

The diagram illustrates a state machine with a hierarchical structure. At the top level is the **outer_state**, which contains a sub-state **p**. The **outer_state** has an initial state (represented by a circle with a downward arrow) and a final state (represented by a circle with a downward arrow). The **outer_state** has an entry event and an exit event. The **outer_state** has a transition labeled **to_p** leading to the **p** state. The **p** state has an entry event and an exit event. The **p** state contains two regions: **S1 Region** and **S2 Region**. The **S1 Region** contains states **S11** and **S12**. The **S11** state has an entry event and an exit event. The **S12** state has an entry event and an exit event. The **S11** state has a transition labeled **e4** leading to the **S12** state. The **S12** state has a transition labeled **e1** leading to a final state labeled **S1Final**. The **S2 Region** contains states **S21** and **S22**. The **S21** state has an entry event and an exit event. The **S22** state has an entry event and an exit event. The **S21** state has a transition labeled **e1** leading to the **S22** state. The **S22** state has a transition labeled **e2** leading to a final state labeled **S2Final**. The **outer_state** has a transition labeled **to_outer** leading to a state labeled **some_other_state**. The **some_other_state** has an entry event. The **outer_state** has a transition labeled **p_final** leading to the **some_other_state**. A note indicates that the **p_final** transition is sent when **S1Final** and **S2Final** states have been entered.

Equivalent Orthogonal Components Diagram

The diagram illustrates the decomposition of an **outer_state** into two **hidden_state** components (**s1_hidden_state** and **s2_hidden_state**) using orthogonal components.

outer_state (Left):

- Initial state: **entry / exit /**
- Region **p**:
 - entry / self.p_dispatcher(Event(signal=signals.to_p))
 - e1, e2, e4 as e / self.p_dispatcher(e)
 - exit / self.p_dispatch(Event(signal=signals.region_exit))
- Transitions:
 - to_outer (downward)
 - p_final (downward to **some_other_state**)
 - to_p (upward)

s1_hidden_state (Top Right):

- Initial state: **entry / exit /**
- Region **s1_region**:
 - entry / exit /
 - Initial state: **s11** (entry / exit /)
 - Transition e4 to **s12** (entry / exit /)
 - Transition e1 to **s1_region_final** (entry / r.final = True, r.post_final_to_other_if_ready(); exit / r.final = False)
- Transitions:
 - region_exit (downward)
 - to_p (upward)

s2_hidden_state (Bottom Right):

- Initial state: **entry / exit /**
- Region **s2_region**:
 - entry / exit /
 - Initial state: **s21** (entry / exit /)
 - Transition e4 to **s22** (entry / exit /)
 - Transition e1 to **s2_region_final** (entry / r.final = True, r.post_final_to_other_if_ready(); exit / r.final = False)
- Transitions:
 - region_exit (downward)
 - to_p (upward)

«pattern» Orthogonal Components (Dashed oval):

- Indicates the decomposition of the **outer_state** into **s1_hidden_state** and **s2_hidden_state**.

«pattern» Reminder (Dashed oval):

- Indicates the reminder for the **post_p_final_to_outer_if_ready** method.

Code Snippets:

```
def p_dispatcher(self, e):
    status = return_status.HANDLED
    [region.post_fifo(e) for region in self.p_regions]
    [region.complete_circuit() for region in self.p_regions]
    return status
```

```
def post_p_final_to_outer_if_ready(self):
    ready = False if self.regions is None and len(self.regions) < 1 else True
    for region in self.regions:
        ready &= True if region.final else False
    if ready:
        self.outer.post_fifo(self.final_event)
```