As we saw in the last exercise, we can use the Scala library `pilib` to implement concurrent processes described as process calculus terms. This week, we will implement mobile processes using this library. We will use the same environment as last time, including the `Makefile`.

One difference between Scala and the pi-calculus is that the former is statically typed. In other words, we must manually find an explicit type for each channel that we want to use. This can be done in two different ways.

1. We can define each channel to have the type `Chan[Any]`, and then use an explicit typecast whenever we want to use the value that we received. This circumvents the static typing of the language, discovering type errors only at run-time. In Scala, we write `x.as[T]` for the cast of `x` into type `T`.

2. We can use *recursive* channel types. For instance, if we want to do the output action $\overline{a} < a, a >$ then the Scala translation of a could have a type `T` defined as
   `class T extends Chan[(T,T)];`
   Receiving on this channel can then be written (in a sum)
   `...  + a * (case (x,y) => F(x,y)) + ...`

For examples, you can look at the files `handoverCast.scala` and `handoverRecursive.scala` that can be found on the course web site.

0. Previous Session

If you have not yet completed the exercises of the previous session, this is a good opportunity to do so.

1. Cut-when-left

Implement the cut-when-left elastic buffers seen in class. Instrument the code with printouts (`System.err.println` calls) and random-length delays (`sleep(a + random.nextInt(b))`). Look at a few runs of the producer-buffer-consumer system, and see how the shrinking works.

2. Cut-when-right

Also implement the cut-when-right elastic buffers (easy once you have cut-when-left). Do both variants behave in the same way when coupled with a producer and a consumer? What happens if they stand alone with an empty cell on the left? Or on the right?