EECS 391 Project: Forward chaining in Python

John Lambert – jlambert@jlambert.com

May 6, 2002

Contents

1 Introduction

This system is for forward-chaining rule-based systems written in Python. It takes a plain-text file of rules as input and outputs a graph, generated using `dotty` [1], of the relationships between facts and conclusions, as well as how it reached them for a given set of base knowledge. All examples in this report use the Zookeeper rules presented in [2], but the system will work with similar rules in any problem domain.

2 Approach

There are three main sections to the implementation: the rules, knowledge representation, and the algorithm for doing the actual forward chaining.

2.1 Rules

2.1.1 Encoding

Rules are stored as lines in a file. They consist of two parts: an `if` expression and a `then` conclusion and are written like `if_part_expression -> then_conclusion`. The `if` expression is the set of facts which must be true for the `then` conclusion to be true. For example, Rule Z14 from the textbook [2, page 124], is:

If	?x is a bird
	?x does not fly
	?x has long legs
	?x has a long neck
	?x is black and white
then	?x is an ostrich

This rule is encoded in `rules.txt` as:

```
# ostrich rule
is_bird and not flies and (has_long_legs and has_long_neck) \
and has_black_and_white_color -> is_ostrich
```

Several things are worth noting:

1. There are no spaces in any of the terms: "has long neck" becomes `has_long_neck`.
2. Negation is explicitly specified: "does not fly" was represented as `not flies`.
3. The original rule contained implicit **and**s; these are explicitly placed in the `if` part of the rule.
4. There is one rule per line; the \ is only present in this document.
5. One-line comments are allowed and are set off with # as the first character.
6. There are parentheses in the rule, although they serve no functional purpose in this specific rule.
7. The `or` operation is allowed, although it is not present in this specific rule.

The rules in `rules.txt` are verified to make sure they are valid: properly parenthesized with one and only one conclusion.

2.1.2 Program representation

The rules are stored in memory as a list[1] of two-item tuples[2]. The first item is the `if` expression and the second part is the `then` conclusion:

```
[   ('has_hair', 'is_mammal'),
    ('gives_milk', 'is_mammal'),
    ('has_feathers', 'is_bird'),
    ('flies and lays_eggs', 'is_bird'),
    ('is_mammal and eats_meat', 'is_carnivore'),
    ...
]
```

2.2 Knowledge

Now that we have our rules processed, we want to create some baseline knowledge about a concept. This breaks down into two things: things we know to be true and things we know to be false.

For example, we know that Tux has feathers, swims, and has black and white color. Also, Tux does not fly. We represent this with a map[3] from a condition to a true (1) or false (0) value:

```
{
    'flies': 0,
    'has_black_and_white_color': 1,
    'has_feathers': 1,
    'swims': 1
}
```

2.3 Algorithm

Now that we have our base knowledge and our rules, we follow the algorithm presented on [2, page 128].

- While we are still finding new conclusions:[4]
  - For each r in Rules:
    1. Set the antecedants to be the if part of r
    2. Set the conclusion to be the then part of r
    3. For each *item* in our base knowledge:

[1] In Python, a list is a sequence of arbitrary length containing data of any type. It is zero-indexed and displayed as `[4, 53, 'test']`.

[2] In Python, a tuple is a data structure of fixed arity containing data of any type. A two-tuple `t` has items at 0 and 1 and is displayed as `(4, 'test')`.

[3] In Python, a map is an associative container which takes a key to a value, like a hash table. The map `m` displayed as `{'another_key': 0, 'key': 1}` takes `'key'` to the value 1 when accessed as `m['key']`.

[4] Since this is a general system, we use this as a stopping point instead of identifying an animal.

        * Replace every occurance of the item in the antecedant with the known value for that item[5]
    4. If the antecedant is a valid logical expression[6] **and** the antecedant evaluates to 1:
        * Indicate that rule r has been fired
        * Draw links from the antecedants to the conclusion, possibly with a box for the rule
        * Indicate that we made a new conclusion
        * Add the conclusion to our base knowledge with a value of 1 so we can use it for remaining rules

2.4 Bonus: conclusion hierarchy

In order to help test the system, I had it display the "chain of inferences" possible for any given rule set. The algorithm is:

- For each r in Rules:
    - For each term in the antecedant of r:
        * Draw a link from term to r's conclusion
        * If the term is negated, then label the link with "not"

3 Sample runs

The sample runs are for the Zookeeper system.

3.1 Input rules from `rules.txt`

```
has_hair -> is_mammal
gives_milk -> is_mammal
has_feathers -> is_bird

flies and lays_eggs -> is_bird
is_mammal and eats_meat -> is_carnivore

is_mammal and has_pointed_teeth and has_claws
    and has_forward_pointing_eyes -> is_carnivore

is_mammal and has_hoofs -> is_ungulate
is_mammal and chews_cud -> is_ungulate

is_carnivore and has_tawny_color and has_dark_sports -> is_cheetah
```

[5]For example, (`is_mammal and has_hoofs`) becomes (`is_mammal and 0`)

[6]For example, `1 and (1 or not 0)`, or `0 and unknown`, which will evaluate to 0 in all cases

```
is_carnivore and has_tawny_color and has_black_stripes -> is_tiger

is_ungulate and has_long_neck and has_long_legs
     and has_dark_spots -> is_giraffe
is_ungulate and has_black_stripes -> is_zebra

# bird rules
is_bird and not flies and has_long_legs and has_long_neck
     and has_black_and_white_color -> is_ostrich
is_bird and swims
    and (not flies and has_black_and_white_color) -> is_penguin
is_bird and flies -> is_albatross
```

3.2 Graph of rules

See Figure 1 on page 8 for the relationship between facts and rules. Figure 2 on page 9 for the relationship between facts.

3.3 Animals

3.3.1 Tiger

This is the base knowledge for Tiger. (Note that it does not fly; this was just extra information to see how the system would respond.)

```
{   'eats_meat': 1,
    'flies': 0,
    'gives_milk': 1,
    'has_black_stripes': 1,
    'has_tawny_color': 1}
```

Figures 3 on page 10 shows with rule boxes, and Figure 4 on page 10 shows it without rule boxes. It came to the appropriate conclusions: `is_mammal, is_carnivore, is_tiger`. (Any oval that is not in the base knowledge box is an inferred conclusion.)

3.3.2 Penguin

Tux the Penguin has the following base knowledge:

```
{   'flies': 0,
    'has_black_and_white_color': 1,
    'has_feathers': 1,
    'swims': 1}
```

Figures 5 on page 11 shows with rule boxes, and Figure 6 on page 11 shows it without rule boxes. It came to the appropriate conclusions: `is_bird, is_penguin`

3.3.3 Zebra

The Zebra has the following base knowledge:

```
{   'gives_milk': 1,
    'has_black_stripes': 1,
    'has_hair': 1,
    'has_hoofs': 1}
```

Figures 7 on page 12 shows with rule boxes, and Figure 8 on page 13 shows it without rule boxes. It came to the appropriate conclusions: `is_mammal, is_ungulate, is_zebra`

3.3.4 Nightmare

I remembered one of my cousins attempting to describe a creature in one of his dreams. Obviously, this doesn't exist, but it's interesting to see the conclusions drawn.

```
{   'eats_meat': 1,
    'flies': 1,
    'gives_milk': 1,
    'has_black_stripes': 1,
    'has_dark_spots': 1,
    'has_feathers': 1,
    'has_hoofs': 1,
    'has_long_legs': 1,
    'has_long_neck': 1}
```

Figures 9 on page 14 shows with rule boxes, and Figure 10 on page 15 shows it without rule boxes. It came to several conclusions: `is_albatross, is_bird, is_carnivore, is_giraffe, is_mammal, is_ungulate, is_zebra`. These are neither correct nor incorrect; it's a fictional animal.

4 Summary and conclusions

Proud I'm proud of the generic-ness: it works with other systems as well. (I made a simple one to distinguish between vegan, vegetarian, ovo-lacto-vegetarian, and kosher; it worked.) I also like the graphical output format: it makes it a lot easier to understand than pure-text output.

Improvements If I was going to improve it, I would add better input/output. There's no input now since it's forward chaining: the user knows everything ahead of time and they can just input it in the function call. I'm not quite happy with the handling of `not`: the formatting is sub-optimal and I'd like it to be able to draw "negative conclusions" (animal_products -> not vegan), which opens a whole host of issues. It would be neat to "overlay" a specific instance of the system on top of the full rule graph so you can see what paths were taken versus what paths were possible; this is just an input/output issue, though. Also, `or` clauses and `and` clauses show up the same and parentheses aren't indicated on the graph at all.

Implementation Python's `eval` function made things easy: I could just pass in some arbitrary boolean expression and have it figure it out; I'd probably have to write a parser if I did this in C++. Getting the output to be both correct and useful was difficult, and I had some termination problems to start with.

Lessons learned I learned that the general solution was somewhat easier (and definately more fun) to implement than the specific solution. Also, the choice of language can help speed up implementation time. Backward chaining is good for testing specific hypotheses, especially if no data has been collected: "Was that a tiger?" "Well, did it have a tawny color?" Forward chaining is good when there is already data and the goal is to find out as much about it as possible: "It had feathers, and it was swimming; what was it?"

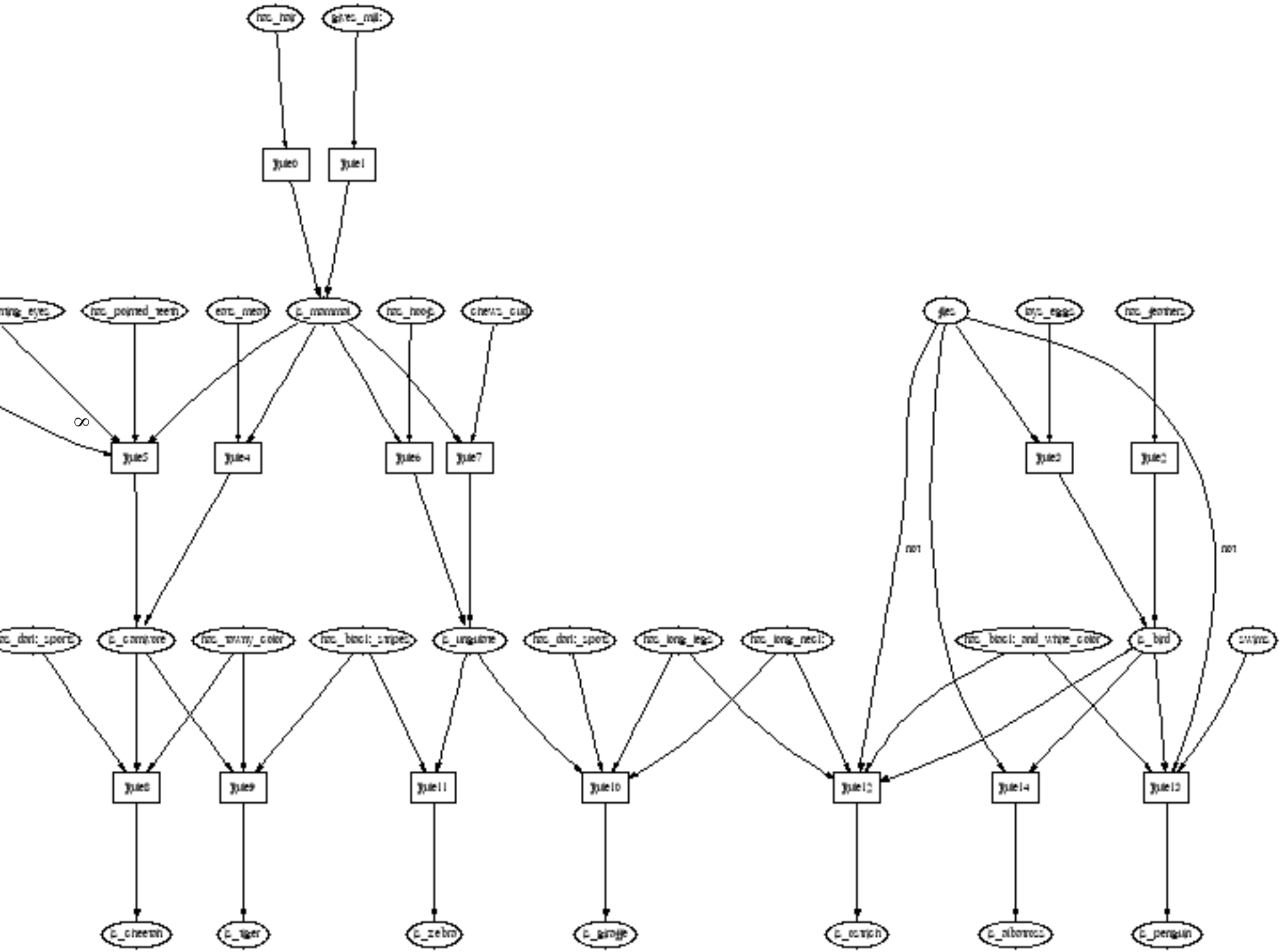
A Source code

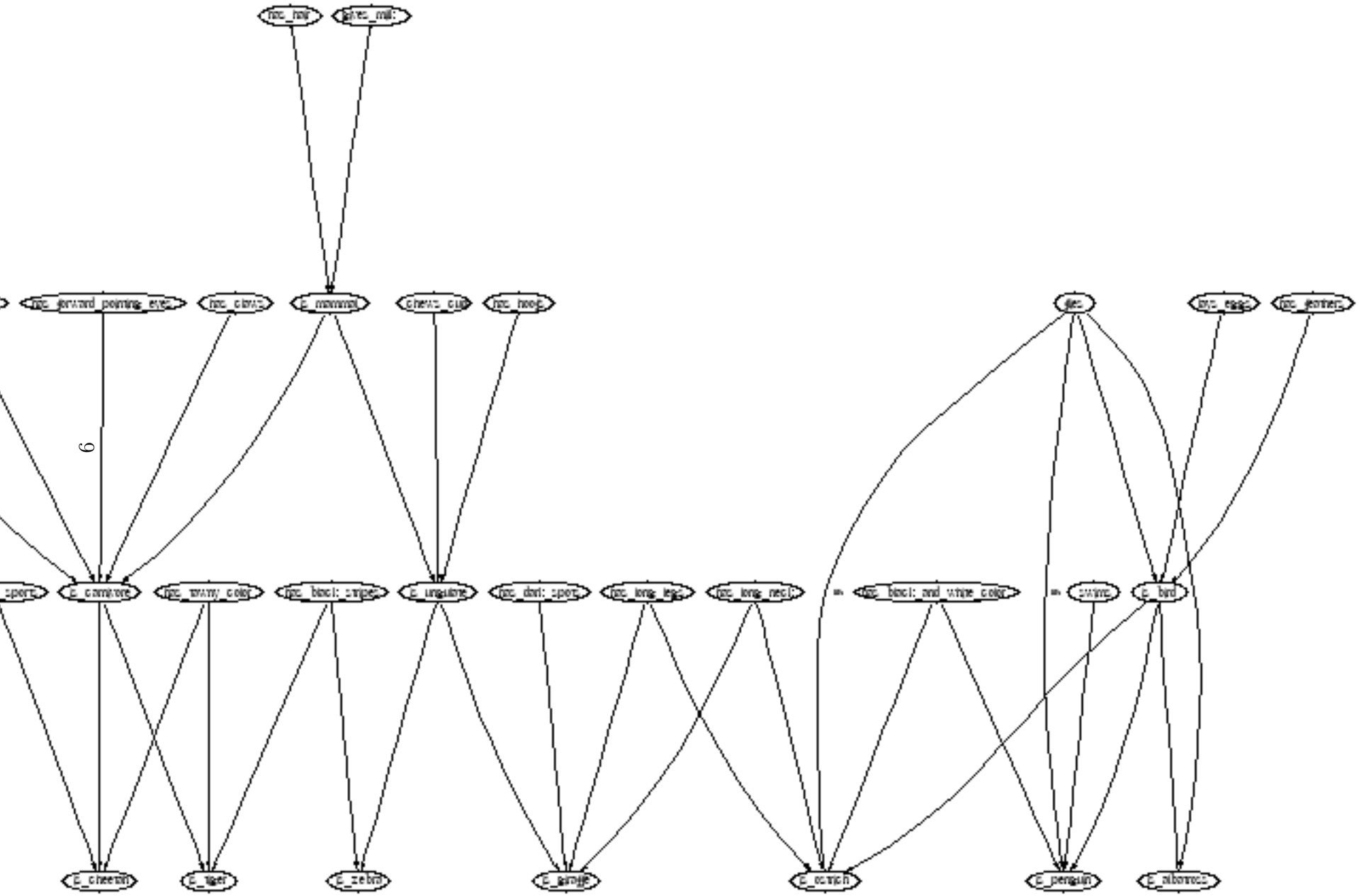
Source code is attached.

References

[1] Stephen C. North and Eleftherios Koutsofios. Application of graph visualization. In *Proceedings of Graphics Interface '94*, pages 235–245, Banff, Alberta, Canada, 1994. http://www.research.att.com/sw/tools/graphviz/download.html. 1

[2] Patrick Henry Winston. *Artificial Intelligence*. Addison Wesley, 3rd edition, 1992. 1, 2.1.1, 2.3


has_hair
gives_milk
rule0
rule1
has_pointed_teeth
eats_meat
is_mammal
has_hoofs
chews_cud
rule5
rule4
rule6
rule7
is_carnivore
has_tawny_color
has_black_stripes
is_ungulate
has_dark_spots
has_long_legs
has_long_neck
rule8
rule9
rule11
rule10
rule12
rule14
rule13
is_cheetah
is_tiger
is_zebra
is_giraffe
is_ostrich
is_albatross
is_penguin
not
not


has_hair
gives_milk
has_forward_pointing_eyes
has_claws
is_mammal
chews_cud
has_hoofs
has_eggs
has_feathers
is_carnivore
has_tawny_color
has_black_stripes
is_ungulate
has_dark_spots
has_long_legs
has_long_neck
has_black_and_white_color
swims
is_bird
is_cheetah
is_tiger
is_zebra
is_giraffe
is_ostrich
is_penguin
is_albatross

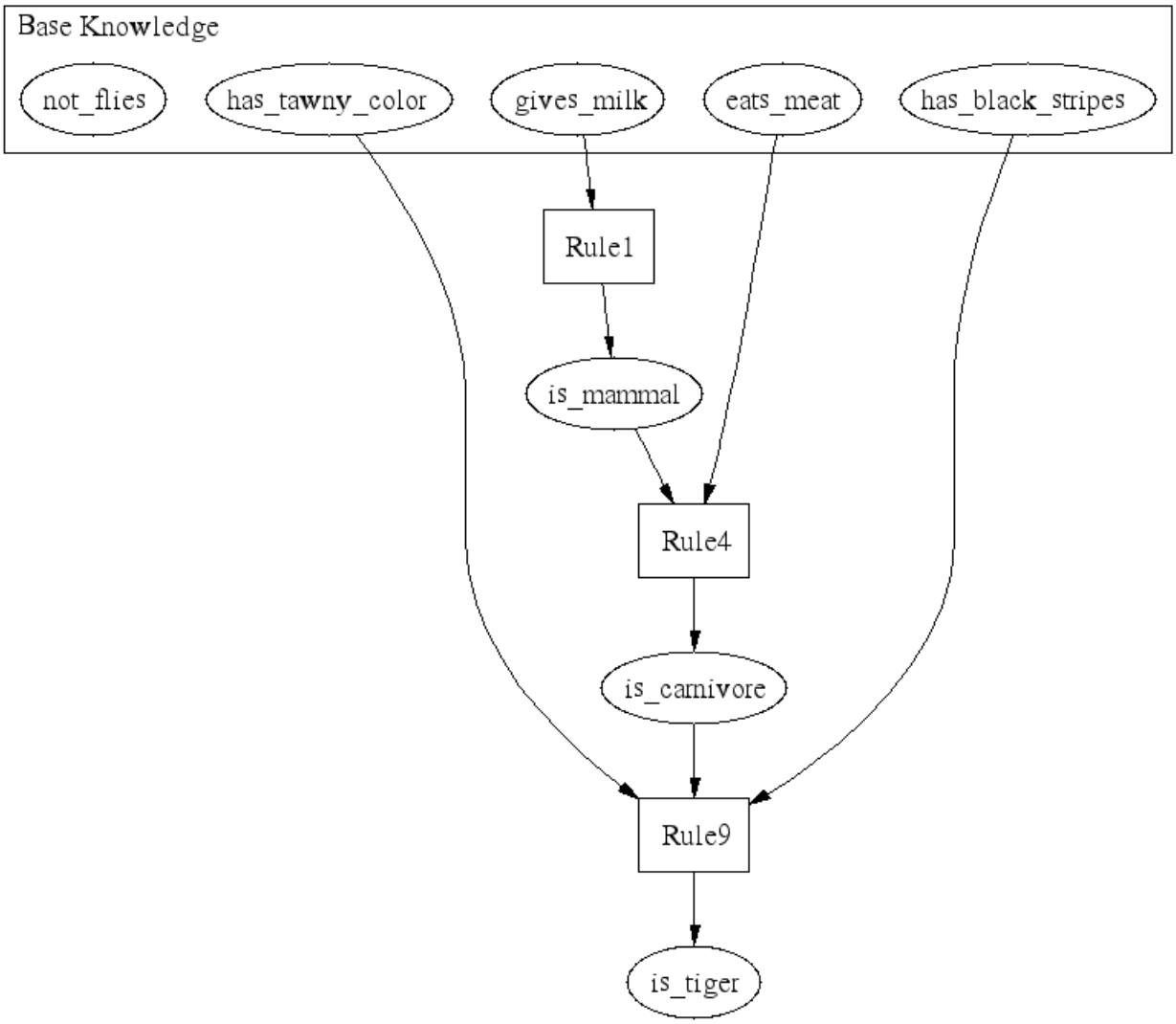


Figure 3: Tiger with rule boxes

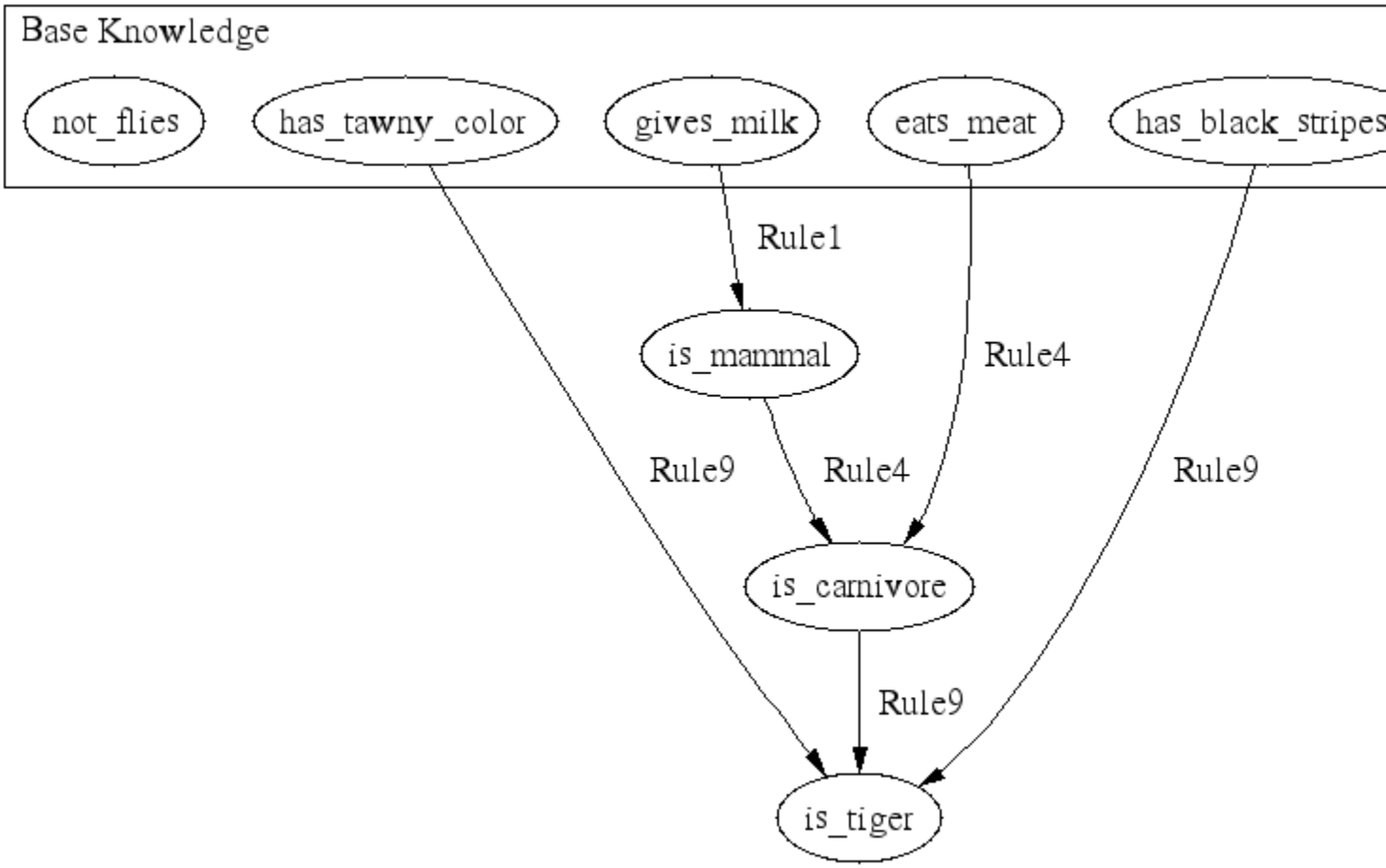


Figure 4: Tiger without rule boxes

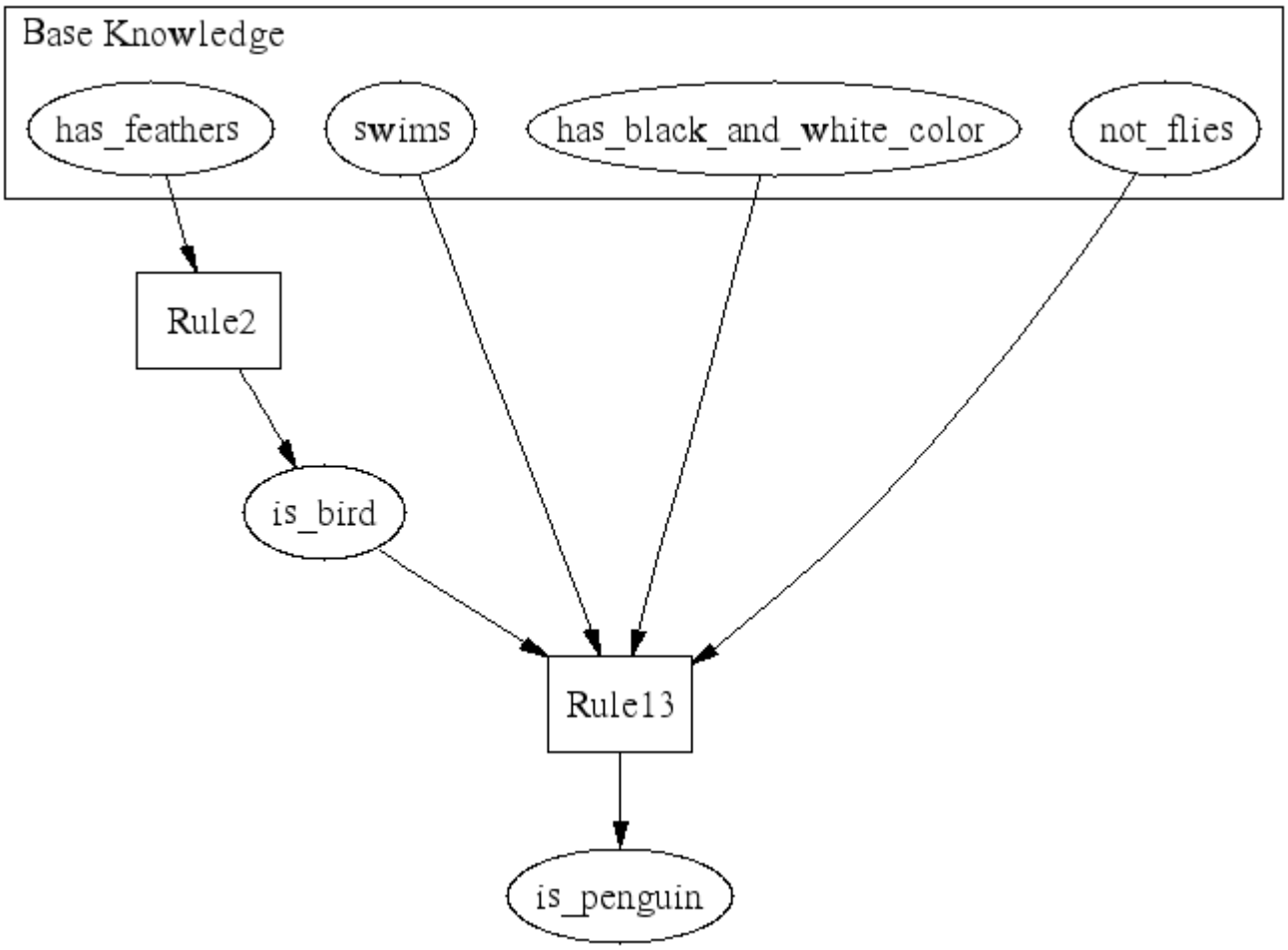


Figure 5: Tux (Penguin) with rule boxes

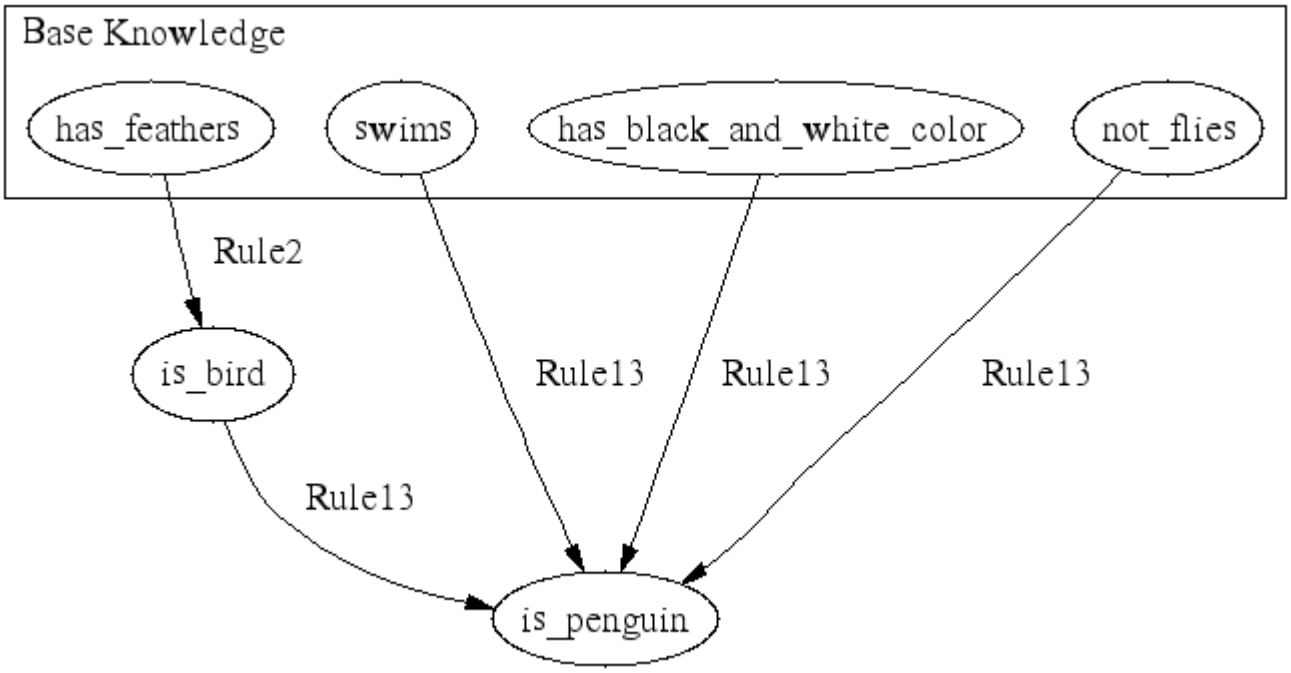


Figure 6: Tux (Penguin) without rule boxes

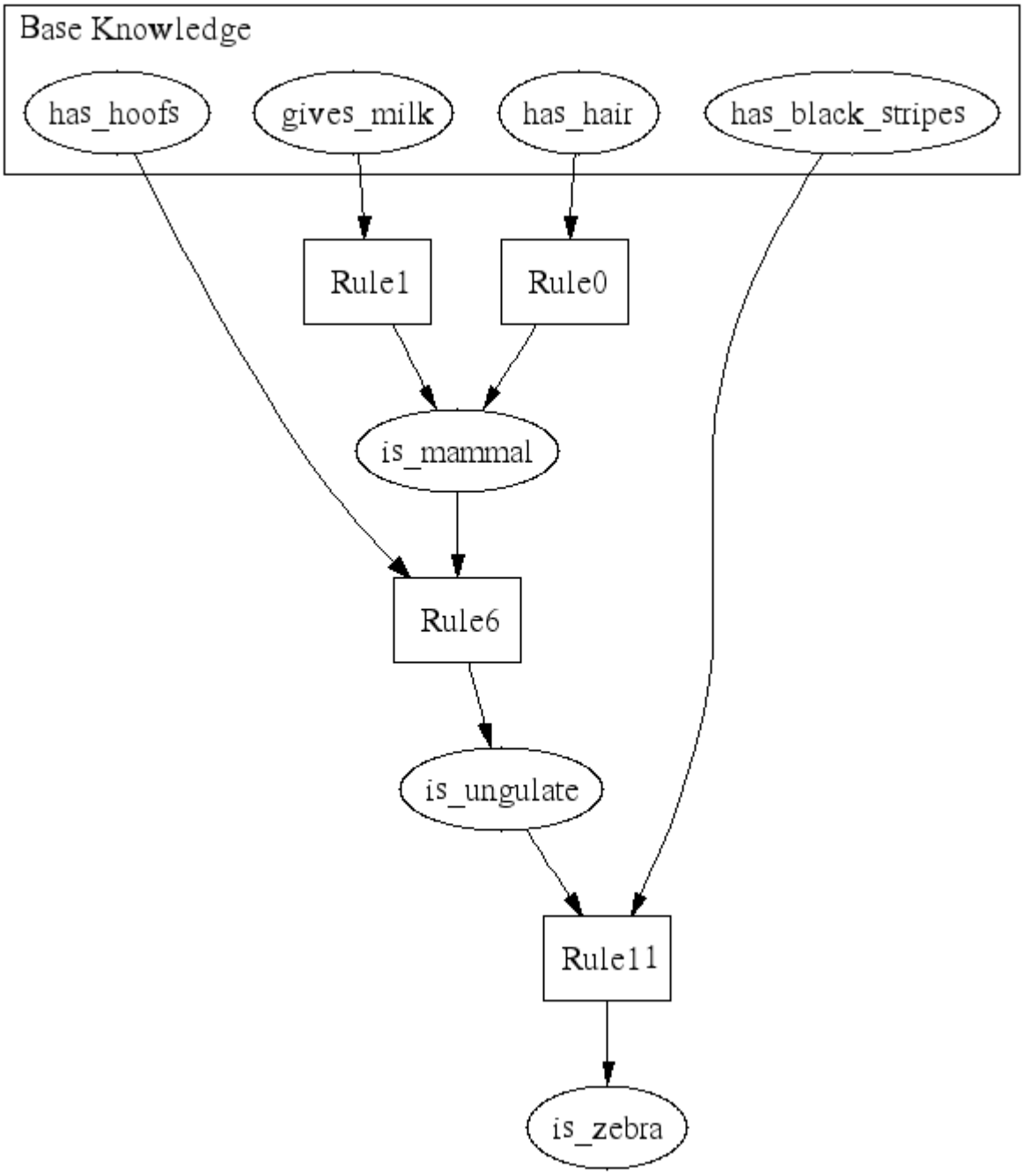


Figure 7: Zebra with rule boxes

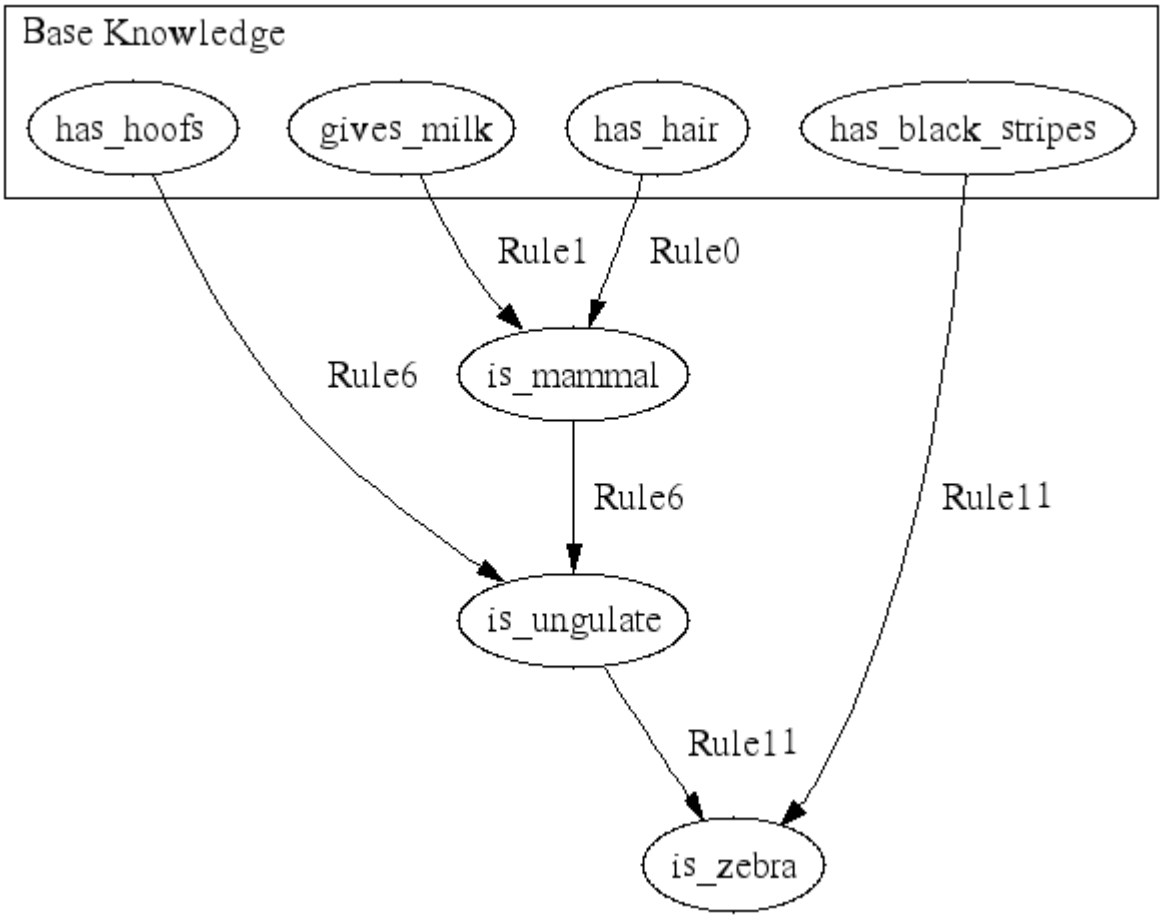


Figure 8: Zebra without rule boxes

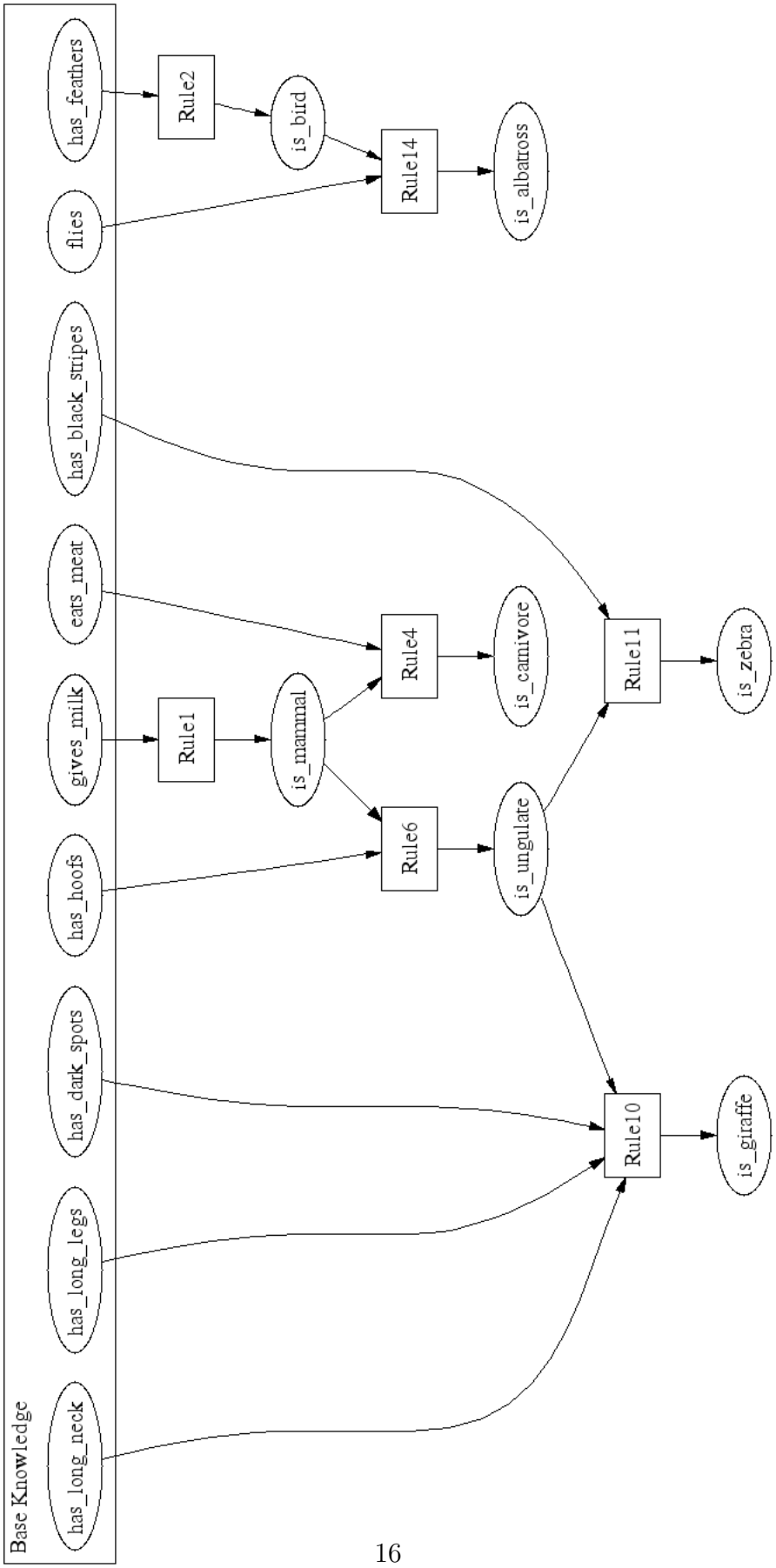


Figure 9: Nightmare creature with rule boxes

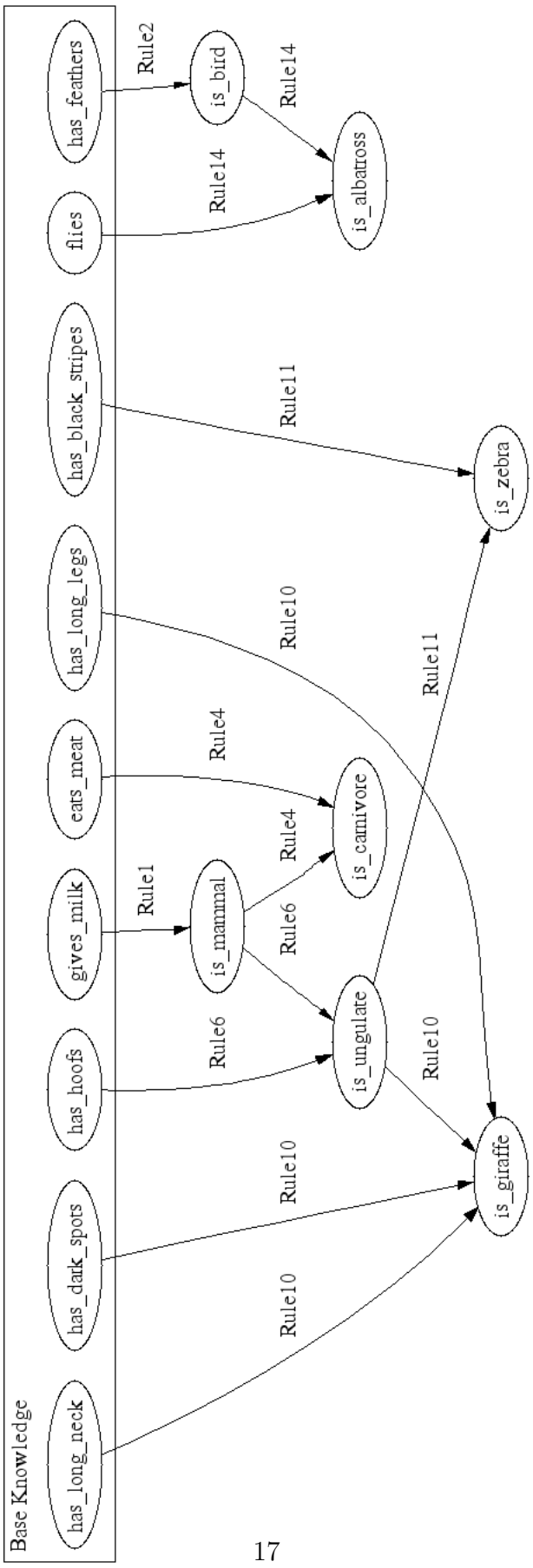


Figure 10: Nightmare creature without rule boxes