

Modeling transitive closure in B/Event-B

Also discussions about first-order and higher-order aspects of B.

```
In [21]: ::load
MACHINE StateGraph
SETS States = {s1,s2,s3}
CONSTANTS next
PROPERTIES
    next = {s1|->s2, s2|->s1, s3|->s3}
END
```

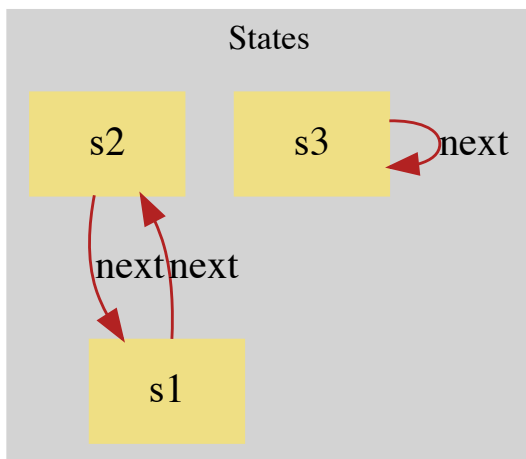
Out[21]: Loaded machine: StateGraph

```
In [22]: :init
```

Out[22]: Machine constants were not set up yet. Automatically set up constants using a
rbitrary transition: SETUP_CONSTANTS()
Executed operation: INITIALISATION()

```
In [23]: :dot expr_as_graph next
```

Out[23]:



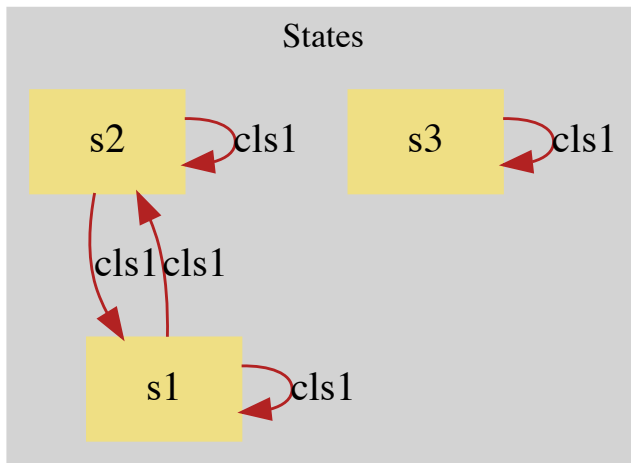
How do we obtain the transitive closure of this relation? In classical B there is the operator `closure1` for this:

```
In [24]: closure1(next)
```

Out[24]: $\{(s1 \mapsto s1), (s1 \mapsto s2), (s2 \mapsto s1), (s2 \mapsto s2), (s3 \mapsto s3)\}$

```
In [25]: :dot expr_as_graph ("cls1",closure1(next))
```

Out [25]:



In Event-B this operator is not available in the core language; one can import a theory for it though. Let us, however, try and axiomatise this ourselves:

In [26]:

```

next ⊆ cls1 & // the next relation is included in the transitive closure
∀s.(s:States ⇒ next[cls1[{s}]] ⊆ cls1[{s}] ) // all successors are also included

```

Out [26]: *TRUE*

Solution:

- $cls1 = \{(s2 \mapsto s1), (s1 \mapsto s2), (s3 \mapsto s3), (s1 \mapsto s1), (s1 \mapsto s3), (s2 \mapsto s2)\}$

As you can see this solution is maybe not what you expected, it contains an edge between s1 and s3. But it satisfies our conditions.

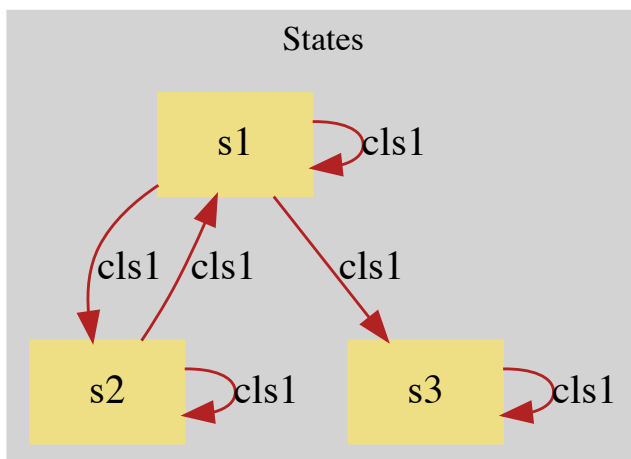
In [27]:

```

:dot expr_as_graph ("cls1", {(s2↦s1),(s1↦s2),(s3↦s3),(s1↦s1),(s1↦s3),(s2↦s2)})

