Exercises #7
Temporal and modal logics
[**Ex. 1**] Two processes $p_1$ and $p_2$ want to access a single shared resource $r$. Consider the atomic propositions:

- $req_i$: holds when process $p_i$ is requesting access to $r$;
- $use_i$: holds when process $p_i$ has had access to $r$;
- $rel_i$: holds when process $p_i$ has released $r$.

with $i \in [1, 2]$. Use LTL formulas to specify the following properties:

1. *mutual exclusion*: $r$ is accessed by only one process at a time;
2. *release*: every time $p_1$ accesses $r$, it releases $r$ after some time;
3. *priority*: whenever both $p_1$ and $p_2$ require $r$, $p_1$ is granted access first;
4. *no starvation*: whenever $p_1$ requires $r$, it is eventually granted access.
Ex. 1, LTL

\[
\begin{align*}
\text{req}_i & \quad \text{use}_i & \quad \text{rel}_i \\
\text{p}_i \text{ requests } r & \quad \text{p}_i \text{ has access to } r & \quad \text{p}_i \text{ releases } r
\end{align*}
\]

mutual exclusion \quad G \neg (\text{use}_1 \land \text{use}_2)

release \quad G (\text{use}_1 \Rightarrow F \text{ rel}_1)

priority \quad G ((\text{req}_1 \land \text{req}_2) \Rightarrow ((\neg \text{use}_2) \cup (\text{use}_1 \land \neg \text{use}_2)))

no starvation \quad G (\text{req}_1 \Rightarrow F \text{ use}_1)

1. mutual exclusion: \( r \) is accessed by only one process at a time;
2. release: every time \( p_1 \) accesses \( r \), it releases \( r \) after some time;
3. priority: whenever both \( p_1 \) and \( p_2 \) require \( r \), \( p_1 \) is granted access first;
4. no starvation: whenever \( p_1 \) requires \( r \), it is eventually granted access.
[Ex. 2] Three dogs live in a house with two couches and a front garden. Let $couch_{i,j}$ represent the predicate “the dog $i$ sits on couch $j$” and $garden_i$ represent the predicate “the dog $i$ plays in the front garden”.

1. Write an LTL formula expressing the fact that whenever dog 1 plays in the garden then he keeps playing until he sits on some couch (but he may also play forever).

2. Write a CTL formula expressing the fact that dog 2 eventually plays in the garden whenever couch 1 is occupied by another dog.

3. Write a $\mu$-calculus formula expressing the fact that no more than one couch is occupied at any time by dog 3.
Ex. 2, logics

\(couch_{i,j}\)  \(\text{i sits on } j\)  \(garden_i\)  \(\text{i plays in the garden}\)

\textbf{LTL}

\(\mathbf{G} (garden_1 \Rightarrow ((\mathbf{G} \ garden_1) \lor (garden_1 \lor (couch_{1,1} \lor couch_{1,2}))))\)

\textbf{CTL}

\(\mathbf{A} \mathbf{G} ((couch_{1,1} \lor couch_{3,1}) \Rightarrow \mathbf{A} \mathbf{F} \ garden_2)\)

\(\mu\text{-calculus}\)

\(\nu x. \left((\neg couch_{3,1} \lor \neg couch_{3,2}) \land \Box x\right)\)

1. Write an LTL formula expressing the fact that whenever dog 1 plays in the garden then he keeps playing until he sits on some couch (but he may also play forever).

2. Write a CTL formula expressing the fact that dog 2 eventually plays in the garden whenever couch 1 is occupied by another dog.

3. Write a \(\mu\)-calculus formula expressing the fact that no more than one couch is occupied at any time by dog 3.
Two processes $p_1$ and $p_2$ want to access a single shared resource $r$.

Consider the atomic propositions:
- $\text{req}_i$: holds when process $p_i$ is requesting access to $r$;
- $\text{use}_i$: holds when process $p_i$ has had access to $r$;
- $\text{rel}_i$: holds when process $p_i$ has released $r$.

Use LTL formulas to specify the following properties:
1. mutual exclusion: $r$ is accessed by only one process at a time;
2. release: every time $p_1$ accesses $r$, it releases $r$ after some time;
3. priority: whenever both $p_1$ and $p_2$ require $r$, $p_1$ is granted access first;
4. no starvation: whenever $p_1$ requires $r$, it is eventually granted access.

Three dogs live in a house with two couches and a front garden.

Let $\text{couch}_{i,j}$ represent the predicate "the dog $i$ sits on couch $j$" and $\text{garden}_i$ represent the predicate "the dog $i$ plays in the front garden".

1. Write an LTL formula expressing the fact that whenever dog 1 plays in the garden then he keeps playing until he sits on some couch (but he may also play forever).
2. Write a CTL formula expressing the fact that dog 2 eventually plays in the garden whenever couch 1 is occupied by another dog.
3. Write a $\mu$-calculus formula expressing the fact that no more than one couch is occupied at any time by dog 3.

[Ex. 3] Given the $\mu$-calculus formula $\Phi = \mu x.((p \land \square x) \lor (\neg p \land \Diamond x))$ write its denotational semantics $[\Phi] \rho$ and evaluate it on the LTS below (where $V = \{s_1, s_2, s_3, s_4\}$ and $P = \{p\}$).

