- The course: Advanced course in languages focusing on concurrency
- First half: look at historical side of things, second half: papers
- Each day there will be a scribe, who will take notes that will be posted on the course website.
- Evaluation:
  - Participation
  - Course project
- 5 minutes at the end of class today to talk about forming up into groups
  - Wants us to be in groups of 2 to 3 for the course projects
- Ideas for Projects:
  - Open to anything that involves concurrency in some form. Should have producing a document that could be sent to some publication venue as a goal.
  - Possible ideas:
    * Survey paper – look at a bunch of papers, put them in some common form, compare them.
    * Hacking project – implementing some general and reuseable system (interpreter?) in another language.
      · π-calculus has an implementation called Pict; could use this to build an interpreter for another language.
      · Proof assistant – modeling a system formally, and checking the logic with a mechanical proof assistant.
  - Also open to other kinds of project ideas.
- On the course webpage:
  - `http://www.cs.cornell.edu/courses/CS6112/2011fa`
  - Number changed to 6112.
  - Course project preliminary proposal is due on September 11th (half page to one page).
  - Project check-up at the end of October.
  - Presentations on the last day of class, final writeup can be turned in then or a few days later.
- Course web page:
  - Tentative plans in grey, solid plans in black.

- If there are things you really want to talk about, but is not on the list, please bring it up.

- Finishing up a few things from last time:

- CSP:

$$
\begin{aligned}
a &::= \text{(arithmetic operations)} \\
b &::= \text{(boolean operations)} \\
c &::= skip | x := a | \alpha?x | \alpha!x | c\ \alpha | c_1 || c_2 | c_1; c_2 | \text{if } g \text{ fi} | \text{do } g \text{ od} \\
g &::= b \rightarrow c | b \wedge \alpha?x \rightarrow c | b \wedge \alpha!a \rightarrow c | g_1 [] g_2
\end{aligned}
$$

  - g – Dijkstra's guarded command syntax.
  - All of the operational semantics rules are represented in the CS4110 lecture notes, lectures 26-27. `http://www.cs.cornell.edu/Courses/cs4110/2010fa/lectures/lecture26.pdf`.

- What do programs look like in this CSP language?

  - $if(x \leq y -> m!y [] y \leq x -> m!x) fi.$
  - This program sends the maximum of x and y over channel m.
  - If they are equal, there is no way to distinguish which was sent.
  - Remember that $[]$ is actually a non-deterministic selection.

- Another example:

  - do (true $\wedge\ \alpha?x \rightarrow \beta!x$) od
  - Hooks up α to β
  - Buffering!
  - Remember that when receivers receive, they block

- One more example:

  - do (true $\wedge\ \alpha?x) -> \gamma!x []$ true $\wedge\ \beta?x -> \gamma!x$) od
  - Nondeterministic merge from α and β.

- Are guards atomic?

  - Yes! Once a guarded command begins, the two occurrences of x cannot interfere with each other.

- We might look at CHP at some point in the future

  - Note to Async crew: figure out a good paper to look for that

- Starting point for the pi calculus:

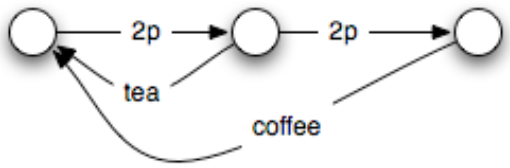
  - Finite automata!

- Typically, we will present these graphically:

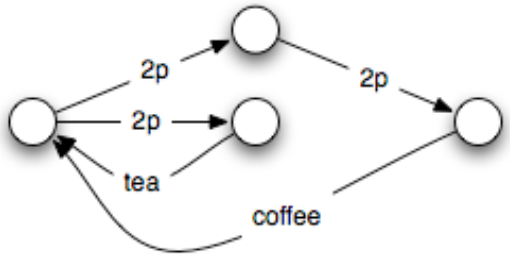
- $L(A)$ – language of strings over an automata, with internal symbols Σ

- Two automata, A_1 and A_2 are considered to be equivalent if their languages are equivalent, $L(A_1) = L(A_2)$
- The language that can be accepted by a finite automata can also be expressed as a regular expression (e.g., $a((bc + c)a)*$).
- Kleene proved that the set of things that you can match with a regex is the same as the one that you can match with finite automata
  - There are rich equational theories—e.g., Prof. Kozen has done work on this.
  - Some examples of laws include:
    * $R\epsilon = R$
    * $R|S = S|R$
    * $R|$
    * $T(R|S) = TR|TS$
- Expected to familiar with several operations on finite automata.
  - There are a handful of operations and axioms that can be used to prove many things about them.
- Sequential computation: a trace through a finite automata.
  - Can change a nondeterministic machine and embed it in a deterministic machine.
- Milner thinks that this is problematic:
  - We will have graphs of a process, and we will interact with it by doing certain actions.
  - Accepting alphabet is things that we can do, and bring us into different states.
  - Different model of interaction with computers.
  - Don't want to be so committed to equivalences that we get through some of the well-known transforms.
- Deterministic systems are very different to interact with than non-deterministic systems.
  - Do not want axioms like $T(R|S) = TR|TS$, since which T you pick matters in the second.
- Theory of simulations and bisimulations will be talking about today.
- First thing:
  - tweak our basic model of computation
- A Labeled Transition System (LTS)
  - Call it that instead of a finite automata.
  - Get rid of final states – any possible states are final states.
  - Accepting strings are not the right kind of equivalence for LTS.
  - Also, we will be getting rid of the start state.
    * Will only be talking about what we can do from some arbitrary states.

- An LTS is just a set of states and a transition relation.
- Why is language equivalence not what we want for these kind of systems?
- Milner likes to think about all the processes as black boxes.
  - All you can see is what buttons you can press in each state.
  - If you are in a state, you cannot press a button that is not active.
- State diagram for vending machine:



  - From some state, you can insert a 2p coin in, and then you can get tea out of it, or you can insert another 2p coin and then get coffee out.
  - Two kinds of actions:
    * Actions that we can initiate (will be written normally), and actions that the system can initiate (will be written with an overbar).
- Another automaton:



  - Inserting a 2p coin, you either go into a state where you can get tea, or into a state where you can insert another 2p coin, and then get coffee.
  - The original is very much the one we want.
    * This system, if the box chooses that you can get tea, then you can buy tea. However, it might force you to put in another 2p coin to get coffee.
    * Nondeterministic if you can get tea or coffee.
- We don't need to distinguish bar actions from unbared actions in this example, but it will be useful in future systems.
  - An example might be in a lottery system, where a bared action would be a win dispensing coins.
- Language equivalence doesn't work for these two systems:

- The two system we made up accept the same languages, however, they don't allow us to have the same modes of interactions.

- David Park and Milner worked out: theory of simulation and bisimulation (lots of varieties of these will be defiend in the next few weeks) that allows you to compare two different (possibly nondeterministic) systems, and say when they are behaving the same to some outside observer.
  - We will be able to say that the two are equivalent or non-equivalent in some formal way

- Even more boiled down example: (really strong tea, eh?)

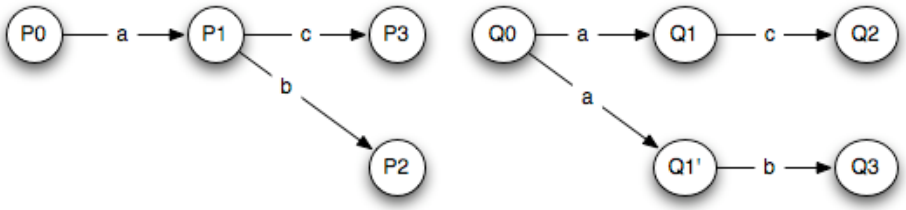
  - These two will accept the same language, but they will not be the same, since the nondeterminism in the second state screws things up.

- In our formalization, we will be able to say if states are able to simulate other states.