```

Out [27]:



As you can see there are eight solutions to this "axiomatisation" of transitive closure for this graph:

In [28]:

```

:table {cls1 | next ⊆ cls1 & ∀s.(s:States ⇒ next[cls1[{s}]] ⊆ cls1[{s}] )}

```

Out [28]:

cls1

```
{(s1→s1),(s1→s2),(s1→s3),(s2→s1),(s2→s2),(s3→s3)}
{(s1→s1),(s1→s2),(s2→s1),(s2→s2),(s3→s3)}
{(s1→s1),(s1→s2),(s2→s1),(s2→s2),(s2→s3),(s3→s3)}
{(s1→s1),(s1→s2),(s1→s3),(s2→s1),(s2→s2),(s2→s3),(s3→s3)}
{(s1→s1),(s1→s2),(s1→s3),(s2→s1),(s2→s2),(s3→s1),(s3→s2),(s3→s3)}
{(s1→s1),(s1→s2),(s2→s1),(s2→s2),(s3→s1),(s3→s2),(s3→s3)}
{(s1→s1),(s1→s2),(s1→s3),(s2→s1),(s2→s2),(s2→s3),(s3→s1),(s3→s2),(s3→s3)}
{(s1→s1),(s1→s2),(s2→s1),(s2→s2),(s2→s3),(s3→s1),(s3→s2),(s3→s3)}
```

Can we encode transitive closure in B?

What we want is the smallest relation satisfying our axioms. In higher-order logic we can quantify over predicates, and for example ask that we want the smallest solution.

Transitive closure can actually not be axiomatised in first-order logic. As B is based on first-order logic, can we do this without resorting to the built-in operator `closure1`? The answer is yes, because B has higher-order values and we can arbitrarily quantify over sets and relation values. We have to specify that all other relations contained in `cls1` are not a solution. With this we get a single solution, encoding our expected transitive closure of the next relation:

In [29]:

```
:table {cls1 |
    next ⊆ cls1 & ∀s.(s:States ⇒ next[cls1[{s}]] ⊆ cls1[{s}]) //
    & // all smaller relations do not satisfy our axioms:
    ∀other.(other ⊂ cls1 & next ⊆ other ⇒
        ¬ ( ∀s.(s:States ⇒ next[other[{s}]] ⊆ other[{s}]) )
    )
}
```

Out [29]:

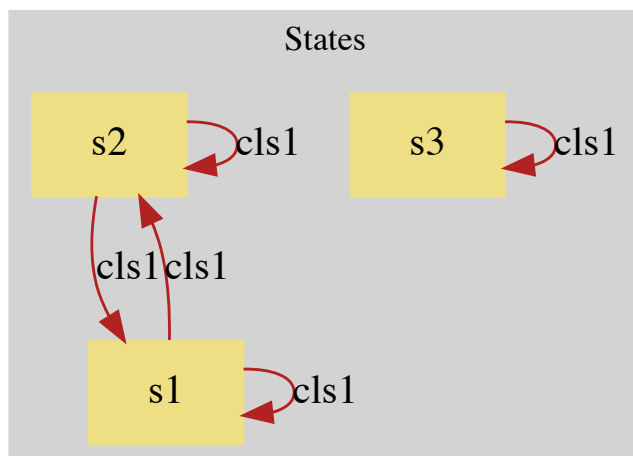
cls1

```
{(s1→s1),(s1→s2),(s2→s1),(s2→s2),(s3→s3)}
```

In [30]:

```
:dot expr_as_graph ("cls1", {(s1→s1),(s1→s2),(s2→s1),(s2→s2),(s3→s3)})
```

Out [30]:



Digression: Encoding transitive closure in Prolog

In Prolog we can encode transitive closure as this:

```
cls1(A,B) :- next(A,B).  
cls1(A,B) :- next(A,C), cls1(C,B).
```

So even though Prolog is built on top of first-order logic, it has a "minimization" built-in: the semantics of the above Prolog program are not **all** models satisfying the clauses, but the minimal Herbrand model, which here encodes exactly the transitive closure.

