```
import openpathsampling as paths
storage = paths.AnalysisStorage("tps_file.nc")

# load states from storage
C7eq = storage.volumes['C7eq']
alpha_r = storage.volumes['alpha_r']

# create the ensemble
recrossing = paths.SequentialEnsemble([
    paths.LengthEnsemble(1) & paths.AllInXEnsemble(alpha_r),
    paths.OptionalEnsemble(paths.AllOutXEnsemble(C7eq | alpha_r)),
    paths.LengthEnsemble(1) & paths.AllInXEnsemble(C7eq)
])
```

```
# define function to identify accepted recrossings
def accepted_recrossing(step):
    trial_recrossings = recrossing.split(step.trials[0].trajectory)
    return len(trial_recrossings) > 0 & step.accepted

# find all relevant MC steps
recrossing_steps = [step for step in storage.steps
                    if accepted_recrossing(step)]

print len(recrossing_steps)  # output

# look at the first trajectory
trajectory = recrossing_steps[0].active[0].trajectory
```

```
import openpathsampling as paths
storage = paths.AnalysisStorage("tps_file.nc")

# load states from storage
C7eq = storage.volumes['C7eq']
alpha_r = storage.volumes['alpha_r']
```

```
# create the ensemble
recrossing = paths.SequentialEnsemble([
    paths.LengthEnsemble(1) & paths.AllInXEnsemble(alpha_r),
    paths.OptionalEnsemble(paths.AllOutXEnsemble(C7eq | alpha_r)),
    paths.LengthEnsemble(1) & paths.AllInXEnsemble(C7eq)
])
```

```
# define function to identify accepted recrossings
def accepted_recrossing(step):
    trial_recrossings = recrossing.split(step.trials[0].trajectory)
    return len(trial_recrossings) > 0 & step.accepted

# find all relevant MC steps
recrossing_steps = [step for step in storage.steps
                    if accepted_recrossing(step)]

print len(recrossing_steps)  # output

# look at the first trajectory
trajectory = recrossing_steps[0].active[0].trajectory
```