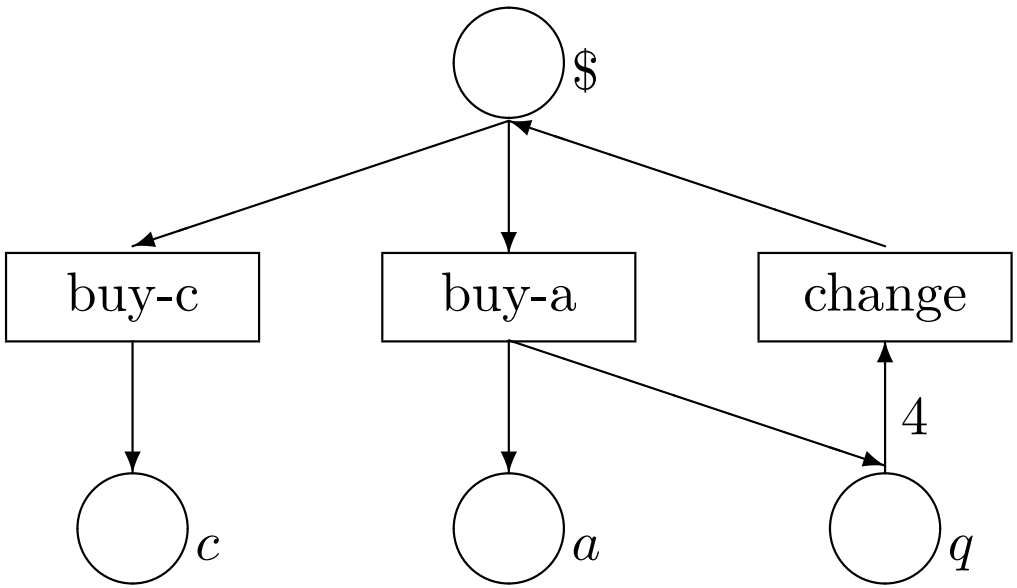
Petri Nets

So far so good, but we have not yet seen any concurrency. The simplest concurrent system examples are probably the concurrent automata called Petri nets. Consider for example the picture,



Petri Nets (II)

The previous picture represents a concurrent machine to buy cakes and apples; a cake costs a dollar and an apple three quarters.

Due to an unfortunate design, the machine only accepts dollars, and it returns a quarter when the user buys an apple; to alleviate in part this problem, the machine can change four quarters into a dollar.

The machine is concurrent, because we can push several buttons at once, provided enough resources exist in the corresponding slots, which are called places

Petri Nets (III)

For example, if we have one dollar in the \$ place, and four quarters in the q place, we can simultaneously push the buy-a and change buttons, and the machine returns, also simultaneously, one dollar in \$, one apple in a, and one quarter in q.

That is, we can achieve the concurrent computation,

$$\text{buy-a change} \ : \ \$\ q\ q\ q\ q \longrightarrow a\ q\ \$.$$

Petri Nets (IV)

This has a straightforward expression as a rewrite theory (system module) as follows:

```
mod PETRI-MACHINE is
  sort Marking .
  ops null $ c a q : -> Marking .
  op _ _ : Marking Marking -> Marking [assoc comm id: null] .
  rl [buy-c] : $ => c .
  rl [buy-a] : $ => a q .
  rl [chng] : q q q q => $ .
endm
```

Petri Nets (V)

That is, we view the distributed state of the system as a multiset of places, called a marking, with identity for multiset union the empty multiset `null`.

We then view a transition as a rewite rule from one (pre-)marking to another (post-)marking.

Petri Nets (VI)

The rewrite rule can be applied modulo associativity, commutativity and identity to the distributed state iff its pre-marking is a submultiset of that state.

Furthermore, if the distributed state contains the union of several such presets, then several transitions can fire concurrently.

For example, from `$ $ $` we can get in one concurrent step to `c c a q` by pushing twice (concurrently!) the `buy-c` button and once the `buy-a` button.

