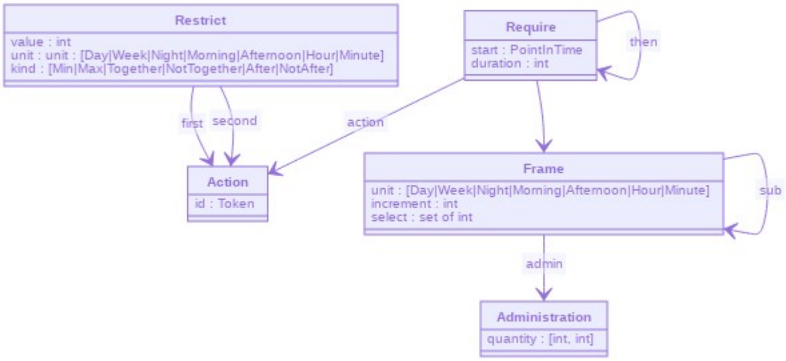
```
(ns chaos.core-test
  (:require [chaos.core :refer :all]
            [chaos.gts :refer :all]
            [chaos.parser :refer :all]))

(use 'grape.core)
```

```
nil
```

Relative Temporal Frame (RTF) Meta Model

The meta model for RTF structures is shown below:

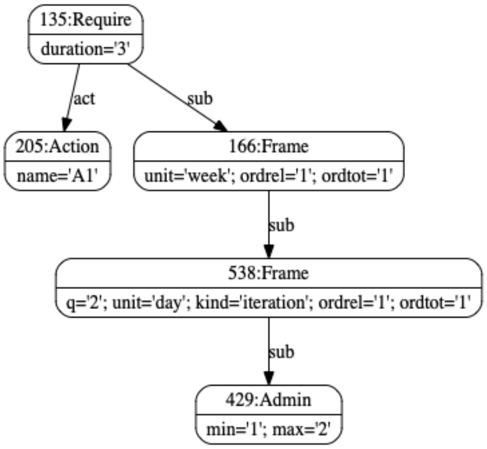


```
(parse "require A1 for 3 weeks every 2nd day administer 1-2")
```

```
{:tag :prescription, :content ({:tag :requirement, :content ({:tag :action, :content ("A1")} {:tag
:duration, :content ({:tag :number, :content ("3")} {:tag :unit, :content ("weeks")})} {:tag :frame,
:content ({:tag :iteration, :content ({:tag :number, :content ("2")} {:tag :unit, :content ("day")})}
{:tag :administer, :content ({:tag :number, :content ("1")} {:tag :number, :content ("2")})})})})}
```

Create an RTF Graph for a prescription

```
(clear!)
(-> (parse "require A1 for 3 weeks every 2nd day administer 1-2")
  createRTF!)
(browse)
```



Transform the RTF Graph into an STN

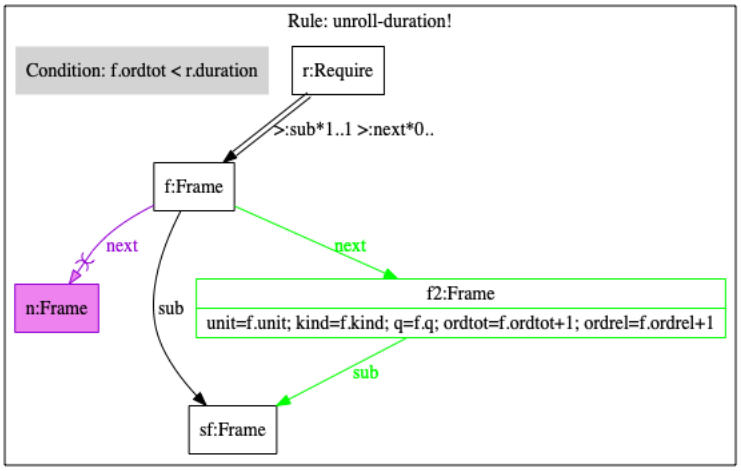
STN Graphs are simply Time Point (TP) nodes connected by eighted edges. The conversion from RTF to STN is done in 7 steps:

1. **Unroll Duration**: we unroll the top level *Frame* for the duration specified in the requirement
2. **Split SubFrame**: we split the subframes of the "unrolled" frames, so that each unrolled frame has its own subframe
3. **Unroll Frame**: we unroll Frames by creating the right number if subframes
4. **Connect Subframes**: we connect the ends of terminal subframes to create a cohesive sequence of subframes
5. **Remove Parent Frame**: we remove the parent frames (which are not longer needed) (this step is not really needed - but makes the graph less cluttered)
6. **Filter Iteration**: this step filters out only those frames where an event needs to happen
7. **Create TimePoints**: create TPs for each filtered frame

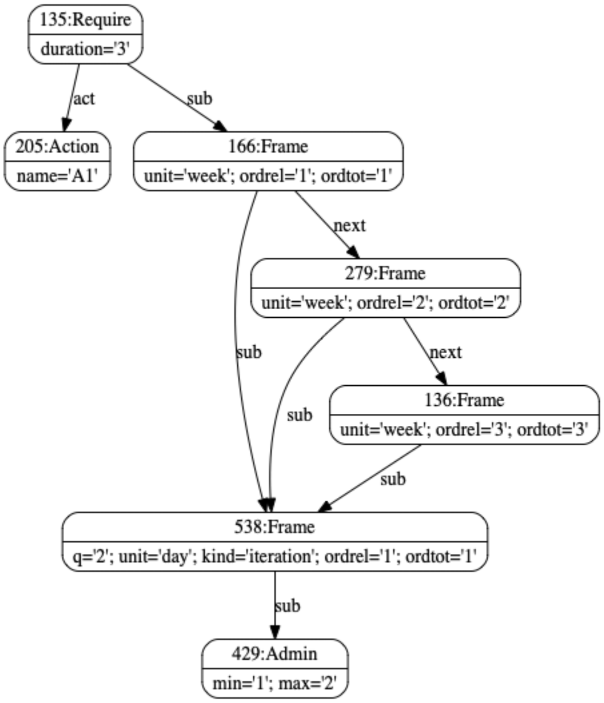
1. Unroll Duration

This step replicates the "top frame" for *duration* times

```
(rule 'unroll-duration!
      :read (pattern (node 'r :label "Require")
                     (node 'f :label "Frame")
                     (path ">:sub*1..1 >:next*0.." :src 'r :tar 'f)
                     (NAC
                      (node 'n :label "Frame")
                      (edge :label "next" :src 'f :tar 'n)
                      )
                     (condition "f.ordtot < r.duration")
                     (node 'sf :label "Frame")
                     (edge :label "sub" :src 'f :tar 'sf)
                     )
      :create (pattern
                (node 'f2 :label "Frame" :asserts {:unit "f.unit" :kind "f.kind" :q "f.q"
:ordtot "f.ordtot+1" :ordrel "f.ordrel+1"})
                (edge :label "next" :src 'f :tar 'f2)
                (edge :label "sub" :src 'f2 :tar 'sf)
               ))
```


Rule: unroll-duration!
Condition: f.ordtot < r.duration
r:Require
>:sub*1..1 >:next*0..
f:Frame
next
next
n:Frame
sub
f2:Frame
unit=f.unit; kind=f.kind; q=f.q; ordtot=f.ordtot+1; ordrel=f.ordrel+1
sub
sf:Frame

```
(while (unroll-duration!))
(browse)
```

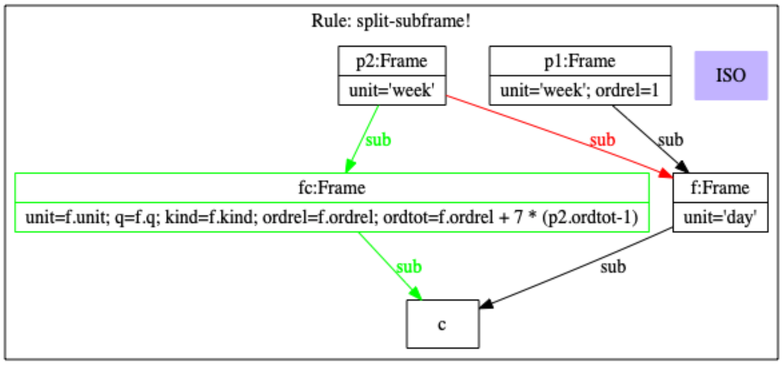

135:Require
duration='3'
act
sub
205:Action
name='A1'
166:Frame
unit='week'; ordrel='1'; ordtot='1'
next
279:Frame
unit='week'; ordrel='2'; ordtot='2'
sub
sub
next
136:Frame
unit='week'; ordrel='3'; ordtot='3'
sub
538:Frame
q='2'; unit='day'; kind='iteration'; ordrel='1'; ordtot='1'
sub
429:Admin
min='1'; max='2'