Note that the Clark completion is an attempt at modelling the semantics of Prolog programs in first-order logic. The Clark completion translates the implications of the two clauses above into a single equivalence formula of the form:

```
In [31]: 
$$\forall(A,B). (A:\text{States} \ \& \ B:\text{States} \Rightarrow$$
  
          
$$(A \mapsto B : \text{cls1}$$
  
          
$$\Leftrightarrow$$
  
          
$$(A \mapsto B : \text{next}$$
  
          
$$\text{or}$$
  
          
$$\exists C. (C:\text{States} \ \& \ A \mapsto C:\text{next} \ \& \ C \mapsto B:\text{cls1})$$
  
          
$$)$$
  
          
$$))$$

```

Out[31]: *TRUE*

Solution:

- $\text{cls1} = \{(s1 \mapsto s2), (s2 \mapsto s1), (s1 \mapsto s1), (s2 \mapsto s2), (s3 \mapsto s3)\}$

As our initial attempt at axiomatising the transitive closure, this encoding admits eightmany solutions:

```
In [32]: :table {cls1|  
          
$$\forall(A,B). (A:\text{States} \ \& \ B:\text{States} \Rightarrow$$
  
          
$$(A \mapsto B : \text{cls1}$$
  
          
$$\Leftrightarrow$$
  
          
$$(A \mapsto B : \text{next}$$
  
          
$$\text{or}$$
  
          
$$\exists C. (C:\text{States} \ \& \ A \mapsto C:\text{next} \ \& \ C \mapsto B:\text{cls1})$$
  
          
$$)$$
  
          
$$))\}$$

```

Out [32]:

cls1

```
{(s1→s1),(s1→s2),(s2→s1),(s2→s2),(s3→s3)}
{(s1→s1),(s1→s2),(s2→s1),(s2→s2),(s3→s1),(s3→s3)}
{(s1→s1),(s1→s2),(s2→s1),(s2→s2),(s3→s2),(s3→s3)}
{(s1→s1),(s1→s2),(s1→s3),(s2→s1),(s2→s2),(s2→s3),(s3→s3)}
{(s1→s1),(s1→s2),(s1→s3),(s2→s1),(s2→s2),(s2→s3),(s3→s1),(s3→s3)}
{(s1→s1),(s1→s2),(s1→s3),(s2→s1),(s2→s2),(s2→s3),(s3→s2),(s3→s3)}
{(s1→s1),(s1→s2),(s2→s1),(s2→s2),(s3→s1),(s3→s2),(s3→s3)}
{(s1→s1),(s1→s2),(s1→s3),(s2→s1),(s2→s2),(s2→s3),(s3→s1),(s3→s2),(s3→s3)}
```

ASCII versions of formulas

In [33]:

```
:prettyprint next <: cls1 & // the next relation is included in the transitive
!s.(s:States => next[cls1[{s}]] <: cls1[{s}] ) // all successors are also included
```

Out [33]:

$next \subseteq cls1 \wedge \forall s \cdot (s \in States \Rightarrow next[cls1[\{s\}]] \subseteq cls1[\{s\}])$

In [34]:

```
:prettyprint ∀(A,B).( A:States & B:States =>
    (A|→B : cls1
    <=>
    (A|→B : next
    or
    #C.(C:States & A|→C:next & C|→B:cls1)
    )
    ))
```

Out [34]:

$\forall(A,B) \cdot (A \in States \wedge B \in States \Rightarrow (A \mapsto B \in cls1) \Leftrightarrow (A \mapsto B \in next \vee \exists C \cdot (C \in$

In [35]:

```
:prettyprint next ⊆ cls1 & ∀s.(s:States => next[cls1[{s}]] ⊆ cls1[{s}]) //
& // all smaller relations do not satisfy our axioms:
∀other.(other <<: cls1 & next ⊆ other =>
    not ( ∀s.(s:States => next[other[{s}]] <: other[{s}] )
    )
```

Out [35]:

$next \subseteq cls1 \wedge \forall s \cdot (s \in States \Rightarrow next[cls1[\{s\}]] \subseteq cls1[\{s\}]) \wedge \forall other \cdot (other \subset cls1,$

In []: