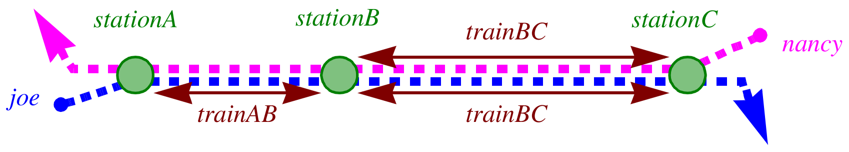
Exercise 1. (From “Abstractions for Mobile Computation”)

This example emphasizes the mobility aspects of ambients, and the fact that an ambient may be transported from one place to another without having to know the exact route to be followed. A passenger on a train, for example, only needs to know the destination station, and need not be concerned with the intermediate stations a train may or may not stop at.

In this example, there are three train stations, represented by ambients: *stationA*, *stationB* and *stationC* (of course, these particular ambients will never move). There are three trains, also represented by ambients: a train from *stationA* to *stationB* originating at *stationA*, and two trains between *stationB* and *stationC*, one originating at each end. There are two passengers, again represented by ambients; *joe* and *nancy*. *Joe* wants to go from *stationA* to *stationC* by changing trains at *stationB*; *nancy* wants to go the other way.



We begin by defining a parametric process that can be instantiated to trains going between different stations at different speeds. The parameters are: *stationX*: the origin station; *stationY*: the destination station; *XYatX*, the tag that the train between *X* and *Y* displays when stationed at *X*; *XYatY*, the tag that the train between *X* and *Y* displays when stationed at *Y*; *tripTime*, the time a train takes to travel between origin and destination.

```
let train(stationX stationY XYatX XYatY tripTime) =
    (ν moving)                    // assumes the train originates inside stationX
        moving[rec T.
            be XYatX. wait 2.0.
            be moving. out stationX. wait tripTime. in stationY.
            be XYatY. wait 2.0.
            be moving. out stationY. wait tripTime. in stationX.
            T];
```

The definition of a train begins with the creation of a new name, *moving*, that is intermittently used as the name of the train. While the train is *moving*, passengers should not be allowed to (dis)embark; this is achieved by keeping *moving* a secret name. The train begins as an ambient with name *moving*, and contains a single recursive thread that shuttles the train back and forth between two stations. Initially, the train declares itself to be a train between *X* and *Y* stationed at *X*, and waits some time for passengers to enter and exit. Then it becomes *moving*, so passengers can no longer (dis)embark. It exits the origin station, travels for the *tripTime*, enters the destination station, and declares itself to be the train between *X* and *Y* at *Y*. Again, passengers can (dis)embark during the wait time at the station. Then the train becomes *moving* again, goes back the other way, and then repeats the whole process.

Next we have the configuration of stations and trains. We create fresh names for the three stations and for the train tags. Then we construct three ambients for the three stations, each containing an appropriately instantiated train.

```
(ν stationA stationB stationC ABatA ABatB BCatB BCatC)
    stationA[train(stationA stationB ABatA ABatB 10.0)] |
    stationB[train(stationB stationC BCatB BCatC 20.0)] |
    stationC[train(stationC stationB BCatC BCatB 30.0)] |
```

Finally, we have the code for the passengers. *Joe*’s itinerary is to enter *stationA*, wait to enter the train from *A* to *B* when it is stationed at *A*, exit at *B*, wait for the train from *B* to *C* when it is stationed at *B*, exit at *C* and finally exit *stationC*. During the time that *joe* is waiting to exit a train, he is blocked waiting for the train to acquire the appropriate tag. The train could change tags at intermediate stations, but this would not affect *joe*, who is waiting to exit at a particular station. When that station is reached, and the train assumes the right tag, *joe* will attempt to exit. However, there is no guarantee that he will succeed. For example, *joe* may have fallen asleep, or there may be such a rush that *joe* does not manage to exit the train in time. In that case, *joe* keeps shuttling back and forth between two stations until he is able to exit at the right station.

```
(ν joe)
    joe[
        in stationA.
        in ABatA. out ABatB.
        in BCatB. out BCatC.
        out stationC] |
```

The code for *nancy* is similar, except that she goes in the other direction. Given the timing of the trains, it is very likely that *nancy* will meet *joe* on the platform at *stationB*.

```
(ν nancy)
    nancy[
        in stationC.
        in BCatC. out BCatB.
        in ABatB. out ABatA.
        out stationA]
```

In all this, *joe* and *nancy* are active ambients that are being transported by other ambients. Sometimes they move of their own initiative, while at other times they move because their context moves. Note that there are two trains between *stationB* and *stationC*, which assume the same names when stopped at a station. *Joe* and *nancy* do not care which of these two train they travel on; all they need to know is the correct train tag for their itinerary, not the “serial number” of the train that carries them. Therefore, having multiple ambients with the same name simplifies matters.

When all these definitions are put together and activated, we obtain a real-time simulation of the system of stations, trains, and passengers. A partial trace looks like this:

```
nancy:  moved in stationC
nancy:  moved in BCatC
joe:    moved in stationA
joe:    moved in ABatA
joe:    moved out ABatB
nancy:  moved out BCatB
joe:    moved in BCatB
nancy:  moved in ABatB
nancy:  moved out ABatA
nancy:  moved out stationA
joe:    moved out BCatC
joe:    moved out stationC
```

Exercise 4.

1)
$\mathcal{A} \mid \mathcal{B} \wedge \mathbf{0} \vdash \mathcal{A}$.

Need to show that $P \vDash \mathcal{A} \mid \mathcal{B}$ and $P \vDash \mathbf{0}$ implies $P \vDash \mathcal{A}$.
$P \vDash \mathcal{A} \mid \mathcal{B}$ means $P \equiv P' \mid P''$ and $P' \vDash \mathcal{A}$ and $P'' \vDash \mathcal{B}$.
$P \vDash \mathbf{0}$ means $P \equiv \mathbf{0}$. Hence $P' \mid P'' \equiv \mathbf{0}$.
This implies (lemma) that $P' \equiv \mathbf{0}$. Hence $P' \equiv P$.
Since $P' \vDash \mathcal{A}$ and $P' \equiv P$ we obtain $P \vDash \mathcal{A}$ by the "Basic Fact".

2)
$\mathcal{A} \mid \mathcal{B} \wedge \mathbf{0} \vdash \mathcal{A}$.

$$
\begin{array}{l}
\lceil\, \lceil\ \}_{(\mid \|)}\ \mathcal{A} \mid \mathcal{B} \wedge \neg(\neg\mathbf{F} \mid \neg\mathbf{0}) \vdash \mathcal{A} \mid \mathbf{0} \vee \mathbf{F} \mid \mathcal{B}\ ; \\
\quad \lceil\ \}_{(\mid \mathbf{0})}\ \mathcal{A} \mid \mathbf{0} \vdash \mathcal{A}\ ;\ \}_{(\mid \mathbf{F})}\ \mathbf{F} \mid \mathcal{B} \vdash \mathbf{F}\ \rceil\ \}_{(\vee \vdash)(\text{X-R})(\mathbf{F})}\ \mathcal{A} \mid \mathbf{0} \vee \mathbf{F} \mid \mathcal{B} \vdash \mathcal{A} \\
\ \rceil\ \}_{(\text{Trans})}\ \mathcal{A} \mid \mathcal{B} \wedge \neg(\neg\mathbf{F} \mid \neg\mathbf{0}) \vdash \mathcal{A} \\
\ \lceil\ \}_{(\mid \neg\mathbf{0})}\ \neg\mathbf{F} \mid \neg\mathbf{0} \vdash \neg\mathbf{0}\ \ \}_{(\neg \vdash)(\ldots)}\ \mathbf{0} \vdash \neg(\neg\mathbf{F} \mid \neg\mathbf{0})\ ; \\
\quad \}_{(\text{Id})}\ \mathcal{A} \mid \mathcal{B} \vdash \mathcal{A} \mid \mathcal{B} \\
\ \rceil\ \}_{(\text{Trans})}\ \mathcal{A} \mid \mathcal{B} \wedge \mathbf{0} \vdash \mathcal{A} \mid \mathcal{B} \wedge \neg(\neg\mathbf{F} \mid \neg\mathbf{0}) \\
\rceil\ \}_{(\text{Trans})}\ \mathcal{A} \mid \mathcal{B} \wedge \mathbf{0} \vdash \mathcal{A}
\end{array}
$$

