# Removal of conditioning

- Idea: restart an infeasible run until all `observe`-statements are passed
- For program variable `x` use auxiliary variable `sx`
  - store initial value of `x` into `sx`
  - on each new loop-iteration restore `x` to `sx`
- Use auxiliary variable `flag` to signal observation violation:

```
flag := true; while(flag)prog
```

- Change `prog` into `modprog` by:
  - `observe(G)` $\rightsquigarrow$ `flag := !G`
  - `abort` $\rightsquigarrow$ `if(!flag) abort`
  - `while(G) prog` $\rightsquigarrow$ `while(G && !flag) prog`

# Resulting program

```
sx1,...,sxn := x1,...,xn; flag := true;
while(flag) {
   flag := false;
   x1,...,xn := sx1,...,sxn;
   modprog
}
```

# Removal of conditioning

the transformation in action:

```
x := 0 [p] x := 1;
y := 0 [p] y := 1;
observe(x != y)
```

```
sx, sy := x, y; flag := true;
while(flag) {
  x, y := sx, sy; flag := false;
  x := 0 [p] x := 1;
  y := 0 [p] y := 1;
  flag := (x = y)
}
```

a simple data-flow analysis yields:

```
repeat {
  x := 0 [p] x := 1;
  y := 0 [p] y := 1
} until(x != y)
```

# Removal of conditioning

## Correctness of transformation

For program $P$, transformed program $\hat{P}$, and post-expectation $f$:

$$cwp(P, f) \;=\; wp(\hat{P}, f)$$