- if $p \; S \; q$, and p can go to p' through a, then there exists a q' such that q can go to q' through a, and $p' \; S \; q'$.
  - In this case, we will say that q simulates p.
  - S needs to be big enough, that if we start to do something in p, then we can do something in q, and the property will still hold.
  - Whenever we have something related by S, then both the state on the lefts and right can do something, and S will still hold.
  - S is directional.

- When S exists, we say that one state can simulate another.
  - S is not an equivalence relation – it could be empty.
  - We would like to build up to a notion of equivalence.

- Example:



  - Would like to show that p_0 simulations q_0.

- S will be a set of pairs of states: $S = \{(q_0, p_0) \cdots\}$.
- Start by adding those states:
  - For every (q, p) in S, then if q can transition through some action, than p can transition through the same action.
  - Since q_0 can trasition on a, the process p_0 must be able to transition on a. It can! It will transition to p_1 in both cases, therefore, S now contains:

$$S = \{(q_0, p_0), (q_1, p_1), (q_1', p_1) \cdots\}$$

  - q_1 can do a b, which p_1 can do, and q_1 can do a c, which p_1 can also do, therefore the complete S is:

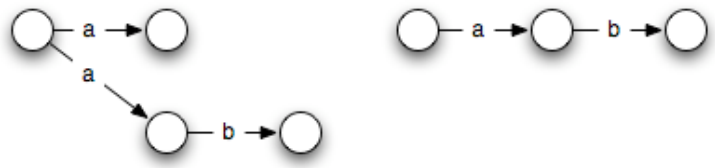
$$S = \{(q_0, p_0), (q_1, p_1), (q_1', p_1), (q_2, p_2), (q_3, p_3)\}$$

    There is a complete simulation in the p_is of the q_is.

- This is a directed notion of simulation
  - Would like to use it to make an equivalence relation, but it is not reflexive or symmetric.
- Strong bisimulation – a (strong) simulation whose converse is also a strong simulation.
- Recall that the converse of a relation S is defined by $S^{-1} \triangleq \{(x, y) | (y, x)\}$.
- We are a tiny bit short on time:
  - Definition: a relation S is a strong bisimulation iff S and S^{-1} is a strong simulation
  - The example we did above is not a strong bisimulation since S^{-1} is not a strong simulation.
  - The core of the reason for this is: "what can simulate p_0 in the other system?" – needs to take to a state that can also accept b and c, which neither q_1 nor q_1' can do.
- Definition:
  - Notation: write $p \sim q$ if there exists a strong bisimulation S s.t. $p \; S \; q$
  - This is the notation that we will be using for the rest of the course, though with a handful of different variations.
    - For example, might not care about the number of internal transitions.
- Two interesting propositions:
  - $\sim$ is an equivalence – it is a binary relation on states.
  - $\sim$ is often called "bisimularity".
- Two case are easy:
  - Reflexive: Have to show that every process p is $\sim$-equivalent to p.
    - Trivially a strong simulation – equality.
  - Symmetric: for all p, q in Q we have $p \sim q \Rightarrow q \sim p$.
    - This holds by definition
  - Transitive: for all p, q, r in Q, if $p \sim q$ and $q \sim r$, then $p \sim r$

    - This can be worked out by a pasting argument – string it together and it works.
    - Left as an exercise.

- Now is where some heads will explode (maybe):
  - Second proposition: $\sim$ is a strong bisimulation.
  - The property itself is a strong bisimulation.
  - $\sim$ defines bisimilarity, and it could have different relationships inside of it.
  - This is the biggest one—the union of all bisimulations—we will be talking about these kinds of problems more later on.
- These definitions are "seductively simple" but actually are quite rich.
  - We will be using them quite a bit over the next few weeks.
  - One thing that people often confuse: to be a bisimulation, the relation and its converse have to be strong simulations.
    - Not the case: that if you have a state that can be simulated by some state, and that state can be simulated by the first state (possibly in a different way) that those things are bisimilar.



- These two systems can strongly simulate each other, but they are still different.
  - The left state can take an a and then deadlock.
- There doesn't exist a bisimulation for this, since the converse doesn't hold.