

Discussion 8 Notes
 CSE 150
 Feb. 27th, 2004
 Prepared by: Anjum Gupta

We will go over an example narrative in the discussion today. You may find that narrative has some similarities to the assignment 3 but it is very different overall. For example, in our narrative, Piotr is a raccoon not a rabbit; also Peter's name is spelled as Piotr.

Piotr, the raccoon, was hungry. He took a basket, went to the store, and bought some bread. He spent all his money but came back with a full basket of bread. Piotr ate the bread and then went to play outside in Mr. McBurger's farm.

First we will lay out the contents in a formal way. You should put similar tables in your report. The otter file printed at the end of the notes, has a suggested structure for laying out different parts of the file. You should also consider splitting the file into different files and using include statements as suggested by Kristin in her grading guidelines. The structure will help you keep your code organized. It will also be easy to debug and modify. Please notice the commented headings in the file to get the feel for the structure you should follow.

If you have another structure or any other way to solve unique names problem, you are welcome to use it. Just make sure that it works.

Constants

Constant to be encoded	Encoded in Otter as
Piotr	Piotr
Bread	Bread
Basket	Basket
Mr. McBurger' farm	MrBurger
BreadStore	Store
Home	Home

Fluents

Fluent used	Meaning
IsHungry(X)	X is hungry
HasMoney(X)	X has money
HasBasket(X)	X has basket
BasketFull(Y)	Basket is full of Y's
At(X, Y)	X is at location Y
IsRaccoon(X)	X is a raccoon

Actions

Action	Meaning
Eating(X, Y)	X is eating Y
Buying(X, Y)	X is buying Y
Went(X, Y)	X goes to location Y
TookBasket(X)	X took a basket.
Playing(X)	X plays

List out all the assumptions that you are making for your implementation, like:

Piotr ate all the bread.

Playing makes you hungry again.

In your report, you will also create the following type of tables for any of the special functions you encode. Make sure you specify if you are using predicates like “actual(x)” or “state(x)” and how you are encoding them.

Causes(action, state, fluent)

Action	Fluents	State or any other condition	Comment
Buying(u,v)	BasketFull(v)	In all states	Buying v fill the basket with v.
Playing(u)	IsHungry(u)	In all states	Playing makes one hungry

Cancel(action, state, fluent)

Action	Fluents	Conditions	Comment
Eating(u,v)	BasketFull(v)	In all states	Eating empties the basket.
Eating(u,v)	IsHungry(u)	In all states	Eating makes you Not Hungry.

Following is the code from the otter file. I edited some of the file while making the notes in the MS word, but a very similar version works in otter. As discussed in the discussion section, there are many things that can be done differently from they way they are implemented in this file. Use this file as you see fit. Please feel free to suggest optimizations or any other modifications to this code. I will try your suggested improvements and will put a response on the boards.

```
set(auto).
```

```
formula_list(usable).
```

```
%%% Define "=" %%%%
all x (x=x).
```

```
%%% Define "<" %%%%
all x y ( (x<y) -> (x != y) ).
all x y z ( ((x<y) & (y<z) ) -> (x<z) ).
```

```
%%%Define Causes and Cancels and holds %%%%
all x all y all z u v (Causes(x,y,z) <->
  (x = Buying(u, v) & z = BasketFull(v) )
  | (x = Playing(u) & z = IsHungry(u) )
  | (x = Went(u, v) & z = At(u,v) )
  | (x = TookBasket(u) & z = HasBasket(u) )
).
```

```
all x all y all z u v (Cancels(x,y,z) <-> ((x = Eating(u,v) & z = BasketFull(v) )
  | (x = Buying(u,v) & z = HasMoney(u)) ) ).
```

```
all x all y all z (holds(z,do(x,y)) <-> ( Causes(x,y,z) | ( holds(z,y) & -
Cancels(x,y,z) ) ) ).
```

%%% define the initial state or any other assumptions and facts %%%%

```
holds(IsHungry(Piotr),s0).
IsRaccoon(Piotr).
-holds(HasBasket(Piotr),s0).
-holds(BasketFull(Bread),s0).
holds(At(Piotr,Home),s0).
```

%%% lay out the state transitions and any other events %%%%

```
s1 = do(TookBasket(Piotr),s0).
s2 = do(Went(Piotr,Store),s1).
s3 = do(Buying(Piotr,Bread),s2).
s4 = do(Went(Piotr,Home),s3).
s5 = do(Eating(Piotr,Bread),s4).
s6 = do(Went(Piotr,MrBurger),s5).
s7 = do(Playing(Piotr),s6).
```

%Encode the queries.

```
%Is basket full in s2. should be false. (just got to the shop)
%holds(BasketFull(Bread),s2).
```

%%% Encode that all functions/fluent return different outputs for different inputs

```
all x all y u v (Eating(x,y) = Eating(u,v) <-> ( x = u ) & ( y = v ) ).
all x all y all u all v (Buying(x,y) = Buying(u,v) <-> ( ( x = u ) & ( y = v ) ) ).
all x all y all u all v (Went(x,y) = Went(u,v) <-> ( ( x = u ) & ( y = v ) ) ).
all x y u v ( ( do(x,y) = do(u,v) ) <-> ( (x=u) & (y=v) ) ).
all x y z u v w ( ( Causes(x,y,z) = Causes(u,v,w) ) <-> ( (x=u) & (y=v) & (z=w) ) ).
all x y u v ( ( holds(x,y) = holds(u,v) ) <-> ( (x=u) & (y=v) ) ).
```

```
all x y u v ( ( At(x,y) = At(u,v) ) <-> ( (x=u) & (y=v) ) ).
all x y ( ( IsHungry(x) = IsHungry(y) ) <-> ( x = y ) ).
all x y ( ( HasMoney(x) = HasMoney(y) ) <-> ( x = y ) ).
all x y ( ( HasBasket(x) = HasBasket(y) ) <-> ( x = y ) ).
all x y ( ( BasketFull(x) = BasketFull(y) ) <-> ( x = y ) ).
```

%%% All the constant symbols are different from each other %%%%

```
s0 < s1.
s1 < s2.
s2 < s3.
s3 < s4.
s4 < s5.
s5 < s6.
s6 < s7.
s7 < Piotr.
Piotr < Bread.
Bread < Basket.
Basket < MrBurger.
MrBurger < Store.
Home < Store.
```

%%% All the functions and fluent names are different from each other. %%%%

```
all u all v (Buying(u,v) < Eating(u,v)).
all u v (Eating(u,v) < Went(u,v)).
all u v z (Went(u,v) < Cancels(u,v,z)).
all u v z (Cancels(u,v,z) < Causes(u,v,z)).
all u v z (Causes(u,v,z) < holds(u,v)).
all u v ( holds(u,v) < do(u,v)).
all u v ( do(u,v) < IsHungry(u) ).
```

```
all u (IsHungry(u) < BasketFull(u) ).  
all u (BasketFull(u) < HasMoney(u) ).  
all u (HasMoney(u) < HasBasket(u) ).  
all u v (HasBasket(u) < At(u,v) ).  
all u v (At(u,v) < TookBasket(u)).
```

```
end_of_list.
```