A  LEMMA ABOUT ADAPTATION AUTOMATON

Lemma A.1 (adaptation automaton Intersection). Let \( M_1 = (Q_1, \Sigma, \Delta_1, q_1, \delta_1, \Pi_1) \) and \( M_2 = (Q_2, \Sigma, \Delta_2, q_2, \delta_2, \Pi_2) \) be SAAs then if \( M_1 \) or \( M_2 \) is an adaptation automaton then \( M_1 \cap M_2 \) is an adaptation automaton where \( M_1 \cap M_2 = (Q_1 \times Q_2, \Sigma, (q_1, q_2), \delta_1 \cap \delta_2, \Pi) \).

Proof. An automata is an adaptation automata iff for all \( q, q' \in Q_1 \times Q_2 \) and transition functions \( \delta \) then \( (q, \delta) \xrightarrow{a} (q', \delta') \) implies

- there not exists \( a \in \Sigma \) such that \( (q, \delta) \xrightarrow{a} (q', \delta) \) Proven by contradiction. Assume that there exists \( q_1, q_2 \in Q_1 \times Q_2 \), \( a \in \Sigma \) and transition function \( \delta \) such that

\[
\delta((q_1, q_2), a) = (q_1', q_2') \tag{1}
\]
\[
\Pi((q_1, q_2)) = (q_1', q_2', \delta') \tag{2}
\]

From the definition of the intersection, it must be the case that

\[
\delta = \delta_1'' \cap \delta_2'' \tag{3}
\]
\[
\delta' = \delta_1' \cap \delta_2' \tag{4}
\]
\[
\delta_1''((q_1, a) = q_1' \text{ and } \delta_2''(q_2, a) = q_2' \tag{5}
\]
\[
\Pi_1(q_1) = (q_1', \delta_1') \text{ and } \Pi_2(q_2) = (q_2', \delta_2') \tag{6}
\]

Without loss of generality assume that \( M_1 \) is a adaptation automaton. An adaptation arise as it cannot be \( \delta_1''(q_1, a) = q_1' \) and \( \Pi_1(q_1) = (q_1', \delta_1') \).

\( (q', \delta') \xrightarrow{a} \) Proven by contradiction, Assume that \( M_1 \) is a adaptation automaton. The transition \( (q, \delta) \xrightarrow{a} (q', \delta') \) could happen because there exists \( q_1 \in Q_1 \) and \( q_2 \in Q_2 \) such that \( q = (q_1, q_2) \)

\[
\Pi_1(q_1, q_2) = (q_1', q_2', \delta_1'' \cap \delta_2'') \tag{7}
\]
\[
\Pi_1(q_1) = (q_1', \delta_1') \text{ and } \Pi_2(q_2) = (q_2', \delta_2') \tag{8}
\]
\[
q' = (q_1', q_2') \text{ and } \delta' = (\delta_1'' \cap \delta_2'') \tag{9}
\]

Similarly, for the second transition \( (q', \delta') \xrightarrow{a} \)

\[
\Pi_1(q_1') = (q_1'', \delta_1'' \cap \delta_2'') \tag{10}
\]
\[
\Pi_2(q_2') = (q_2'', \delta_2'' \cap \delta_2'') \tag{11}
\]

A contradiction arise as \( (q_1'', \delta_1'') \xrightarrow{a} \) contradicts the initial assumption that \( M_1 \) is a adaptation automaton.

\( \square \)

Lemma A.2 (intersection Determinism). For the SAA\( s M_1 = (Q_1, \Sigma, \Delta_1, q_1, \delta_1, \Pi_1) \) and \( M_2 = (Q_2, \Sigma, \Delta_2, q_2, \delta_2, \Pi_2) \)

