For each individual job we have a “bucket,” which we assume starts out empty (though if there is reason we can allow it to start out occupied). We tell certain events to add atoms to the bucket:

```
on started apache add have_apache
on stopped apache add !have_apache
on started tomcat add have_tomcat
on stopped tomcat add !have_tomcat
```

So after started apache had occured, the bucket would be {have_apache}. Note that the bucket is a set, so two of the same atom cannot be added. After another started apache event we would still have {have_apache}.

We then define a series of “decay rules” which cause certain substitutions to be made on the bucket every time an atom is added.

```
{!have_apache, have_apache} => {}
{!have_tomcat, have_tomcat} => {}
{start, have_tomcat} => {start}
{start, have_apache} => {start}
{start, !have_apache} => {have_tomcat}
{start, !have_tomcat} => {have_apache}
{have_apache, have_tomcat} => {start}
{!have_apache} => {}
{!have_tomcat} => {}
```

Each line simply states “Whenever everything on the left can be found in the bucket, remove it and add everything on the right.” We assume that the job should be running whenever the “start” atom is in the bucket, so we start the job when it is added and stop it when it is removed.

The rules are run in sequence, and repeated until all of them run through without altering the contents of the bucket.

You may notice that this definition is a little verbose. However, there are many common patterns that emerge here, which means we can install some syntactic sugar to clean things up a bit. For example, the atoms beginning with ! Are assumed to be the inverse of those that do not. The current ruleset treats them the same as ordinary atoms, but if we formalize these “anti-atoms” we can remove the first two and last two rules and simply assume them to be implicit.

The rules we have left can be further optimized. The “start” atom in this context is essentially equivalent to having both “have_apache” and “have_tomcat” being present at the same time. This situation is common enough, and easy enough to define, that we can simply add an operator for it:

```
{start} <=> {have_apache, have_tomcat}
```

Yielding us one rule for the job definition.