

Knowledge Representation

What is it

Attribute-Value Representation

Object-Attribute-Value Representation

Knowledge Representation in CLIPS

Semantic Networks

Inheritance

Inheritance in CLIPS

Knowledge Representation?

How can we express what we know about a problem domain - -

Using symbolic expressions in software –

With which computations can be done –

More or less efficiently

One Type of Knowledge Representation

Example -- what can be said about the knowledge representation scheme of this set of rules ?

if has_sauce = yes
and sauce_type = spicy
then wine_body = full

What does has_sauce and preferred_color represent ?

if has_sauce = no
and main_component = fish
then wine_body = light

Which objects are implicitly represented in this set of rules ?

if wine_body = light
and preferred_color = white
then wine = riesling

Which limitations does this representation have ? Give specific examples

if wine_body = full
and preferred_color = red
then wine = cabernet_sauvignon

Does this representation have any advantages ?

if wine_body = light
and preferred_color = red
then wine = zinfandel

if has_sauce = yes
and main_component = pasta
then wine = chianti

Two Basic Types of Knowledge Representation

1. Attribute – Value representation

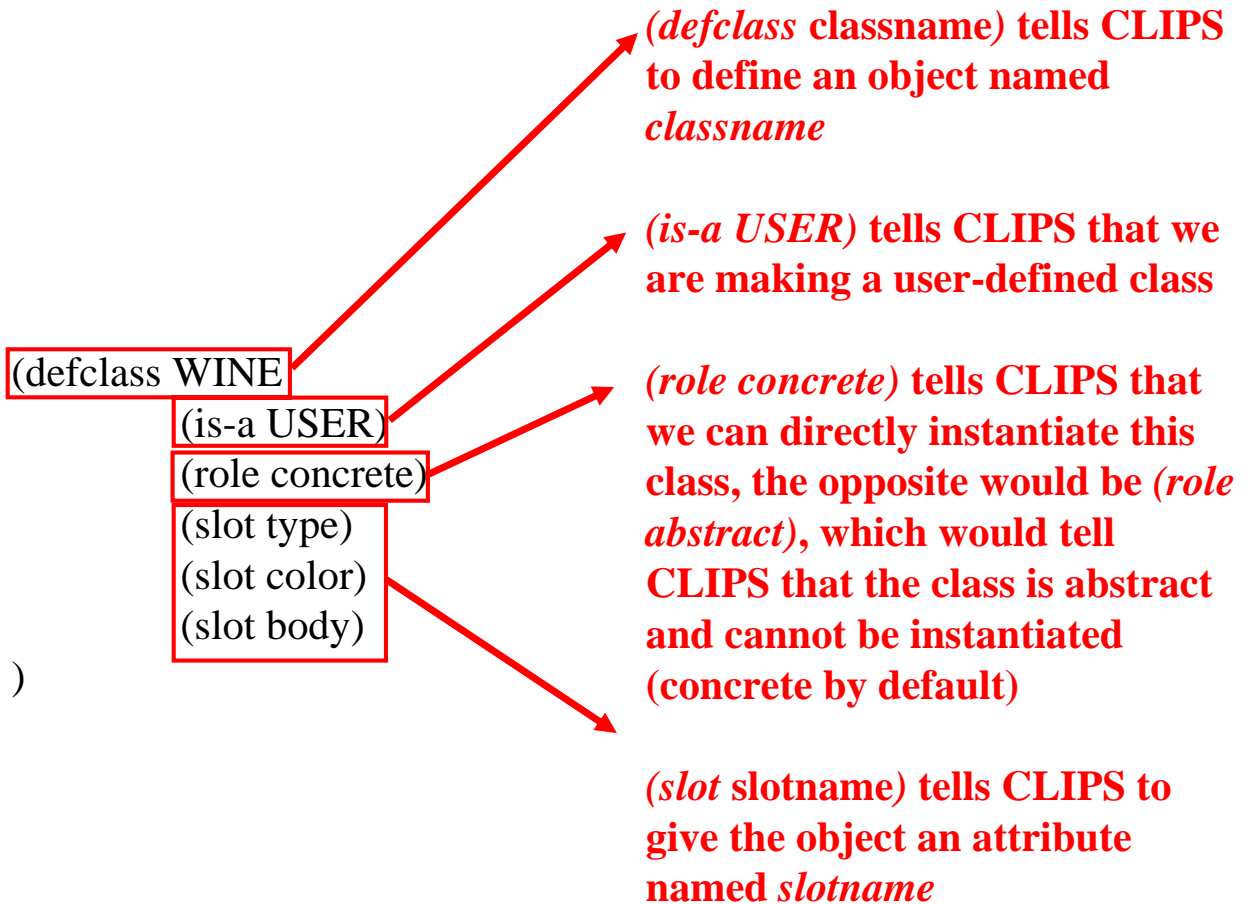
preferred_color	white, red, rose
has_sauce	yes, no