\( M_1 \cap M_2 \) is deterministic

Proof. Proven by contradiction. Assume there is \( t \) such that

\[
(q_1, q_2, \delta_1 \cap \delta_2) \xrightarrow{t} (q_1', q_2', \delta') \tag{1}
\]
\[
(q_1, q_2, \delta_1 \cap \delta_2) \xrightarrow{t} (q_1'', q_2'', \delta'') \tag{2}
\]

such that \( (q_1', q_2') \neq (q_1'', q_2'') \) or \( \delta' \neq \delta'' \). By Lemma A.8

\[
(q_1, \delta_1) \xrightarrow{t} (q_1', \delta_1'') \text{ and } (q_2, \delta_2) \xrightarrow{t} (q_2', \delta_2'') \tag{3}
\]

such that \( \delta' = \delta_1' \cap \delta_2' \) and \( \delta'' = \delta_1'' \cap \delta_2'' \) for some transition functions \( \delta_1', \delta_1'', \delta_2' \text{ and } \delta_2'' \). Since \( M_1 \) and \( M_2 \), it can never be the case that \( q_1' = q_1'' \) and \( \delta' = \delta'' \) (similarly for \( M_2 \)).

\( \square \)

Lemma A.3 (intersection Commutative). For SAA\( s M_1 = (Q_1, \Sigma, \Delta_1, q_1, \delta_1, \Pi_1) \) and \( M_2 = (Q_2, \Sigma, \Delta_2, q_2, \delta_2, \Pi_2) \)

\( M_1 \cap M_2 = a M \cap M_1 \)

Proof. Proven by structural induction on traces derived by each automata.

\( \square \)

Lemma A.4 (intersection Associative). For \( i \in \{1, 2, 3\} \) and SAA\( s M_i = (Q_i, \Sigma, \Delta_i, q_i, \delta_i, \Pi_i) \)

\( (M_1 \cap M_2) \cap M_3 = a M_1 \cap (M_2 \cap M_3) \)

Proof. Proven by structural induction on traces derived by each automata.

\( \square \)

Lemma A.5 (intersection Idempotent). Let \( M = (Q, \Sigma, \Delta, q_0, \delta, \Pi) \) be an SAA then

\( M \cap M = a M \)

Proof. Proven by structural induction on traces derived by each automata.

\( \square \)

Lemma A.6 (intersection Id). Let \( M_{\top \top} = (\{ \top \}, \Sigma, \{ \delta \}, \top, \delta, \Pi_{\top \top}) \) where \( \delta = \text{fsn}(\top, a) \Rightarrow \top \text{ for any } a \in \Sigma \) and \( \Pi_{\top \top} = \text{fsn}(\top, \delta) \Rightarrow \top, \delta \) then for all SAA \( M = (Q, \Sigma, \Delta, q_0, \delta_0, \Pi) \) implies

\( M_{\top \top} \cap M = a M \)

Proof. The empty automaton is defined as \( (Q_2, \Sigma, q_2, \emptyset, M_2) \) where the transition function and adaptation function are undefined for all \( q \in Q_2 \).

We define \( M \cap \emptyset = (Q \times Q_2, \Sigma, (q_0, q_2), \delta_0 \cap \delta, \Pi, \emptyset) \). Note that \( \delta_0 \cap \emptyset = \emptyset \) as has a transition has to be in both. Similarly \( \Pi = \emptyset \) because the adaptation function has to be defined in both automata.

\( \square \)

Lemma A.8. Assume \( M_1 = (Q_1, \Sigma, \Delta_1, q_1, \delta_1, \Pi_1) \) and \( M_2 = (Q_2, \Sigma, \Delta_2, q_2, \delta_2, \Pi_2) \) and \( M_1 \cap M_2 = (Q_1 \times Q_2, \Sigma, (q_1, q_2), \delta_1 \cap \delta_2, \Pi_1) \) be SAA\( s \)

\[
(q_1, \delta_1) \xrightarrow{t} (q_1', \delta_1'') \text{ and } (q_2, \delta_2) \xrightarrow{t} (q_2', \delta_2'') \tag{4}
\]

such that \( \delta' = \delta_1' \cap \delta_2' \) for some transition functions \( \delta_1', \delta_1'', \delta_2' \text{ and } \delta_2'' \). Since \( M_1 \) and \( M_2 \), it can never be the case that \( q_1' = q_1'' \) and \( \delta' = \delta'' \) (similarly for \( M_2 \)).

Proof. Proven by structural induction on \( t \)

Case \( t = \epsilon \) This means that \( (q_1, q_2) = (q_1', q_2') \). Result follows by reflexivity of the reduction semantics.

\( \square \)
case $t = t' \cdot a \ast$ By transitivity, we can decompose into $((q_1, q_2), \delta_1 \cap \delta_2) \xrightarrow{t} (q_1', q_2', \delta')$.

By IH, (1) implies $((q_1, \delta_1) \xrightarrow{t} (q_1', \delta_1'))$ (3) and $((q_2, \delta_2) \xrightarrow{t} (q_2', \delta_2'))$ (4) where $\delta' = \delta_1' \cap \delta_2'$.

By case-analysis on the structure of $a$:
• $a \ast = \ast$ This means that the reductions (2) and (2) happen because

\begin{align*}
\Pi_1(q_1') &= (q_1', \delta_1') \\
\Pi_2(q_2') &= (q_2', \delta_2')
\end{align*}

This allows us to extend the reduction in (5) by case-analysis on the structure of $a \ast$:
• $a \ast = \ast$ This means that $\Pi_1(q_1') = (q_1', \delta_1')$ and $\Pi_2(q_2') = (q_2', \delta_2')$.

This scenario is analogous to the previous case.

B. TRANSLATIONS FOR \( \mathcal{A} \)-EM

Lemma B.1. For an SAA $M = (Q, \Sigma, \Delta, \delta_0, \Pi)$ and $EM[M] = (Q, \Sigma, \delta_0, \Pi)$ such that for all $\sigma \in \Sigma^*$

\[(q_0, \delta_0) \xrightarrow{t} (q', \delta') \implies \exists q'' \in Q.q_0 \xrightarrow{t'} \delta' \rightarrow \sigma^{\delta'} \]

where the structure of $\sigma^{\delta'}$ can be one of the following:

1. $\{p' = q', \delta_0 = \delta'\}$
2. $\{\exists x \in Q.q_0.p' = q'(x)\}$
3. $\{\exists x \in Q.q_0.p' = q'(x) \land (\delta', \delta') = \Pi(x)\}$
4. $\{\exists p' \in Q.q_0.p' = q'(\delta') = \Pi(p')\}$

Proof. Proven by structural induction on $t$

case $t = \epsilon$ follows by reflexivity such that $q_0 = q' = q''$ and $\delta_0 = \delta'$.

case $t = t' \cdot a \ast$ By transitivity, we can decompose into $((q_0, \delta_0) \xrightarrow{t} (q', \delta'))$ can be broken down into $((q_0, \delta_0) \xrightarrow{t} (q'', \delta''))$ (1) and $((q'', \delta'') \xrightarrow{a \ast} (q', \delta'))$ (2).

By IH and (1) and (2) $q_0 \xrightarrow{t'} \sigma^{\delta'}$.

By case-analysis on the structure of $a \ast$:
• $a \ast = \ast$ and $\Pi(q''') = (q', \delta')$. Result follows from IH, such that $\Pi(q''') = (q', \delta')$ as required.

• $a \ast = \ast$ and $\Pi(q''') = (q', \delta')$. Result follows from IH, such that $\Pi(q''') = (q', \delta')$ as required.

• $a \ast = \ast$ and $\Pi(q''') = (q', \delta')$. This contradicts the initial assumption that $M$ is a translation automaton as $x$.
\[ a_e = \star \text{ and } (\exists p'' \in Q_M, q''(x), \delta''(x) = \Pi p''(x)) (2) \text{ could only happen if } \Pi q''(x) = (q', \delta'). \text{ This contradicts the initial assumption that } M \text{ is adaptation automaton as from } p'' \text{ we can derive the transitions } (p'', a) = (q'', \delta''). \]

- \( a_e \neq \star \text{ and } (p'' = q'', \delta_0 = \delta') (2) \text{ could only happen if } \delta_0 = q''(x), a = q'. \text{ Result follows from translation } \delta(p'', a) = \delta_0 = q''(x), a = q'. \]
- \( a_e \neq \star \text{ and } (\exists x \in Q_M, p'' = q''(x) \text{ and } (\delta''(x) = \Pi(x)) (2) \text{ could only happen if } \delta''(q'', a) = q' \text{ and } \delta'' = \delta''. \text{ Result follows from translation } \delta(p'', a) = q''(x) \)
- \( a_e \neq \star \text{ and } (\exists x \in Q_M, p'' = q''(x) \text{ and } (\delta''(x) = \Pi(x)) (2) \text{ could only happen if } \delta''(q'', a) = q' \text{ and } \delta'' = \delta''. \text{ Result follows from translation } \delta(p'', a) = q''(x) \)
- \( \Pi q''(x) = \text{undefined and } \delta''(q'', a) = q' \text{ We can construct the derivation } \]

\[ (q_0, \delta_0) \xrightarrow{a'} (q'', \delta'') \xrightarrow{a} (y', \delta') \]

**Lemma 2.** For an adaptation automaton \( M = (Q_M, \Sigma, \Delta, q_0, \delta_0, \Pi) \) and \( EM[M] = (Q, \Sigma, q_0, \delta) \) such that for all \( t \in \Sigma^* \)

\[ q_0 \xrightarrow{t} p' \text{ implies } \exists q'' \in Q_M, \delta'' \in \Delta, (q_0, \delta_0) \xrightarrow{\sigma} (q'', \delta'') \]

for some \( \sigma \in \Sigma^* \) such that \( \sigma = t'' \). The structure of \( p' \) can be either \( p' = q', \delta_0 = \delta' \) or \( (\exists x \in Q_M, p' = q''(x) \text{ and } (\delta'') = \Pi(x)) \)

**Proof.** Proven by structural induction on \( t \)

**Case** \( t = \epsilon \) follows by reflexivity such that \( q_0 = q' = q'' \) and \( \delta_0 = \delta' \).

**Case** \( t = t', a \)

By transitivity, \( q_0 \xrightarrow{t'} p' \) can be broken down into

\[ q_0 \xrightarrow{\nu} p'' \]

\[ p'' \xrightarrow{a} p' \]

By \( HI \) and (1)

\[ (q_0, \delta_0) \xrightarrow{\sigma'} (q'', \delta'') \]

for some \( \sigma' \in \Sigma^* \) such that \( \sigma'' = t' \). By case-analysis on the structure of \( p'' \) and \( \delta(p'', a) \)

- \( (p'' = q'', \delta_0 = \delta''), \delta(p'', a) = y'(y) \text{ such that } \Pi(p'') = (y, \delta') \) and \( \delta'(y, a) = y' \text{ We can construct the derivation } \]

\[ (q_0, \delta_0) \xrightarrow{\sigma'} (p'', \delta_0) \xrightarrow{a} (y', \delta') \]

- \( (p'' = q'', \delta_0 = \delta''), \delta(p'', a) = \delta_0(p'', a) = p'' \text{ such that } \Pi(p'') \text{ is undefined. We can construct the derivation } \]

\[ (q_0, \delta_0) \xrightarrow{\sigma'} (p'', \delta_0) \xrightarrow{a} (p'', \delta_0) \]

- \( (\exists x \in Q_M, p'' = q''(x) \text{ and } (\delta'') = \Pi(x)), \delta(p'', a) = y'(y) \text{ such that } \Pi(q'') = (y, \delta') \) and \( \delta'(y, a) = y' \text{ We can construct the derivation } \]

\[ (q_0, \delta_0) \xrightarrow{\sigma'} (q'', \delta'') \xrightarrow{a} (y', \delta') \]

**Lemma 3.** For a deterministic SAA \( M = (Q_M, \Sigma, \Delta, q_0, \delta_0, \Pi) \) and \( EM[M] = (Q, \Sigma, q_0, \delta) \)

\( \delta(q, a) = q' \text{ and } \delta(q, a) = q'' \text{ implies } q' = q'' \)

**Proof.** By case-analysis on \( \delta(q, a) \)

**Case** \( \delta(q, a) = y'(y) \text{ such that } q = x(p), \Pi(x) = (y, \delta) \) and \( y' = \delta(y, a) \text{ Result follows from determinism of } \Pi \text{ and } \delta \)

**Case** \( \delta(q, a) = y'(p) \text{ such that } q = x(p), \Pi(x) \text{ is undefined, } \Pi(p) = (y, \delta) \text{ and } y' = \delta(x, a) \text{ Result follows from determinism of } \Pi \text{ and } \delta \)

**Case** \( \delta(q, a) = y' \text{ such that } q \in Q_M, \Pi(q) \text{ is undefined, } \delta_0(q, a) = y' \text{ Result follows from determinism of } \Pi \text{ and } \delta_0 \)
C  FDR CODE

-- Configurations

size = 7

Pos = ((a,b) | a <-> {0..size+1}, b <-> {0..size+1})
GoodPos = ((a,b) | a <-> {1..size}, b <-> {1..size})

loc1 = (2,2)
loc2 = (4,5)

channel g, s : Pos.Pos
channel adapt

Loc = ((a,b) | a <-> Pos, b <-> Pos)

---------------------------------------------
-- Wiring
---------------------------------------------

Sys(strategy, buff) =
let
adaptation = s?x: GoodPos ?y: GoodPos -> Adapt(x,y,buff)
Mach = (V(loc1,loc2) |\ {{s}} \adaptation)
abstractMach = Mach [[s.x,y <- adapt | x <-> GoodPos, y <-> GoodPos]]
within
abstractMach [|| {{g, adapt}}] | strategy

---------------------------------------------
-- Adaptation Function
---------------------------------------------

Adapt(x,y,t) =
let
(a,b) = fix(x,y,t)
continuation = V(a,b) |\ {{s}} \\ s?a:GoodPos?b:GoodPos -> Adapt(a,b,t)
within
g!a!b -> continuation

---------------------------------------------
-- Transition Functions
---------------------------------------------

V(p1,p2) = let
correctMoves = \p \ q1 \ q1 <-> Pos, dis(q1,p) <= 1
within
(g?q1: correctMoves(p1) ?q2: correctMoves(p2) -> V(q1,q2))
[| (s?p1?p2 -> STOP)]

---------------------------------------------
-- Adaptation Strategies
---------------------------------------------

-- adapts when the vehicles are b units apart

adapt(b) = let

--- --
wrongMoves = \x @ diff(GoodPos, correctMoves(x))
within
  (g?x : GoodPos ?y : correctMoves(x) -> close(b))
[] (g?x : GoodPos ?y : wrongMoves(x) -> adapt -> close(b))

-- adapts every 2 steps
--(we skip the first g after adapt as that is from the co-ordinatoor)

every2Steps = let
correctMoves = \p @ {q1 | q1 <- Pos, dis(q1,p) <= 1}
within
  g?x : GoodPos ?y : GoodPos
-> g?x2 : correctMoves(x) ?y2 : correctMoves(y)
-> adapt
-> g?x : GoodPos ?y : GoodPos
-> every2Steps

-- adapts every other step
--(we skip the first g after adapt as that is from the co-ordinatoor)

everyStep = g?_ : GoodPos ?_ : GoodPos
-> adapt
-> g?x : GoodPos ?y : GoodPos
-> everyStep

transparent sbisim, wbisim, diamond, normal

---------------------------------------------
-- Helper Functions
---------------------------------------------

fix(p1,p2,buffer) =
  if (not goodLoc(p1,p2,buffer) )
  then getLoc(p1,p2,buffer)
  else (p1,p2)

goodLoc(q1,q2,buffer) = dis(q1,q2) > buffer and not outOfBound(q1,buffer) and not outOfBound(q2,buffer)

max(x,y) = if(x > y) then x else y
min(x,y) = if (x > y) then y else x
minPos(x,y) = if x < y then True else False
disA(x,y) = max(x,y) - min(x,y)

getLoc2(p1,p2,d,buf) =
  let
    q1 = min(p1,p2)
    q2 = max(p1,p2)
    d1 = disA(q1,q2)
    t1 = moveLeft(q1,buf -d1 -d + 1,buf+1)
    d2 = disA(t1,q2)
    t2 = moveRight(q2,buf -d2+1, size-buf)
  within
    if(p1 <= p2) then (t1,t2) else (t2,t1)

moveToBuffer(p1,buf) = min(max(p1,buf+1),size-buf)

moveLeft(x,dist,min) =
  if(x == min or dist <= 0) then x else moveLeft(x-1,dist-1,min)

moveRight(x,dist,mx) =
  if(x == mx or dist <= 0) then x else moveRight(x+1,dist-1,mx)
collide((a,b),(c,d), (x,y),(w,z)) =
minPos(a,c) \not\leq minPos(x,w) \lor minPos(b,d) \not\leq minPos(y,z)

outOfBound((x,y),buff) = (x + buff > size) \lor (x - buff \leq 0) \lor (y + buff > size) \lor (y - buff \leq 0)

getLoc((x1,y1),(x2,y2),buf) =
let
  x3 = moveToBuffer(x1,buf)
  x4 = moveToBuffer(x2,buf)
  y3 = moveToBuffer(y1,buf)
  y4 = moveToBuffer(y2,buf)
  d1 = max(disA(y3,y4)-1,0)
  (p1,p2) = getLoc2(x3,x4,d1,buf)
  d2 = max(disA(p1,p2)-1,0)
  (q1,q2) = getLoc2(y3,y4,d2,buf)
within
  ((p1,q1),(p2,q2))

---------------------------------------------
-- Assertions
---------------------------------------------

SEveryStep = Sys(everyStep,2)

-- Safety Properties
NoCollision = let
  correctMoves = \x @ { y | y <- Pos, dis(y,x)>0 }
within (g?x : Pos ?y : correctMoves(x) \rightarrow NoCollision)
[] (adapt \rightarrow g?x : Pos ?y : correctMoves(x) \rightarrow NoCollision)

WithinBounds =
let correctMoves = \x @ { y | y <- GoodPos }
within (g?x : GoodPos ?y : correctMoves(x) \rightarrow WithinBounds)
[] (adapt \rightarrow g?x : GoodPos ?y : correctMoves(x) \rightarrow WithinBounds)

assert NoCollision [T= SEveryStep]
assert WithinBounds [T= SEveryStep]

-- Adaptation strategy and composed automata are strongly adaptable
StronglyAdaptable =
(g7_ : GoodPos ?_: GoodPos \rightarrow StronglyAdaptable)
[] (adapt \rightarrow g?x : GoodPos ?y:GoodPos \rightarrow g7_ : GoodPos ?_:GoodPos \rightarrow StronglyAdaptable)
assert StronglyAdaptable [T= everyStep]
assert StronglyAdaptable [T= SEveryStep]

-- Composition follows the adaptation strategy
assert everyStep [T= SEveryStep]

-- Composition yields a deterministic, deadlock and livelock free system
assert SEveryStep :[deterministic \[F\]]
assert SEveryStep :[deadlock free \[F\]]
assert SEveryStep :[divergence-free \[F\]]

-- We show that changing the strategy is still safe
assert Sys(close(2),2) \{\{adapt\}\} [T= SEveryStep \{\{adapt\}\}
Eventually =
let correctMoves = \((g,a,b) \mid a \Leftarrow \text{GoodPos}, b \Leftarrow \text{GoodPos}, \text{dis}(a,b) > 0\)

testCases = \([\{\text{correctMoves}\} | p : \text{GoodPos} \implies T(p, \text{correctMoves})]\)

SuccessProc = (\(\text{SEveryStep} [\{\text{correctMoves}\} \text{testCases}] \setminus \{\text{success}\}\))

OkProc = (\(\; i : \text{seq(\text{GoodPos})} \implies \text{success} \rightarrow \text{SKIP}\); (\(\text{ok} \rightarrow \text{STOP}\))
within (\(\text{SuccessProc} [\{\text{success}\}] \setminus \text{OkProc}\) ) \(\setminus \{\text{ok}\}\)

\(T(p, \text{correctMoves}) = g?x : \text{GoodPos} \; ?y : \{ y \mid y \Leftarrow \text{GoodPos}, \text{member}(g.x.y, \text{correctMoves})\}
\rightarrow \text{if}(x == p \text{ or } y == p) \text{ then success} \rightarrow \text{RUN(correctMoves)} \text{ else } T(p, \text{correctMoves})\)

assert \(\text{ok} \rightarrow \text{STOP} \; [\text{F= Eventually}]\)
assert \(\text{ok} \rightarrow \text{STOP} \; [\text{T= Eventually}]\)

-- Eventually reaches every location
channel success, ok

-- Because the vehicles choose the next goto position at random, the implementation contains divergences
-- These assertions rightfully fail
assert Eventually :[\text{divergence-free}]\)
assert \(\text{ok} \rightarrow \text{STOP} \; [\text{F0= Eventually}]\)