Appendix: alphabet extension

- Remember that an alphabet is implicitly associated with each process
- If an action appears in alphabets of multiple processes, it is shared and always has to happen concurrently in all those processes ie. the processes synchronize on that action.

- In the Garden example, we wanted to prevent any free action by the variable **VAR** alone, such as `write[0]`, from ever occurring.
- This could be done by extending the alphabet of **TURNSTILE** with the required action, ie `+{write[0]}.`
- This was actually done by extending the alphabet of the turnstiles with the full alphabet set of the shared variable `{VarAlpha]}.`

Example of extending alphabet (1)

```
WORKER = (work -> WORKER | play -> WORKER).
MANAGER = STOP.
||COMPANY = (WORKER || MANAGER).
```

- **COMPANY** consists of a worker (who can either work or play) and a manager (who does nothing!)
- Therefore, the composed process **COMPANY** can do either work or play actions
- Problem: we don't want the **COMPANY** to play, ever!

Example of extending alphabet (2)

```
WORKER = (work -> WORKER | play -> WORKER).
MANAGER = STOP + {play}.
||COMPANY = (WORKER || MANAGER).
```

- Solution: we extend the alphabet of **MANAGER** to include action **play**
- **play** is now a shared action and can only occur if both **WORKER** and **MANAGER** do it
- ...and since **MANAGER** never does so, neither can the **WORKER**.

Example of extending alphabet (3)

- Having decided that play is an undesirable action, we can use the ERROR state to check for it. The following finds an error trace:

```
WORKER = (work -> WORKER | play -> WORKER).
MANAGER = STOP.
TEST = (play -> ERROR).
||COMPANY = (WORKER || MANAGER|| TEST).
```

- while extending the alphabet fixes it:

```
WORKER = (work -> WORKER | play -> WORKER).
MANAGER = STOP + {play}.
TEST = (play -> ERROR).
||COMPANY = (WORKER || MANAGER|| TEST).
```

Garden example

- Back in the Garden example, by adding `write[0]` to the alphabet of **INCREMENT**, we ensured that any `write[0]` action in **VAR** would have to happen in **INCREMENT** as well.
- But, since **INCREMENT** only does `write[x+1]`, for `x:0..4`, this can never happen
- What this prevented is the `write[0]` defined in **VAR** from occurring autonomously!