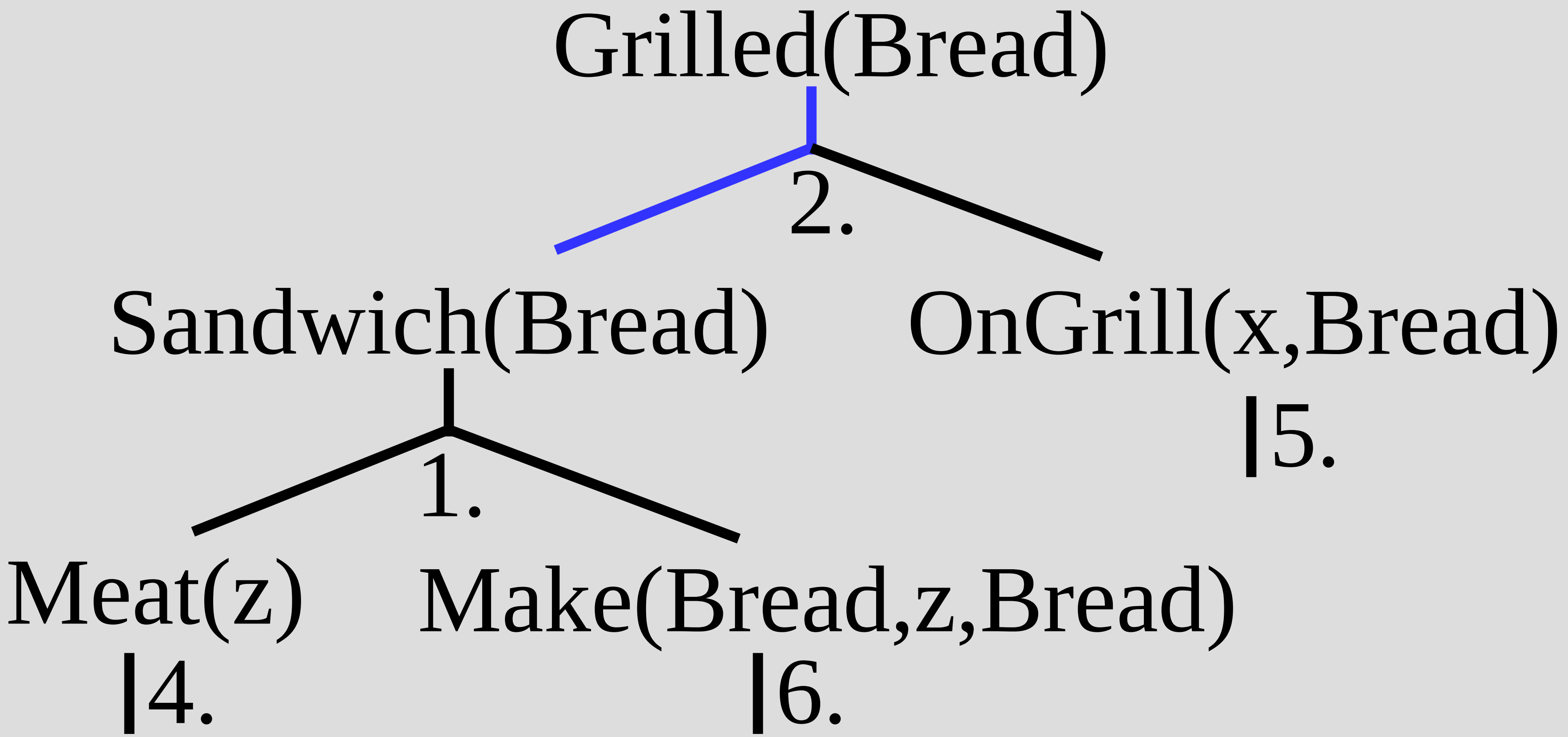
These variables have no correlation,
so relabel one to be different

Backward chaining



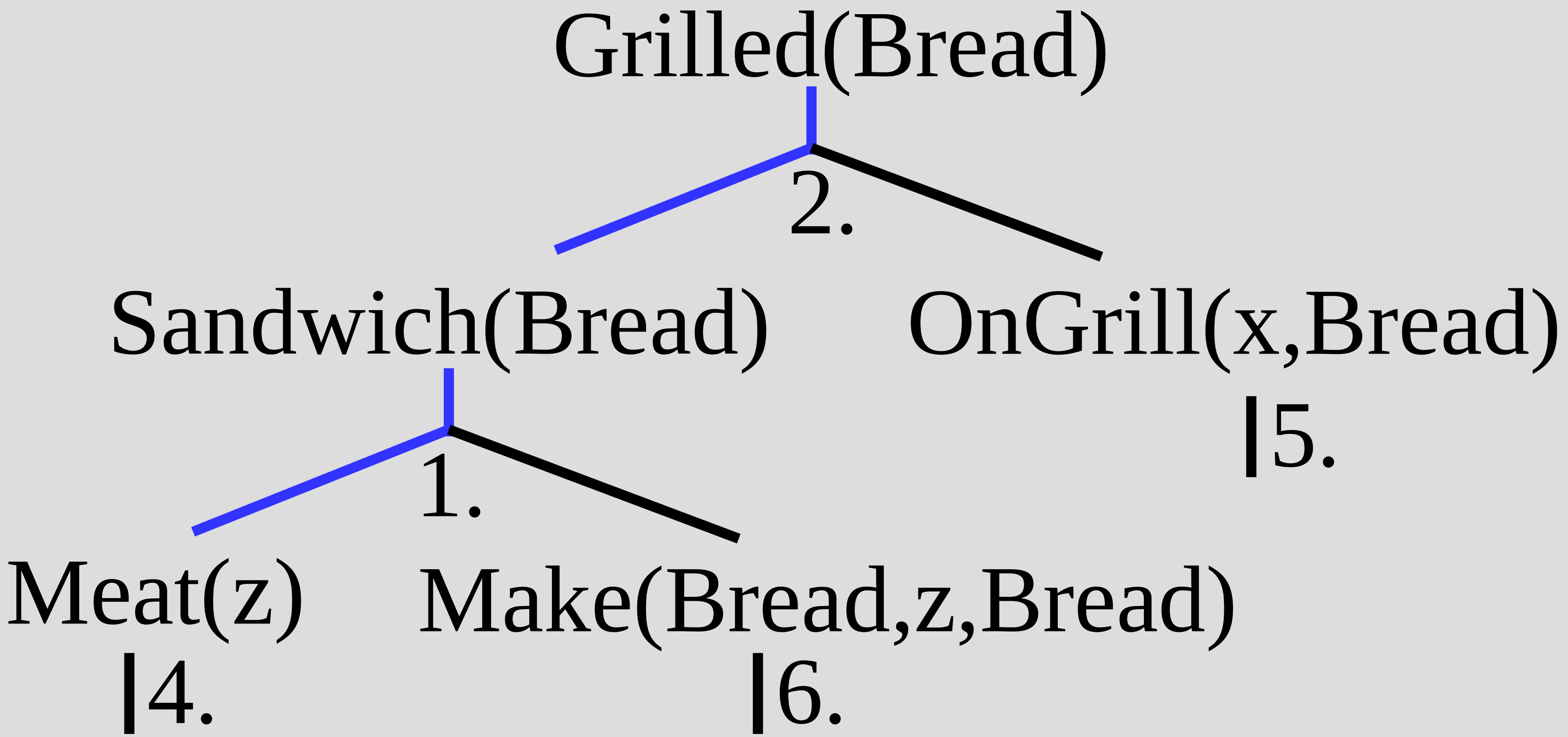
Begin DFS (left branch first)

Backward chaining



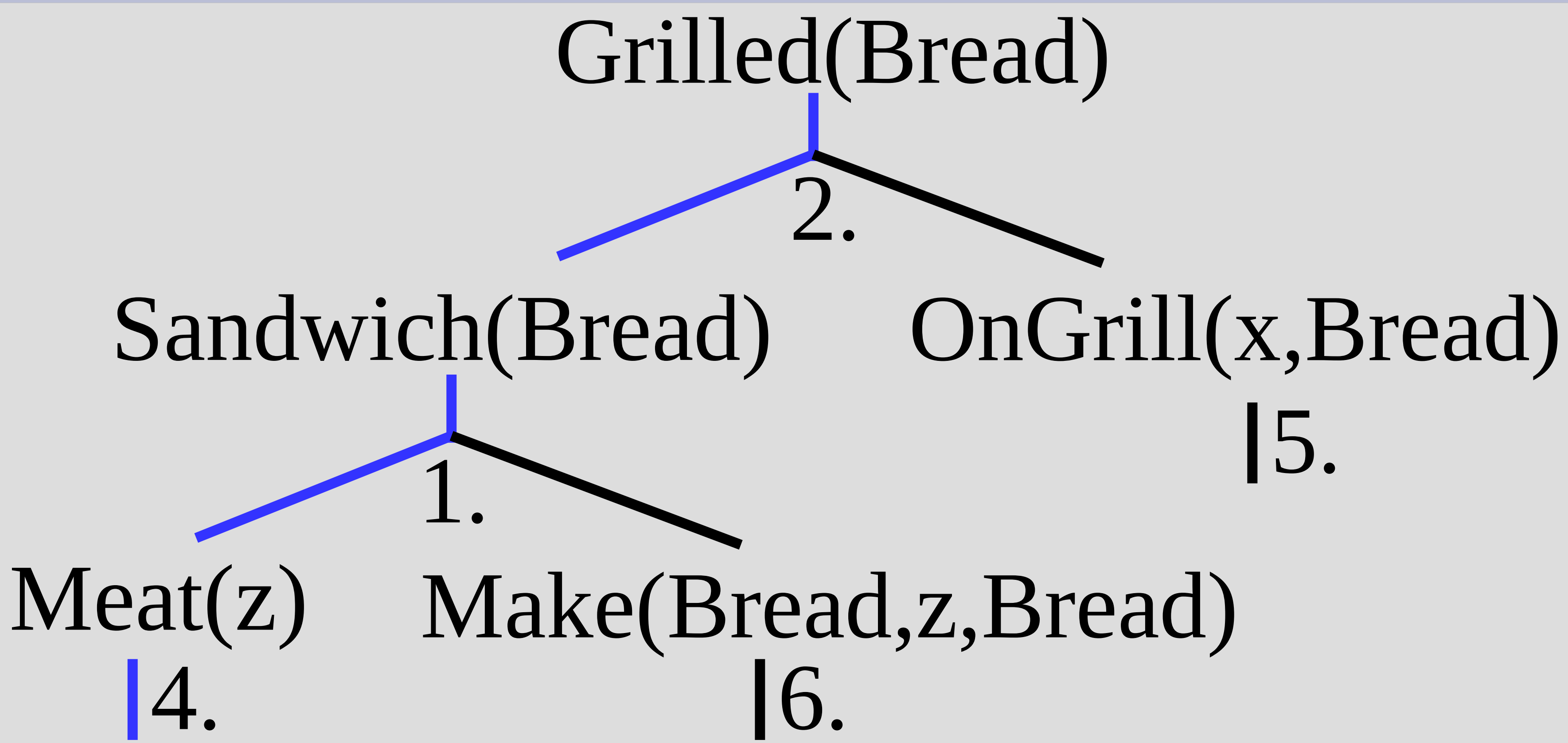
Begin DFS (left branch first)

Backward chaining



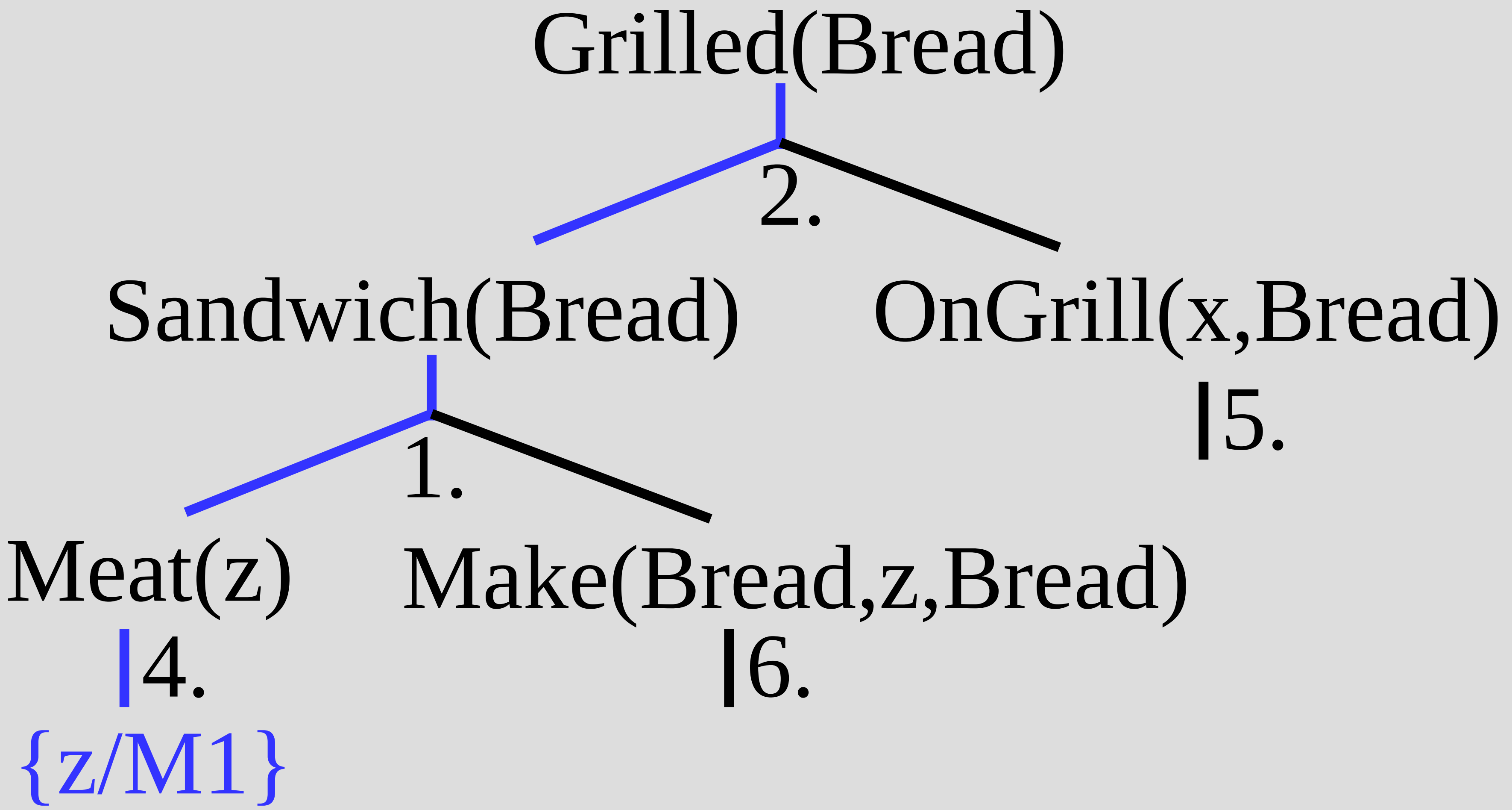
Begin DFS (left branch first)

Backward chaining



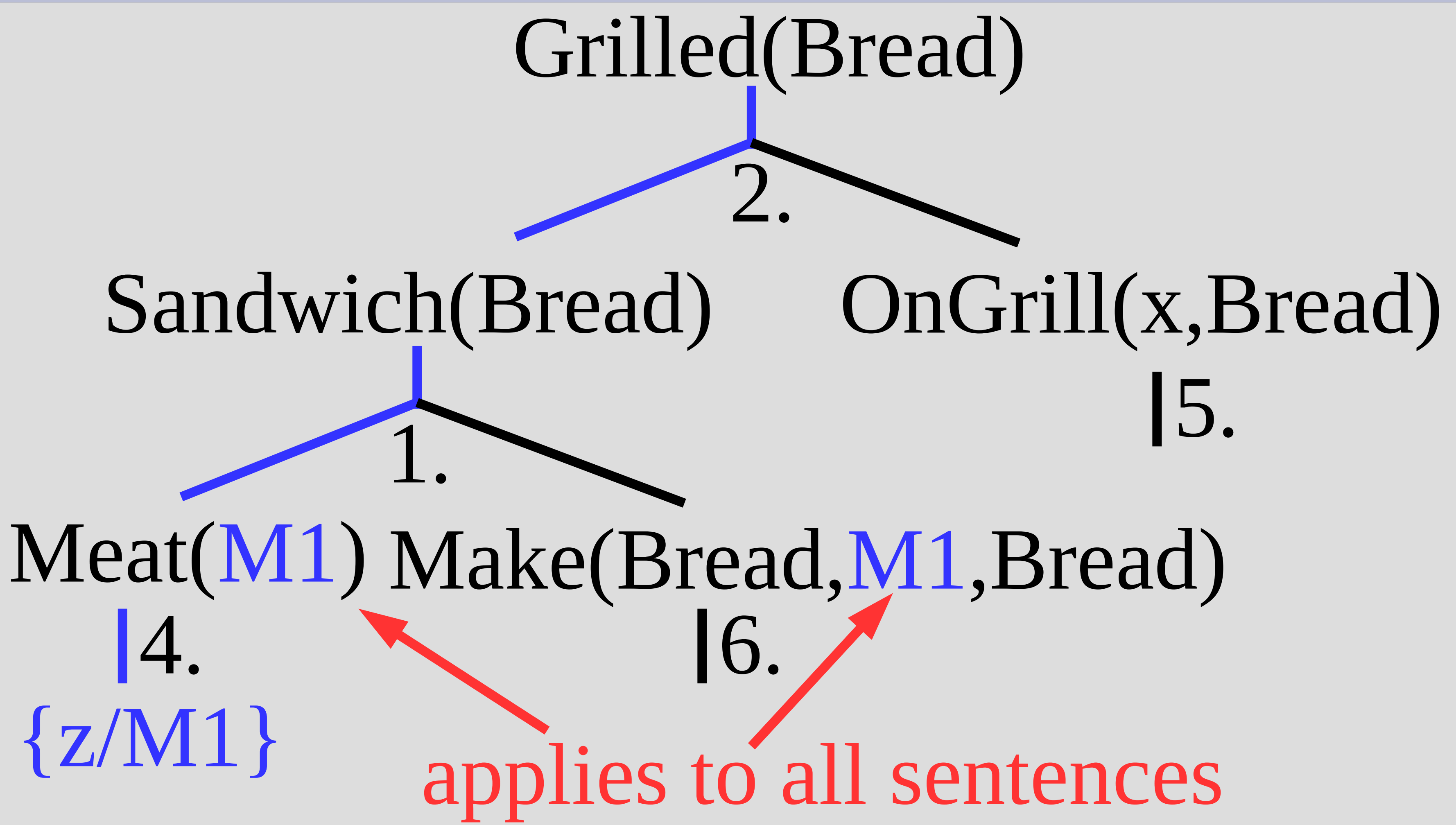
Begin DFS (left branch first)

Backward chaining



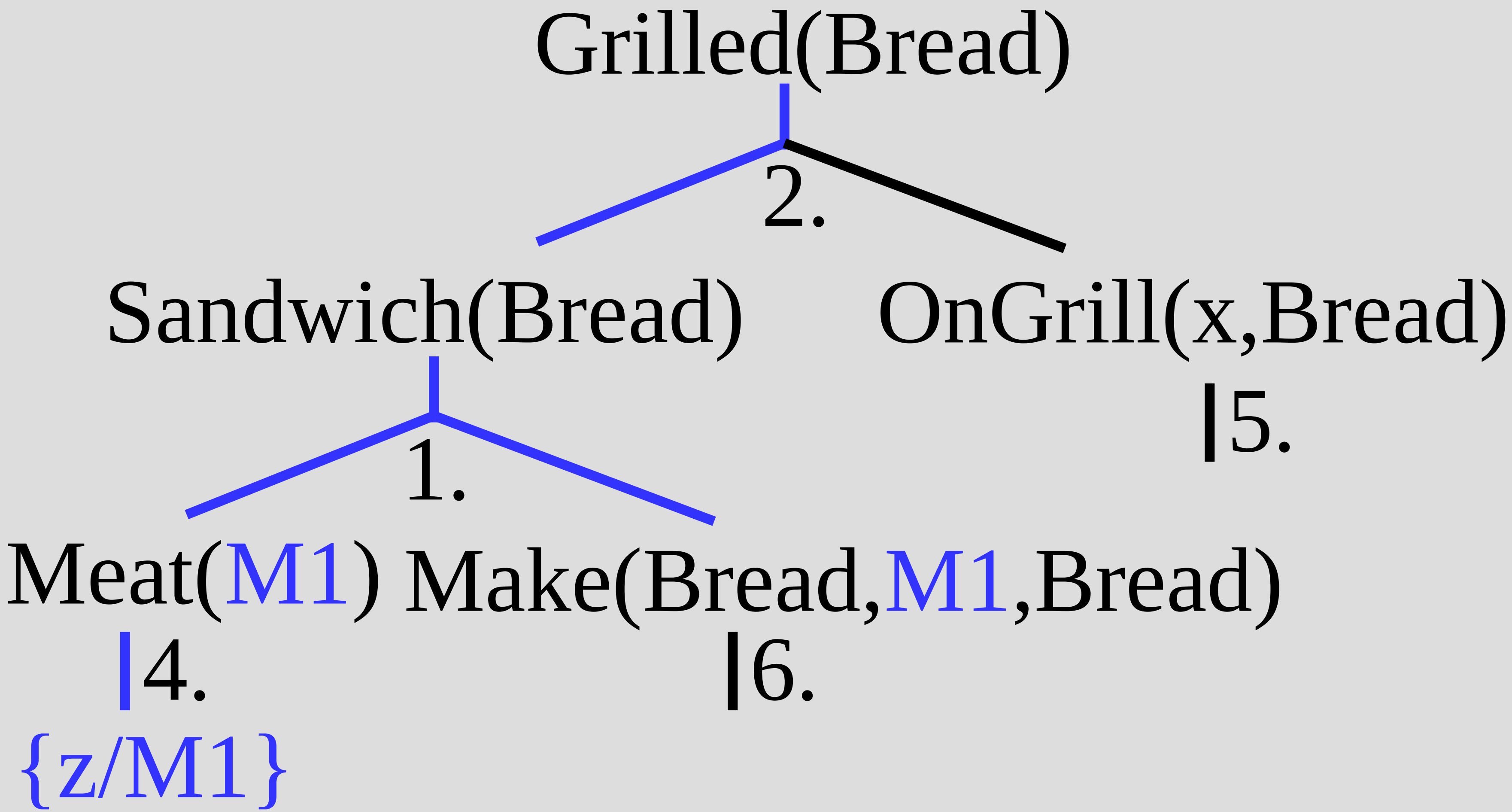
Begin DFS (left branch first)

Backward chaining



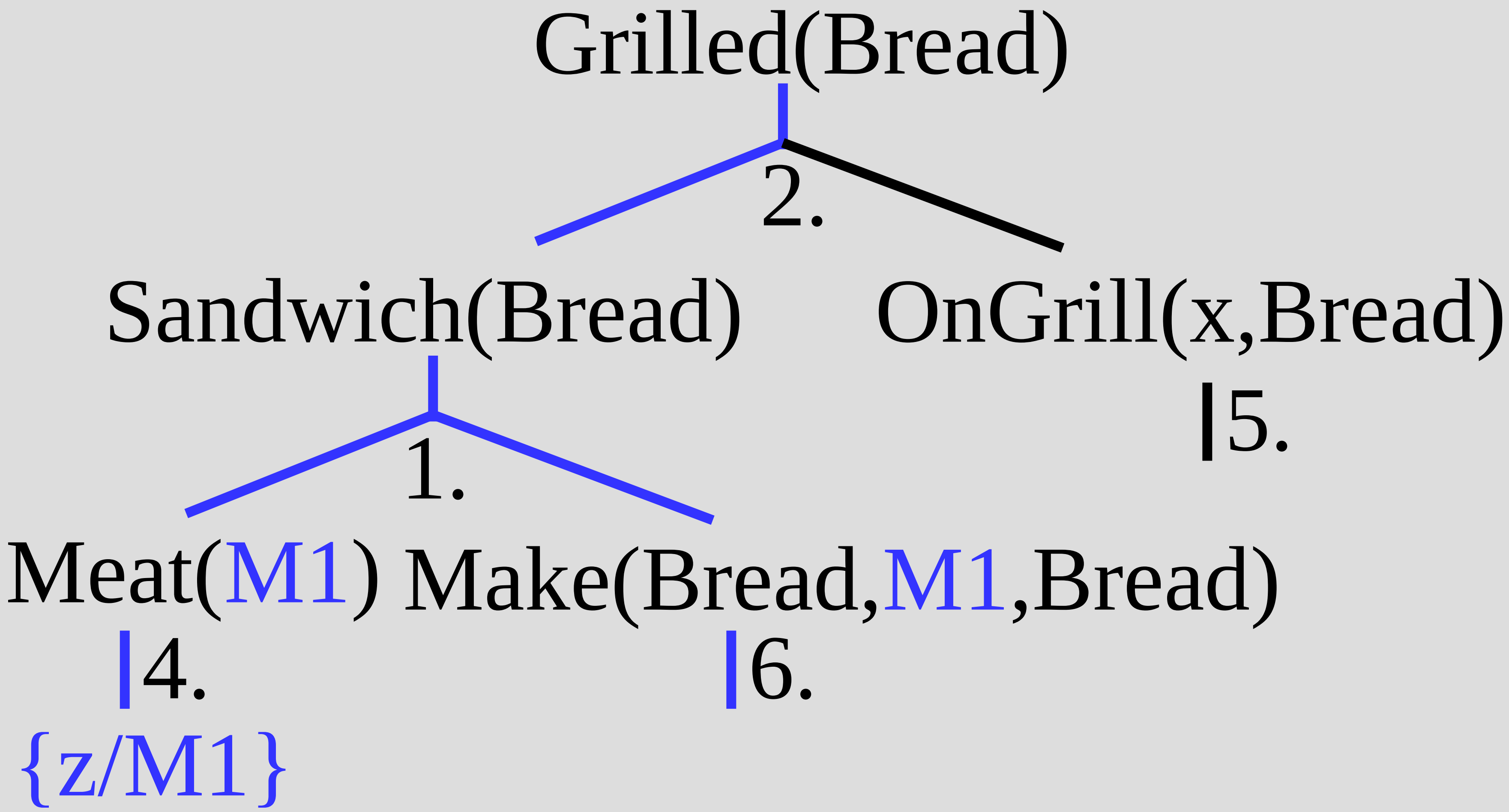
Begin DFS (left branch first)

Backward chaining



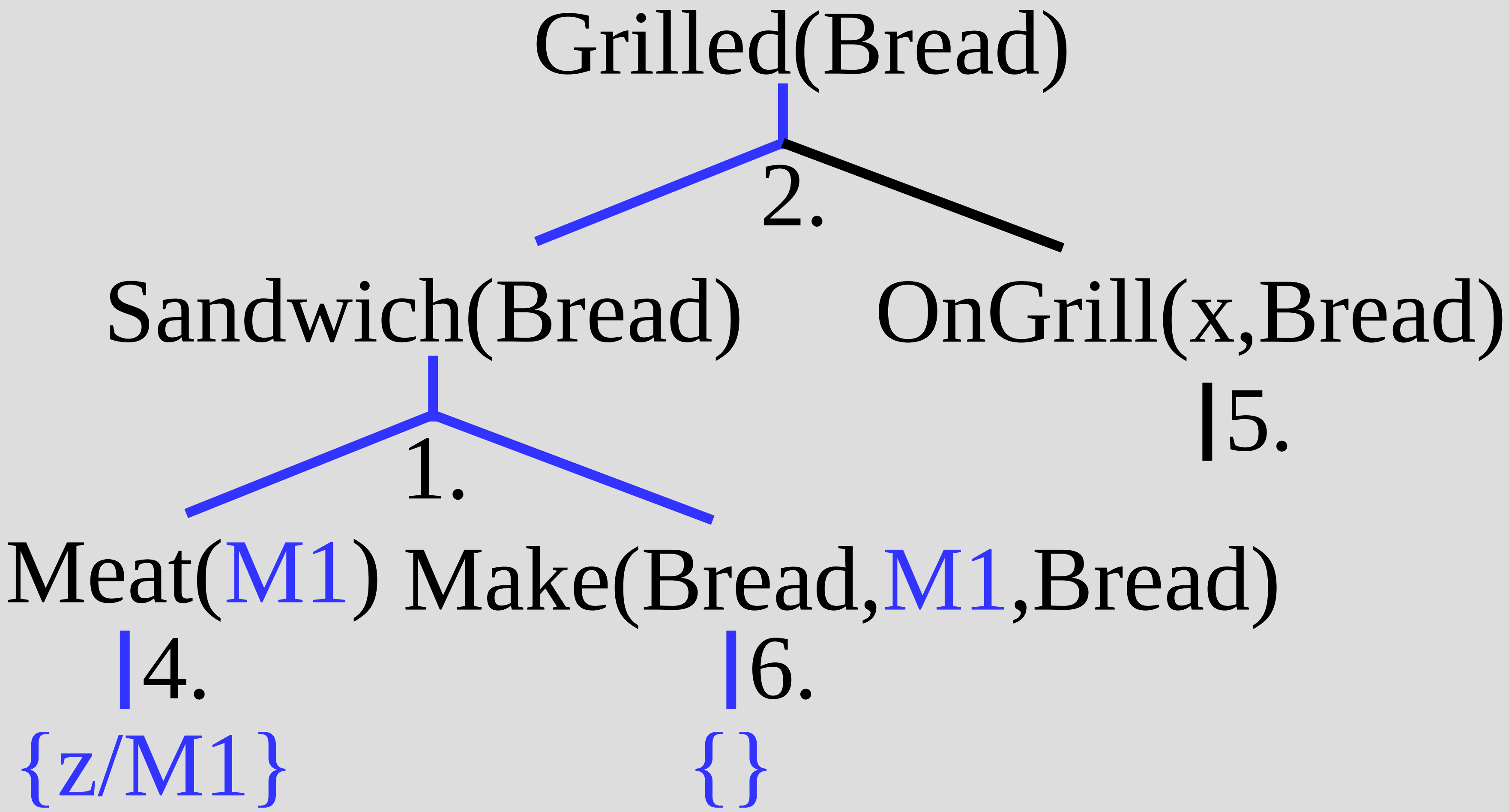
Begin DFS (left branch first)

Backward chaining



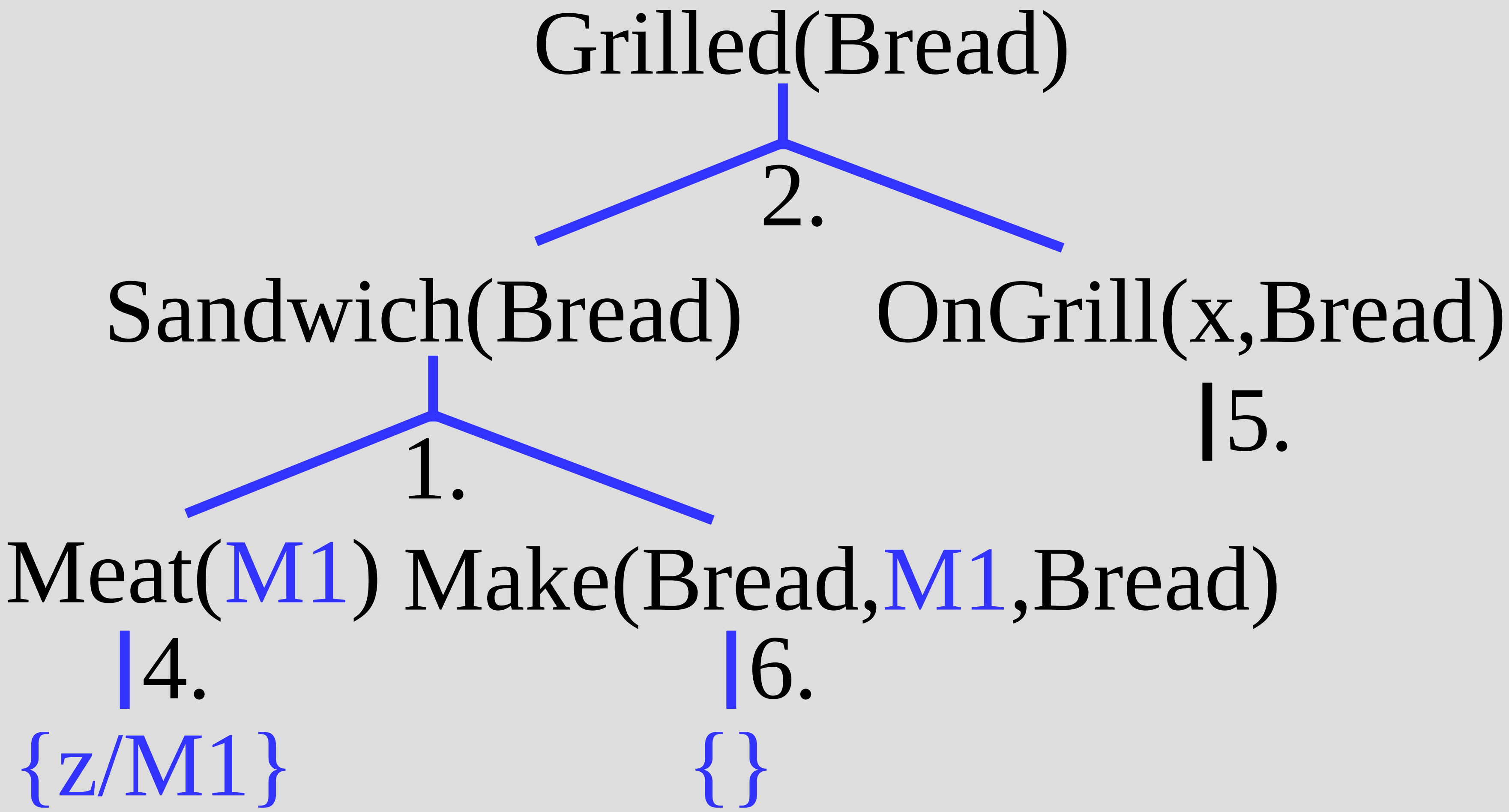
Begin DFS (left branch first)

Backward chaining



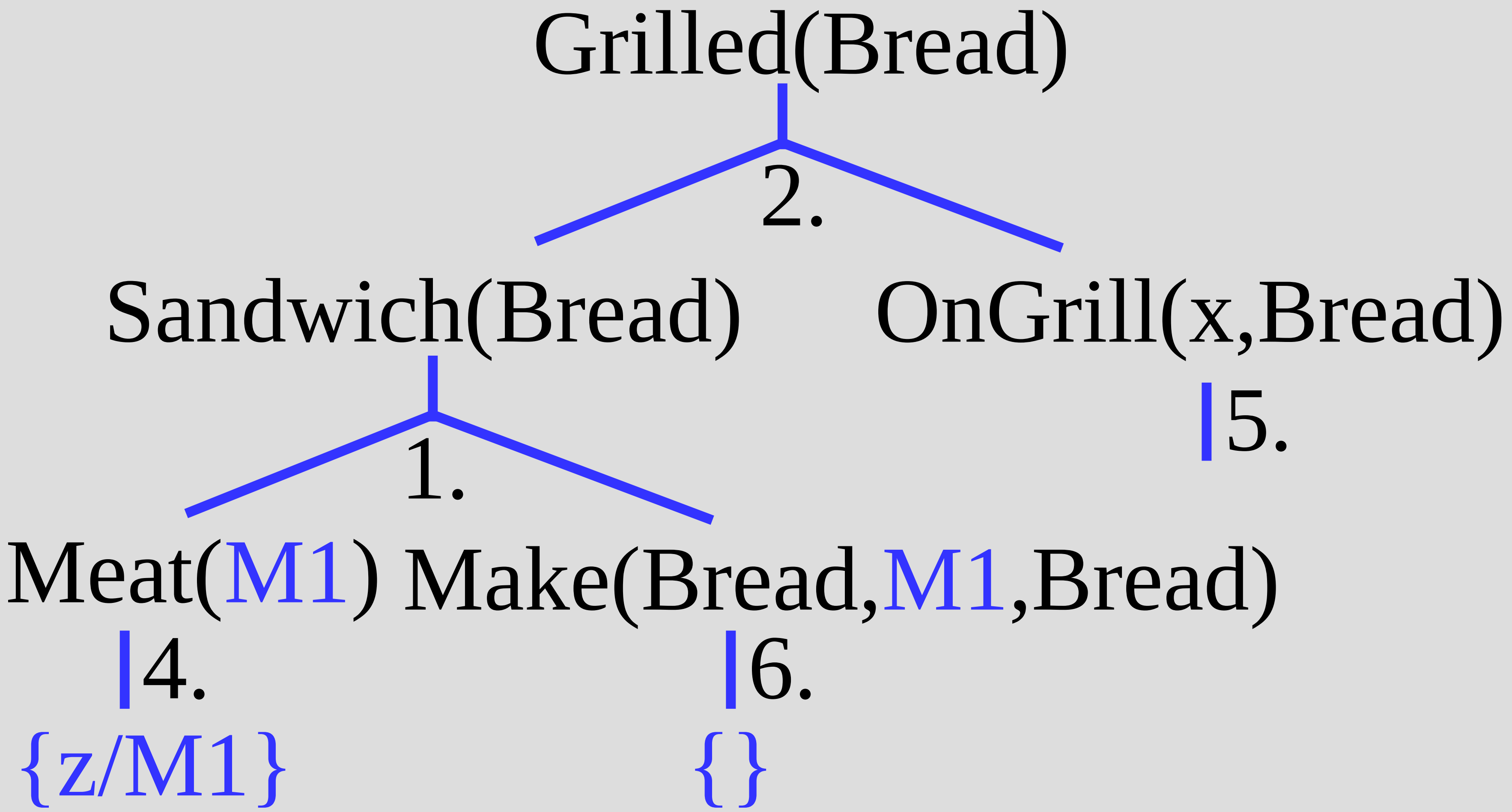
Begin DFS (left branch first)

Backward chaining



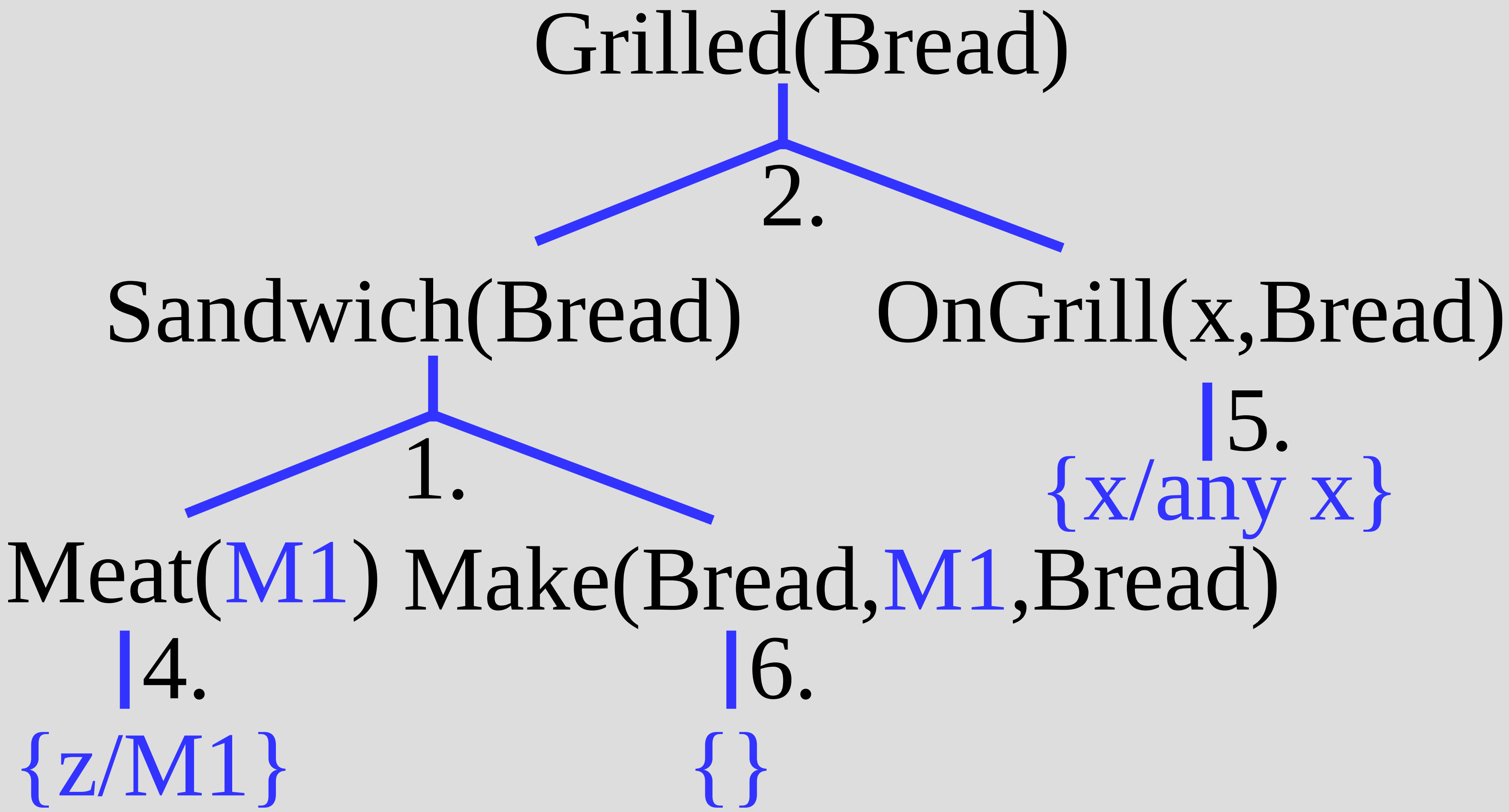
Begin DFS (left branch first)

Backward chaining



Begin DFS (left branch first)

Backward chaining



Begin DFS (left branch first)

Backward chaining

The algorithm to compute this needs to mix between going deeper into the tree (ANDs) and unifying/substituting (ORs)

For this reason, the search is actually two different mini-algorithms that intermingle:

1. FOL-BC-OR (unify)
2. FOL-BC-AND (depth)

Backward chaining

FOL-BC-OR(KB, goal, sub)
1. for each rule (lhs => rhs) with rhs == goal
2. standardize-variables(lhs, rhs)
3. for each newSub in FOL-BC-AND(KB, lhs, unify(rhs, goal sub))
4. yield newSub

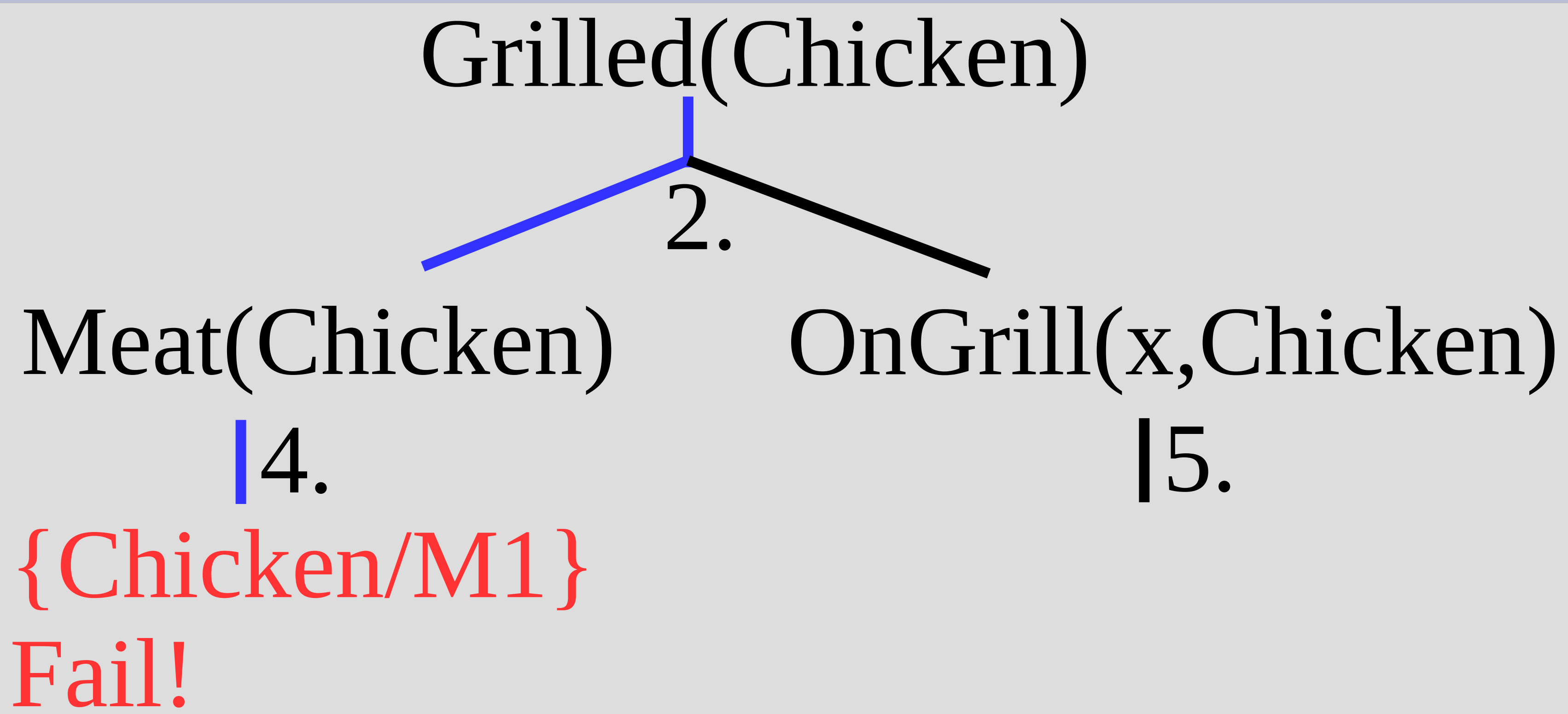
FOL-BC-AND(KB, goals sub)
1. if sub = failure, return
2. else if length(goals) == 0 then yield sub
3. else
4. first,rest ← First(goals), Rest(goals)
5. for each newSub in FOL-BC-OR(KB, substitute(sub, first), sub)
6. for each newNewSub in FOL-BC-AND(KB, rest, newSub)
7. yield newNewSub

Backward chaining

Use backward chaining to infer:
Grilled(Chicken)

1. $\forall x\ Meat(x) \wedge Make(Bread, x, Bread) \Rightarrow Sandwich(Bread)$
2. $\forall x, y\ OnGrill(x, y) \wedge Sandwich(y) \Rightarrow Grilled(y)$
3. $\forall x, y\ OnGrill(x, y) \wedge Meat(y) \Rightarrow Grilled(y)$
4. $\exists x\ Meat(x)$
5. $\forall x, y\ OnGrill(x, y)$
6. $\forall x, y, z\ Make(x, y, z)$

Backward chaining



Begin DFS (left branch first)

Backward chaining

Similar to normal DFS, this backward chaining can get stuck in infinite loops (in the case of functions)

However, in general it can be much faster as it can be fairly easily parallelized (the different branches of the tree)