Principles Of Model Checking Exercises Solutions

Dario Bekic[1]

[1] *University of Pisa*

Introduction

These are my solutions for the exercises of the book Principles Of Model Checking by Christel Baier and Joost-Pieter Katoen

Linear Time Properties

Exercise 3.1

This is like giving a regular grammar.

$$
\begin{aligned}
&\rightarrow \quad {}'| \quad {}'' \\
{}' &\rightarrow \quad {}'''| \; {}'''| \\
{}'' &\rightarrow \quad {}'''| \; {}''' \\
{}''' &\rightarrow \quad {}'
\end{aligned}
$$

Exercise 3.2

a) (set of transitions systems with terminal states and is the one without). $(\; , \; , \rightarrow, \; , \; , \; , \;) = (\; \{ \; \}, \; , \rightarrow \; \{(\; , \; , \;)|$ $\}, \; , \; , \; \{(\; , \;)\})$ by viewing everything as a set.

b) Suppose $(\; _1)$ $(\; _2)$. Let $_1$ be a $_1$ path such that $(\; _1) =$. Either . $=$ or not. Assume the first case, and let be the index of the first element in $_1$ such that $_1 \neq$. The path fragment $'_1 =$ $_{1,1}, \ldots, \; _{1,}$ is a maximal path in $_1$ since $_{1,}$ is a terminal state by construction. By hypothesis there must exist a path $'_2$ $_2$ such that $(\; '_1) =$ $(\; '_2)$ thus also $'_2$ must be a maximal initial finite fragment path i.e. must end in a terminal state of $_2$. Now we can extend the path $'_2$ with infinite loops and since i.e. the phantom state added to $_2$ has no atomic proposition its trace will be equal to that of $_1$. Namely, $=$ $(\; _1) =$ $(\; '_1) =$ $(\; '_2) =$ $(\; _2)$, contradicting the claim that $(\; _2)$.

Exercise 3.3

Exercise 3.4

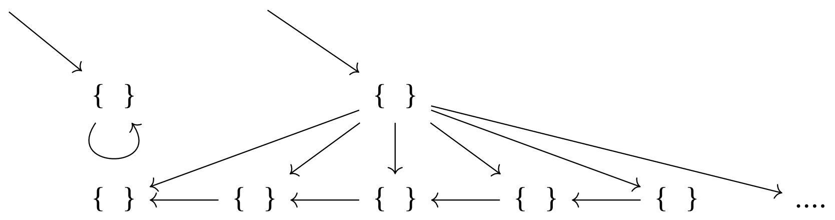
a) : pick a finite path fragment $(\;)$ such that $(\;)$ $(\;)$ $(\;)$ $(\; ')$. Call the size of . I can extend to a maximal path $'$ such that $(\; ')$ $(\;)$ $(\; ')$ $(\; ')$. By $(\; ')$ $(\; ')$ there must exist a path $'$ $(\; ')$ such that $(\; ') =$ $(\; ')$. Thus, the prefix of length of $'$ is a finite path

Algorithm 1 BFS for Invariant Checking (shortest counterexample)

```
procedure BFSInvariant(  ( ,   , →, ,   , ),  )
      ← emptyQueue()
      ← emptyMap()                    [ ] is predecessor of  ; absent means unseen
    for all      do
        if  ( ) /  then
            return  false, [ ]
        end if
          .enqueue( )
            [ ] ←                                mark as seen;    denotes start of path
    end for
    while ¬  .empty() do
          ←  .dequeue()
        for all        ( ) do
            if     keys(     ) then                              first time seen    shortest
                    [ ] ←
                if  ( ) /  then
                    return  false, Reconstruct(     , )
                end if
                  .enqueue( )
            end if
        end for
    end while
    return  true, emptyList                        invariant holds on all reachable states
end procedure
procedure Reconstruct(     , )
      ← emptyList
    ←
    while   ≠    do
          .append( )
          ←     [ ]
    end while
    return reverse(     )
end procedure
```

fragment in ′ such that ( ) = ( ), contradicting the hypothesis ( ) (′). The same can be argued by swapping and ′ roles, completing the proof.

: pick a path ( ) such that ( ) ( ) ( ) (′). I argue that for any there exists a path fragment in ′ such that its trace its equal to the prefix of length of and that it can be extended to path fragment of size + 1 which is equal to the trace of the prefix of length + 1 of . The first part of the statement comes directly from the assumption that the finite traces sets equal each other. The second part comes from the fact any trace in both transition systems can be given by only one path (or path fragment), thus for the prefix of length + 1 of i need necessarily to extend because if there was another path different from , call it , then their prefix trace of size would be the same, contradicting the claim. Suppose that in fact such existed, then it may share a prefix of states with , let < be the last state they share, i.e. = then there should be two states $_{+1}$ and $_{+1}$ such that ($_{+1}$) = ($_{+1}$) because , share the same trace up to the first states, but this contradicts AP determinism.

b) 

Exercise 3.5

a) `false`

b) `initEqualZero` { $_0$ $_1$, ... (2) | . = { = 0} < . = }

c) `initDiffZero` (2) `initEqualZero`

d) This seems not to be safety nor liveness.

e) Liveness property

f) Liveness property

g) `initEqualZero` { $_0$ $_1$, ... (2) |(= { = 0} = { > 0} i even, j odd) (= { = 0} = { > 0} i odd, j even)}

h) `true` (2)

Exercise 3.6

a) Invariant: `Never` $\{ {}_0 \; {}_1, \dots \; (2 \;) \mid \; \}$

b) Safety: `Once` $\{ {}_0 \; {}_1, \dots \; (2 \;) \mid \; . \; .(\; < \; > \;) \; \}$

c) Liveness: `AlternateInfinitely` $\{ {}_0 \; {}_1, \dots \; (2 \;) \mid \; , \; , \; . \; > \; = \{ \; \} \; > \; = \{ \; \} \; (\; > \; . \; = \{ \; \} \; . \; = \{ \; \} \; . \; < \; < \; . \;) \; (\; > \; . \; = \{ \; \} \; . \; = \{ \; \} \; . \; < \; < \; . \;)\}$ (its a liveness property because the existential witness can be large as one wishes).

d) Liveness `AlternateInfinitely` $\{ {}_0 \; {}_1, \dots \; (2 \;) \mid \; .((\; < \; . \; . \; < \; < \; . \;) \;) \; (\; . \; > \; (\; . \; > \; . \;))\}$

Exercise 3.7

Initial values: $_1 = 0$, $_2 = 1$. value is given by the value of register $_2$ and the formula is $(({}_1 \;) \; (\; {}_1))$. $_1$ value is given by $({}_1 \;) \; ({}_2 \;)$

a) Let be a word in and be a word in $(2 \;)$ $. \; {}_1 \; \{ {}_2, \; , \; \}(\;)$ where $, \; {}_2 \; \{ {}_1, \; \} \; (\;) \; , \; {}_3 \; (\{ \; \}, \;) \; , \; {}_4 \; , \; {}_1 \; \{ \; \} \; , \; {}_2 \; \{ {}_2, {}_1\} \; , \; {}_3 \; \{ \; \} \; , \; {}_4 \; \{ \; , {}_1\}$

b) P1) If $= 1$ is high will be high regardless of $_1$: if $_1 = 0$, $_1$ will be 1, if $_1 = 1$ then $_1$ will be 1. Thus it is satisfied.

P2) If $_2 = 0$, $_1 = 0$ in any state, in the next state we will $_1 = 1$ if $= 1$ so it is not satisfied.

P3) This is not satisfied because $_1$ is satisfied.

P4) False, it is a possible initial state

c) P2, P3, P4 are safety properties and only P4 is an invariant.

i) Let be equal to any set in $\{ \; , {}_1, {}_2, \; \}$ of size i. is the terminal state of the grammar: with that symbol we can expand to any set of properties.

P2)

$$\begin{aligned} &\rightarrow \quad \{ {}_2\} \; ' \mid \quad \{ {}_2\} \\ ' &\rightarrow \quad ' \mid \quad \{ {}_1\} \end{aligned}$$

P3)

$$\begin{aligned} &\rightarrow \quad \mid \quad \{ \; \} \; ' \\ ' &\rightarrow \quad \{ \; \} \end{aligned}$$

P4)

$\rightarrow \quad | \quad \{ \} \quad \{ _1 \}$

ii) There is only 4: $\neg \quad _1$

Exercise 3.8

: this direction holds and directly comes from the definition of closure of a property: $( ) = \{ \quad (2 \quad) \mid \quad ( ) \quad\quad ( ) = \{ \quad (2 \quad) \mid \quad ( ) \quad\quad (') = (')\}$.

: suppose $( ) = \quad (')$ but $( ) \quad\quad (')$. Since $( )$ there exists such that is a prefix of . By definition of closure, $( )$ which implies $(')$ by hypothesis. But $(')$ $( ) \quad (')$ by definition of closure, and thus we have $(')$.

Exercise 3.9

We need to show that i) the set $\Phi \quad (\quad (\quad))$ is a LT safety property and ii) that satisfies it.

i) imagine there were $= \ _0 \ _1, \dots \ (2 \quad) \quad \Phi$ such that there exists $( )$ such that $\Phi \cap \{ \quad (2 \quad) \mid \quad ( )\} \neq$. Since $(\quad (\quad))$ it means there is a finite prefix of that is not a prefix of some trace in TS. Let this trace be our . Then if $\Phi \cap \{ \quad (2 \quad) \mid \quad ( )\}$ it means $( ) \quad (\quad (\quad))$ which means $(\quad (\quad))$.

ii) A word is in Φ if and only if $( ) \quad (\quad (\quad))$. Any path in TS has $( ) \quad (\quad)$ thus $(\quad ( ))$ $(\quad (\quad))$.

Exercise 3.10

Suppose $(\quad (\quad))$ then $( )$ such that a finite prefix of is . But any element in $( )$ is just a word whose prefixes are elements of $( )$ thus any prefix of is a an element of $( )$, even , thus $( )$.

Exercise 3.11

a) UNION_SAFE $'$ is a safety property if , $'$ are safety properties. Consider UNION_SAFE. For it to be a safety property, it is sufficient to prove that for any $(2 \quad)$ (UNION_SAFE) and for any finite prefix of it (UNION_SAFE) $\cap$ $(\{ \quad (2 \quad) \mid$ is a finite prefix of $\}) =$. This equals to say, by distributivity, PREFIX $(\ \cap$ PREFIX$) \quad (\ ' \cap$ PREFIX$) =$. Since , $'$ are safety properties we have $=$.

b) INTER_SAFE ∩ ′ is a safety property if , ′ are safety properties. Take any word (2) . There are 2 cases to consider:

   i) ′: we wish to show that for any finite prefix ( ) , ∩ ′ ∩ { (2) | is a finite prefix of }. But since ′ is a safety property PREFIX ′ ∩ PREFIX = , thus ∩ = .

   ii) ′ : we wish to show again that for any finite prefix ( ) , INTER_SAFE ∩ PREFIX = . By commutativity of intersection, ∩ ′ ∩ PREFIX = ′ ∩ ∩ PREFIX = ′ ∩ = . The case of ′ is identical.

c) UNION_LIVE ′ is a liveness property if , ′ are liveness properties. This is trivially arguable informally: if i can extend any finite prefix to a trace in or to a trace ′ ′ then i can extend any prefix to a trace in ′ (just pick either or ′).

d) INTER_LIVE ∩ ′ is not necessarily a liveness property. To see a concrete counter-example consider { from one point there will be only 1s} and ′ { from one point there will be only 0s}. Their intersection is empty, and the empty set is not a liveness property.

Exercise 3.12

1. ( ) . Assume there exists ( ) such that . Since ( ), for any finite prefix of , ( ) ( ). So pick any prefix of , since is a safety property we have ∩ { | ( )} = . But we also proved ( ), and since contains a subset of it means . ( ). But this would imply ∩ { | ( )} ≠ .

2. ((2) ( )). Suppose but not in EXPR. This can only happen if EXPR which implies . We wish to prove EXPR if . This is equal to proving ( ) by definition of EXPR. We can prove the contrapositive: ( ) which we proved in the first point.

Exercise 3.13

One can of course use the Decomposition theorem, but there is a way of giving explicitly the two properties by using intuition.

One would of course first start from characterizing : maybe the exercise is tricky and is a safe or liveness property. Proving one such result would let us quickly end the exercise.

a. Is a safety property? One could argue that the set of bad prefixes is $\{ {}_1, {}_2, ..$ $(2 \quad) \mid \quad . \quad = \{ \ , \ \} \quad . < \quad \}$. However, the condition to have *infinitely* many . cannot be captured by a finite prefix. In conclusion, is not a safety property.

b. Is a liveness property? No, clearly $' = \{\{ \ \}, \{ \ , \ \}\}$ cannot be extended to a word satisfying .

Even though we were unsuccessful in proving safety or liveness, we identified the parts that were problematic during our proof attempt.

In particular:

$$(2 \quad) \quad \{ {}_1, {}_2, .. \quad (2 \quad) \mid \quad .(\quad = \{ \ , \ \} \quad . < \quad)\}$$

and

$$\{ {}_1, {}_2, ... \mid^{\infty} . \quad \}$$

The correctness is immediate.

Exercise 3.14

First, let's put here the transition graphs. Consider the TS for Peterson's algorithm in Figure 41 and for the Semaphore-based mutual exclusion in Figure 42. Note that we have ${}_1$, ${}_2$ labels even though they are not in AP just for ease of read. In general we can solve this exercise by reasoning on the LT properties that the transition systems satisfy or not. As we know, the semaphore transition systems does not guarantee starvation freedom i.e. it is possible that one process will never enter its critical region.

Thus, the trace:

$$\{\{ \ {}_1, \ {}_2\}, \{ \ {}_1, \ {}_2, \}, \{ \ {}_1, \ {}_2\}\}$$

is in ( ) but not in ( ).

Exercise 3.15

(a) Regarding satisfaction of $'$: without *unconditional* fairness assumptions it is a matter of verifying if there exists a path which contains $\{\{ \ \}, \{ \ , \ \}, \ \}$ and this is true ${}_1 \to {}_2 \to {}_3 \to {}_1$, thus this property is never satisfied. For ${}_1$ only because we may loop on ${}_1 \to {}_2 \to {}_1$. For ${}_2$ both and as we need to exit the loop we mentioned because ${}_1 \to {}_3$ is infinitely enabled. For ${}_3$ only , because of the loops ${}_1 \to {}_2 \to {}_1$ and ${}_1 \to {}_2 \to {}_4 \to {}_1$ we always pass by ${}_1$ which has enabled the transition to ${}_3$.

(b) The same reasoning for $'$ applies. Now, recall that weaker assumptions restrict *the same* or more traces, thus we do not check again those we proved are not satisfied by strong assumptions. For example, $_1$ now looses as -transitions from $_1$ are not continuously enabled, as we can loop on $_1 \to {}_3 \to {}_1$. The same goes for $_3$. And we also loose both , for $_2$ as we can loop on $_1 \to {}_2 \to {}_1$ continuously *alternating* between having and transitions.

Exercise 3.16

(a)
1. With no assumption this is false. Let $_1 = \{\ \}$ and $_2 = \{\ \}$. Then let $_2.$ $(\ )$, $'$ $_1.$ $(\ ')$. Then $_1, {}_2 \dots$ $(\ _1 || \ _2)$ we have but no trace in $_1$ contains such label (as it is not part of $_1$ label set).
2. This is false. Consider Figure 1 where $_1, {}'_1$ is the initial state and consider $= \{\ \}$. It will be impossible to move out of the initial state as $'_1$ has no outgoing transition to synchronize on. We have only have one trace: $(\ _1 || \ _2) = \{\{ \ (\ _1, {}'_1)\}\} = \{\{\ _1(\ _1)\}\}$

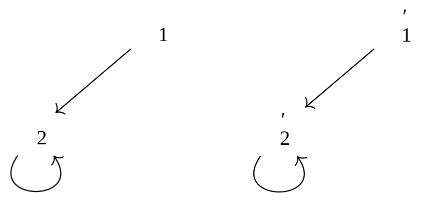


Figure 1: Exercise 3.16. point (a)

(b)
1. With no assumption it is the same as (a).
2. Assuming $_2 =$ the result is true. Let us restrict to the subset of paths in $_1||\ _2$ that use only a transition from $_1$. We argue that for any path $_1 = {}_0, {}_1, \dots$ $(\ _1)$ we have a path $_2 = {}'_0, {}'_1, \dots$ such that $_1(\ ') =$, where $_1$ is the left projection. This can be proved easily by induction.

(c)
1. With no assumption it is the same as (a).
2. Assuming $=$ it is still false. Consider Figure 2. The transition system $_1||\ _2$ allows for the path $(\ _1, {}'_1 \to {}_1, {}'_1)$ with $(\ ) =$. However this trace is not available to $_1$ whose trace set is $\{\{\ \}\}$.

(d) False. Consider Figure 3 and the two transition systems, where we just write the Atomic Propositions that hold: and call the TS on the left $_1$ and the one on the right $_2$. It is obvious that $(\ _1)$ $(\ _2)$ as $(\ _1) =$ $(\ _2)$. Clearly for $= \{\{\ \}\}$ $_2$ has no infinite fair traces as it *has* to take the transition while $_1$ does.

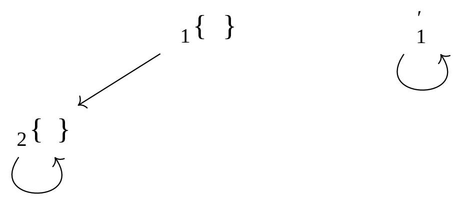


Figure 2: Exercise 3.16. point (c)

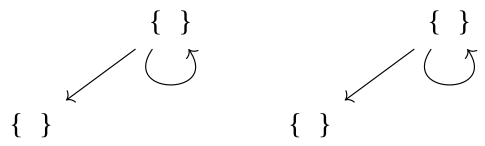


Figure 3: Exercise 3.16. point (d)

(e) We simply make a little change to the transitions systems of the point (d). As you can see in figure 4., we have clearly ($_1$) = { } { | } = ($_2$) thus ($_1$) ($_2$). Consider the liveness property:

$$\texttt{EVENTUALLY\_B} \quad .$$

Under = { , {{ }}, } we have $_2$ `EVENTUALLY_B` but $_1$ / `EVENTUALLY_B`.

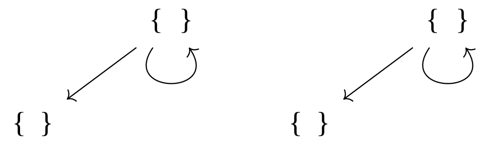


Figure 4: Exercise 3.16. point (e)

Exercise 3.17

It is enough to count all the cycles and see if one of those two non-mutually exclusive (but sufficient) conditions hold:) to access the cycle by using any path we encounter *a* or) in at least one state of the cycle we have available either a or as outgoing transitions to states where *a* holds (or where we have proved that it eventually holds). (Some) cycles are:

1 $_1 \rightarrow$ $_1$: since in $_1$ *a* holds we conclude by criterion).

2 $_2 \rightarrow$ $_3 \rightarrow$ $_2$: we have infinitely many times transition available which sends us to $_1$, thus by criterion) and point 1. we conclude.

3 $_3 \rightarrow\ _4 \rightarrow\ _5 \rightarrow\ _4 \rightarrow\ _6 \rightarrow\ _3$ both criterion do not hold, thus the property is not satisfied.

Exercise 3.18

(a) It is not realizable as there is no fair path starting from $_1$ which is reachable with an transition from $_0$ because of the unconditional assumption { }.

(b) By having now the unconditional assumption { , } the loop $_1 \rightarrow\ _4 \rightarrow\ _1$ is a fair path. Thus, $_2$ is realizable.

(c) This is too realizable. The loop $_1 \rightarrow\ _4 \rightarrow\ _1$ will enable infinitely many times and by strong fairness { , } we will be forced to take it infinitely many times thus satisfying the unconditional assumption { }.

Exercise 3.19

(a) (i)

$$\begin{aligned} &= {}_1(\ _2) \\ _1 &= {}_1'| \\ _1' &= {}_1'| \\ _2 &= \{\ \}\ _2'|\{\ \}\ _2|\{\ ,\ \}\ _2 \\ _2' &= \{\ \}\ _2 \end{aligned}$$

(ii) By the decomposition theorem we have

$$\begin{aligned} &= \{\ _0\ _1 \ldots\ (2\ )\ |\ldots\} \\ \ldots &= \ .(\ = \{\ \}\ .<\ = \ .>\ .\ = \{\ \}\ _{+1} = \{\ \}) \\ &= {}_1\ ((2\ )\ (\ )) \\ &= (\{\ \} + \{\ ,\ \})\ _1\ \{\ _0\ _1\ldots\ (2\ )\ |\ldots\}\ \{\{\ \}\ _1\ldots|\ \} \\ \ldots &= \ .(\ > 0\ = \{\ \}\ <\ .\ \{\{\ ,\ \},\{\ \}\}) \\ &= \ > 0.\ = \{\ \}\ _{+1} \neq \{\ \} \end{aligned}$$

(iii) 1. Suppose (2) . If . = { } then pick the first non-empty set of : it should be clear that any word with this prefix is not in . If . = { } then it must be:

   i. Either . < $\neq$. Then it is sufficient to pick the prefix $_0 \cdots$.

   ii. Or(they are not mutually exclusive) > . = { } $_{+1} \neq$. Pick the prefix $_0, \cdots\ _{+1}$.

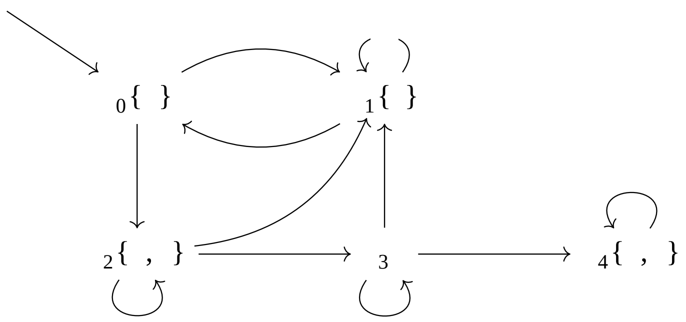


Figure 5: Exercise 3.19. point (b)

(b) First, let's re-draw the transition system which we can see in Figure 5, where each node is denoted with the pair $\{\ (\ )\}$. we work separately for the two fairness assumptions. The first one is:

$$_1 = (\{\{\ \ \}\}, \{\{\ \ \}, \{\ \ ,\ \ \}, \{\ \ \}\},\ \ \})$$

First, state $_4$ is ruled out as it will not allow for infinite transitions that is a condition of the unconditional assumptions.

Notice that we will be forced to take the transition from $_0$ to $_2$ as it is one of our strong assumptions, forcing us out from the hypothetical loop $_0 \rightarrow\ _1 \rightarrow\ _0$.