2. Split Subframe

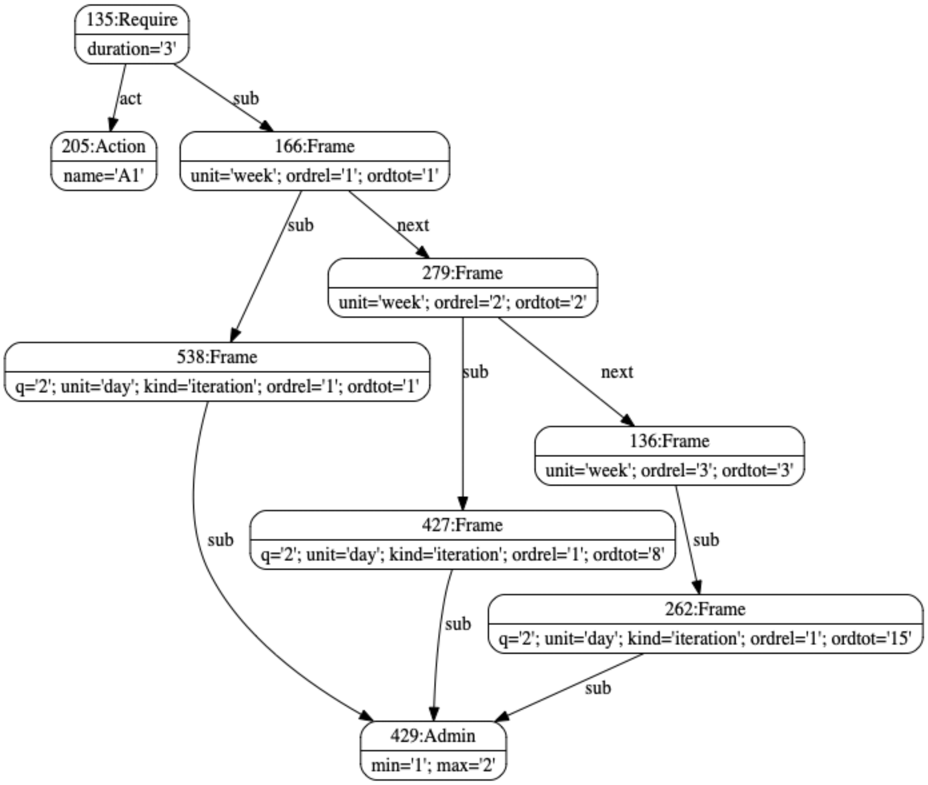
This step splits the *sub* frames.

Note: right now this is implemented for "weeks" only. Todo: implement a more generic version of this operation for other units.

```
(rule 'split-subframe!
      :read (pattern :iso
             (node 'p1 :label "Frame" :asserts {:unit "'week'" :ordrel "1"})
             (node 'p2 :label "Frame" :asserts {:unit "'week'"})
             (node 'f :label "Frame" :asserts {:unit "'day'"})
             (node 'c)
             (edge :label "sub" :src 'p1 :tar 'f)
             (edge 's :label "sub" :src 'p2 :tar 'f)
             (edge :label "sub" :src 'f :tar 'c)
                     )
      :delete ['s]
      :create (pattern
                (node 'fc :label "Frame" :asserts {:unit "f.unit" :q "f.q" :kind "f.kind" :ordrel "f.ordrel" :ordtot "f.ordrel + 7 * (p2.ordtot-1)"})
                (edge :label "sub" :src 'fc :tar 'c)
                (edge 'sn :label "sub" :src 'p2 :tar 'fc)
               ))
```



```
(while (split-subframe!))
(browse)
```

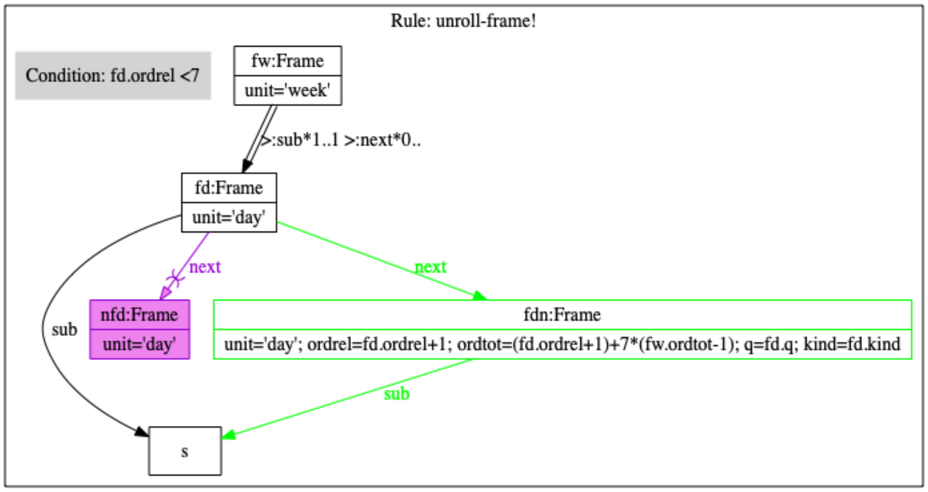

135:Require
duration='3'
act
sub
205:Action
name='A1'
166:Frame
unit='week'; ordrel='1'; ordtot='1'
sub
next
279:Frame
unit='week'; ordrel='2'; ordtot='2'
538:Frame
q='2'; unit='day'; kind='iteration'; ordrel='1'; ordtot='1'
sub
next
136:Frame
unit='week'; ordrel='3'; ordtot='3'
427:Frame
q='2'; unit='day'; kind='iteration'; ordrel='1'; ordtot='8'
sub
sub
262:Frame
q='2'; unit='day'; kind='iteration'; ordrel='1'; ordtot='15'
sub
sub
429:Admin
min='1'; max='2'

3. Unroll Frame

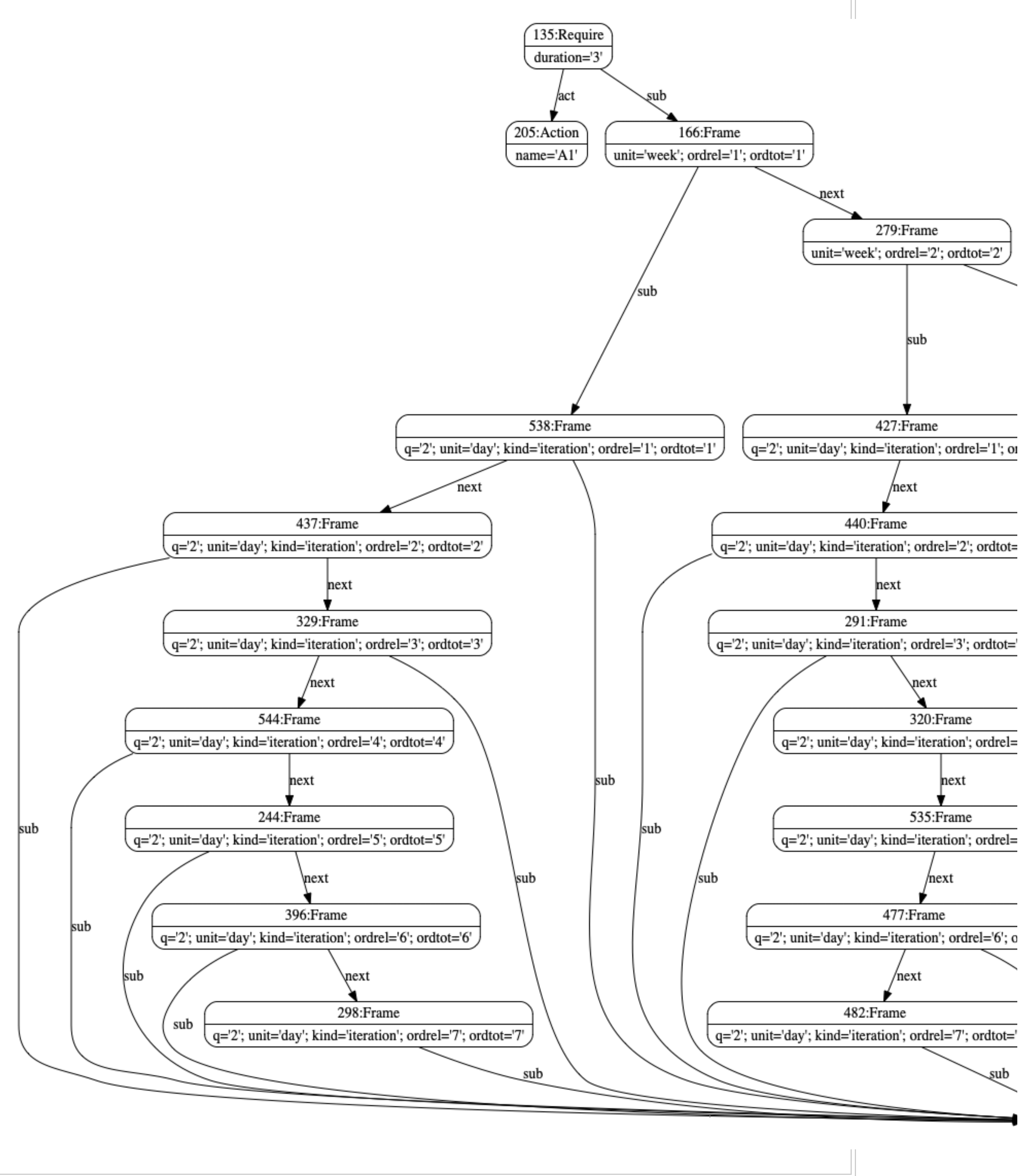
After splitting the sub-frame, we unroll it.