Petri Nets (VII)

We can of course ask and get answers to questions about the behaviors possible in this system. For example, if I have a dollar and three quarters, can I get a cake and an apple?

```
Maude> search $ q q q =>+ c a M:Marking .
search in PETRI-MACHINE : $ q q q =>+ c a M:Marking .

Solution 1 (state 4)
states: 5 in 0ms cpu (0ms real)
M:Marking --> null
```

we can also interrogate the search graph,

Petri Nets (VIII)

```
Maude> show search graph .
state 0, Marking: $ q q q
arc 0 ===> state 1 (rl [buy-c]: $ => c .)
arc 1 ===> state 2 (rl [buy-a]: $ => a q .)

state 1, Marking: c q q q

state 2, Marking: a q q q q
arc 0 ===> state 3 (rl [chng]: q q q q => $ .)

state 3, Marking: $ a
arc 0 ===> state 4 (rl [buy-c]: $ => c .)
arc 1 ===> state 5 (rl [buy-a]: $ => a q .)

state 4, Marking: c a

state 5, Marking: a a q
```

Petri Nets (IX)

```
Maude> show path 4 .
state 0, Marking: $ q q q
===[ rl [buy-a]: $ => a q . ]===>
state 2, Marking: a q q q q
===[ rl [chng]: q q q q => $ . ]===>
state 3, Marking: $ a
===[ rl [buy-c]: $ => c . ]===>
state 4, Marking: c a
```

What is Concurrency?

Why was concurrency impossible in our `CANDY-AUTOMATON` example, but possible in our little `PETRI-MACHINE` example?

The problem with `CANDY-AUTOMATON`, and with any LTS having unstructured states, is that its states are atomic, and, having no smaller pieces, cannot be distributed.

By contrast, a Petri net marking is made out of smaller pieces, namely its constituent places, and therefore can be distributed, so that several transitions can happen simultaneously.

What is Concurrency? (II)

Then what, is concurrency about multisets?

Not necessarily; this is the very common fallacy of taking the part for the whole; for example, "Logic Programming = Prolog," or "Concurrency = Petri Nets".

A more fair and open-minded answer is to give the rewriting logic motto:

Concurrent Structure = Algebraic Structure.

What is Concurrency? (III)

That is, any algebraic structure in the set of states, other than atomic constants, even a single unary operator, will open the possibility for the states to be distributed, and therefore for transitions being concurrent.

Of course that potential for concurrency may be frustrated by the specific transitions of a system forcing a sequential execution, but the potential is there if we use other transitions.

In summary, there are as many possible styles of concurrent systems as there are signatures Σ and equations E. For example: multiset concurrency, tree concurrency, string concurrency, and many, many other possibilities.

Petri Nets in General

I give the Meseguer-Montanari "Petri nets are monoids" definition, instead than the usual, but less enlightening, multigraph definition.

A place-transition Petri net N consists of:

- a set P of places; we then call markings to the elements in the free commutative monoid $M(P)$ of finite multisets of P.
- a labeled transition system $N = (M(P), L, T)$.

Petri Nets in General (II)

The general transformation associating a rewrite theory $R(N)$ to each Petri net N is then obvious. $R(N)$ has:

- a single sort, named, say $M(P)$, or just Marking, with constants the elements of P and a null constant.

- a binary operator
  $\_\,\_ : \text{Marking Marking} \longrightarrow \text{Marking}\ [assoc\ comm\ id:\ null]$

- for each $(m, l, m') \in T$ a rewrite rule $l : m \longrightarrow m'$.

Petri Net Computations

The computations of a net N are not just paths, since we can now take several concurrent steps at once. They are generated as follows:

- Reflexivity.

$$\frac{m \in M(P)}{m \xrightarrow{m} m}$$

- Basic Transition.

$$\frac{(m, l, m') \in T}{m \xrightarrow{l} m'}$$

- Congruence.

$$\frac{m \xrightarrow{\alpha} m' \qquad u \xrightarrow{\beta} u'}{m\ u \xrightarrow{\alpha\ \beta} m'\ u'}$$

Petri Net Computations (II)

- Transitivity.

$$\frac{m \xrightarrow{\alpha} u \qquad u \xrightarrow{\beta} v}{m \xrightarrow{\alpha;\beta} v}$$

We will see later that, when we view Petri nets as rewite theories, the above inference system generating all Petri net computations of a net N coincides with the specialization of the general inference system of rewriting logic to the rewrite theory $R(N)$.

This illustrates a general point, namely, that rewriting logic is a very expressive semantic framework, in which many different concurrency models can be naturally specified.