Now the strong assumption $\{\ ,\ \}$ forces us only to eventually take either or an infinite number of times as we will pass $_2$ an infinite number of times.

Thus, the first part property is satisfied as we will pass an infinite number of times to $_2$ as we will re-enter the loop either through $_2 \rightarrow\ _1$ or $_2 \rightarrow\ _3 \rightarrow\ _1$.

For the second part of the property, notice that if we pass to $_3$ an infinite amount of time, we will need to take thus invalidating that path. So any fair path passes trough $_3$ a finite number of times, so pick as the last time we are in $_3$. Then any fair path passes only through $_0$, $_1$, $_2$ and thus cannot invalidate the second part of the property as any outgoing transition from states such that $(\ )$ and for $'.\ '$ $(\ )$ $(\ ')$.

Now recall the other assumption:

$$_2 = (\{\{\ \ \}\}, \{\{\ \ \}, \{\ \ \}\}, \{\{\ \ \}\})$$

Now we are not forced to move with even if we pass through $_3$ an infinite number of times. So say such existed, we can always loop back to $_2$ and then move to $_3$(as we can pass through to $_3$ an infinite number of times).

Exercise 3.20

Regular Properties

Exercise 4.1

(a) yes, see Figure 6

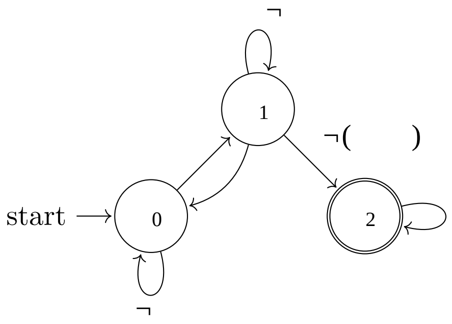


Figure 6: Exercise 4.1 point (a)

(b) yes, see Figure 7

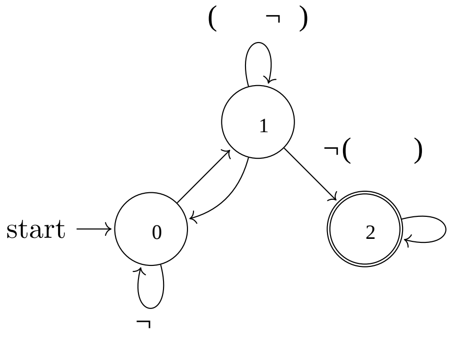


Figure 7: Exercise 4.1 point (b)

(c) No, as $\; = \; ^{+1}, \; \geq 0$ is a subset of the property but is not a regular language. To prove it one can use the Pumping Lemma: let be the pumping length and let $\; = \; ^{+1}$, clearly $\; ^{+} \; ^{+1}$for $\; > 0$ is not in the language.

(d) Yes, see Figure 8

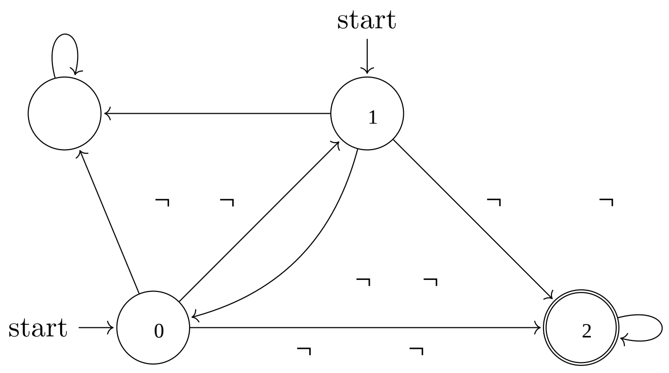


Figure 8: Exercise 4.1 point (b)

Exercise 4.2

1. Let = 3 for simplicity.

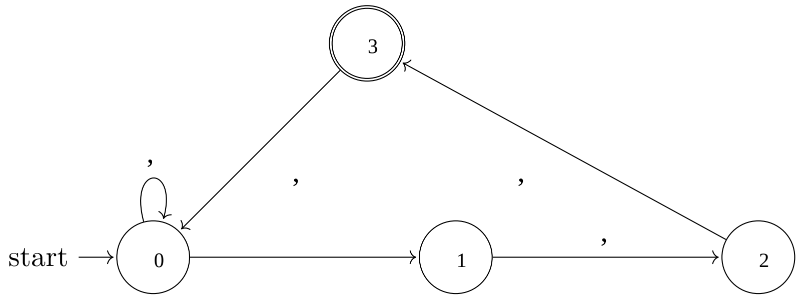


Figure 9: Exercise 4.2 point (a)

2. To determinize the NFA we start from $_0$ and with a transition we can only stay in $_0$ while using we can either move to $_0$ or $_1$, we call this state $_{0,1}$.Then from this state by either or we go in the state $_{0,2}$. Then, again by either or , we go to state $_{0,3}$ and then again to state $_0$. See Figure 10.

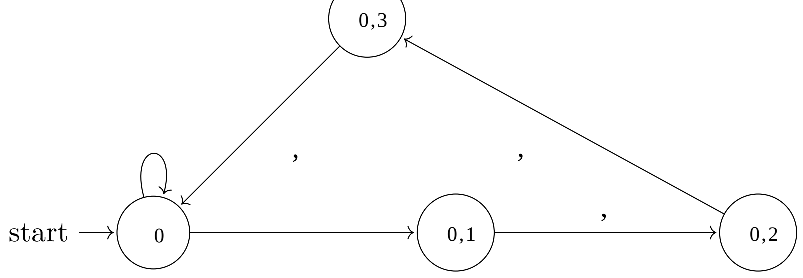


Figure 10: Exercise 4.2 point (b)

Exercise 4.3

(a) (i) See Figure 11

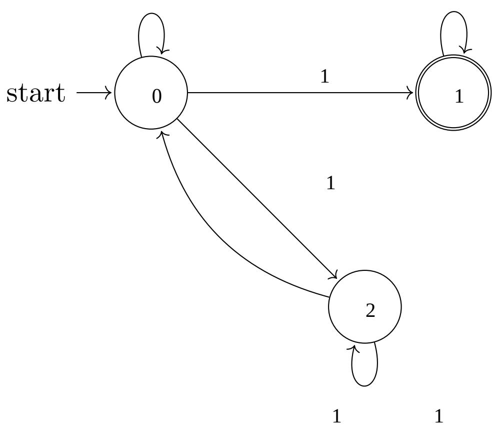


Figure 11: Exercise 4.2 point (b)

(ii) Recall the Semaphore based TS(Figure 42).The product with the previous point's NFA is given in Figure 12, as one can see no state has the right projection equal to $_1$.

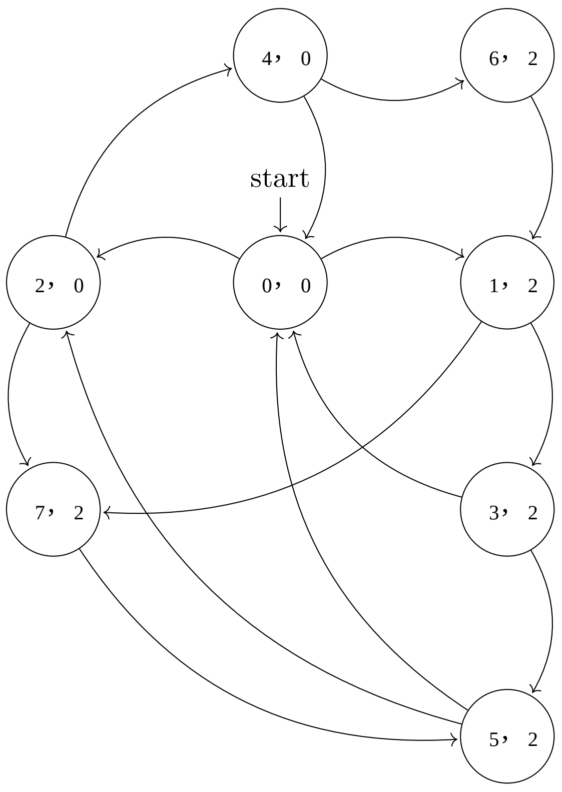


Figure 12: Exercise 4.3. (a) point (ii): Product Graph for Semaphore Algorithm

(b) (i) The NFA can be seen in Figure 13.

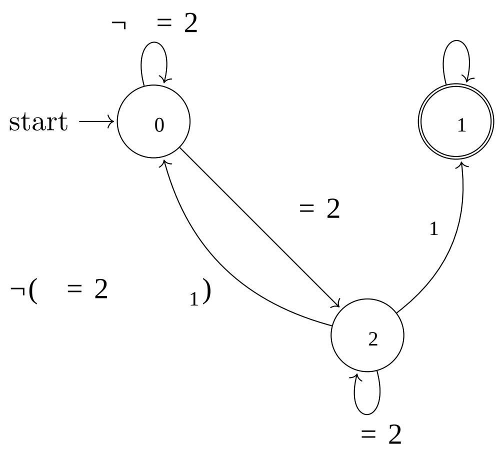


Figure 13: Exercise 4.3 point (b) the NFA of Bad Prefixes for "process 1 never enters its critical section from a state with x=2"

(ii) Recall Peterson's TS in Figure 41 and then we show (part of) the product graph in Figure 14

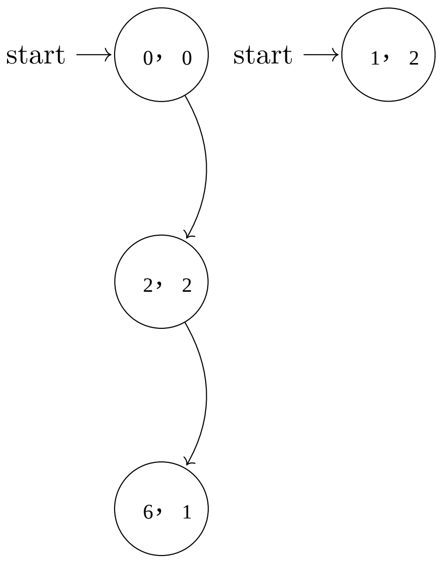


Figure 14: Exercise 4.3 point (b) (ii) part of the product automata for the property "process 1 never enters its critical section from a state with x=2". Clearly we can reach a state where ( ) = $_1$. So we conclude the TS does not satisfy the property.

Exercise 4.4

(a) It is true. Take the NFA = (, Σ, , $_0$,) corresponding to , then by hypothesis this NFA accepts the minimal bad prefixes of and some other bad prefixes. It is enough to remove any outgoing edge from end states in , obtaining the NFA ′, to get an NFA accepting the minimal bad prefixes of . If (| | =) is a minimal bad prefix then it is accepted by by construction, which means there exists a run such that . and $_1$ → ^{1} $_2$ → ^{2} … → . Then it is obvious that there cannot be another such that < and because this would mean ′ = $_1$… is a bad prefix (since ( )) of a minimal bad prefix . Then we know that ′ can imitate this same path and thus accept any minimal bad prefix of . For the other direction, namely that ′ accepts only the (minimal) bad prefixes: since ′ is after we remove some edges it cannot expand the language accepted, thus ′ accepts only Bad Prefixes of and because no end state has outgoing transition it accepts only the minimal ones.

(b) False. Consider the regular safety property , then ( ) = and ( ) = $(2^{\{0,1\}})$. Now consider = {{0} {1} + 1| ≥ 0} which is clearly not regular.

Exercise 4.5

First, some considerations on the NFA: it is described by the RE:

(( ) .(.(+). .(+)) .) .( ) .

. One possible way to describe the bad prefixes represented by this NFA could be: $_{-1}$ = { }.

