In [1]:

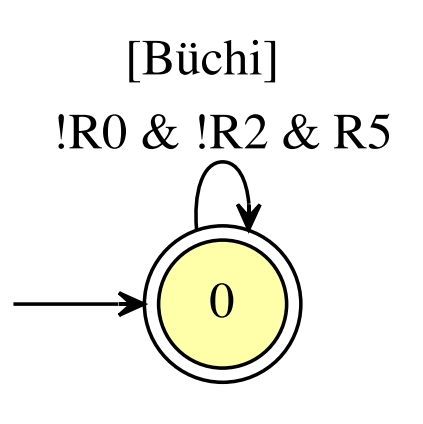
```python
from IPython.display import display
import spot
from spot.jupyter import display_inline
spot.setup()
```

In [2]:

```python
#Univeral / Global Reachability
a = spot.formula(' G ((R0)V(!R0&R2)V(!R0&!R2&R5))')
#b = spot.translate(a, 'BA', 'complete');
b = spot.translate(a, 'BA');

print("b is safety automaton: ",spot.is_safety_automaton(b))
print("b is liveness automaton:",spot.is_liveness_automaton(b))

print("a is safety: ",a.is_syntactic_safety())
print("a is liveness: ",spot.is_liveness(a))

b
```

```
b is safety automaton:  True
b is liveness automaton: False
a is safety:  True
a is liveness:  False
```

Out[2]:

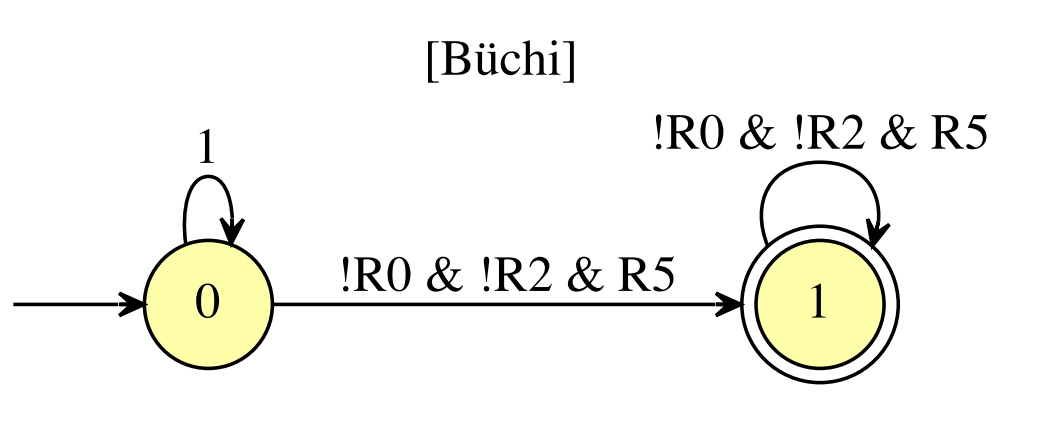


In [3]:

```python
#Future Reachability
a1 = spot.formula(' F ((R0)V(!R0&R2)V(!R0&!R2&R5))')
#b1 = spot.translate(a1, 'BA', 'complete');
b1 = spot.translate(a1, 'BA');

print("b1 is safety automaton: ",spot.is_safety_automaton(b1))
print("b1 is liveness automaton:",spot.is_liveness_automaton(b1))

print("a1 is safety: ",a1.is_syntactic_safety())
print("a1 is liveness: ",spot.is_liveness(a1))

b1
```

```
b1 is safety automaton:  False
b1 is liveness automaton: True
a1 is safety:  False
a1 is liveness:  True
```

Out[3]:

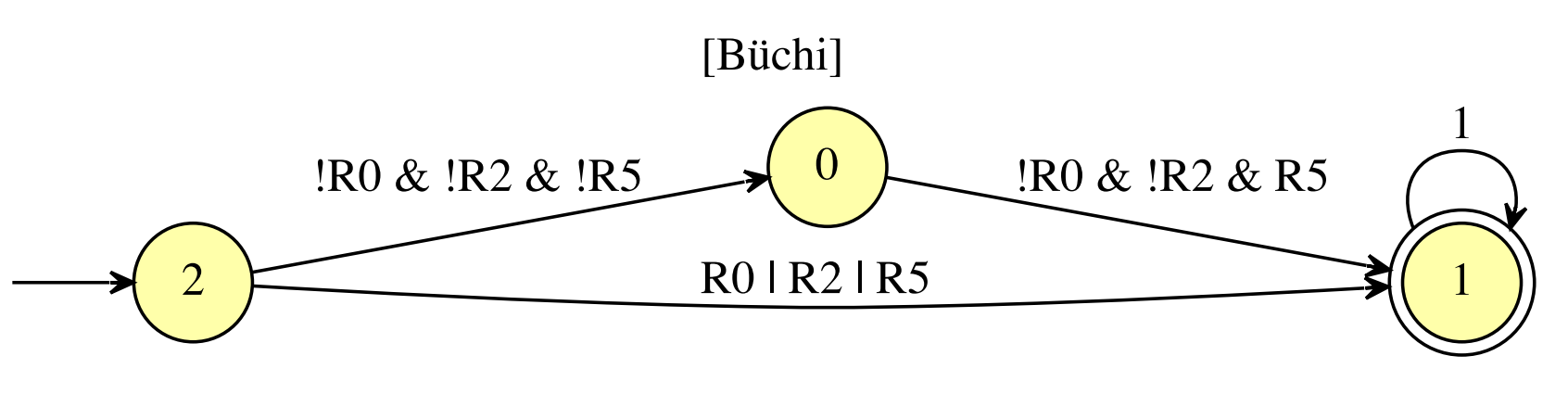


In [4]:

```python
#Immediate Reachability / non-reachability
a2 = spot.formula(' !R0&!R2&!R5 ->X ( !R0&!R2&R5)')
#b2 = spot.translate(a2, 'BA', 'complete');
b2 = spot.translate(a2, 'BA');

print("b2 is safety automaton: ",spot.is_safety_automaton(b2))
print("b2 is liveness automaton:",spot.is_liveness_automaton(b2))

print("a2 is safety: ",a2.is_syntactic_safety())
print("a2 is liveness: ",spot.is_liveness(a2))

b2
```

```
b2 is safety automaton:  True
b2 is liveness automaton: False
a2 is safety:  True
a2 is liveness:  False
```

Out[4]:

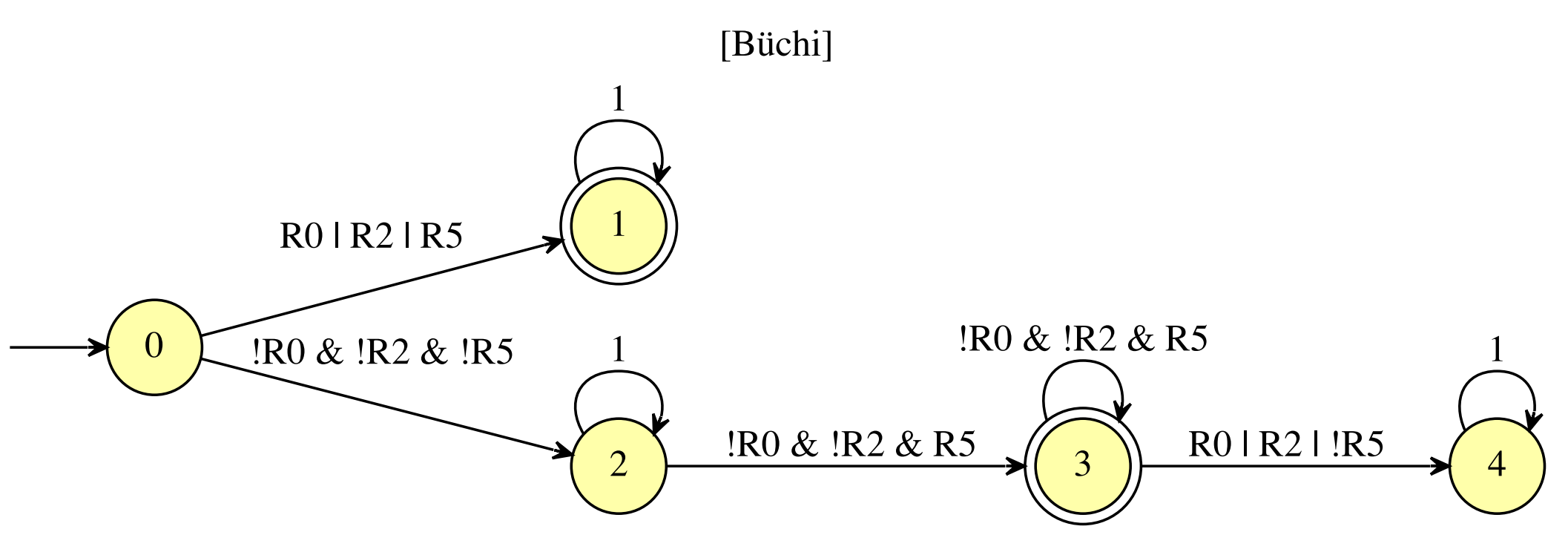


In [5]:

```python
#Conditional Reachability
a3 = spot.formula(' !R0&!R2&!R5 ->F ( (R0)V(!R0&R2)V(!R0&!R2&R5))')
b3 = spot.translate(a3, 'BA', 'complete');
#b3 = spot.translate(a3, 'BA');

print("b3 is safety automaton: ",spot.is_safety_automaton(b3))
print("b3 is liveness automaton:",spot.is_liveness_automaton(b3))

print("a3 is safety: ",a3.is_syntactic_safety())
print("a3 is liveness: ",spot.is_liveness(a3))

b3
```

```
b3 is safety automaton:  False
b3 is liveness automaton: True
a3 is safety:  False
a3 is liveness:  True
```

Out[5]:

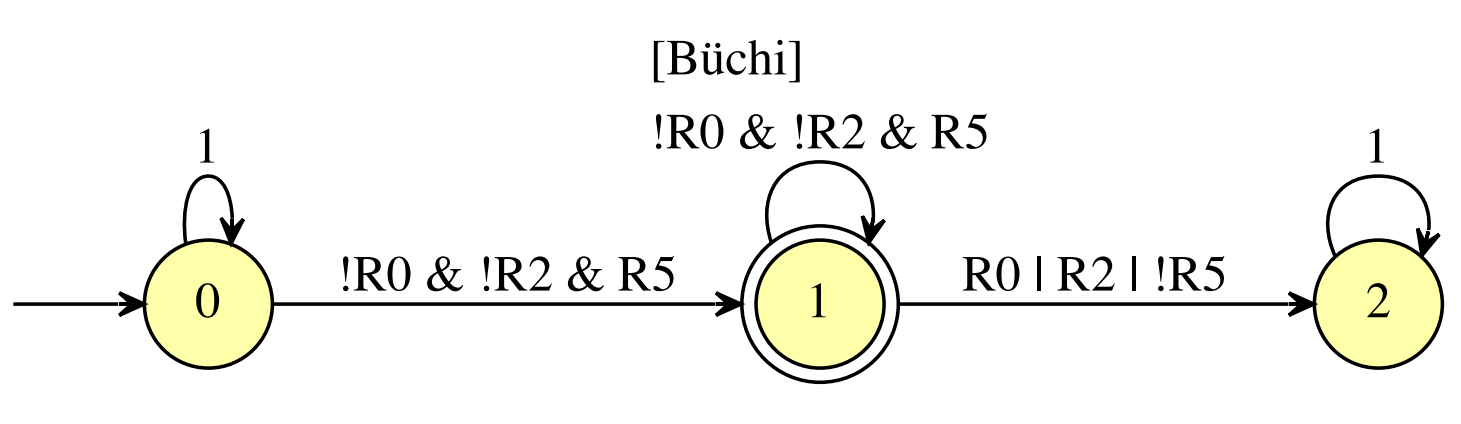


In [6]:

```python
#Always in Future / After
a4 = spot.formula(' F (G( !R0&!R2&R5))')
b4 = spot.translate(a4, 'BA', 'complete');
#b4 = spot.translate(a4, 'BA');

print("b4 is safety automaton: ",spot.is_safety_automaton(b4))
print("b4 is liveness automaton:",spot.is_liveness_automaton(b4))

print("a4 is safety: ",a4.is_syntactic_safety())
print("a4 is liveness: ",spot.is_liveness(a4))

b4
```

```
b4 is safety automaton:  False
b4 is liveness automaton: True
a4 is safety:  False
a4 is liveness:  True
```