where

$(\vee \vdash)$ $\quad \mathcal{A} \vdash \mathcal{B};\ \mathcal{A}' \vdash \mathcal{B}'\ \}\ \mathcal{A} \vee \mathcal{A}' \vdash \mathcal{B} \vee \mathcal{B}'$

$\lceil\lceil\lceil\lceil \mathcal{A}' \vdash \mathcal{B}'\ \}_{(\text{W-L})}\ \mathcal{A}' \wedge \mathbf{T} \vdash \mathcal{B}'\ \}_{(\text{X-L})}\ \mathbf{T} \wedge \mathcal{A}' \vdash \mathcal{B}' \rceil\ ;\ \}_{(\text{Id})}\ \mathcal{A}' \vee \mathcal{A} \vdash \mathcal{A}' \vee \mathcal{A} \rceil\ \ \}_{(\text{Cut }\mathcal{A}')}\ (\mathcal{A}' \vee \mathcal{A}) \wedge \mathbf{T} \vdash \mathcal{A} \vee \mathcal{B}' \rceil\ \}_{(\mathbf{T})}\ \mathcal{A}' \vee \mathcal{A} \vdash \mathcal{A} \vee \mathcal{B}' \rceil;\ \ \lceil \mathcal{A} \vdash \mathcal{B}\ \}_{(\text{W-L})}\ \mathcal{A} \wedge \mathbf{T} \vdash \mathcal{B}\ \}_{(\text{X-L})}\ \mathbf{T} \wedge \mathcal{A} \vdash \mathcal{B} \rceil\rceil\ \ \}_{(\text{Cut }\mathcal{A})}\ (\mathcal{A}' \vee \mathcal{A}) \wedge \mathbf{T} \vdash \mathcal{B}' \vee \mathcal{B}\ \}_{(\mathbf{T})}\ \mathcal{A}' \vee \mathcal{A} \vdash \mathcal{B}' \vee \mathcal{B}$

$(\neg \vdash)$ $\quad \mathcal{B} \vdash \mathcal{A}\ \}\ \neg\mathcal{A} \vdash \neg\mathcal{B}$

$\mathcal{B} \vdash \mathcal{A}\ \}_{(\text{W-L})}\ \mathcal{B} \wedge \mathbf{T} \vdash \mathcal{A}\ \}_{(\text{X-L})}\ \mathbf{T} \wedge \mathcal{B} \vdash \mathcal{A}\ \}_{(\neg\text{-R})}\ \mathbf{T} \vdash \neg\mathcal{B} \vee \mathcal{A}\ \}_{(\text{X-R})}\ \mathbf{T} \vdash \mathcal{A} \vee \neg\mathcal{B}$
$\}_{(\neg\text{-L})}\ \mathbf{T} \wedge \neg\mathcal{A} \vdash \neg\mathcal{B}\ \}_{(\text{X-L})}\ \neg\mathcal{A} \wedge \mathbf{T} \vdash \neg\mathcal{B}\ \}_{(\mathbf{T})}\ \neg\mathcal{A} \vdash \neg\mathcal{B}$