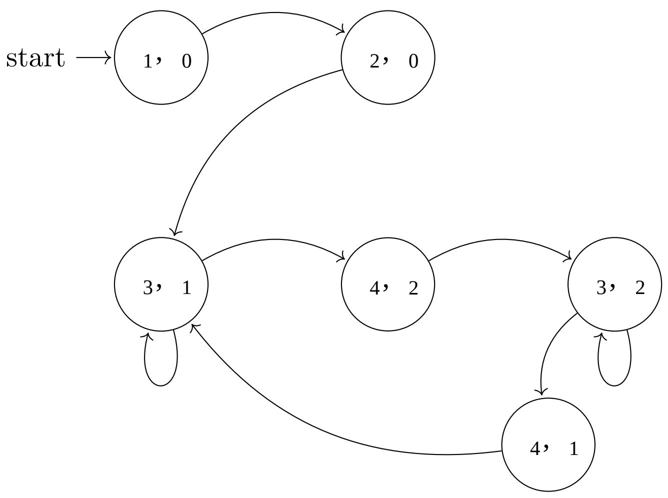


Figure 15: Exercise 4.5, the reachable fragment of the product

Exercise 4.6

I read this property as "Always, if becomes valid and ¬ holds in one of the states *before* then in every state *after* neither nor hold until a holds". I consider the set = { , , }.

(a) See Figure 6.

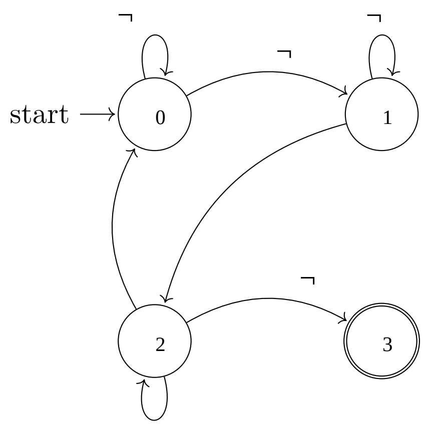


Figure 16: Exercise 4.6 part (a): The NFA accepting MinBadPref( )

(b) The TS satisfies the property as a state of the form $, {}_3$ is not reachable, see Figure 17.

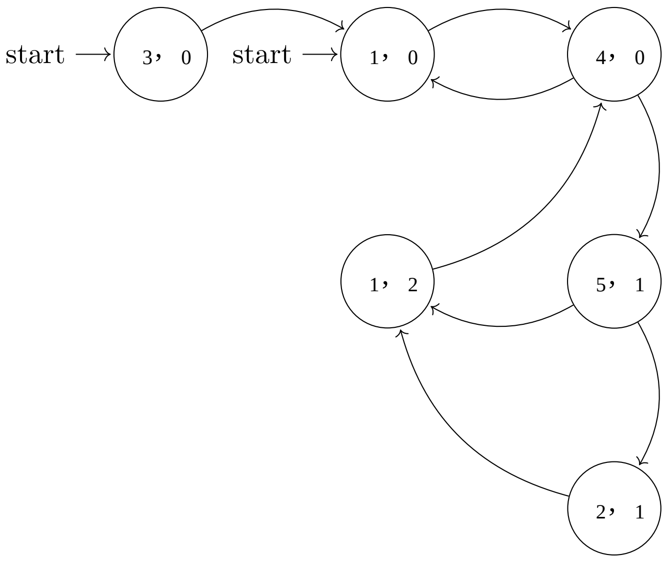


Figure 17: Exercise 4.6 part (b): The Product

Exercise 4.7

(a) True.

$$({}_1 \cdot \quad \underset{1}{+} \quad {}_2 \cdot \quad) = \quad ({}_1) \underset{2}{.} \; (\;) \qquad ({}_2) \underset{3}{.} \; (\;)$$

so any word 1 is either a word in 2 or 3 (non exclusive or).

$$((\ _1 + \ _2). \ ) = (\ _1 + \ _2). (\ )$$
$$4 \qquad 5$$

Suppose, without loss of generality, 2, then it is made of a prefix in ($_1$) and an -suffix in ( ) . Since ($_1$) ($_1$ + $_2$) we conclude. For the other direction: if 4 then it is made of a prefix in ($_1$ + $_2$) and a suffix in ( ), since ($_1$) or ($_2$) we conclude.

(b) False. Take Σ = { , , } as an alphabet, = , $_1$ = , $_2$ = . Then .(.) ( ).($_1$ + $_2$) by definition of the operator. But (. $_1$) + (. $_2$) = { .( ) , .( ) }

(c) True. We can just show

$$(\ .\ ) = (\ )$$
$$1 \qquad 2$$

Take any word 1. , is by definition, an infinite concatenation of words . . We prove that there exists a word $'$ 2 such that . $_1$... is a prefix of $'$. This equals to say that $'$ = . By induction:

- i=1: by definition $_1$ is a concatenation of one or more (finite) words from , i.e. $_1$ = $_{1,1}$· $_{1,2}$ ⋯ $_{1,}$. Let $'_1$ = $_{1,1}$, $'_2$ = $_{1,2}$ ⋯ $'$ = $_{1,}$.
- i=k+1: suppose, by inductive hypothesis, that the prefix $_1$... is long . suppose $_{+1}$ = $_{+1,1}$· $_{+1,2}$ ⋯ $_{+1,}$ then let $'_{+1}$ = $_{+1,1}$, $'_{+2}$ = $_{+1,2}$ ⋯ $'_{+}$ = $_{+1,}$.

For the other direction, namely that 2 1 we note that ( ) (.) ( ) (.) .

(d) False. Take Σ = { , } as an alphabet, = , = . Then .(.) (.) but there is no word with infinite s in . as the operator is applied only to the language { }.

Exercise 4.8

We need to proceed by cases on the form of , let be an operator which takes in input a generalized -regular expression and returns one equivalent of the form + $_1$· $_1$ + ... :

1. ( ) =
2. ( ) =
3. ( ) = { }
4. ($_1$ + $_2$) = ($_1$) + ($_2$)

5. $(\ _1.\ _2)$: we know by inductive hypothesis that $(\ _1) = \ + \ _1.\ _1 + \dots\ .$ and $(\ _2) = \ ' + \ _1'.(\ _1') + \dots\ '.(\ ')$ then we need to concatenate finite words of $_1$ with finite and infinite words of $_2$ and append the infinite words of $_1$:

$$(\ _1.\ _2) = \ .\ ' + \ _1.\ _1 + \dots\ .\ + \ .\ _1'.(\ _1') + \dots\ .\ '.(\ ')$$

6. $(\ )$: by inductive hypothesis $(\ ) = \ + \ _1.\ _1 + \dots\ .$ thus

$$(\ ) = \ + \ .\ _1.\ _1 + \dots\ .\ .$$

7. $(\ )$: by inductive hypothesis $(\ ) = \ + \ _1.\ _1 + \dots\ .\ .$ Following the same reasoning as for the Kleene closure, $(\ ) = (\ )\ .$

Exercise 4.9

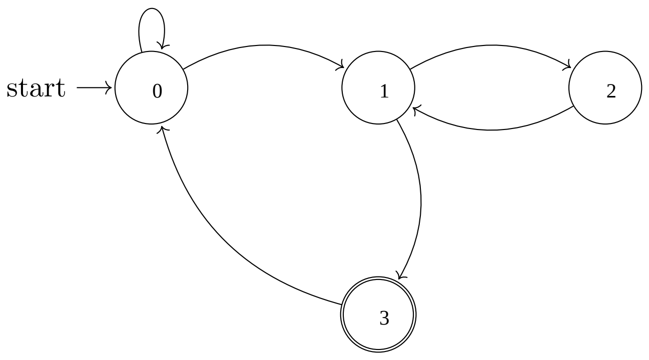


Figure 18: Exercise 4.9. NBA for the property ”infinitely many A, and between two A there is an odd number of B”

Exercise 4.10

(a) See Figure 19 for a visual description of the NBA. An -regular expression is: $(\ +\ )\ .(\ .((\ .\ )\ .\ + (\ .\ )\ .\ ))$

(b) See the NBA in Figure 20, the -regular expression becomes

$$((\ +\ )\ .(\ .((\ .\ )\ .\ + (\ .\ )\ .\ ))).(\ .((\ .\ )\ .\ ))$$

(c)

(d)

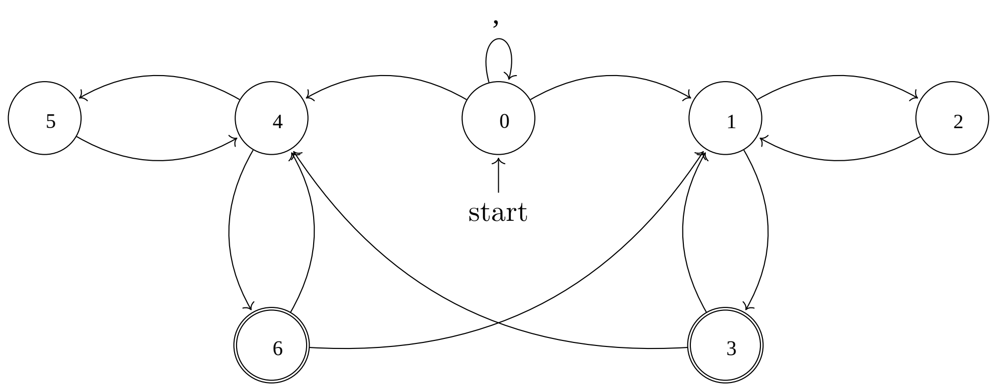


Figure 19: Exercise 4.10 NBA for point (c)

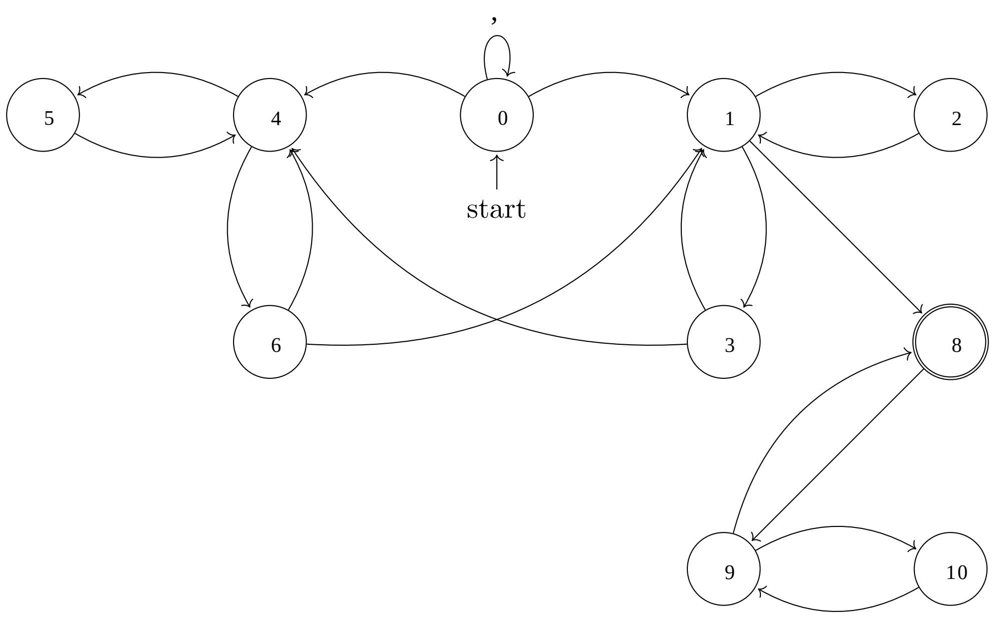


Figure 20: Exercise 4.10 NBA for point (b)

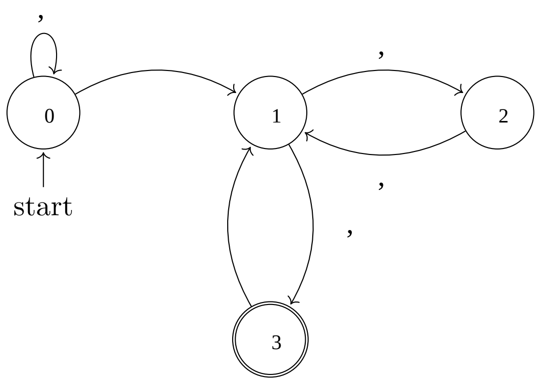


Figure 21: Exercise 4.10 NBA for point (c)

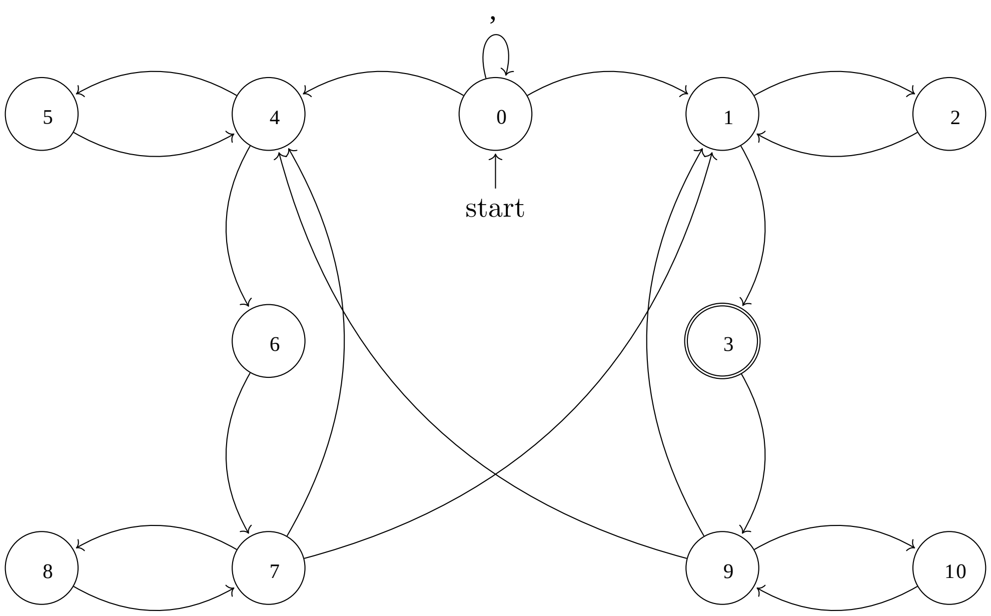


Figure 22: Exercise 4.10 NBA for point (d)

Exercise 4.11

The Automata has two initial states and two final states and it is depicted in Figure 23.

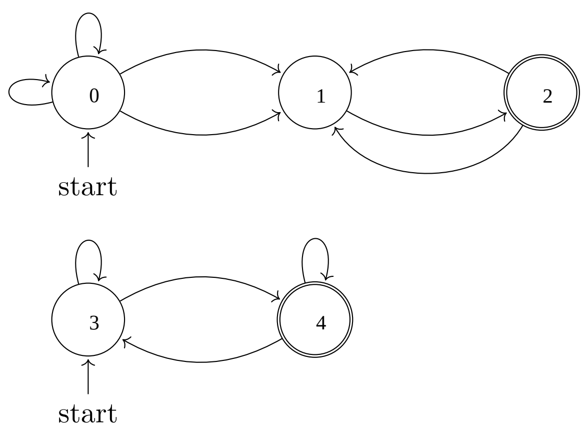


Figure 23: Exercise 4.11 NBA for the -regular expression (+) ((+)) + ( )

Exercise 4.12

For $_1$ (.(+ +) . + .(+ +) .) , for $_2$ (+) . .(.) + (+) . .(.) . . + (+) . .

Exercise 4.13

First we apply operator to $_2$. Intuitively, this means that from each end state we can operator as any initial state but keeping in mind that this holds only if the end state has no outgoing edges. Since $_2$ has outgoing edges we need to duplicate him into a new node $_5$ which takes all its outgoing edges, see Figure 24. Then to make the concatenation NBA, we need to make the end state of $_1$ act as any initial state of $_2$: in this case we just need to give the outgoing transitions of $_0$ to $_0$.

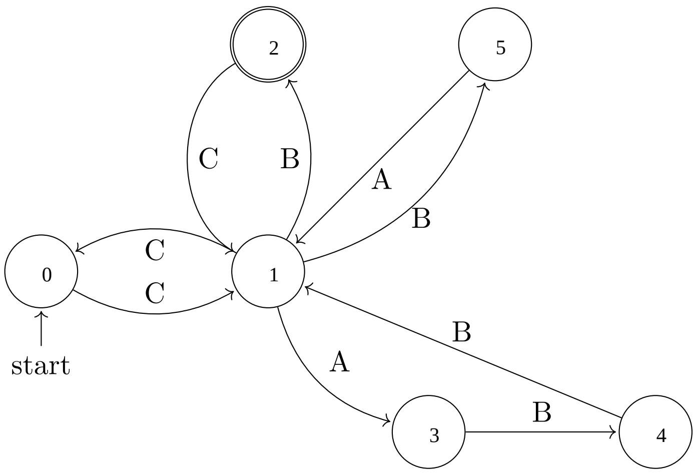


Figure 24: Exercise 4.13 NBA for $_2$

Exercise 4.14

This property means ”infinitely often but at least once ¬ . Let = { } + { } + { } + . So .{ }.(.{ , }) .

Exercise 4.15

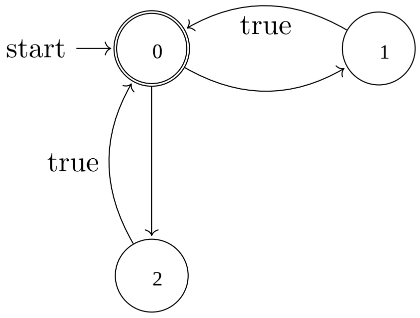


Figure 25: Exercise 4.15 NBA’s

Exercise 4.16

($_1$) = , ($_2$) = , however the powerset construction yields in both cases an automata with language accepted .

Exercise 4.17

(a) $(2 \quad . \,_1 .(2 \quad ^{\{\ _1\}}) \ . \,_2 .(2 \quad ^{\{\ _2\}}) \ . \,_3 \dots \quad)$ when i write 2 i mean the regular language of this finite set, i.e. the + of each of its members.

(b) Suppose there was one such automata $'$. One word accepted by the language is $= (\,_1 . \,_2 . \,_3 \dots \quad)$ and so it must be accepted also by the automata $'$. It can be shown that there exists a path with *Traces*( ) = such that a final state appears in the first positions of the states traversed. The proof relies on the fact that we *need* to encounter an accepting state while consuming an $.1 \leq \quad \leq \quad$. Say this first happens on the th state we traverse, then it means we arrived in the $-$ $+$ 1th state with $_1$. So we can take the path $\,_{-\ +1} \,_{-} \dots$ **as before onward** as from $\,_{-\ +1}$ we exit with $_1$, from $_{-}$ with $_2$ and so on. Remember that since is accepted also Σ is accepted so we do not care about any finite prefix, we therefore always talk about acceptance of for clarity. Let us reference the new path as $_1$, $_2, \dots$ ... and say the $, \leq$ -th state is the final one. By pigeon-hole principle there must be at least one state repeating in the prefix $_1, \dots$, say on indexes and $+$. It is clear that once we exit the state with label and the other time with one labeled $_+$ mod (here the modulo moves us back to 1 not 0). So we divide the problem by 3 cases:

- $\leq \quad \leq \quad + \quad$: but wait...we can now create the path

$$(\quad \rightarrow \dots \quad \dots \,_+ \overset{+}{\rightarrow} \dots) = (\quad , \dots , \quad _{+\ -1})$$

  which is not in our language as $_+$ will not be repeated infinitely many times.

- $+ \quad < \quad$: but then I can do the path $_1 \dots \quad \overset{+}{\rightarrow} \dots \quad \dots$ but the string $_1 \dots \,_{-1} . \,_+ \dots \quad .(\,_1 \dots \quad)$ is not in our language as it is not accepted by the automata.

- $> \quad$: similar to the previous case.

Exercise 4.18

Pick any NFA such that its end states do not have outgoing edges, then the language accepted by the NBA is empty.

Exercise 4.19

TODO

Exercise 4.20

The NBA has to contain a strongly connected component $_1$ where a final state of the NBA is reachable and for each state in this SCC the path gives a trace which is a string in $(\quad + \quad)$. It is not possible for $_1$ to allow for transitions of the like $\to \quad ' \to$ as it would enable us to accept strings of the like ”there are infinitely many BBs or AAs”. Now from some state we can consume and then we should be able to access $_1$ (to express) but also to consume another which cannot go in $_1$ (because in $_1$ we cannot express the language $(\quad + \quad)$). Thus, has to have two transitions labeled , hence it is not a DBA.

Exercise 4.21

TODO

Exercise 4.22

TODO

Exercise 4.23

Consider two generic non-blocking DBAs $_1 = (\;, \Sigma, \;, \;_0, \;), \;_2 = (\;', \Sigma, \;', \;'_0, \;')$ I claim that the DBA $_1\;_2 = (\quad \times \quad ', \Sigma, \quad, \;_0 \times \;'_0, \quad \times \quad ' \qquad ')$ recognizes $(\;_1\;_2) = \quad (\;_1) \qquad (\;_2)$ where is defined as:

$$(\;, \;, \;) = \;(\;, \;), \;'(\;, \;)$$

Let $\qquad (\;_1) \qquad (\;_2)$, then assume $\qquad (\;_1)$, the proof does not care whether $\qquad (\;_2)$ and it is totally symmetric if we assumed $\qquad (\;_2)$ instead. We have $\qquad (\;_1) \qquad . \qquad (\;_1). \qquad (\;) = \;$. Take the path $'$ in $_1\;_2$ such that $(\;, \;) = \;' \to \;'_{+1} = (\;, \;)$ if and only if $\; = \qquad = \;'($`take_right(`$\;'_{-1}, \;$`))`. Now it is evident that `take_left(`$\;'$`)` $= \;, \;$ ÿ

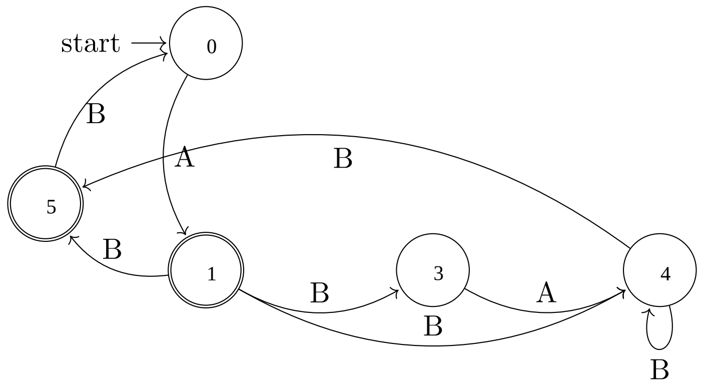


Figure 26: Exercise 4.24 NBA's

Exercise 4.25

TODO

Exercise 4.26

(a) $= (\quad + \quad (\quad)^{+} \quad) \quad + (\quad + \quad (\quad)^{+} \quad) \quad .(\quad)$

(b) The GNBA $= (\;, \Sigma, \;, \;_0, \;)$ accepts the same language as the Muller Automaton $(\;, \Sigma, \;, \;_0, \;')$ where $' = \{ \; | \qquad . \qquad \}$.

Exercise 4.27

(a) You can view the NBA for in Figure 27 and the one for its complement in Figure 28.

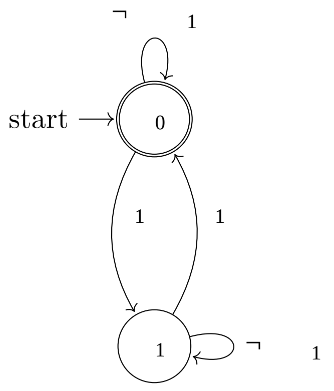


Figure 27: Exercise 4.27 NBA for

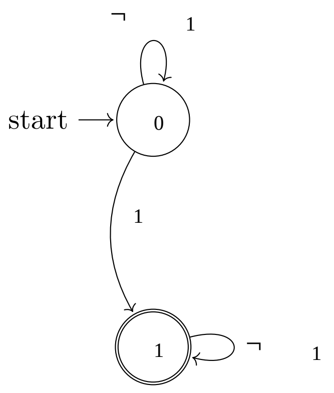


Figure 28: Exercise 4.27 NBA for the complement of

(b) (i) Consider in Figure 42. The reachable fragment of ˆ is depicted in Figure 29. The nested DFS computes in which reachable states ¬ ¬ $=$ $_1$. For example $_{7,\ 1}$. Then from this state runs a DFS and it first adds on the stack $_{6,\ 1}$ and checks if there is an edge $_{6,\ 1} \rightarrow$ $_{7,\ 1}$ since it is negative, it checks $_{1,\ 1}$ which has an outgoing edge to $_{7,\ 1}$ and thus it returns $_{7,\ 1}$, $_{6,\ 1}$, $_{1,\ 1}$, $_{7,\ 1}$.

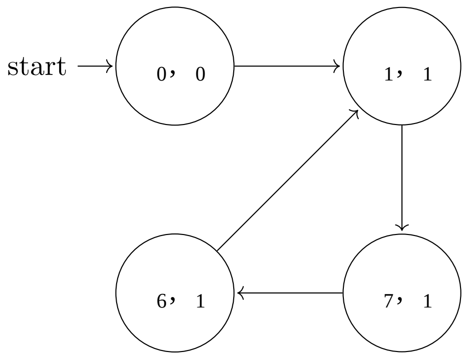


Figure 29: Exercise 4.27

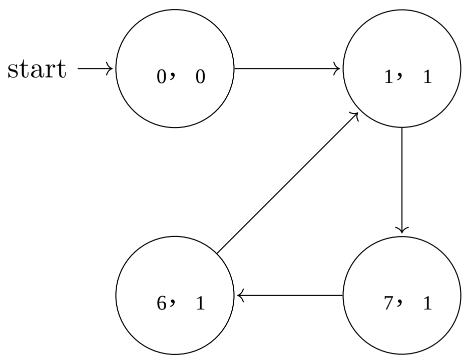


Figure 30: Exercise 4.27

Linear Time Logic

Exercise 5.1

Let $= \{ {}_1, {}_2, {}_3, {}_4\}$.

(1) $\circ$a is satisfied by $_4$

(2) $\circ\circ\circ$resolves in computing all paths of length 3 starting from a certain state and checking whether they end up in $_4$, clearly this happens for any path starting from any state, so the whole state space satisfies the formula.

(3) This formula is satisfied by no state as eventually we enter in a state in $\{ {}_1, {}_2\}$

(4) It is satisfied by

(5) It is satisfied by

(6) Any state satisfying $\Diamond$ also satisfies $\Diamond($ $)$ and all the states satisfy $\Diamond$

Exercise 5.2

- $\neg$ $_1$: there is no state which satisfies $\Box$.
- $_2$: the only way not to satisfy this formula would be to loop through states which do not have in their label set. The only two states which do not have are $_1$ (which is not reachable apart from when the path starts from it) and $_4$ which, however, has no outgoing edges to $\{ {}_1, {}_4\}$
- $\neg$ $_3$: $_1 \rightarrow {}_4 \rightarrow {}_5 \rightarrow {}_4$ $($ $)$ but $\neg$ $_3$.
- $\neg$ $_4$ because $({}_2)$
- $_5$ because all paths in the SCC $' =$ $_1$ satisfy $\Box$ and for a path starting from $_1$ the only two outgoing edges are into $'$.
- $_6$ by the same reasoning as for $_5$.

Exercise 5.3

(a) $\square(\quad \bigcirc\neg \;)$

(b) $\square(\quad \bigcirc\bigcirc \quad \bigcirc \quad)$

(c) $\square(\quad (\; _1 \quad \bigcirc \; _1 \quad \neg \; _1 \quad \bigcirc\neg \; _1 \quad _2 \quad \bigcirc \; _2 \quad \neg \; _2 \quad \bigcirc\neg \; _2))$

(d) $\square\Diamond \; _1$

Exercise 5.4

Let us introduce the following definitions: PR = Peter.Request, PU = Peter.Use, PL = Peter.Release and with BR, BU, BL the correspondent for Betsy.

(a) $\square(\neg \quad \neg \quad)$

(b) $\square(\quad \Diamond\neg \quad) \quad (\quad \Diamond\neg \quad) \quad (\neg \quad \neg \quad)$

(c) $\square(\quad \Diamond \quad) \quad (\quad \Diamond \quad) \quad (\neg \quad \neg \quad)$

(d) $\square(\Diamond \quad \Diamond \quad)$

(e) $\square((\quad \bigcirc(\neg \quad)) \quad (\quad \bigcirc(\neg \quad))) \quad (\;)$

Exercise 5.5

Let   denote the atomic proposition "the elevator is at the i-th floor",   denote the atomic proposition "the i-th floor's door is open",   denote "the elevator is requested at the i-th floor".

(a) $(\quad _{=0}) \quad ^{-1}$

(b) $\square \quad _{=0}^{-1}(\square\neg(\quad) \quad \neg \quad)$

(c) $\square\Diamond \quad _0$

(d) $\square((\neg \quad _{=0}^{-2}(\neg \quad) \quad _{-1}) \quad \neg \quad _{-1})$

Exercise 5.6

Consider   =   and   = .

(a) They are not equivalent. { } { } satisfies the formula on the left but not the one on the right.

(b) They are equivalent. If $\Diamond\Box \rightarrow \Box\Diamond$ then either is not eventually always true or it is and this implies is always eventually true. Let's put ourselves in the first case, then consider any .

- If then take the first such that $\neg$, thing we can do because $\neg$ $\Diamond\Box$ and moreover $\neg$ $\Box$. Then $<$ $<$. because was chosen minimal, and thus ($\neg$).
- If $\neg$ then we are immediately done because we chose as prefix where holds the empty one.

Now, consider the case where $\Diamond\Box$, that is, there exists minimal such that $\Box$. Then it must also be true that $\Box\Diamond$. Consider any :

- If $<$ we use the same reasoning as above as we are sure . $\leq$ $<$ such $\neg$ as, if that was not the case, $\Box$ but was chosen to be minimal index such that $\Box$ and $<$.
- If $\geq$ the reasoning above does not apply as there is no next state where $\neg$ holds. But we know $\Diamond$ so there must exist $\geq$ such that so $\neg$ and since $\leq$ $<$ we have ($\neg$).

Now lets prove the opposite direction, namely that ($\Box($ $(\neg$))) ($\Diamond\Box \rightarrow \Box\Diamond$). Let's work by contradiction, so we assume $\Diamond\Box$ $\neg$ $\Box\Diamond$. By the latter part of the conjugation we know there exists a such that $\neg$ $\Diamond$, i.e. $\Box\neg$. By the first part of the conjugation we know there exists such that $\Box$ so let $=$ (,). Then, $\Box($ $\neg$) so $\neg$ $\Diamond(\neg$) $\Diamond$ thus it cannot satisfy $(\neg$) but we assumed $\Box$ $(\neg$), contradiction.

(c) $\Box$ is idempotent, so $\Box\Box($ $\neg$) $\equiv$ $\Box($ $\neg$). The rest is DeMorgan, so they are equivalent.

(d) { }{ } $\Diamond$ $\Diamond$ but $\neg$({ }{ } $\Diamond($))

(e) They are equivalent. In particular any structure satisfying $\Box$ also satisfies $\bigcirc\Diamond$.

(f) { } $\Diamond$ but $\neg${ } $\Diamond$ $\bigcirc\Box$ as $\neg${ } $\bigcirc\Box$ because $\neg$ $\Box$.

(g) They are not equivalent. { } $(\Box\Diamond$) $\rightarrow$ $(\Box\Diamond$) vacuously but $\neg($ $\rightarrow \Diamond$) as it requires at least an index such that but clearly no set contains .

(h) They are equivalent, this is a basic duality notion.

(i) If $_1 \not\equiv$ $_2$ this is easily not always true. If $_1 \equiv$ $_2$ then they are equivalent: Suppose > 0 and $_1$ then take $' =$ $- 1$ and $'$ $\bigcirc$ $_1$ $'$ $\bigcirc$ $_2$ that is $\Diamond\bigcirc$ $_2$. Where we used equivalence in the .

(j) They are equivalent. If $(\Diamond\Box$ $_1)$ $(\Diamond\Box$ $_2)$ then there are two indexes , such that $\Box$ $_1$ and $\Box$ $_2$. So $_{\max(,)}$ $(\Box($ $_1$ $_2))$. For the other direction by the same reasoning you obtain $=$.

(k) They are equivalent.

- : assume $(\ _1\ \ _2)\ \ _2$ then $.\ \ _2\ \ <\ \ (\ _1\ \ _2)$, where such is minimal. Then $\ \neg\ _2$ which implies $\ _1$, by until semantics. Thus $<\ .\ \ _1$.
- : assume $(\ _1\ \ _2)$ then there $.\ \ _2\ \ <\ .\ \ _1\ \neg\ _2$. We need to prove $<\ .\ \ _1\ \ _2$ i.e. that there exists $\geq$ such that $\ _2$ and $\leq\ <\ .\ \ _1$. Take $=\ .$

Exercise 5.7

(a) N $\equiv \neg\ \ (\ \ )$

(b) W $\equiv\ \ \neg$

(c) B $\equiv (\Diamond\ \ \rightarrow (\ \ \ )$ if the *before* is to be intended as non-strictly before otherwise if needs to be satisfied in a moment strictly before I propose $\Diamond\ \ \rightarrow$ $(\ \ (\ \ \neg\ ))$

Exercise 5.9

(a)

$$\begin{aligned}
&\neg(\neg\ \ \neg\ ) \equiv \\
&(\neg\ \ \ )\ (\ \ \ )\ \ \Box(\neg\ \ \ ) \equiv \\
&(\ \ \ )\ \ (\neg\ \ \ \ \circ(\ ))\ \ (\neg\ \ \ \ \circ\Box(\neg\ \ \ )) \equiv \\
&\quad(\ \ (\neg\ \ \circ(\ )))\ \ (\ \ (\neg\ \ \circ\Box(\neg\ \ \ ))) \equiv \\
&\quad(\ \ (\neg\ \ \circ(\ ))\ \ (\neg\ \ \circ\Box(\neg\ \ \ ))) \equiv \\
&\quad(\ \ (\neg\ \ (\circ(\ )\ \ \circ\Box))) \\
&\quad(\ \ (\circ(\ \ \ )))
\end{aligned}$$

(b) Just take the first equivalence of point a.

Exercise 5.10

Assume $+\ 1$ atomic propositions: $\ _0, \ldots,$ then let $\neg(\ \ _{-1}\mathsf{U}\ \ )$ and $\ _0 = \ _0$. Then we get $|\ _0| = 1 = 2^0$. And for the recursive case we have $|\ \ | = ((\ \ _{-1}$ $\neg\ \ )\mathsf{U}(\neg\ \ _{-1}\ \ \neg\ \ ))\ \ \Box(\ \ _{-1}\ \ \neg\ \ )$ whose length is $2\ \ |\ \ _{-1}| + |\neg\ \ _{-1}| + 1(\mathsf{U}) +$ $2(\ ) + 2(\neg)$ which by inductive hypothesis is $2\ \ 2^{\ -1} + |\neg\ \ _{-1}| + 5 > 2\ $.

Exercise 5.17

(a)

= □(○(¬)) ≡

□((¬ ○) (○¬)) ≡

¬(¬(¬ ○) ¬(○¬)) =

(b)

$_1$ = { , ○¬ , ○¬ , ¬(¬ ○), ¬ }

$_2$ = { , ○ , ¬(○¬), ¬(¬ ○), }

$_3$ = { , ○¬ , ○¬ , ¬(¬ ○), }

$_4$ = {¬ , ○¬ , ¬(○¬), ¬(¬ ○), }

$_5$ = {¬ , ○}

Computation Tree Logic

Exercise 6.8

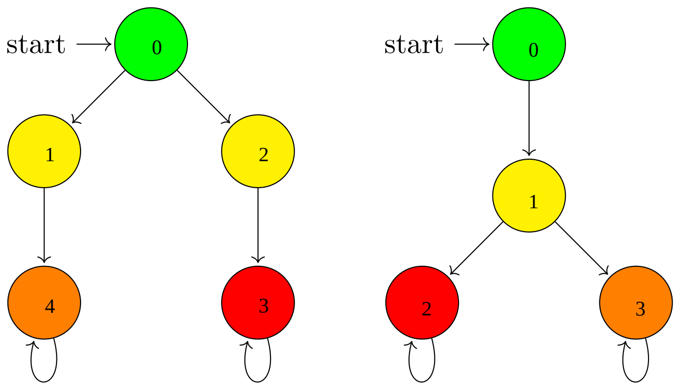


Figure 31: Exercise 6.8

The two transition systems are trace equivalent but the one on the left does not satisfy the CTL formula $\bigcirc \;\; \bigcirc \mathit{red}$

Exercise 6.13

The implication is false, as removing paths in $\;'$ may invalidate state formulas previously true in . Suppose $\Phi = \mathit{green}$ and $\Psi = \;\bigcirc \mathit{red}$. Then removing the transition from $\;_1$ to $\;_2$ makes $\;_1$ not satisfy anymore Ψ and as a consequence no reachable state satisfies $\;\bigcirc \mathit{red}$.

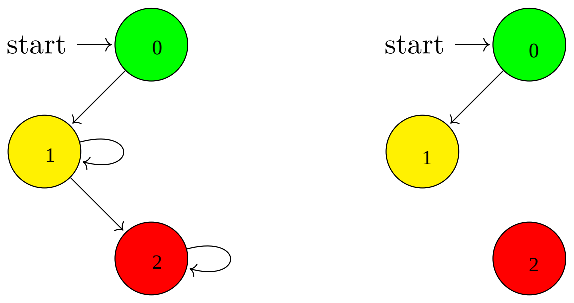


Figure 32: Exercise 6.13; implication counterexample

The other direction is true: suppose there is a path that leads in *TS'* to a state where Ψ

Exercise 6.17

See the pseudo code here 2

Algorithm 2 Computing $((\Phi \cup \Psi))$

```
procedure SatComputing($= ( , , \rightarrow, , , ), \Phi, \Psi$)
    $\leftarrow$ Sat($\Phi$)
    $\leftarrow$ Sat($\Psi$)
    $\leftarrow$
    $\leftarrow$ HashMap[ , ]
    for all do
        $[\ ] \leftarrow |succ(\ )|$
    end for
    while $\neq$ do
        $\leftarrow$ Extract( )
        for all $pred(\ )$ do
            if then
                $[\ ] \leftarrow [\ ] - 1$
                if $[\ ] = 0$ then
                    Add( , )
                    $\leftarrow \{\ \}$
                end if
            end if
        end for
    end while
    return
end procedure
```

Exercise 6.18

(a) We need to prove that for any state , $((\Phi\ \ \Psi))$ where is the greatest subset of such that it satisfies the recurrence:

$$(\Psi)\ \ \{\ \ (\Phi)|\ \ (\ ) \cap\ \ \neq\ \ \}$$

One can find by finding the greatest fix point of a function $(\ )\ \ (\ )$ defined as $(\ ) = (\Psi)\ \ \{\ \ |\ \ (\ ) \cap\ \ \neq\ \ \}$. The existence of a fixed point is guarantee because $(\ )$ is a complete lattice and is monotone on the order defined as , using Knaster-Tarski.

- : assume is in the satisfying set and suppose . By hypothesis we know there exists a path $(\ )$ such that $\Phi\ \ \Psi$, this means that either Ψ is true for a and $< .\ \ \Phi\ \ \neg\Psi$ or $.\ \ \neg\Psi\ \ \Phi$.

a) In the case such a exists it implies that since (Ψ) and is maximal. Then by backward induction we can prove that all states $_1, ...,$ are part of , where the base case is $=$ and the inductive case is -1 assuming $\{ \, , ..., \, \}$ and by using the right hand side of the equation, since $_{-1}$ satisfies Φ by hypothesis, and therefore maximality forces $_{-1}$.

b) In the case there is no such state, all the states in satisfy Φ. We need to show that adding all states in , and thus adding $_1 =$, does not violate the recurrence, and hence all those states must already be present in by maximality. Suppose $' =$ $\{ \, _1, ...\}$. Take any state in , since we added $_{+1}$ and we assumed (Φ) by the right hand side condition (Ψ) $\{$ $(\Phi) |$ $(\,) \cap \, ' \neq \, \}$ and thus $'$ satisfies the condition. Since we can repeat this reasoning for all states traversed in and was the maximal set that satisfied the recurrence it implies $'$ i.e. .

- : We will prove the contra-positive: $(\neg\Psi \quad \neg\Psi \quad \neg\Phi)$ and we will do it by induction on the iteration of the operator, which we define as $^0(\,) =$, $^{+1}(\,) = (\, (\,))$. Suppose is the iteration number of the fixed point, i.e. $^{+1}(\,) = (\,)$. Take $(\,) = \{ \, _1, ... \}$, by our assumption, in every such path it exists a minimal index such that $\neg\Psi \quad \neg\Phi$, call the set of those indexes "boundaries". Take the maximum one and call it . Then we want to prove that $(\,)$. Notice that after this proof we know as our operator is monotone. We want to use induction but to do this we need to define a partial order: let be the set of states $\{(\, , \, , \,) | \, \leq$ boundary index for $\}$ and define a total order $(\, , \, , \,) \quad (\, ', \, ', \, ') \quad < \, ' \, (\, = \, ' \quad < \, ')$. That is each state of each path is labeled by an unique integer (second component) and its position in its own path(third component). Now by induction on this partial order (where the base case is going to be the maximal element) we prove that each of those states is eventually removed. We use naturals to denote the elements of the partial order. Let be the maximal element.

  - Base case: $=$. The maximal element of this total order is a state which has the third component maxed, i.e. it is the boundary index of some path. Thus it satisfies $\neg\Psi \quad \neg\Phi$ and by the same reasoning it is eliminated after the first iteration of .
  - Inductive case: -1. Suppose all states represented by labels greater than $-1 = (\, , \, , \,)$ have been eliminated. Then take any path such that is equal to up to the state, then any path that extend from have been eliminated by inductive hypothesis. Thus also is removed.

The other point is similar.

Exercise 6.20

For $(\Phi \quad \Psi) = \neg \quad ((\neg\Phi) \quad (\neg\Psi))$:

$$\Phi \quad \Psi \equiv \quad ((\neg\Phi \quad \Psi) \quad (\Phi \quad \Psi)) \qquad (\Box(\Psi \quad \neg\Phi))$$

so an expansion law for this operator is simply:

$$\Phi \quad \Psi \equiv ((\neg\Phi \quad \Psi) \qquad \bigcirc \quad \Phi \quad \Psi) \quad (\Phi \quad \Psi) \equiv \Psi \quad (\Phi \quad (\quad \bigcirc \quad \Phi \quad \Psi)).$$

for $(\Phi \quad \Psi) \equiv \Psi \quad (\Phi \qquad \bigcirc \quad \Phi \quad \Psi)$.

Exercise 6.21

(a) $(\Phi_1) = \{ \ _2, \ _3\}$. To compute (Ψ_1) one needs to take only states satisfying $\neg \quad = \{ \ _5\}$ and then take the preceding nodes satisfying , i.e. $(\Psi_1) = \{ \ _0, \ _2\}$.

(b) To compute $(\ \Box \quad)$ one takes all the states in the transition system (the ones satisfying), and then compute the SCCs: $_1 = \{ \ _0, \ _1\}$, $_2 = \{ \ _2, \ _5\}$, $_3 = \{ \ _3, \ _4\}$. Then we check those against :

- Since $_1 = \{ \ _0, \ _1\} \cap \quad (\Phi_1) =$ we find that both of those states are in $(\ \Box \quad)$.
- Since $_2 = \{ \ _2, \ _5\} \cap \quad (\Phi_1) = \{ \ _2\}$ and $_2 \cap \quad (\Psi_1) \neq$ we have $_2 \qquad (\ \Box \quad)$.
- Since $_3 = \{ \ _3, \ _4\} \cap \quad (\Phi_1) \neq$ and $_3 \cap \quad (\Psi_1) =$ we need to remove $_3$, however, $\{ \ _4\}$ is not anymore a connected component.

So in conclusion $(\ \Box \quad) = \{ \ _0, \ _1, \ _2, \ _5\}$

(c) First we put the formula in **ENF**: $\Phi \equiv \quad \Diamond \quad \Box\neg \ $. So we compute first $(\neg \ ) = \{ \ _1, \ _2, \ _3, \ _4\}$ and then remove all but $(\neg \ )$ states and then search for SCCs: $_3$ is the only SCC such that all of its states are present in $(\neg \ )$. However, no state in $_3$ has fair paths, so $(\quad \neg \ ) = \ $. By the same reasoning no state satisfies $\Diamond \quad \Box\neg$ so we conclude $(\Phi) = \ $.

Exercise 6.22

(a) This is false: the right part is strictly stronger as it requires all paths exiting the start node to be fair. Assume the unconditional fair assumption $\Box$. A counter-example is shown in Figure 35 where the given transition systems satisfies that for any fair path *green U yellow* but it does not satisfy, for the path $_0, (\ _2)$ the path formula *green U (yellow)*.

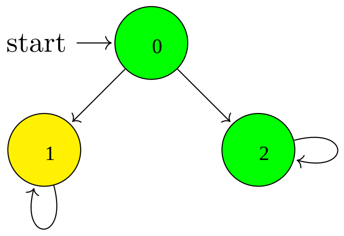


Figure 35: Exercise 6.15 b)

(b) Consider the fairness assumption □◇ and = , = . Consider the transition system shown in Figure 36: it satisfies (( )) since is forever true and stays forever false (as is false). However, this transition system does not have any fair path satisfying ( ).

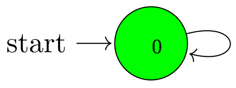


Figure 36: Exercise 6.15 b)

(c) We show the two direction separately:

- : we want to prove that ( ) we have (W( )), meaning it must be either:
  - If is a fair path then by assumption we know either:
    - there is a index $'$ such that and $<$ $'$. then take $=$ $'$, then and $<$ we have .
    - for all states it holds $\neg$ so will always be false (as is always true on a fair path) but this is accepted by the weak until semantics.
  - If is not a fair then either:
    - There is a finite prefix $_1, ...,$ where holds, and, thus $_1, ...,$ is a prefix of some fair path and thus either has been satisfied in $_1, ...,$ (and then the reasoning above applies) or it will be satisfied by following some other fair path (or it will never be satisfied). In both latter cases, we have $_1, ...,$ and thus, since $_{+1}$ $\neg$ we have $_{+1}$ ( ).
    - If . but it is an unfair path, then each is part of some fair path with prefix $_1, \dots,$. Which means either W is satisfied by some prefix $_1, \dots,$ and then we fall in the first case of the preceding point, or . $\neg$. Which is ok for the formula W( ).
- : take any fair path then we want to prove W assuming W( ). By our assumption we know two cases can occur:

- . $(\neg$). Which also means . $\neg$ which implies W .
- . $(\neg$). In this case if it easy to finish the proof. But if $\neg$?. Then we just know $\neg$, but remember is a fair path, i.e. . . So it can only be that .

Exercise 6.23

- ($\square$true): first we rewrite the fairness assumptions by replacing subformulas with appropriate atomic propositions:
  - $_1 = \square\Diamond(\ \Diamond\ _1\ \ _3)\ \ \square\Diamond\ _4$. So we first compute $(\ \Diamond\ _1\ \ _3)$ by using a standard CTL Model Checker procedure. We put it in **ENF** obtaining $\square\neg\ _1\ \neg\ _3$. Then $(\neg\ _1\ \neg\ _3) = \{\ _1,\ _3,\ _8\}$. Then $(\ \square\{\ _1,\ _3,\ _8\})$ is obtained as greatest fix point of the operator ( ) defined as:

    $$(\ ) = \{\ _1,\ _3,\ _8\}\ \ \bigcirc\ (\ )$$

    which is the set $(\ \Diamond\ _1\ \ _3) = \{\ _8\}$. Then, we can rewrite $_1$ as:

    $$\square\Diamond\{\ _8\}\ \ \square\Diamond\{\ _3,\ _8\}$$

    where { , , ... } is a short-hand for an atomic proposition valid only on states , , ...
  - $_2 = \square\Diamond(\ _3\ \neg\ _4)\ \ \square\Diamond\ _5$ can be rewritten as:

    $$\square\Diamond\{\ _2,\ _7\}\ \ \square\Diamond\{\ _4,\ _7\}$$

  - $_3 = \square\Diamond(\ _2\ \ _5)\ \ \square\Diamond\ _6$ can be rewritten as

    $$\square\Diamond\{\ _7\}\ \ \square\Diamond$$

  Then, we proceed by finding the SCCs: $_1 = \{\ _2,\ _3,\ _4\}$, $_2 = \{\ _5,\ _6,\ _7,\ _8\}$ and check whether those satisfy the fairness assumptions:

  - $_1$ has non-empty intersection with the antecedent of the second fairness assumption, but it has also non-empty intersection with the consequent so we conclude that all states that can reach $_1$ are in $(\ \square\ )$.
  - $_2$ has non-empty intersection with the third antecedent, thus $_7$ must be removed, as *false* is not satisfied by any state in the SCC. After this operation we obtain the only sub-SCC $_3 = \{\ _8\}$ which we have to recursively check for fairness.
    - $_3$ has non-empty intersection with the first antecedent, however, it has also non-empty intersection with the consequent and thus all states that can reach $_8$ are in $(\ \square\ )$.

So we conclude that all states have a fair path, $(\Box \ ) =$

- To compute $= \Phi$ we need:
  1. $(\ _1 \mathsf{U} \ _2)$, so we first compute $(\ _1 \mathsf{U} \ _2)$ by least fixed point over the monotone operator $(\ )$ defined as:

  $$(\ ) = \ _2 \ ((\ \circ \ (\ )) \ \ _1)$$

  which is $\{ \ _1, \ _2\}$. Since every state has a fair path, $(\ _1 \mathsf{U} \ _2) = (\ _1 \mathsf{U} \ _2) = \{ \ _1, \ _2\}$.
  2. We can then rewrite Φ as

  $$\circ(\{ \ _2, \ _4, \ _5, \ _6\} \qquad \{ \ _1, \ _2\})$$

  We then search for $(\ \{ \ _2, \ _4, \ _5, \ _6\} \ (\ \{ \ _1, \ _2\})) = \{ \ _2\}$. So $\circ(\ \{ \ _2\}) = \ .$

Exercise 6.26

(a) They are equivalent. $\circ \ \Box\Phi \qquad (\ ). \ ^1 \ \Box\Phi$. But any path, $(\ ^1_1)$, is the suffix of a path that starts from , i.e. , so this is equivalent to $(\ ). \ ^1 \ \Box\Phi$ which is exactly the semantics of $\circ\Box\Phi$.

(b) They are equivalent by same reasoning of (a)

(c) They are equivalent:

$$(\qquad)$$
$$(\ ).$$
$$(\ ).$$

(d) They are not equivalent. Suppose $= \circ \quad , \quad = \circ$. The transition system in Figure 37 satisfies only the formula on the right.

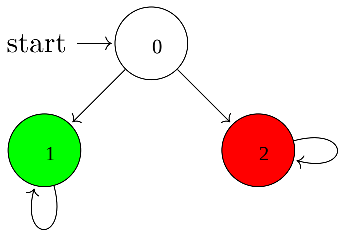


Figure 37: Exercise 6.26 d)

(e) They are equivalent. $\neg$ $\neg$ and then we apply DeMorgan on $\neg$.

(f) They are not equivalent. Suppose $\Phi =$. Consider the transition system in page 45. It satisfies the left formula as the path $(_0 \ _1)$, in any state, is allowed a path such that in the next state is allowed: in case we are in $_0$ simply $(_0 \ _1)$, in case of $_1$ we have $(_1 \ _2 \ _0)$ as $_2$. However there is no path such that $\circ$ is continuously satisfied.

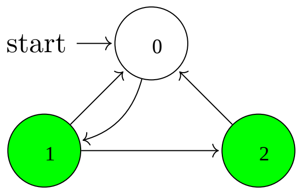


Figure 38: Exercise 6.26 f)

(g) They are not equivalent. Consider $\Phi =$, $\Psi =$, then Figure 39 satisfies $(\Diamond\Psi \quad \Box\Phi)$ as the (only) path $_0(_1)$ satisfies eventually red (by $_0$) and always green (because both states satisfy green). However, the transition system does not satisfy $\Box(\Phi \quad \Diamond\Psi)$ because the path $= _0(_1)$ does not satisfy $\Box(\Phi \quad \Diamond\Psi)$: take ^{1}, from now on, it is true that it satisfies Φ but $\Diamond\Psi$ is not satisfied.

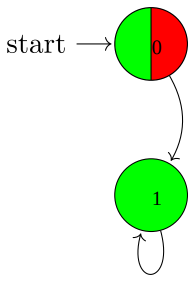


Figure 39: Exercise 6.26 g)

(h) They are different. Take $\Phi =$, $\Psi =$ and see Figure 40. As $_0$ does not satisfy Φ the transition system does not satisfy $(\Diamond\Psi \quad \Box\Phi)$. However, there is the (only) path $_0(_1)$ such that eventually Ψ is satisfied (in the second state) and from there onward Φ is always satisfied.

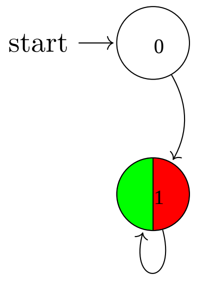


Figure 40: Exercise 6.26 h)

0.0.1 Appendix

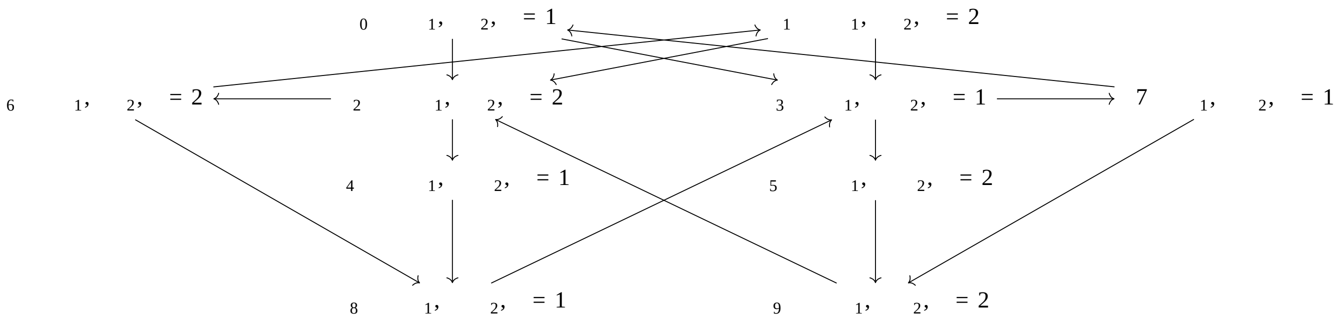


Figure 41: Transition Graph for Peterson Algorithm,

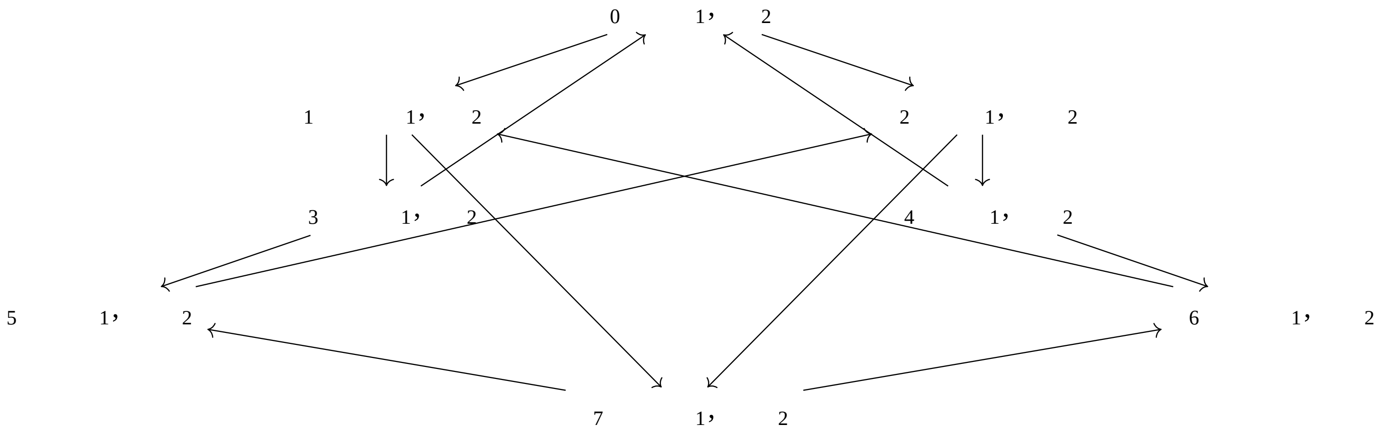


Figure 42: Transition Graph for Semaphore Algorithm,