Note: this is currently only done for _week frames (that have *day* subframes). Todo: general implementation._

```
(rule 'unroll-frame!
      :read (pattern
             (node 'fw :label "Frame" :asserts {:unit "'week'"})
             (node 'fd :label "Frame" :asserts {:unit "'day'"})
             (path ">:sub*1..1 >:next*0.." :src 'fw :tar 'fd)
             (node 's)
             (edge :label "sub" :src 'fd :tar 's)
             (NAC
              (node 'nfd :label "Frame" :asserts {:unit "'day'"})
              (edge :label "next" :src 'fd :tar 'nfd)
              )
             (condition "fd.ordrel <7")
             )
      :create (pattern
               (node 'fdn :label "Frame" :asserts {:unit "'day'" :ordrel "fd.ordrel+1" :ordtot "(fd.ordrel+1)+7*(fw.ordtot-1)" :q "fd.q" :kind "fd.kind"})
               (edge :label "next" :src 'fd :tar 'fdn)
               (edge :label "sub" :src 'fdn :tar 's)
               ))
```



```
(while (unroll-frame!))
(browse)
```

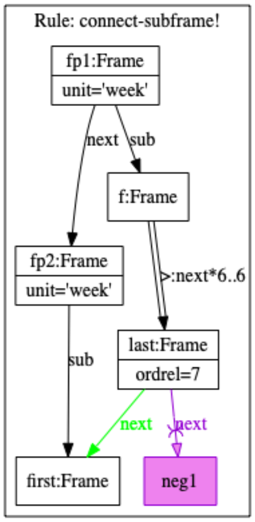

135:Require
duration='3'
act
sub
205:Action
name='A1'
166:Frame
unit='week'; ordrel='1'; ordtot='1'
next
279:Frame
unit='week'; ordrel='2'; ordtot='2'
sub
sub
538:Frame
q='2'; unit='day'; kind='iteration'; ordrel='1'; ordtot='1'
427:Frame
q='2'; unit='day'; kind='iteration'; ordrel='1'; o
next
next
437:Frame
q='2'; unit='day'; kind='iteration'; ordrel='2'; ordtot='2'
440:Frame
q='2'; unit='day'; kind='iteration'; ordrel='2'; ordtot=
next
next
329:Frame
q='2'; unit='day'; kind='iteration'; ordrel='3'; ordtot='3'
291:Frame
q='2'; unit='day'; kind='iteration'; ordrel='3'; ordtot='
next
next
544:Frame
q='2'; unit='day'; kind='iteration'; ordrel='4'; ordtot='4'
320:Frame
q='2'; unit='day'; kind='iteration'; ordrel=
next
next
244:Frame
q='2'; unit='day'; kind='iteration'; ordrel='5'; ordtot='5'
535:Frame
q='2'; unit='day'; kind='iteration'; ordrel=
next
next
396:Frame
q='2'; unit='day'; kind='iteration'; ordrel='6'; ordtot='6'
477:Frame
q='2'; unit='day'; kind='iteration'; ordrel='6'; o
next
next
298:Frame
q='2'; unit='day'; kind='iteration'; ordrel='7'; ordtot='7'
482:Frame
q='2'; unit='day'; kind='iteration'; ordrel='7'; ordtot='
sub
sub
sub
sub
sub
sub
sub
sub
sub
sub

4. Connect Sub Frames!

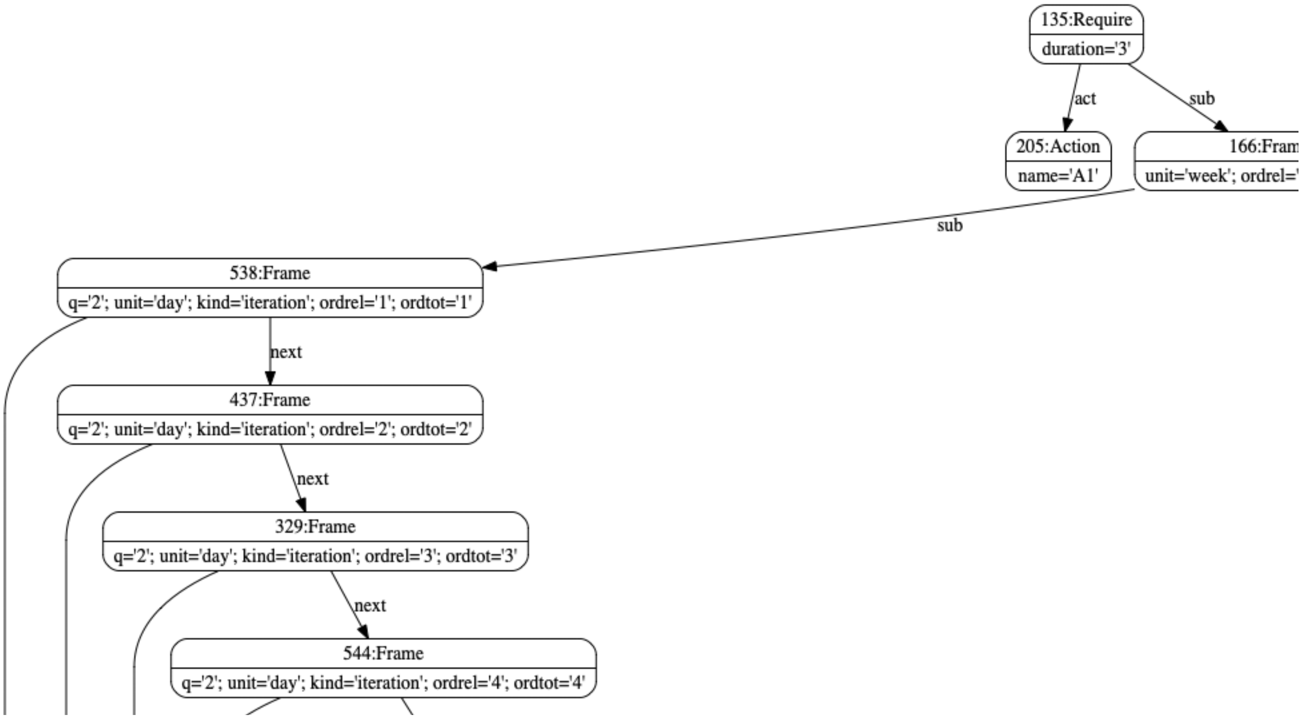
After unrolling a frame, we connect the end-points of it's sub-frames.

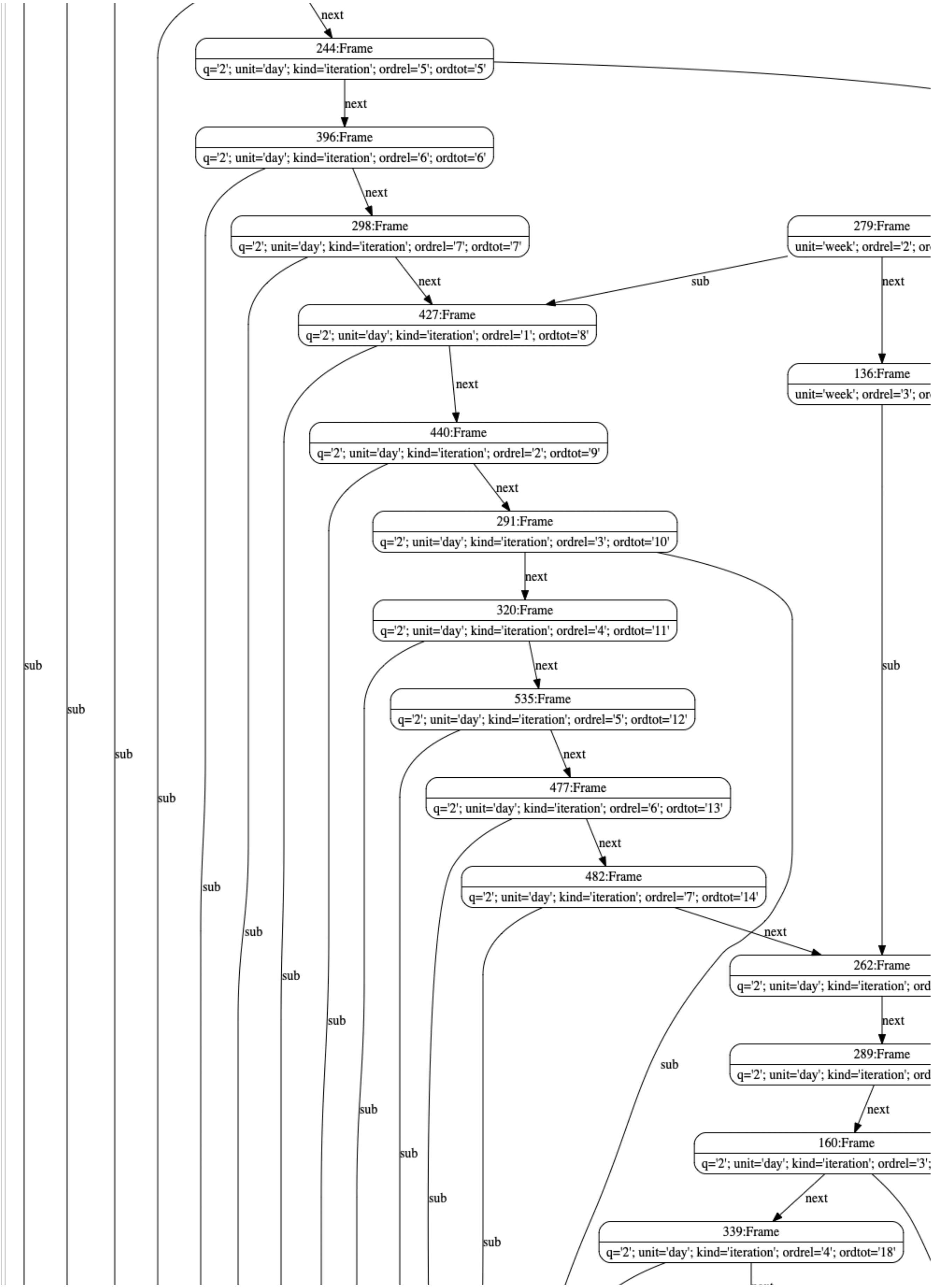
Note: this is currently only implemented for _day subframes. Todo: general implementation._

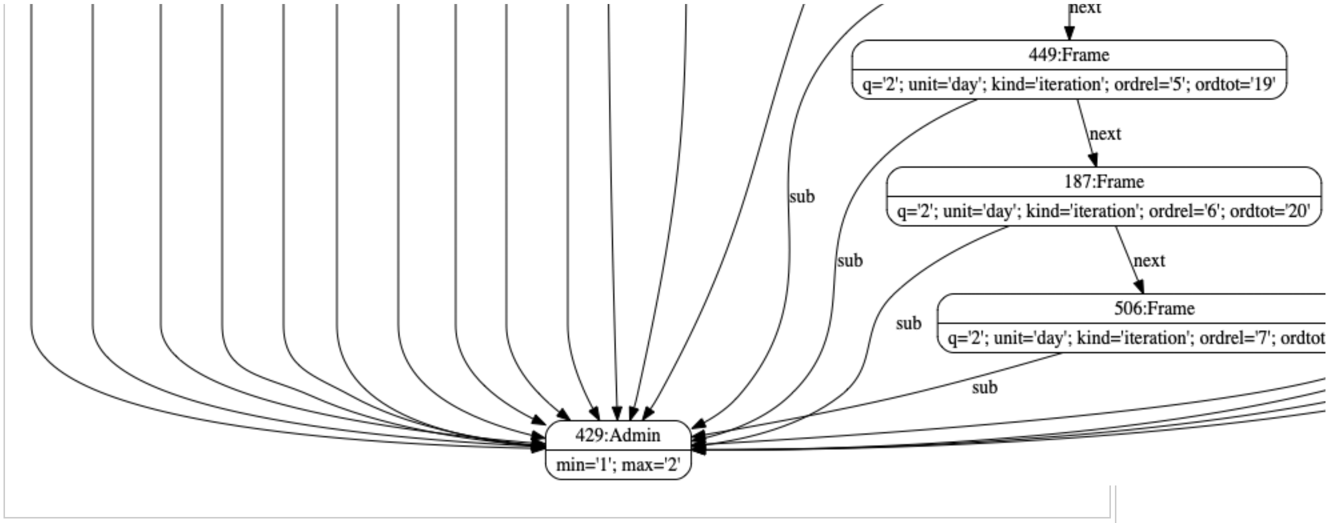
```
(rule 'connect-subframe!
  :read (pattern
          (node 'fp1 :label "Frame" :asserts {:unit "'week'"})
          (node 'fp2 :label "Frame" :asserts {:unit "'week'"})
          (edge :label "next" :src 'fp1 :tar 'fp2)
          (node 'last :label "Frame" :asserts {:ordrel "7"})
          (node 'f :label "Frame")
          (edge :label "sub" :src 'fp1 :tar 'f)
          (node 'first :label "Frame")
          (path ">:next*6..6" :src 'f :tar 'last)
          (edge :label "sub" :src 'fp2 :tar 'first)
          (NAC
            (node 'neg1)
            (edge :label "next" :src 'last :tar 'neg1))
          )
  :create (pattern
            (edge :label "next" :src 'last :tar 'first)))
```



```
(while (connect-subframe!))
(browse)
```


135:Require
duration='3'
act
sub
205:Action
name='A1'
166:Fram
unit='week'; ordrel='
sub
538:Frame
q='2'; unit='day'; kind='iteration'; ordrel='1'; ordtot='1'
next
437:Frame
q='2'; unit='day'; kind='iteration'; ordrel='2'; ordtot='2'
next
329:Frame
q='2'; unit='day'; kind='iteration'; ordrel='3'; ordtot='3'
next
544:Frame
q='2'; unit='day'; kind='iteration'; ordrel='4'; ordtot='4'


next
244:Frame
q='2'; unit='day'; kind='iteration'; ordrel='5'; ordtot='5'
next
396:Frame
q='2'; unit='day'; kind='iteration'; ordrel='6'; ordtot='6'
next
298:Frame
q='2'; unit='day'; kind='iteration'; ordrel='7'; ordtot='7'
279:Frame
unit='week'; ordrel='2'; or
next
sub
next
427:Frame
q='2'; unit='day'; kind='iteration'; ordrel='1'; ordtot='8'
next
136:Frame
unit='week'; ordrel='3'; or
440:Frame
q='2'; unit='day'; kind='iteration'; ordrel='2'; ordtot='9'
next
291:Frame
q='2'; unit='day'; kind='iteration'; ordrel='3'; ordtot='10'
next
320:Frame
q='2'; unit='day'; kind='iteration'; ordrel='4'; ordtot='11'
sub
next
sub
535:Frame
q='2'; unit='day'; kind='iteration'; ordrel='5'; ordtot='12'
sub
next
sub
477:Frame
q='2'; unit='day'; kind='iteration'; ordrel='6'; ordtot='13'
next
482:Frame
q='2'; unit='day'; kind='iteration'; ordrel='7'; ordtot='14'
sub
sub
next
262:Frame
q='2'; unit='day'; kind='iteration'; ord
sub
next
sub
289:Frame
q='2'; unit='day'; kind='iteration'; ord
sub
next
sub
160:Frame
q='2'; unit='day'; kind='iteration'; ordrel='3';
sub
sub
next
339:Frame
q='2'; unit='day'; kind='iteration'; ordrel='4'; ordtot='18'
sub

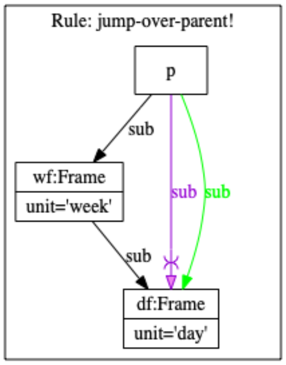

next
449:Frame
q='2'; unit='day'; kind='iteration'; ordrel='5'; ordtot='19'
next
187:Frame
q='2'; unit='day'; kind='iteration'; ordrel='6'; ordtot='20'
next
506:Frame
q='2'; unit='day'; kind='iteration'; ordrel='7'; ordtot
sub
sub
sub
sub
429:Admin
min='1'; max='2'

5. Remove Parent Frame

Now we can remove the parent frames.

Note: this is currently only implemented for parent frames of unit "week". Todo: general implementation.

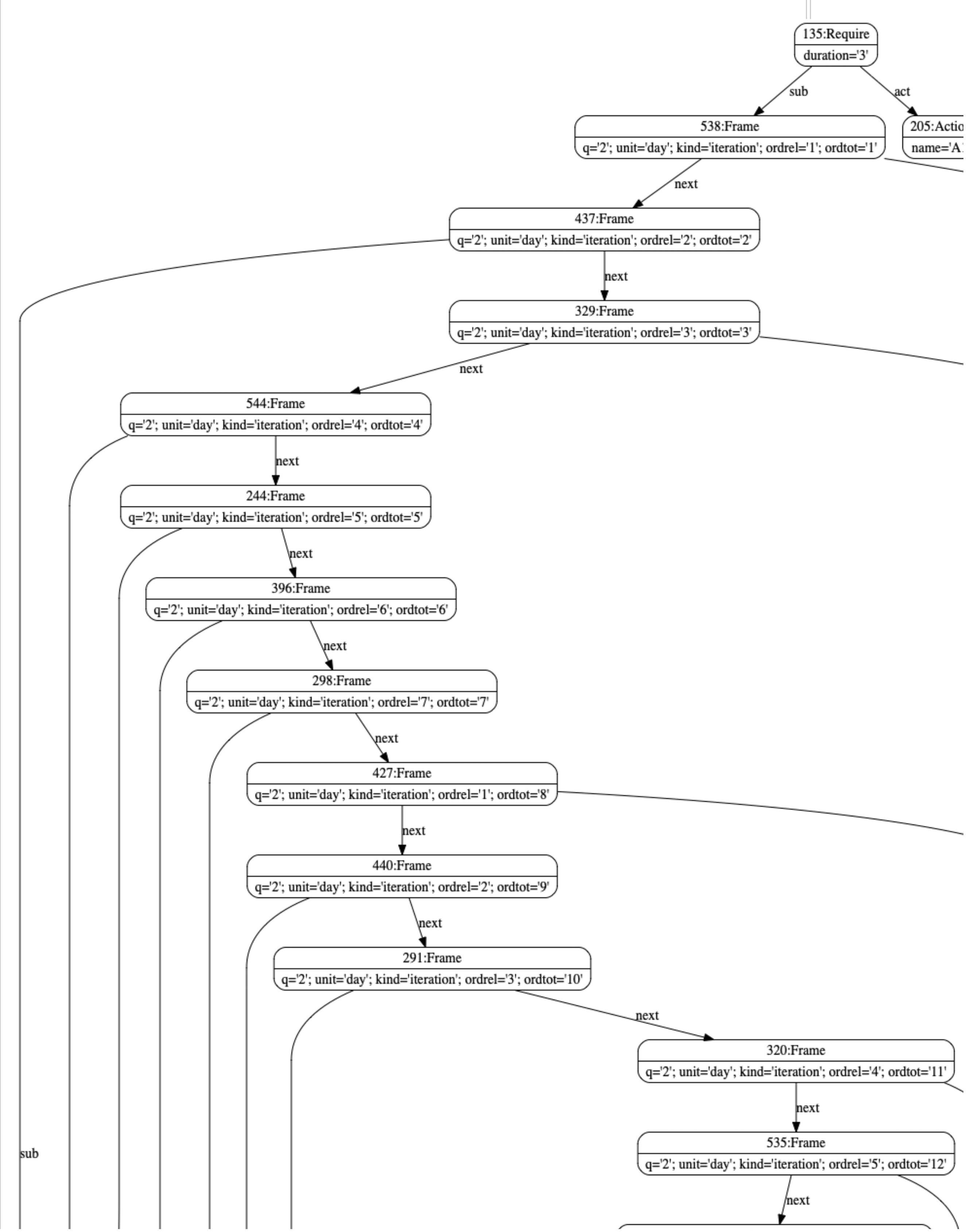
```
(rule 'jump-over-parent!
      :read (pattern
             (node 'p)
             (node 'wf :label "Frame" :asserts {:unit "'week'"})
             (edge :label "sub" :src 'p :tar 'wf)
             (edge :label "sub" :src 'wf :tar 'df)
             (node 'df :label "Frame" :asserts {:unit "'day'"})
             (NAC
              (edge :label "sub" :src 'p :tar 'df)
              ))
      :create (pattern
               (edge :label "sub" :src 'p :tar 'df))
      )
```

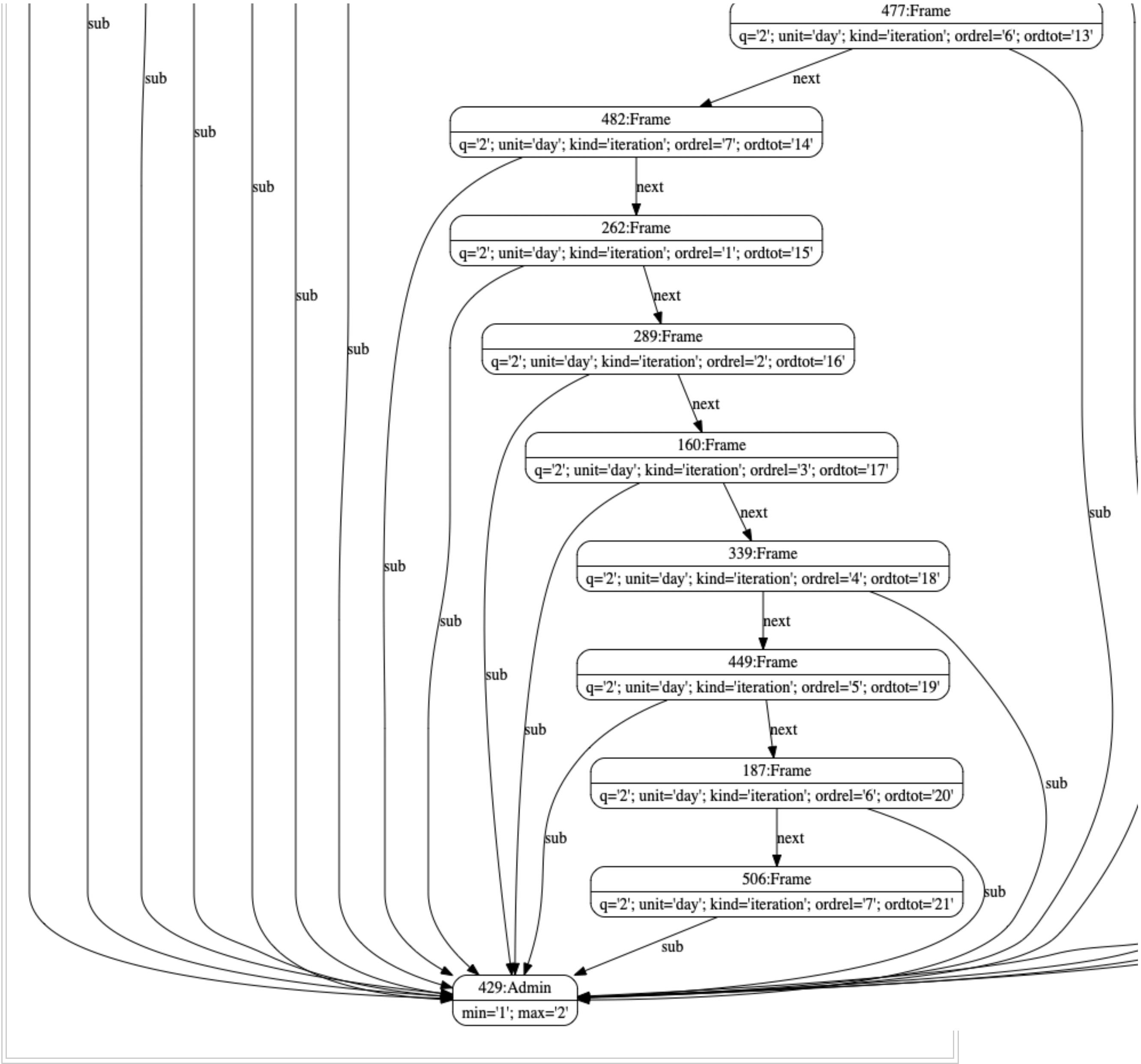


```
(rule 'delete-parent!
      :read (pattern
             (node 'wf :label "Frame" :asserts {:unit "'week'"}))
      :delete ['wf]
      )

(defn remove-parent-frames! []
    (while (jump-over-parent!))
    (while (delete-parent!)))
```

```
#'chaos.core-test/remove-parent-frames!
```

```


(remove-parent-frames!)
(browse)

```


135:Require
duration='3'
sub
act
538:Frame
q='2'; unit='day'; kind='iteration'; ordrel='1'; ordtot='1'
205:Actio
name='A
next
437:Frame
q='2'; unit='day'; kind='iteration'; ordrel='2'; ordtot='2'
next
329:Frame
q='2'; unit='day'; kind='iteration'; ordrel='3'; ordtot='3'
next
544:Frame
q='2'; unit='day'; kind='iteration'; ordrel='4'; ordtot='4'
next
244:Frame
q='2'; unit='day'; kind='iteration'; ordrel='5'; ordtot='5'
next
396:Frame
q='2'; unit='day'; kind='iteration'; ordrel='6'; ordtot='6'
next
298:Frame
q='2'; unit='day'; kind='iteration'; ordrel='7'; ordtot='7'
next
427:Frame
q='2'; unit='day'; kind='iteration'; ordrel='1'; ordtot='8'
next
440:Frame
q='2'; unit='day'; kind='iteration'; ordrel='2'; ordtot='9'
next
291:Frame
q='2'; unit='day'; kind='iteration'; ordrel='3'; ordtot='10'
next
320:Frame
q='2'; unit='day'; kind='iteration'; ordrel='4'; ordtot='11'
next
535:Frame
q='2'; unit='day'; kind='iteration'; ordrel='5'; ordtot='12'
next
sub

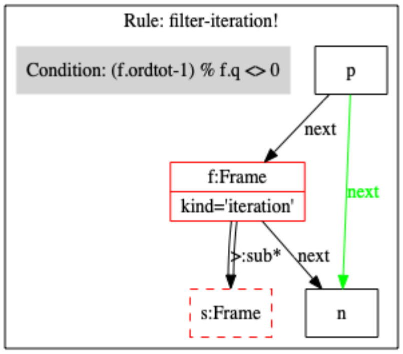

477:Frame
q='2'; unit='day'; kind='iteration'; ordrel='6'; ordtot='13'
next
482:Frame
q='2'; unit='day'; kind='iteration'; ordrel='7'; ordtot='14'
next
262:Frame
q='2'; unit='day'; kind='iteration'; ordrel='1'; ordtot='15'
next
289:Frame
q='2'; unit='day'; kind='iteration'; ordrel='2'; ordtot='16'
next
160:Frame
q='2'; unit='day'; kind='iteration'; ordrel='3'; ordtot='17'
next
339:Frame
q='2'; unit='day'; kind='iteration'; ordrel='4'; ordtot='18'
next
449:Frame
q='2'; unit='day'; kind='iteration'; ordrel='5'; ordtot='19'
next
187:Frame
q='2'; unit='day'; kind='iteration'; ordrel='6'; ordtot='20'
next
506:Frame
q='2'; unit='day'; kind='iteration'; ordrel='7'; ordtot='21'
sub
429:Admin
min='1'; max='2'
sub

6. Filter iteration

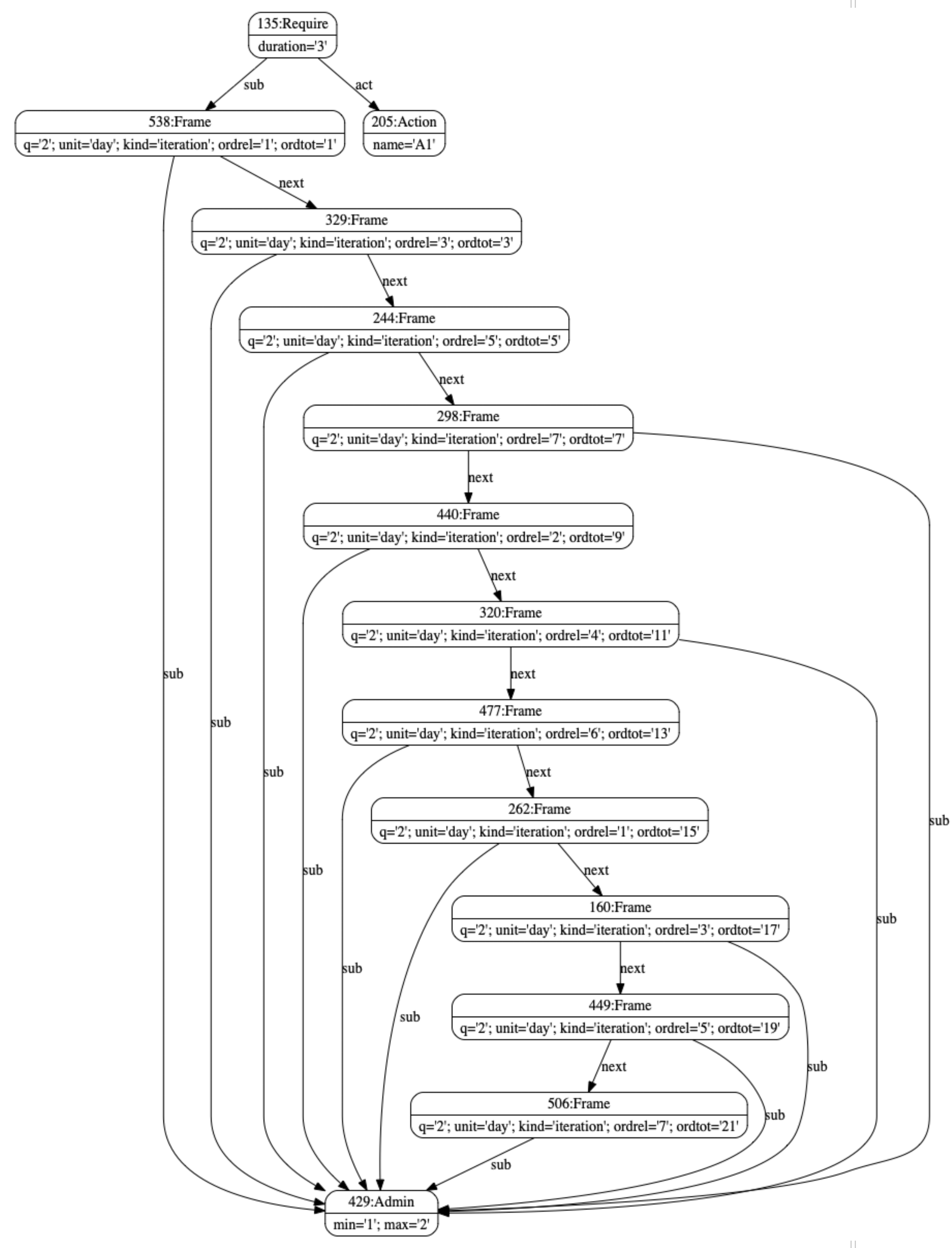
Now we filter out those frames that are targeted by the prescribed iteration.

Note: todo filter by selection

```
(rule 'filter-iteration!
      :read (pattern
             (node 'f :label "Frame" :asserts {:kind "'iteration'"})
             (node 'p)
             (node 'n)
             (edge :label "next" :src 'p :tar 'f)
             (edge :label "next" :src 'f :tar 'n)
             (node 's :label "Frame" :opt true)
             (path ">:sub*" :src 'f :tar 's :opt true)
             (condition "(f.ordtot-1) % f.q <> 0"))
      :delete ['f 's]
      :create (pattern
               (edge :label "next" :src 'p :tar 'n)))
```



```
(while (filter-iteration!))
(browse)
```

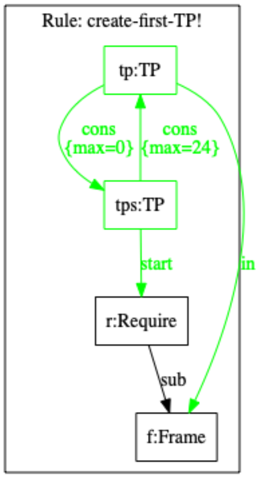

135:Require
duration='3'
sub
act
538:Frame
q='2'; unit='day'; kind='iteration'; ordrel='1'; ordtot='1'
205:Action
name='A1'
next
329:Frame
q='2'; unit='day'; kind='iteration'; ordrel='3'; ordtot='3'
next
244:Frame
q='2'; unit='day'; kind='iteration'; ordrel='5'; ordtot='5'
next
298:Frame
q='2'; unit='day'; kind='iteration'; ordrel='7'; ordtot='7'
next
440:Frame
q='2'; unit='day'; kind='iteration'; ordrel='2'; ordtot='9'
next
320:Frame
q='2'; unit='day'; kind='iteration'; ordrel='4'; ordtot='11'
next
477:Frame
q='2'; unit='day'; kind='iteration'; ordrel='6'; ordtot='13'
next
262:Frame
q='2'; unit='day'; kind='iteration'; ordrel='1'; ordtot='15'
next
160:Frame
q='2'; unit='day'; kind='iteration'; ordrel='3'; ordtot='17'
next
449:Frame
q='2'; unit='day'; kind='iteration'; ordrel='5'; ordtot='19'
next
506:Frame
q='2'; unit='day'; kind='iteration'; ordrel='7'; ordtot='21'
sub
sub
sub
sub
sub
sub
sub
sub
sub
sub
sub
sub
429:Admin
min='1'; max='2'

7. Create STN

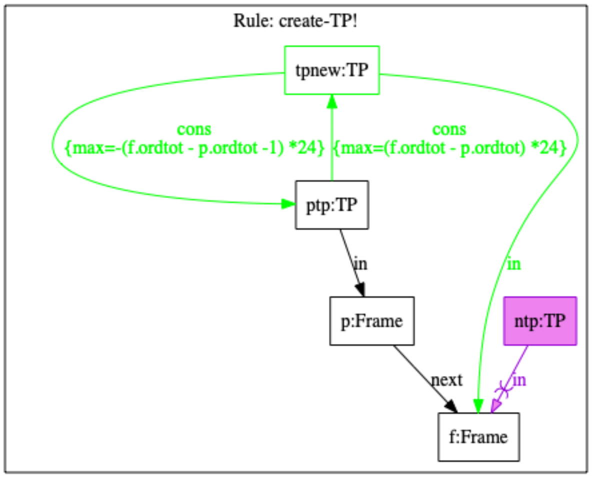
We finally create the STN nodes and edges

Note: this is currently done based on "day" frames and with the resultion of hours only

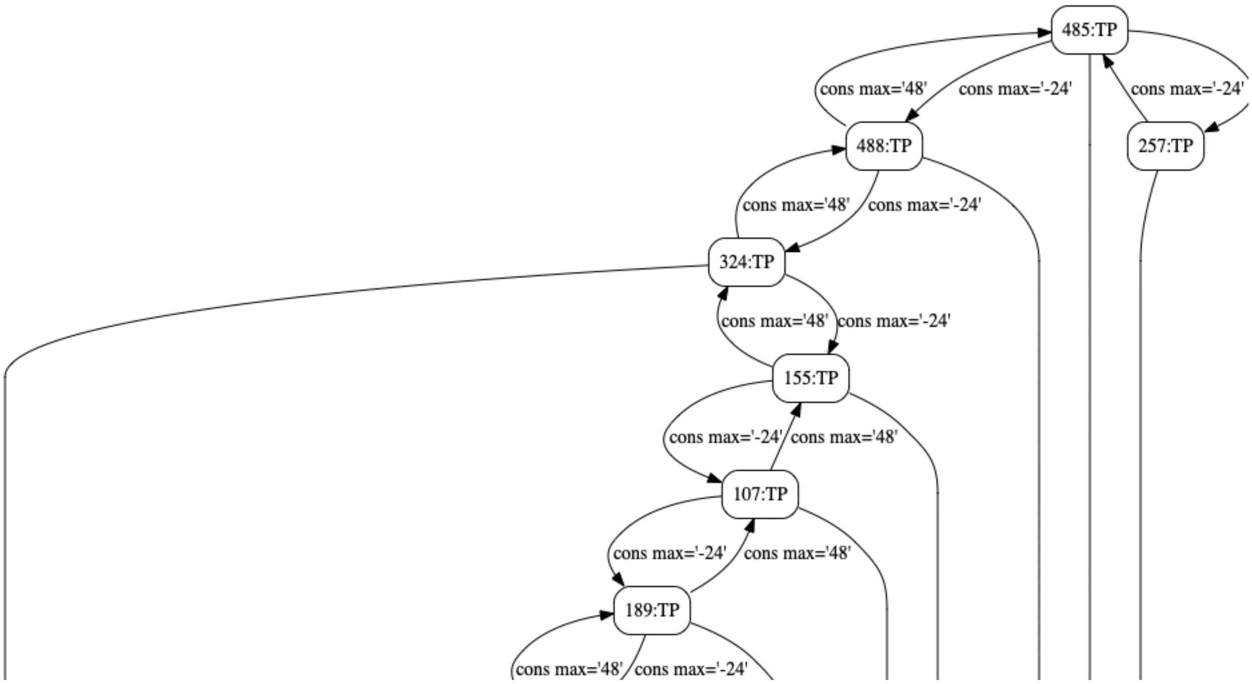
```
(rule 'create-first-TP!
      :read (pattern
             (node 'r :label "Require")
             (node 'f :label "Frame")
             (edge :label "sub" :src 'r :tar 'f)
             )
      :create (pattern
               (node 'tps :label "TP")
               (edge :label "start" :src 'tps :tar 'r)
               (node 'tp :label "TP")
               (edge :label "in" :src 'tp :tar 'f)
               (edge :label "cons" :src 'tps :tar 'tp :asserts {:max "24"})
               (edge :label "cons" :src 'tp :tar 'tps :asserts {:max "0"})

       ))
```

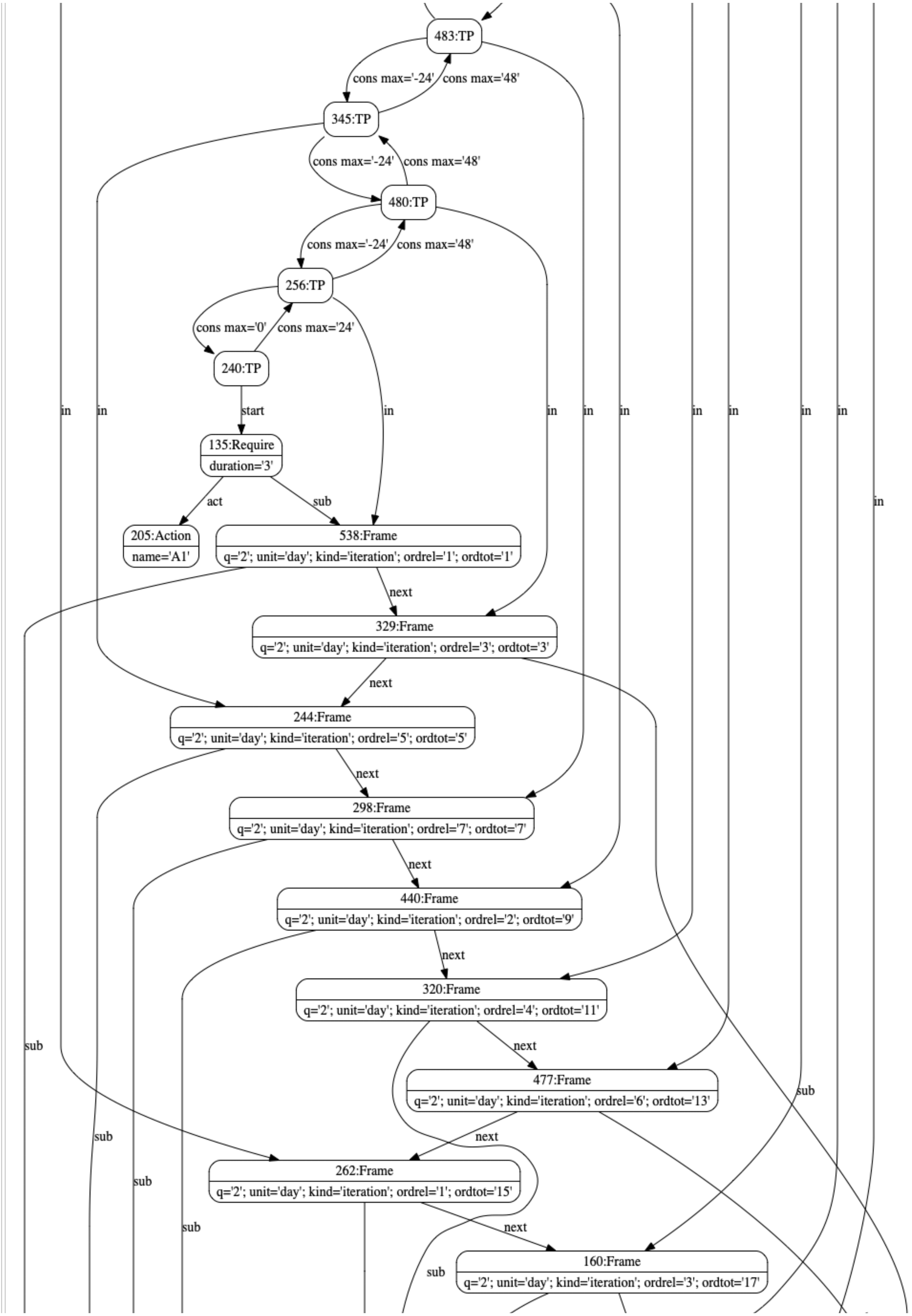


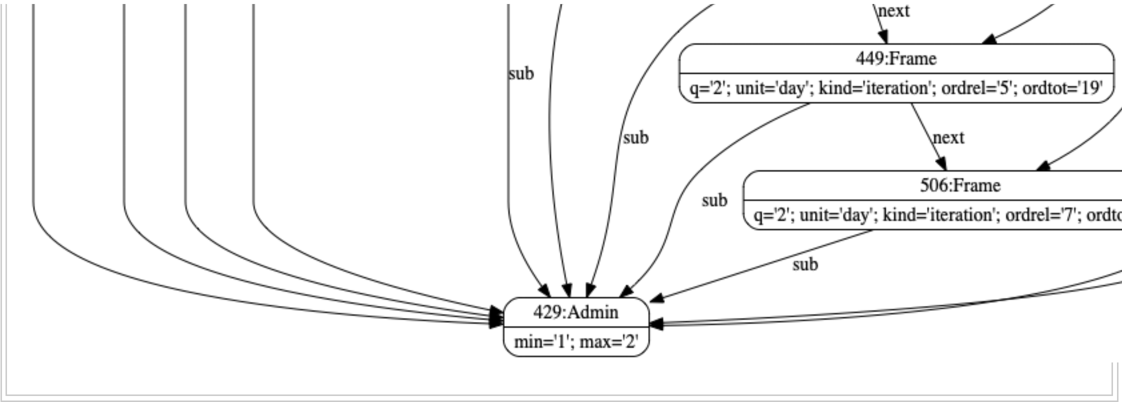
```
(rule 'create-TP!
      :read (pattern
             (node 'p :label "Frame")
             (edge :label "next" :src 'p :tar 'f)
             (node 'f :label "Frame")
             (node 'ptp :label "TP")
             (edge :label "in" :src 'ptp :tar 'p)
             (NAC
              (node 'ntp :label "TP")
              (edge :label "in" :src 'ntp :tar 'f)
              ))
       :create (pattern
               (node 'tpnew :label "TP")
               (edge :label "in" :src 'tpnew :tar 'f)
               (edge :label "cons" :src 'ptp :tar 'tpnew :asserts {:max "(f.ordtot - p.ordtot) *24"})
               (edge :label "cons" :src 'tpnew :tar 'ptp :asserts {:max "-(f.ordtot - p.ordtot -1) *24"})
             )
       )
```



```
(create-first-TP!)
(while (create-TP!))
(browse)
```




483:TP
cons max='-24'
cons max='48'
345:TP
cons max='-24'
cons max='48'
480:TP
cons max='-24'
cons max='48'
256:TP
cons max='0'
cons max='24'
240:TP
in
in
start
in
in
in
in
in
in
in
in
135:Require
duration='3'
in
act
sub
205:Action
name='A1'
538:Frame
q='2'; unit='day'; kind='iteration'; ordrel='1'; ordtot='1'
next
329:Frame
q='2'; unit='day'; kind='iteration'; ordrel='3'; ordtot='3'
next
244:Frame
q='2'; unit='day'; kind='iteration'; ordrel='5'; ordtot='5'
next
298:Frame
q='2'; unit='day'; kind='iteration'; ordrel='7'; ordtot='7'
next
440:Frame
q='2'; unit='day'; kind='iteration'; ordrel='2'; ordtot='9'
next
320:Frame
q='2'; unit='day'; kind='iteration'; ordrel='4'; ordtot='11'
sub
next
477:Frame
q='2'; unit='day'; kind='iteration'; ordrel='6'; ordtot='13'
sub
sub
next
262:Frame
q='2'; unit='day'; kind='iteration'; ordrel='1'; ordtot='15'
sub
sub
next
sub
160:Frame
q='2'; unit='day'; kind='iteration'; ordrel='3'; ordtot='17'

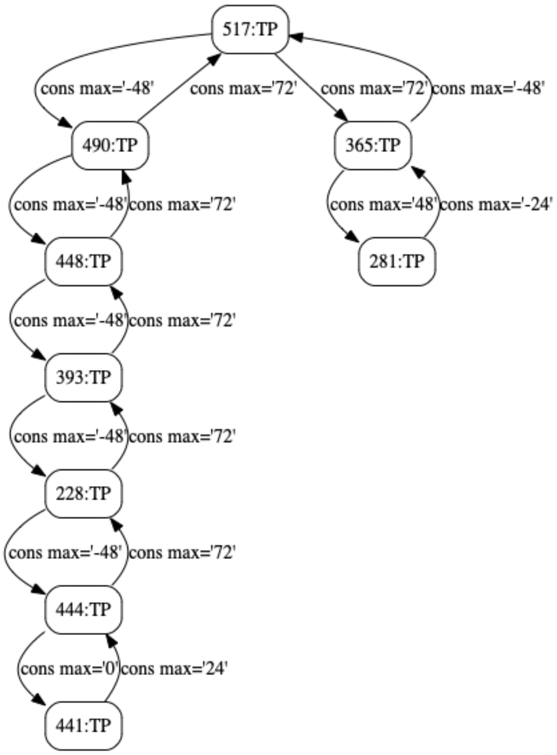

next
sub
sub
449:Frame
q='2'; unit='day'; kind='iteration'; ordrel='5'; ordtot='19'
sub
next
sub
506:Frame
q='2'; unit='day'; kind='iteration'; ordrel='7'; ordtot='21'
sub
429:Admin
min='1'; max='2'

Wrap all steps in a function:

```
(rule 'TP?
      :read (pattern
                 (node 'n :label "TP")
                 (node 'm :label "TP")
                 (edge :src 'n :tar 'm :opt true)))

(defn transform [s]
  (clear!)
  (createRTF! (parse s))
  (while (unroll-duration!))
  (while (split-subframe!))
  (while (unroll-frame!))
  (while (connect-subframe!))
  (remove-parent-frames!)
  (while (filter-iteration!))
  (create-first-TP!)
  (while (create-TP!))
  (-> TP? matches view))
```

```
#'chaos.core-test/transform
```

```
(transform "require A1 for 3 weeks every 3rd day administer 1-2")
```



Checking consistency

We can now use Neo4J to search for cycles with the shortest weighted path. If that is negative, the STN is inconsistent: (http://127.0.0.1:7474/browser/)

```
MATCH (n:TP), path = (n)-[:cons*]->(n)
RETURN path AS shortestPath,
    reduce(max = 0, r in relationships(path) | max+r.max) AS total
    ORDER BY total ASC
    LIMIT 1
```

The above query delivers "24" as the shortest path. Hence the STN is consistent.