2. Object - Attribute – value representation

<u>Object</u>	<u>Attribute</u>	<u>Value</u>
wine	color	white, red, rose
	type	riesling, zinfandel cabernet-sauvignon
	body	light, full, medium
meal	has_sauce	yes, no
	sauce_type	spicy, other
	main_component	fish, poultry, beef
	suggested_wine_body	light, full, medium
person	preferred_color	white, red, rose

Rules and O-A-V Representation in CLIPS

1. Defining objects in CLIPS



```
(defclass MEAL
  (is-a USER)
  (role concrete)
  (slot main_component)
  (slot mealname)
  (slot has_sauce)
  (slot sauce_type)
  (slot suggested_wine_body) )
```

```
(defclass PERSON
  (is-a USER)
  (role concrete)
  (slot firstname)
  (slot preferred_color) )
```

Rules and O-A-V Representation in CLIPS

2. Making instances of objects in CLIPS

a. From the CLIPS command line:

(make-instance mywine of WINE (color white) (body light))

*** Note: Any instances defined in this manner will be deleted when CLIPS is reset or cleared.**

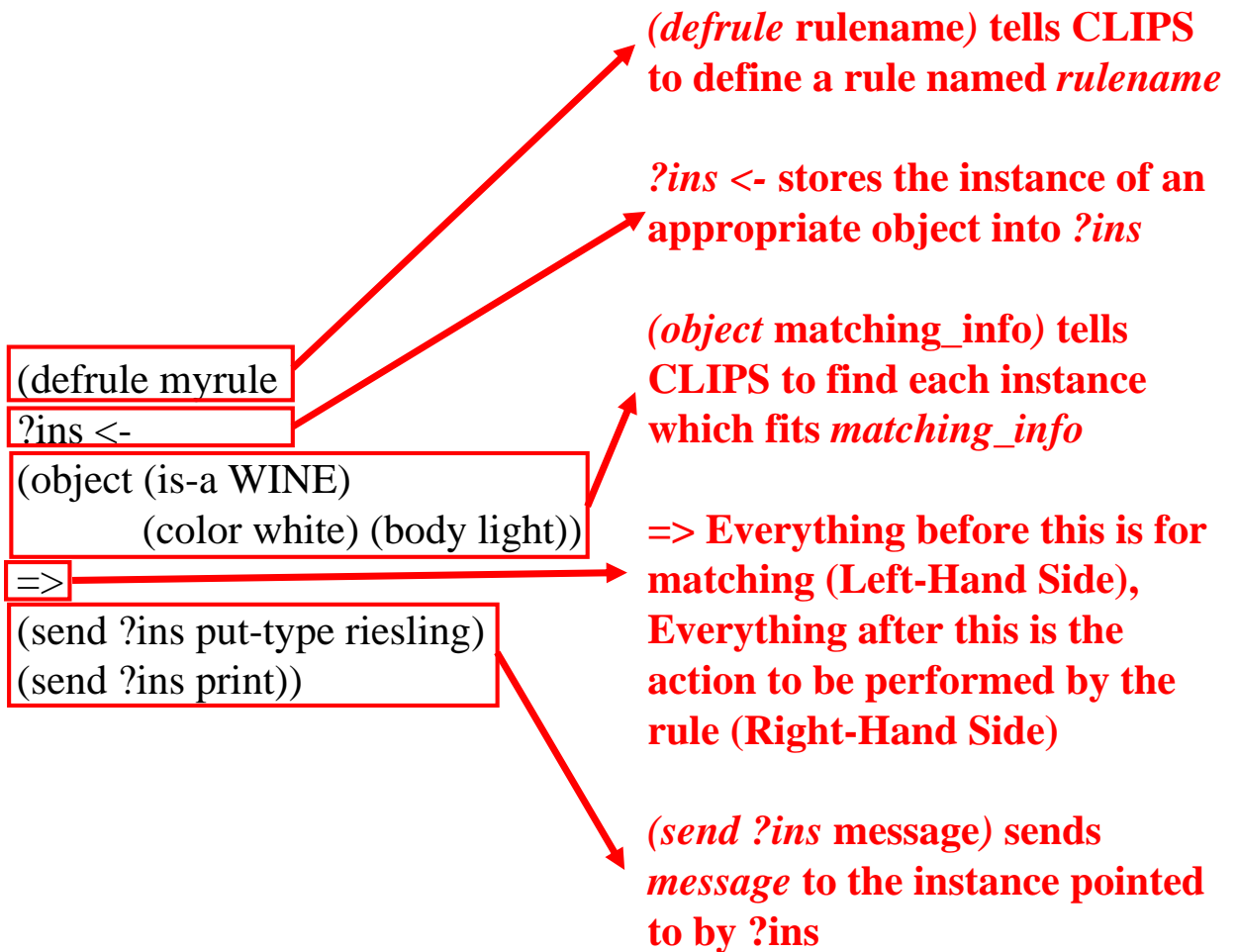
b. In a definition file to be loaded into CLIPS:

```
(definstances WINE-INSTANCES
  (firstwine of WINE (color white) (body light))
  (secondwine of WINE (color red) (body full))
  (thirdwine of WINE (type chianti))
  (fourthwine of WINE (color red) (body light))
)
```

*** Note: You must reset CLIPS for the instances in a definstances command to take effect after the definitions file is loaded.**

Rules and O-A-V Representation in CLIPS

3. Defining Rules that work on objects in CLIPS



Rules and O-A-V Representation in CLIPS - Example

```
(defrule suggest_full_body
?ins <- (object (is-a MEAL) (has_sauce yes) (sauce_type spicy))
=>
(send ?ins put-suggested_wine_body full) )
```

```
(defrule suggest_light_body
?ins <- (object (is-a MEAL) (has_sauce no) (main_component fish))
=>
(send ?ins put-suggested_wine_body light) )
```

```
(defrule is_riesling
?ins <- (object (is-a WINE) (color white) (body light))
=>
(send ?ins put-type riesling) )
```

```
(defrule is_cabernet_sauvignon
?ins <- (object (is-a WINE) (color red) (body full))
=>
(send ?ins put-type cabernet_sauvignon) )
```

```
(defrule is_zinfandel
?ins <- (object (is-a WINE) (color red) (body light))
=>
(send ?ins put-type zinfandel) )
```

Rules and O-A-V Representation in CLIPS - Example

```
(definstances WINE-INSTANCES
  (firstwine of WINE (color white) (body light))
  (secondwine of WINE (color red) (body full))
  (thirdwine of WINE (type chianti))
  (fourthwine of WINE (color red) (body light))
)

(definstances MEAL-INSTANCES
  (firstmeal of MEAL (has_sauce yes) (sauce_type spicy)
  (mealname thai_chicken))
)

(definstances PERSON-INSTANCES
  (john of PERSON (preferred_color red) (firstname john))
)

(defrule choose_wine (declare (salience -10))
  (object (is-a MEAL) (suggested_wine_body ?swb) (mealname ?mn))
  (object (is-a PERSON) (preferred_color ?pc) (firstname ?fn))
  (object (is-a WINE) (color ?pc) (body ?swb) (type ?t))
=>
  (printout t "If " ?fn " chooses " ?mn " then he should take " ?t " for
  his wine selection." crlf)
)
```

Objects Also Need Procedures

Attributes represent “declarative” knowledge –

“WHAT” you know about objects

type of wine

type of sauce

main component of meal

Some knowledge about objects is “procedural” –

“HOW” an object does something

gas mileage for a car for a particular trip

how a robot moves

Example

In CLIPS, procedures are implemented using message-handlers, and called by sending a message to the object. (send [instance] print) and (send [instance] put-type riesling) as used in the example several slides ago were examples of messages, and the results came from the corresponding message-handlers. CLIPS assigns some default message-handlers, and these were used in that prior example. In this example, we will define our own procedure by defining our own message-handler.

```
(defclass ANIMAL
  (is-a USER)
  (slot myname)
  (slot skin_covering)
  (slot step_length)
  (slot step_frequency)
)

(defmessage-handler ANIMAL speed ()
  (* ?self:step_length ?self:step_frequency) )
```

defmessage-handler is the CLIPS function which assigns a message handler to a specific object

ANIMAL is the object to which we're assigning the message-handler

speed is the message we will be handling

() would be used for handling arguments to the message

(* ...) is the procedure we're performing, in this case multiplication

?self is a reserved variable which refers to the specific instance of the object to which this message is being passed

Example

To call this procedure, we would send the message “speed” to an instance of the object ANIMAL. Below we will create an instance of ANIMAL called [a] and have it run the procedure “speed”.

```
CLIPS> (make-instance a of ANIMAL (step_length 10)
(step_frequency 2))
```

[a]

```
CLIPS>(send [a] speed)
```

20

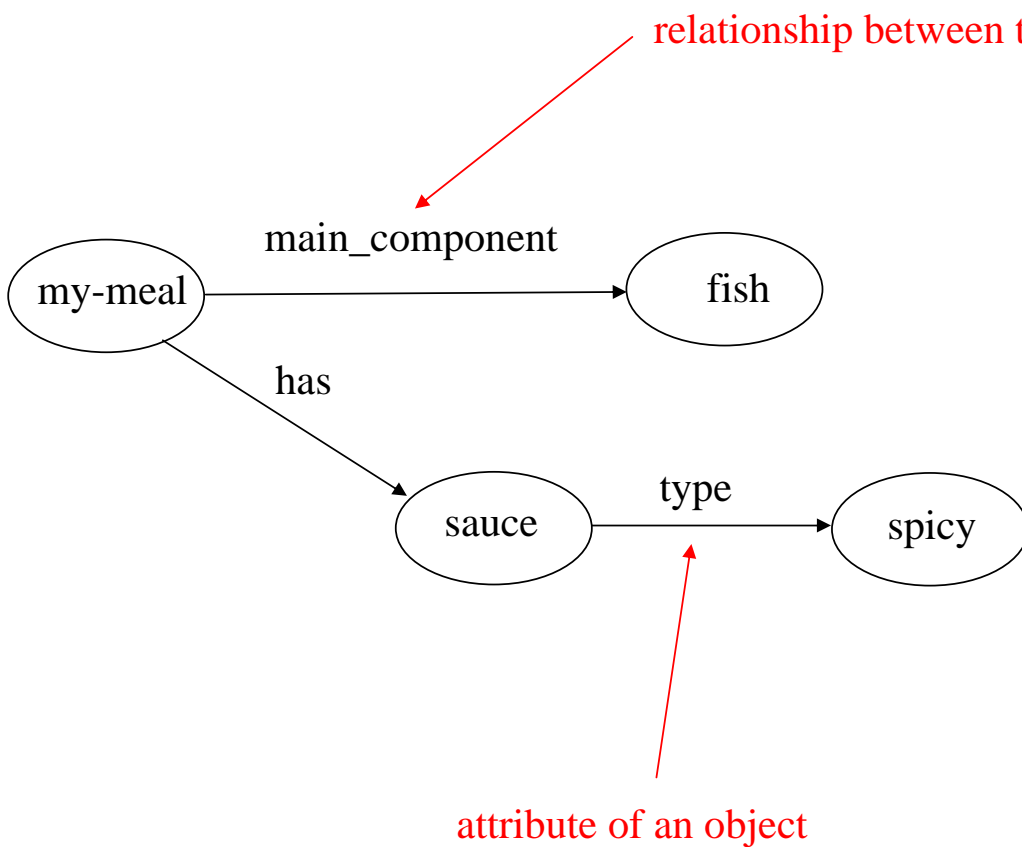
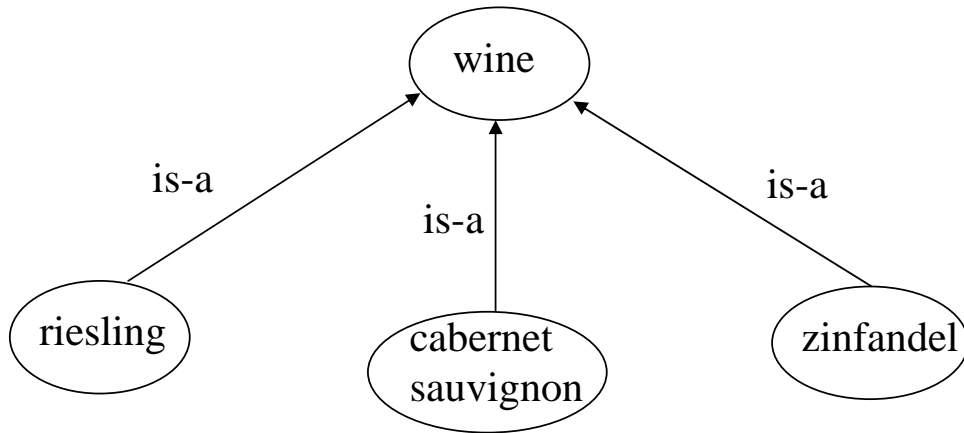
Example

The rule “veryfast” is based on the procedure that we defined in the last slide called “speed”. (test ...) is a command which tests the boolean statement inside it and returns true or false.

```
(defrule veryfast
  ?ins <- (object (is-a ANIMAL) (myname ?mn))
  (test (> (send ?ins speed) 100))
  =>
  (printout t ?mn " is a very fast animal!" crlf)
)
```

Semantic Networks

Represent knowledge as a directed graph



Semantic Networks vs Predicate Logic

Predicate Logic

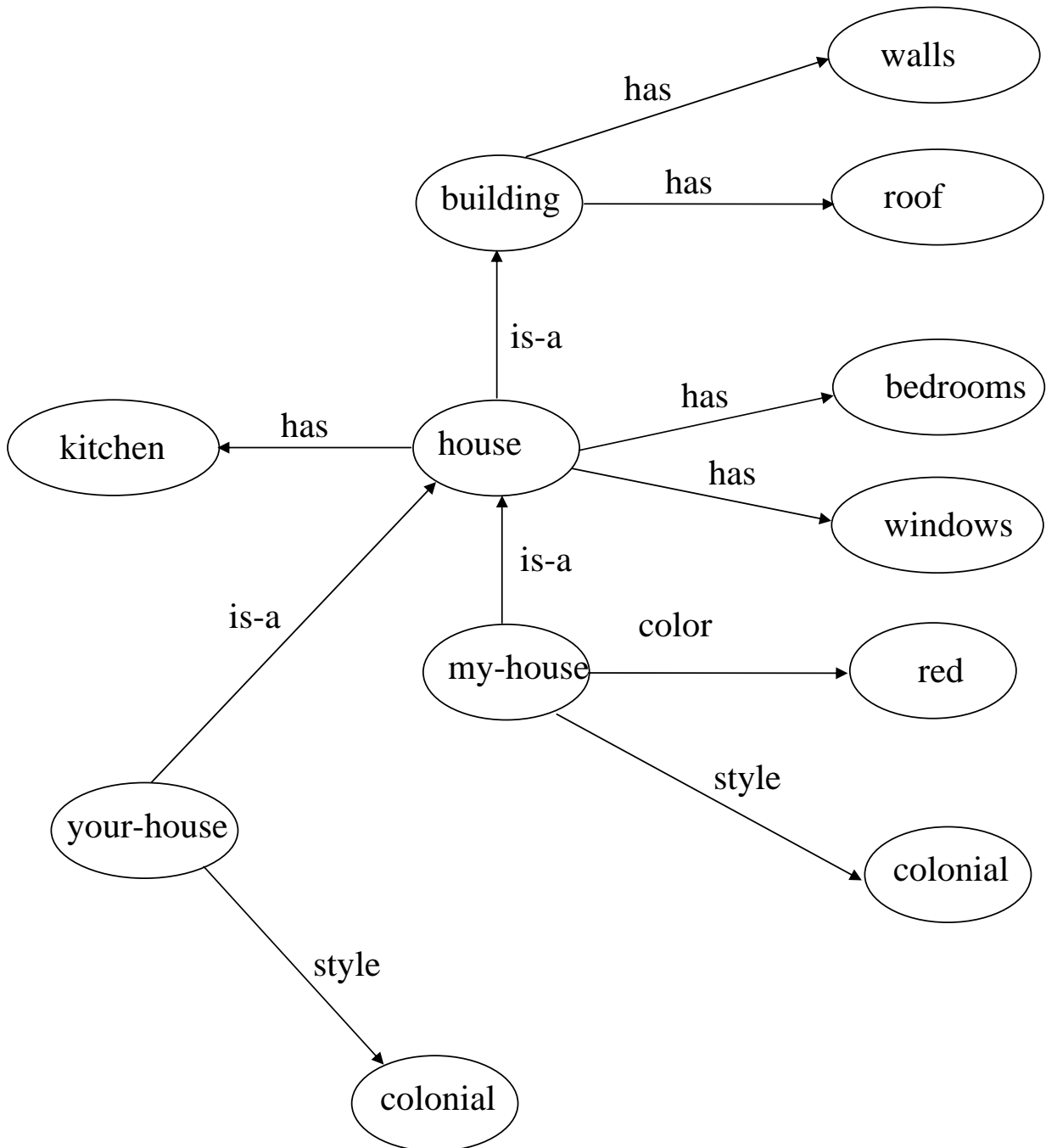
Main goal is to provide ways to make correct inferences

Semantic Networks

Main goal is to investigate meaning

Origins of Semantic Networks

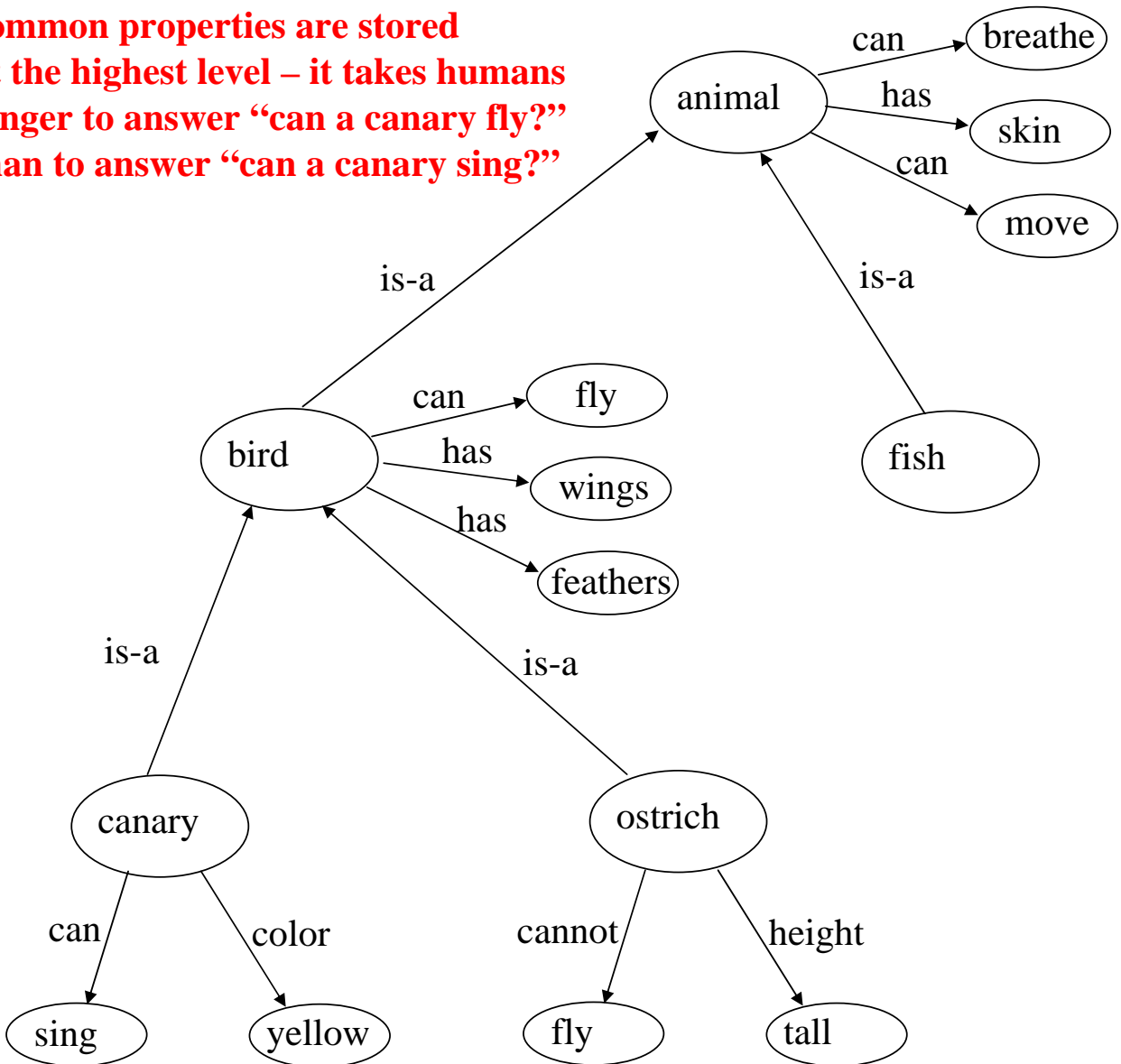
An attempt to capture meaning – meaning of a symbol is captured by the association of that symbol with other symbols



Hierarchical Organization in Semantic Networks

It seems that humans organize knowledge hierarchically

common properties are stored at the highest level – it takes humans longer to answer “can a canary fly?” than to answer “can a canary sing?”



Exceptions are stored at the level of the specific node – it took humans longer to answer “can an ostrich breathe?” than to answer “can an ostrich fly?”

Advantages of Hierarchical Organizations

If knowledge is simply organized as a network of relationships then a semantic network is equivalent to a predicate calculus style of knowledge representation

The power of semantic network representations comes from the inference rules that can be associated with the different types of links

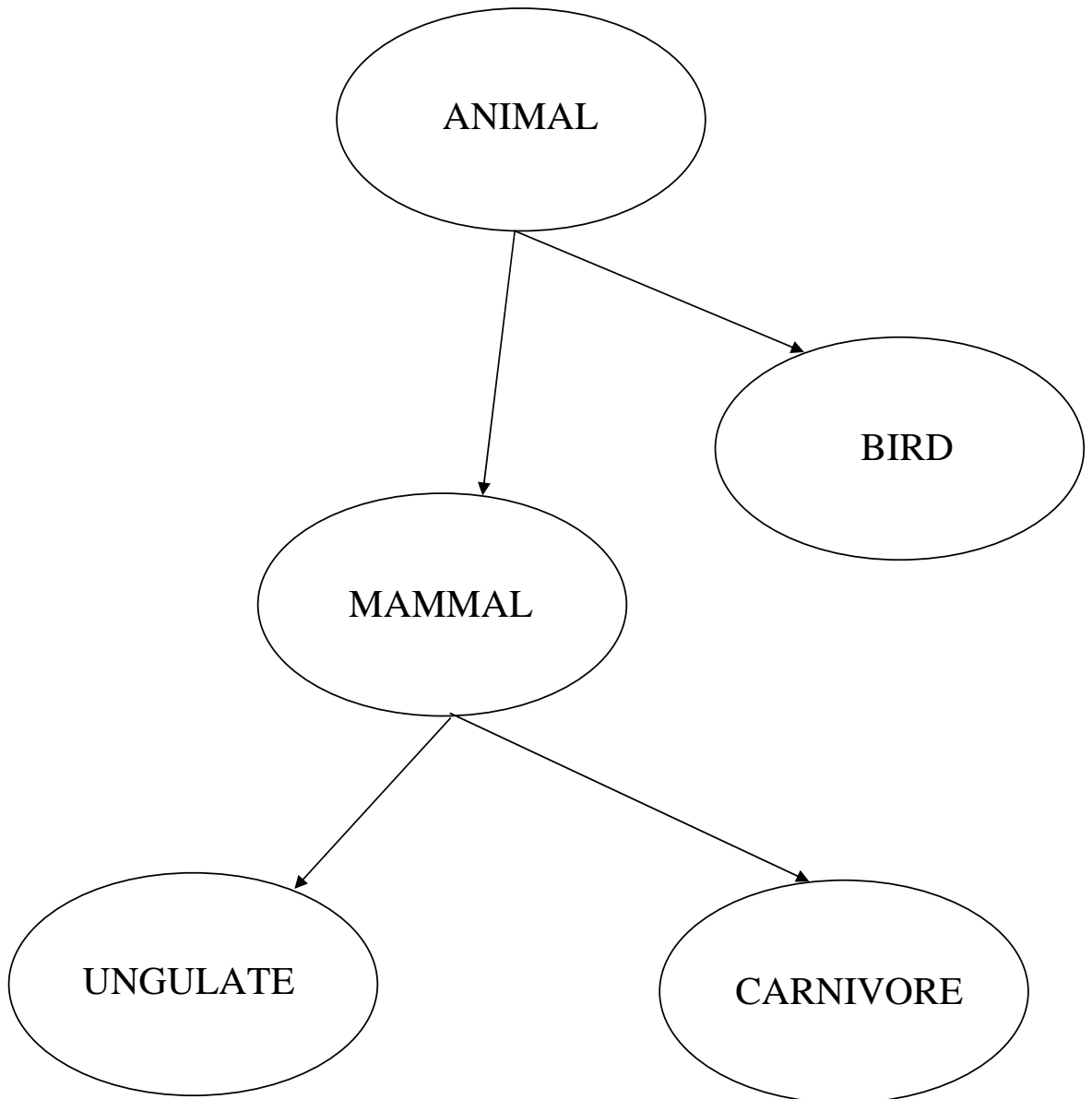
Inference rules associated with “is-a” links provide a lower level node with properties of higher level nodes

Inference rules associated with “default” links provide properties in the absence of other information

Hierarchies ensure that all members of a class have the same properties and mistakes during updates are reduced

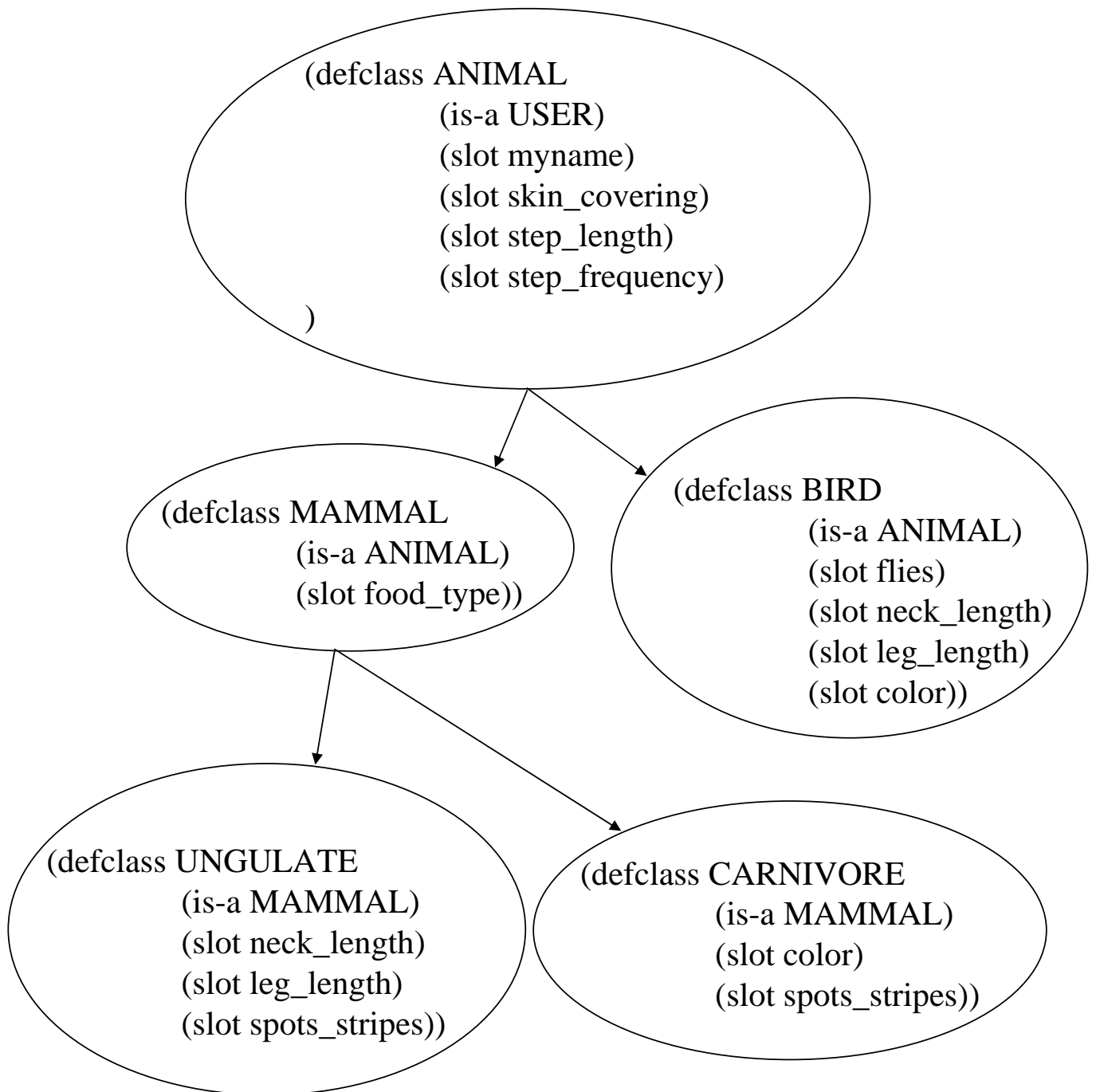
The size of the knowledge base is reduced since properties are stored once at the highest levels of the hierarchy

Inheritance



Each of the children in this hierarchy inherit all of the attributes and procedures of the parents linked to it from above.

Inheritance Example



This command still applies to the ANIMAL object, and all objects that inherit from ANIMAL will inherit this procedure:

```
(defmessage-handler ANIMAL speed primary ()
  (* ?self:step_length ?self:step_frequency) )
```