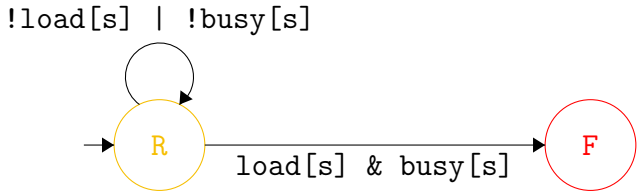
Automated Testing and Dynamic Analysis (2020/21)
FIT, Brno University of Technology
Vladimír Marcin
xmarci10@stud.fit.vutbr.cz

Homework 2

Monitor 1 *The cart mustn't load anything into an occupied slot.*

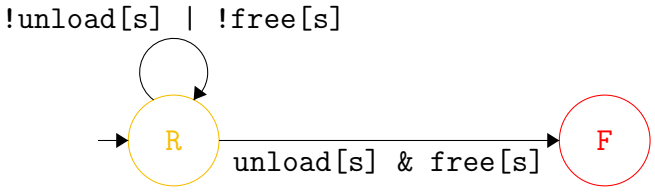


`s` ∈ {0, 1, 2, 3}
□`!(load[s] & busy[s])`

- `load[s]`: indicates whether the cart wants to load something into slot `s`
- `busy[s]`: indicates if there is something on the cart in slot `s`

  - Implemented as a four-element vector representing the current state of the monitor (see `Monitoring.monitor1._slots`).

Monitor 2 *The cart mustn't unload anything from a free slot.*

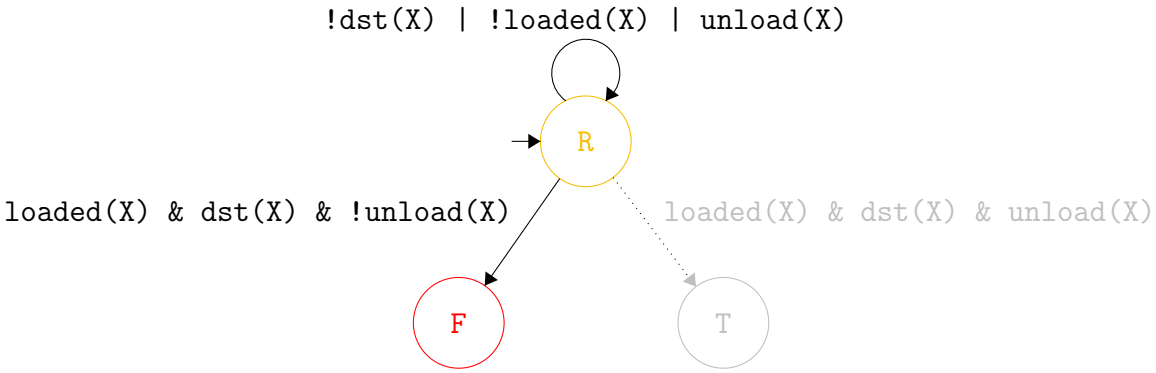


`s` ∈ {0, 1, 2, 3}
□`!(unload[s] & free[s])`

- `unload[s]`: indicates whether the cart wants to unload something from slot `s`
- `free[s]`: indicates whether slot `s` is free

  - Implemented as a four-element vector representing the current state of the monitor (see `Monitoring.monitor2._slots`).

Monitor 3 *The cart must unload the cargo at its destination.*

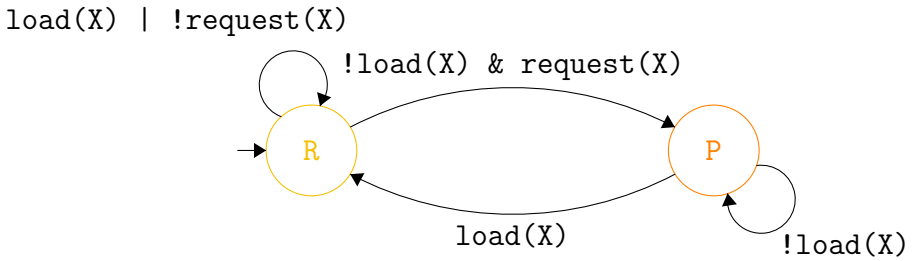


$\forall$ X: $\square$[(dst(X) & loaded(X)) $\Rightarrow$ unload(X)]

- `dst(X)`: indicates whether the cart is at the destination stop of the cargo `X`
- `loaded(X)`: indicates whether the cargo `X` is loaded on the cart
- `unload(X)`: indicates whether the cargo `X` has been unloaded at the current stop

- Implemented by the monitor instance with each new request (see `onrequest()`). The monitor then waits for the cargo to be loaded and this event is logged within the monitor (see `Monitoring.monitor3[X].content_loaded()`). Then, when leaving the stop where the cargo `X` were to be unloaded, it is checked whether this has happened (see `Monitoring.move()` or `Monitoring.stop()`). If the cargo `X` has been successfully unloaded at its destination stop (gray out branch, instance specific), the property is fulfilled for the monitor instance, and thus it is removed from the list of instances to save a space. However, the global state of **Monitor 3** remains in the `Status.Risk` state.

Monitor 4 (optional) *Each transfer request must sometimes result in the load.*

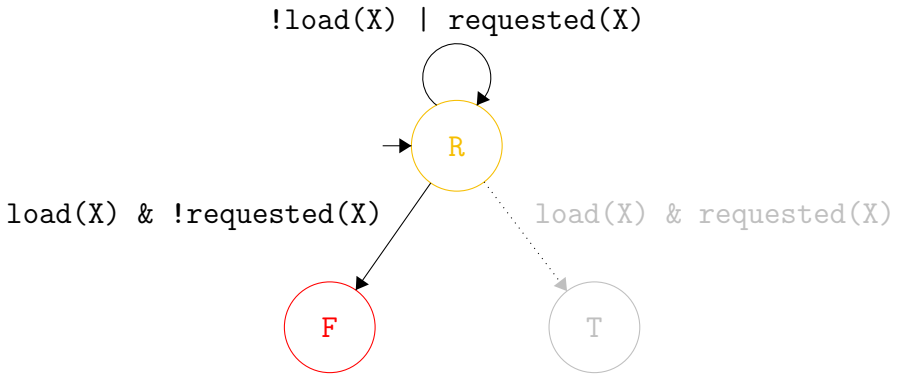


$\forall$ X: $\square$(request(X) $\Rightarrow$ $\Diamond$load(X))

- `load(X)`: indicates whether the cargo `X` has been loaded
- `request(X)`: indicates whether a request for the carriage of cargo `X` exists

- Again, implemented by instantiating monitor at each request. When loading `X` (see `Monitoring.load()`), the corresponding monitor instance is removed as the property is fulfilled for it (same idea as in **Monitor 3**). Since this is a liveness property, its violation is checked only at the `stop` event (see `Monitoring.stop()`).

Monitor 5 *The cart must not be loaded at the station unless there is a request to do so.*

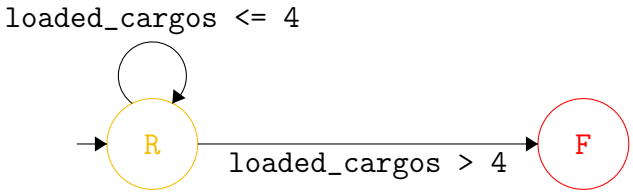


∀ X: □!(!request(X) & load(X))

- `load(X)`: indicates whether the cargo `X` has been loaded at the current stop
- `requested(X)`: indicates whether a request for the carriage of cargo `X` has arisen

- Like **Monitor 3**, a new request creates a new instance of **Monitor 5**. Subsequently, when loading a cargo `X`, it is checked whether there was a request for loading of cargo `X` (see `Monitoring.load()`, `X in Monitoring.monitor5` ?). If so, it means that the given instance is fulfilled and, as in **Monitor 3**, this instance is deleted.

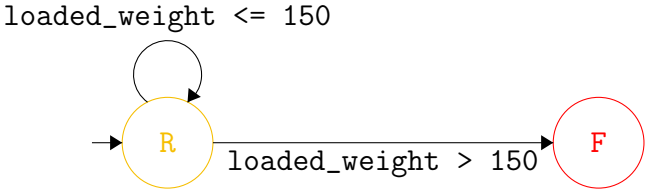
Monitor 6 *No more than 4 cargos may be loaded.*



□!(loaded_cargos > 4)

- Implemented by a simple counter that increments with each load (see `Monitoring.monitor6.add_cargo()`).

Monitor 7 *The cart must not be overloaded.*



□!(loaded_weight > 150)

- Again, implemented by a counter that remembers the currently loaded weight and checks before each loading that the permitted weight has not been exceeded (see `Monitoring.monitor7.add_weight(weight)`).

Error reporting

Run the following command to reproduce the errors found.

input:

```
./cartctl_test.py requests.csv | ./cart_monitor.py
```

output:

```
272:error:  the gold requested at A but never loaded
CartCoverage 100%
```