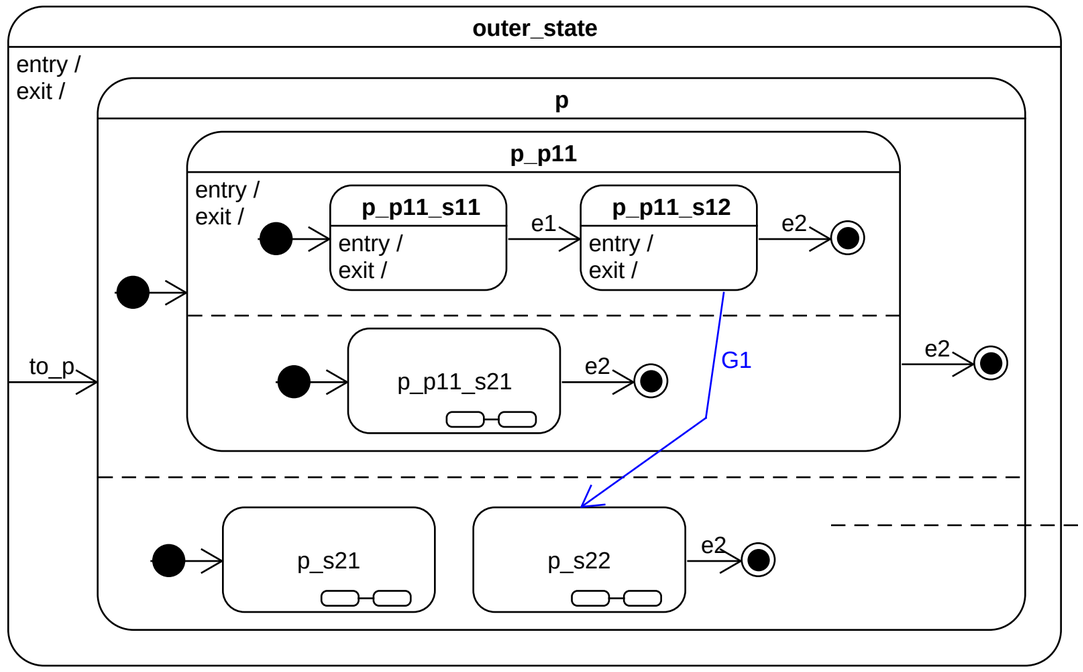


It is p_p11 has cause transitions in p_p11_r1 and p_p11_r2 to their under_hidden_regions, but this happens with an exit

It is p that has cause to cause transitions in p_r1 to under_hidden_region, but this happens with an exit

It is p that has cause to cause a transition in p_r2 to p_s21 via its INIT_SIGNAL

Idea:
* try making an onion from p_s21 to p_p11_s12
* reverse it
* for each item in the onion assign a META_EXIT or META_INIT
* An injector receiving a META_EXIT will only inject an exit_region if the function is in the regions it is injecting to
* If receiving a META_INIT, use peak_meta and see if the function is in the region, if so post_lifo force_region_init

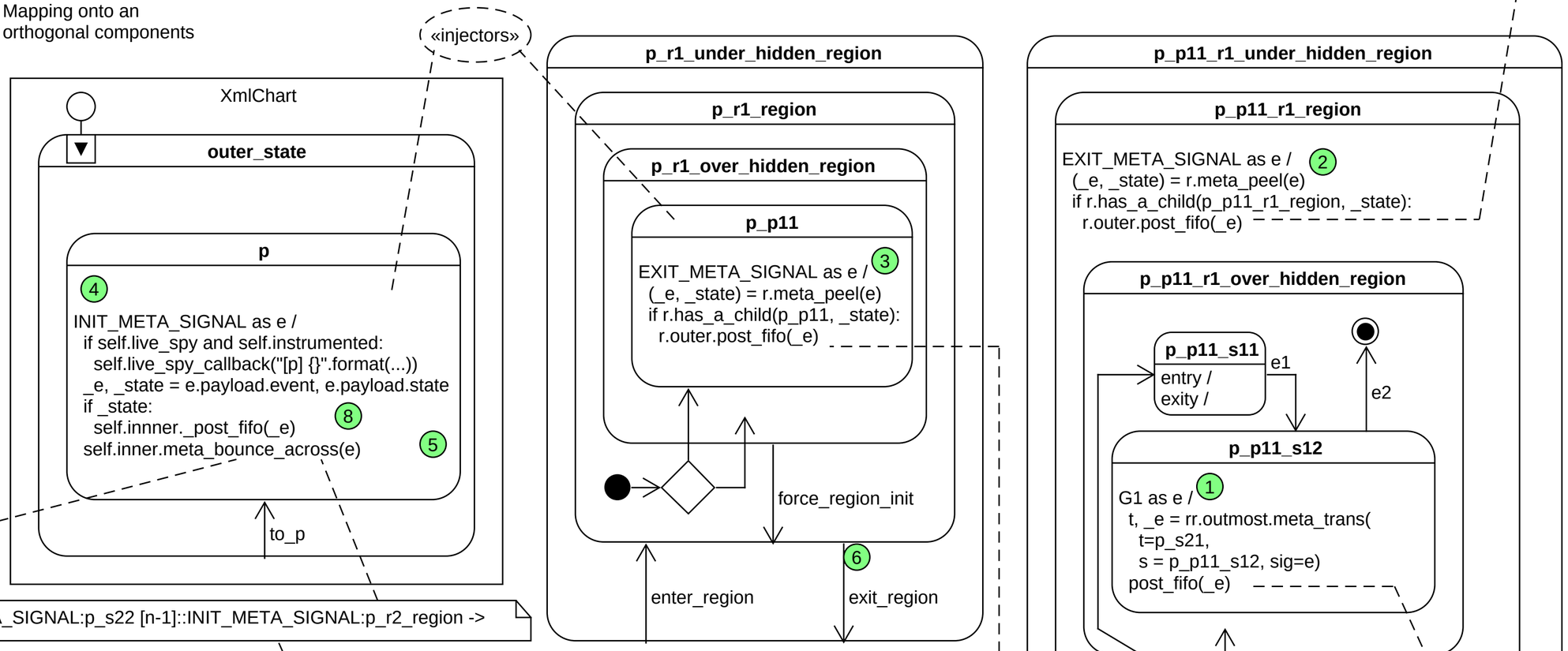
Introducting the second WTF event

[n]::EXIT_META_SIGNAL:p_p11 [n-1]::EXIT_META_SIGNAL:p_r2_region ->
[n]::INIT_META_SIGNAL:p_r2_region [n-1]::EXIT_META_SIGNAL:p_p11 ->
[n]::INIT_META_SIGNAL:p_s22 [n-1]::INIT_META_SIGNAL:p_r2_region ->

Harel Diagram

Mapping onto an orthogonal components

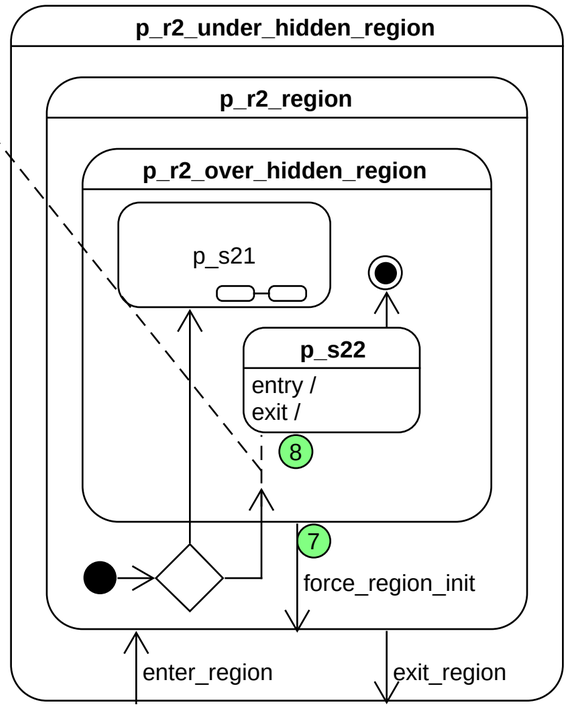
```
payload = namedtuple(
 'META_SIGNAL_PAYLOAD',
 [
  'event',
  'state',
  'previous_state',
  'previous_event'
 ]
)
```



[n]::INIT_META_SIGNAL:p_s22 [n-1]::INIT_META_SIGNAL:p_r2_region ->

[n]::INIT_META_SIGNAL:p_r2_region [n-1]::EXIT_META_SIGNAL:p_p11 ->
[n]::INIT_META_SIGNAL:p_s22 [n-1]::INIT_META_SIGNAL:p_r2_region ->

```
def meta_bounce_across(self, e):
 trans_init = False
 if e.payload.event.signal != e.payload.previous_signal:
  trans_init = True
 if trans_init:
  for region in self._regions:
   if region.has_state(e.payload.previous_state):
    region.pop_event():
    region.post_lifo(Event(signal=signals.region_exit))   6
   else:
    region.post_lifo(Event(signal=signals.force_region_init))
  self._complete_circuit()   7
 else:
  self.post_lifo(Event=signals.force_region_init)
```



[n]::EXIT_META_SIGNAL:p_p11_r1_region [n-1]::G1:p_p11_s12 ->
[n]::EXIT_META_SIGNAL:p_p11 [n-1]::EXIT_META_SIGNAL:p_r2_region ->
[n]::INIT_META_SIGNAL:p_r2_region [n-1]::EXIT_META_SIGNAL:p_p11 ->
[n]::INIT_META_SIGNAL:p_s22 [n-1]::INIT_META_SIGNAL:p_r2_region ->