Out[6]:

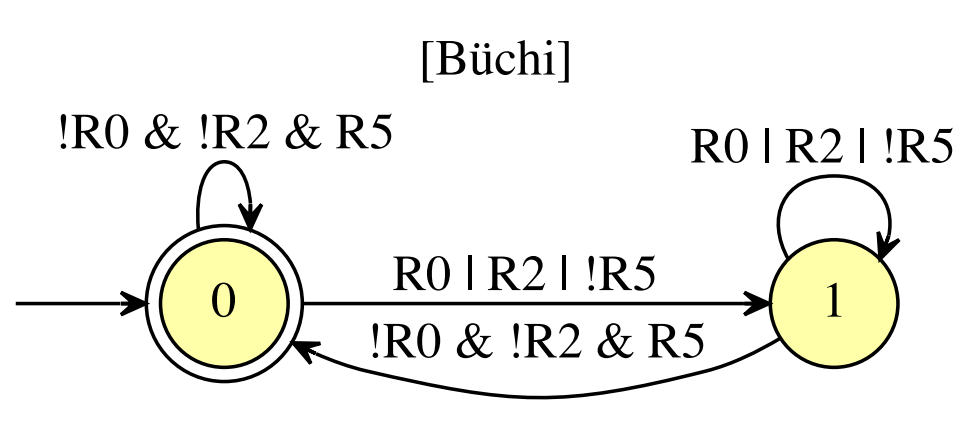


In [7]:

```python
#Not sure what this is. but...
a5 = spot.formula(' G (F( !R0&!R2&R5))')
b5 = spot.translate(a5, 'BA', 'complete');
#b5 = spot.translate(a5, 'BA');

print("b5 is safety automaton: ",spot.is_safety_automaton(b5))
print("b5 is liveness automaton:",spot.is_liveness_automaton(b5))

print("a5 is safety: ",a5.is_syntactic_safety())
print("a5 is liveness: ",spot.is_liveness(a5))

b5
```

```
b5 is safety automaton:  True
b5 is liveness automaton: True
a5 is safety:  False
a5 is liveness:  True
```

Out[7]:

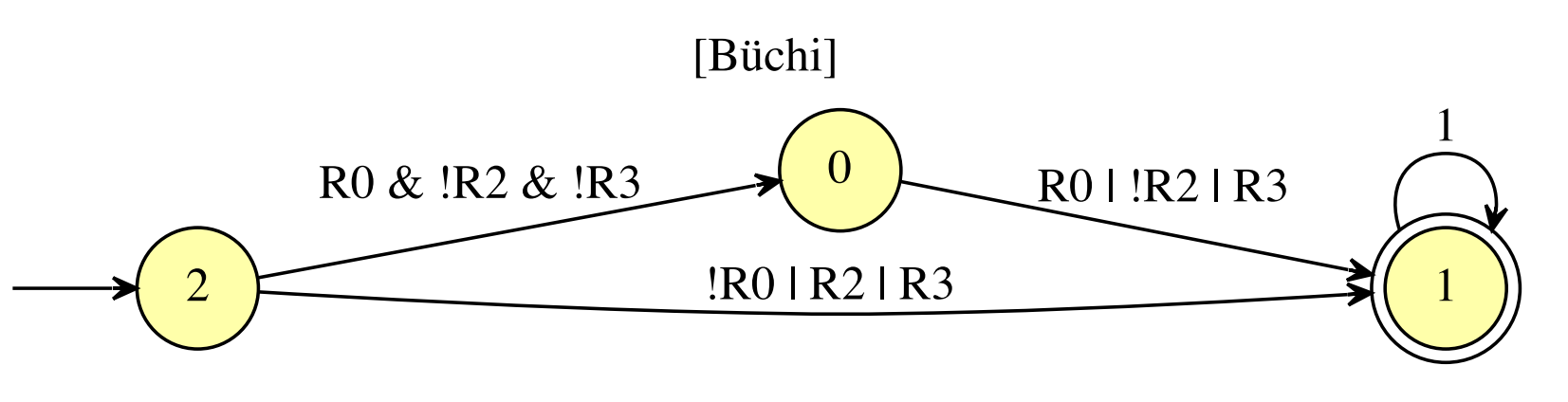


In [8]:

```python
#Chain Events 1
a6 = spot.formula(' R0&!R2&!R3 -> X(!R0&R2&!R3)->X(!R0&!R2&R3)')
#b6 = spot.translate(a6, 'BA', 'complete');
b6 = spot.translate(a6, 'BA');

print("b6 is safety automaton: ",spot.is_safety_automaton(b6))
print("b6 is liveness automaton:",spot.is_liveness_automaton(b6))

print("a6 is safety: ",a6.is_syntactic_safety())
print("a6 is liveness: ",spot.is_liveness(a6))

b6
```

```
b6 is safety automaton:  True
b6 is liveness automaton: False
a6 is safety:  True
a6 is liveness:  False
```

Out[8]:

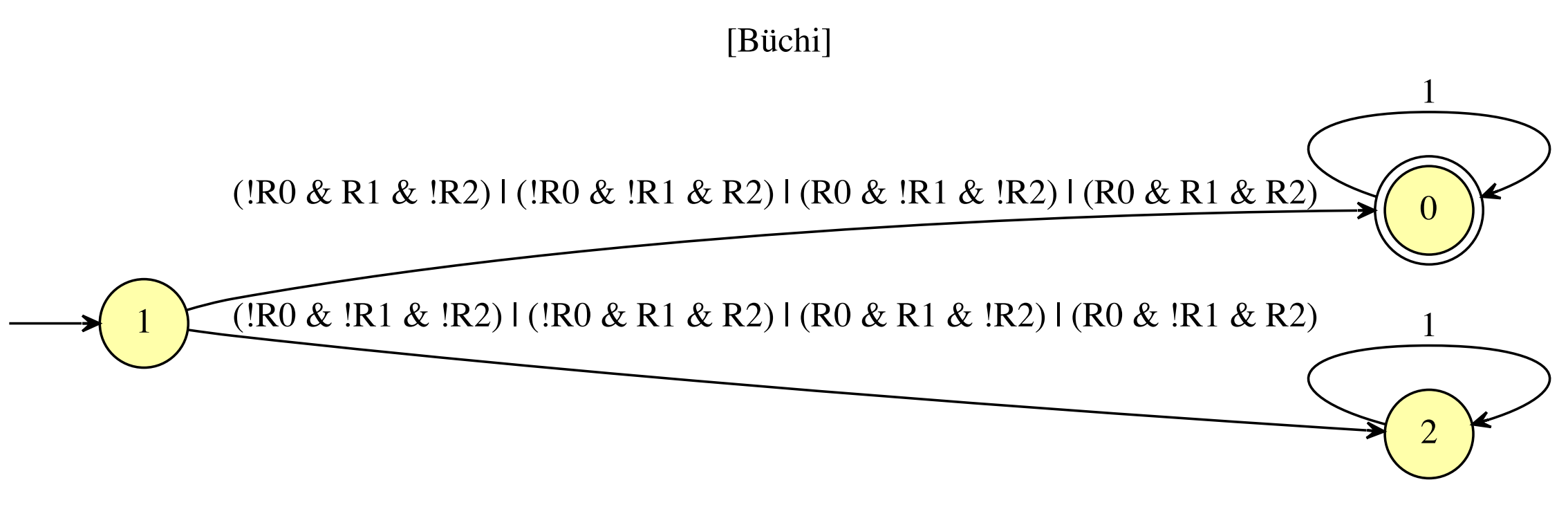


In [9]:

```python
#Service property like rate 1 xor rate2 xor rate3
#Service Property like a user is either permanent user or guest but not both
a7 = spot.formula(' R0 xor R1 xor R2')
b7 = spot.translate(a7, 'BA', 'complete');
#b7 = spot.translate(a7, 'BA');

print("b7 is safety automaton: ",spot.is_safety_automaton(b7))
print("b7 is liveness automaton:",spot.is_liveness_automaton(b7))

print("a7 is safety: ",a7.is_syntactic_safety())
print("a7 is liveness: ",spot.is_liveness(a7))

b7
```

```
b7 is safety automaton:  False
b7 is liveness automaton: False
a7 is safety:  True
a7 is liveness:  False
```

Out[9]:

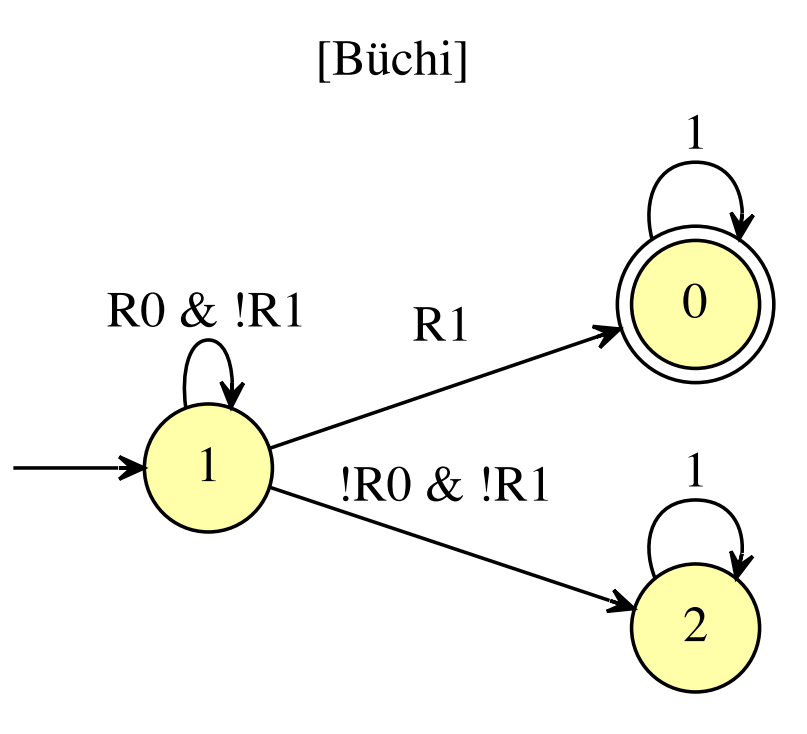


In [10]:

```python
#Chain Events / Sequence / Until
#Eg1: Allow connection until blacklisted
#Eg2: Rate 1 until downgraded to Rate2
#Eg3: Clocked until authentication
#Eg4: FW send to IDPS until whitelisted

#TODO: How to express this in our network model

a8 = spot.formula(' R0 U R1')
b8 = spot.translate(a8, 'BA', 'complete');
#b8 = spot.translate(a8, 'BA');

print("b8 is safety automaton: ",spot.is_safety_automaton(b8))
print("b8 is liveness automaton:",spot.is_liveness_automaton(b8))

print("a8 is safety: ",a8.is_syntactic_safety())
print("a8 is liveness: ",spot.is_liveness(a8))

b8
```

```
b8 is safety automaton:  False
b8 is liveness automaton: False
a8 is safety:  False
a8 is liveness:  False
```

Out[10]:



In [11]:

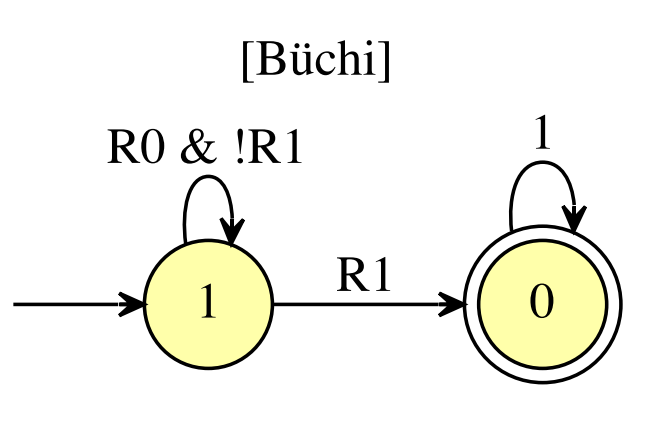
```python
print(b8.to_str())
```

```
HOA: v1
States: 3
Start: 1
AP: 2 "R0" "R1"
acc-name: Buchi
Acceptance: 1 Inf(0)
properties: trans-labels explicit-labels state-acc complete
properties: deterministic stutter-invariant terminal
--BODY--
State: 0 {0}
[t] 0
State: 1
[1] 0
[0&!1] 1
[!0&!1] 2
State: 2
[t] 2
--END--
```

In [12]:

```python
#Safety Decomposer : Every cycle contains an infinite accepting state
#1.Delete all states from which no infinite accepting state is reachable
#2.Make all remaining states infinite accepting states

#Step1
c8=spot.scc_filter(b8)
c8
```

Out[12]:

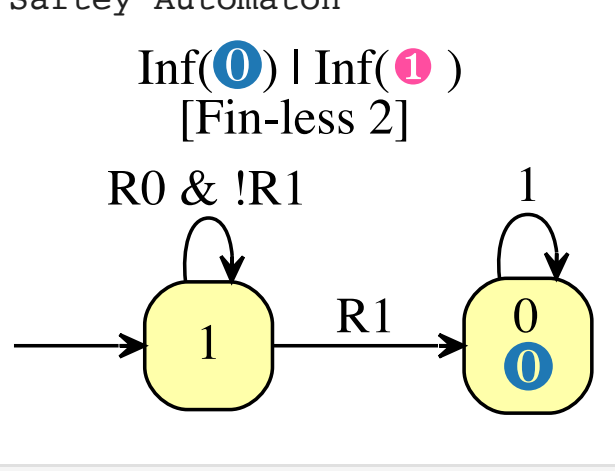


In [13]:

```python
#Step2
c8.set_acceptance(2, "(Inf(0) | Inf(1))");
print("Saftey Automaton")
c8
```

```
Saftey Automaton
```

Out[13]:



In [14]:

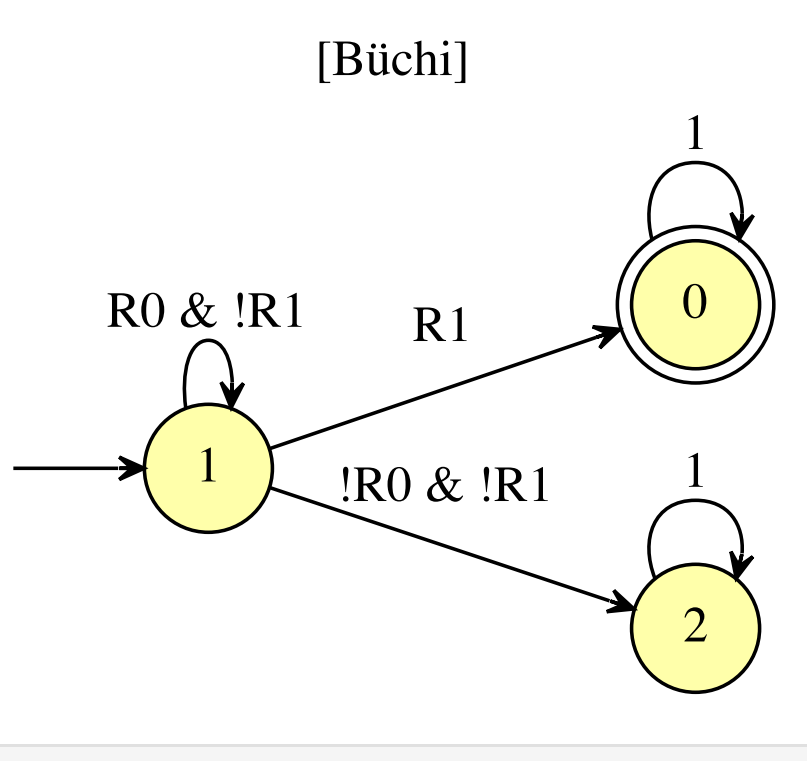
```python
print("Safety Automaton in never claim format that SPIN accepts")
print(c8.to_str("HOA"))
```

```
Safety Automaton in never claim format that SPIN accepts
HOA: v1
States: 2
Start: 1
AP: 2 "R0" "R1"
Acceptance: 2 Inf(0) | Inf(1)
properties: trans-labels explicit-labels state-acc deterministic
properties: stutter-invariant terminal
--BODY--
State: 0 {0}
[t] 0
State: 1
[1] 0
[0&!1] 1
--END--
```

In [15]:

```python
#Liveness Decomposer
#Step1: Delete states from which no infinite-accepting state is reachable
#Step2: Add new infinite accepting state qt that has a transition to itself on all input symbols
#Step3: For every state q that has an undefined transition on any input symbol s, add transition from q to qt under s

#Step1 & 3
#d8=b8
d8=spot.complete(spot.scc_filter(b8))
d8
```

Out[15]:

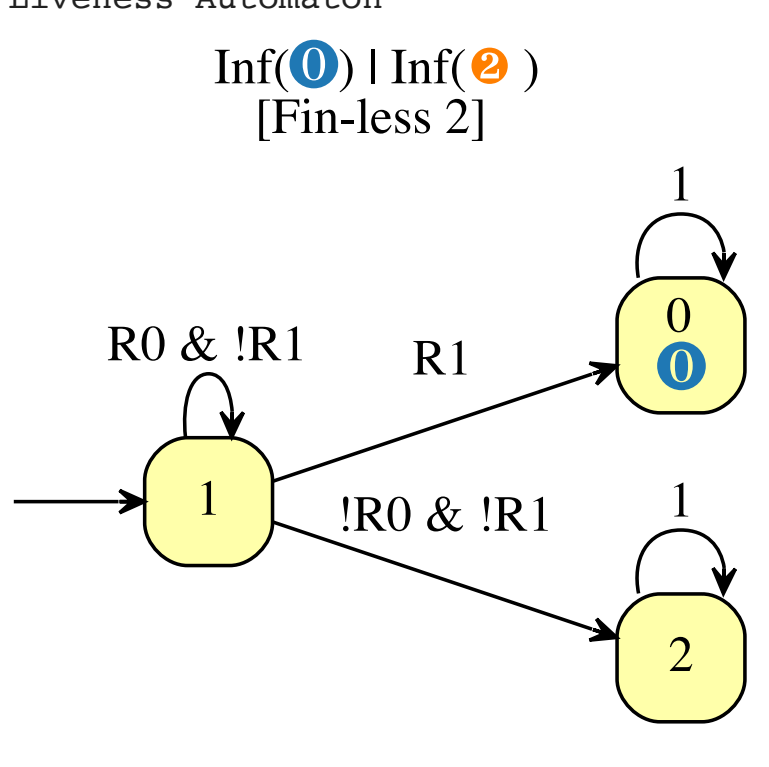


In [16]:

```python
#Step 2
d8.set_acceptance(2, "(Inf(0) | Inf(2))");

print("Liveness Automaton")
d8
```

```
Liveness Automaton
```

Out[16]:



In [17]:

```python
print("Liveness Automaton in never claim format that SPIN accepts")
print(d8.to_str("HOA"))
```

```
Liveness Automaton in never claim format that SPIN accepts
HOA: v1
States: 3
Start: 1
AP: 2 "R0" "R1"
Acceptance: 2 Inf(0) | Inf(2)
properties: trans-labels explicit-labels state-acc complete
properties: deterministic stutter-invariant terminal
--BODY--
State: 0 {0}
[t] 0
State: 1
[1] 0
[0&!1] 1
[!0&!1] 2
State: 2
[t] 2
--END--
```

In [ ]: