

Language Time

Time

Alloy 6 added **temporal operators** to Alloy, making it easier to model dynamic systems.

Variables

A signature or relation can be declared mutable with the var keyword:

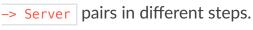
```
sig Server {}

var sig Offline in Server {}

sig Client {
   , var connected_to: lone Server - Offline
}
```

The set of servers that are offline is mutable: different servers can be offline in

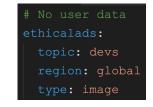
different steps. The connected_to field is also mutable and car





This uses the in modifier as in the Subtyping technique.

The number of steps in a model is specified with the Steps con



Al-powered ad network for devs. Get your message in front of the right developers with EthicalAds.

www.ethicalads.io

Temporal Operators

Ads by EthicalAds

A dynamic model is broken into several Steps. For each step, all var sign; relations may change, depending on the predicates. By default, all predicates facts only hold for the *initial* step of a variable. Eg

```
fact "init" {
  no Offline
  no connected_to
}
```

This fact says that in the initial step, there are no offline servers and no connection between clients and servers. There are no constraints, however, on future steps. To place constraints on future steps, put predicates inside a temporal operator, like

```
always Or eventually.
```

```
pred spec {
   always no Offline
   eventually some connected_to
}
run {spec}
```

- always no Offline means that no Offline is true now, and in all future steps.
- eventually some connected_to means that some connected_to is true in at least one step, now or in the future.

Temporal operators can be combined: eventually always some Offline means that there's a step where, from that step forward, there is some Offline server.

To model a "change", we relate the values of a variable between two steps. If connected_to is a var field, then connected_to is the value of connected_to in the next step.

```
pred connect[c: Client, s: Server] {
  c -> s not in connected_to
  connected_to' = connected_to ++ c -> s
}
```

In this example, **connect** is true or false in every step. In steps where it is true, the client is not connected to the server *and* in the next step, it is connected to the server. This represents the state of the system changing.

is also called the *prime* operator. Combining primed predicates with temporal operators gives us a simple way to model system dynamics.

```
pred spec {
    -- all servers always online
    always no Offline

    -- initially no connections
    no connected_to

    -- every step, a client connects to a new server
    always some c: Client, s: Server {
        c.connect[s]
    }
}
run {spec}
```

List of Operators

Alloy operators include both *future* and *past* operators. Operators are true and false for a specific step.

Future	tempo	ral ope	erators
--------	-------	---------	---------

Operator	Meaning
always P	P is true and true in all future steps
eventually P	P is true or true in at least one future step
after P	P is true in the next step
P; Q	Shorthand for P && after Q
Q releases P	P is true until Q is true, then P may become false
P until Q	Equivalent to (Q releases P) and eventually Q

(P' is special: instead of being true or false, it's simply the value of the P in the next step.)

There are also *past* operators corresponding to each future operator. once P is the past-version of eventually P: P is true or true in at least one *previous* step.

Past temporal operators

Future Operator	Past Version	
always	historically	
eventually	once	
after	before	
triggered	releases	
since	until	