![LTS diagram]

$$s_1 \xrightarrow{p} s_2 \xrightarrow{} s_3 \xleftarrow{p} s_4$$
Ex. 3, mu-calculus

$$\rho(p) = \{s_4\} \quad \rho(p) = \{s_1, s_2, s_3\}$$

$$\left[\mu x. ((p \land \Box x) \lor (\neg p \land \Diamond x))\right] \rho$$

$$\triangleq \text{fix } \lambda S. \left[\left(\left(p \land \Box x\right) \lor \left(\neg p \land \Diamond x\right)\right)\right] \rho[S/x]$$

$$= \text{fix } \lambda S. \left\{ p \land \Box x \right\} \rho[S/x] \cup \left\{ \neg p \land \Diamond x \right\} \rho[S/x]$$

$$= \text{fix } \lambda S. \left( \left[ p \right] \rho[S/x] \cap \left[ \Box x \right] \rho[S/x] \right) \cup \left( \left[ \neg p \right] \rho[S/x] \cap \left[ \Diamond x \right] \rho[S/x] \right)$$

$$= \text{fix } \lambda S. \left( \rho(p) \cap \{ v \mid \forall w. v \rightarrow w \Rightarrow w \in \left[x\right] \rho[S/x]\} \right) \cup \left( \rho(p) \cap \{ v \mid \exists w \in \left[x\right] \rho[S/x]. v \rightarrow w \} \right)$$

$$= \text{fix } \lambda S. \left( \{s_4\} \cap \{ v \mid \forall w. v \rightarrow w \Rightarrow w \in S\} \right) \cup \left( \{s_1, s_2, s_3\} \cap \{ v \mid \exists w \in S. v \rightarrow w \} \right)$$
Ex. 3, mu-calculus

\[
\text{fix } \lambda S. \ (\{s_4\} \cap \{v \mid \forall w. \ v \rightarrow w \implies w \in S\}) \cup \\
(\{s_1, s_2, s_3\} \cap \{v \mid \exists w \in S. \ v \rightarrow w\})
\]

\[S_0 = \emptyset\]

\[S_1 = \ (\{s_4\} \cap \{v \mid \forall w. \ v \rightarrow w \implies w \in S_0\}) \cup \\
(\{s_1, s_2, s_3\} \cap \{v \mid \exists w \in S_0. \ v \rightarrow w\})
\]

\[= \ (\{s_4\} \cap \{v \mid \forall w. \ v \rightarrow w \implies w \in \emptyset\}) \cup \\
(\{s_1, s_2, s_3\} \cap \{v \mid \exists w \in \emptyset. \ v \rightarrow w\})
\]

\[= \ (\{s_4\} \cap \{s_2, s_4\}) \cup \\
(\{s_1, s_2, s_3\} \cap \emptyset)
\]

\[= \ \{s_4\}\]
**Ex. 3, mu-calculus**

\[
\begin{align*}
\text{fix } \lambda S. & \quad \left( \{ s_4 \} \cap \{ v \mid \forall w. v \rightarrow w \Rightarrow w \in S \} \right) \cup \\
& \quad \left( \{ s_1, s_2, s_3 \} \cap \{ v \mid \exists w \in S. v \rightarrow w \} \right) \\
S_0 = & \emptyset \quad S_1 = \{ s_4 \} \\
S_2 = & \quad \left( \{ s_4 \} \cap \{ v \mid \forall w. v \rightarrow w \Rightarrow w \in S_1 \} \right) \cup \\
& \quad \left( \{ s_1, s_2, s_3 \} \cap \{ v \mid \exists w \in S_1. v \rightarrow w \} \right) \\
= & \quad \left( \{ s_4 \} \cap \{ s_2, s_4 \} \right) \cup \\
& \quad \left( \{ s_1, s_2, s_3 \} \cap \{ s_3 \} \right) \\
= & \quad \{ s_3, s_4 \}
\end{align*}
\]

- **deadlock or can reach only** \( s_4 \)
- **has a transition to** \( s_4 \)
Ex. 3, mu-calculus

\[ \text{fix } \lambda S. \ (\{ s_4 \} \cap \{ v \mid \forall w. \ v \rightarrow w \Rightarrow w \in S \}) \cup \ (\{ s_1, s_2, s_3 \} \cap \{ v \mid \exists w \in S. \ v \rightarrow w \}) \]

\[ S_0 = \emptyset \quad S_1 = \{ s_4 \} \quad S_2 = \{ s_3, s_4 \} \]

\[ S_3 = \ (\{ s_4 \} \cap \{ v \mid \forall w. \ v \rightarrow w \Rightarrow w \in S_2 \}) \cup \ (\{ s_1, s_2, s_3 \} \cap \{ v \mid \exists w \in S_2. \ v \rightarrow w \}) \]

\[ = \ (\{ s_4 \} \cap \{ v \mid \forall w. \ v \rightarrow w \Rightarrow w \in \{ s_3, s_4 \} \}) \cup \ (\{ s_1, s_2, s_3 \} \cap \{ v \mid \exists w \in \{ s_3, s_4 \}. \ v \rightarrow w \}) \]

\[ = \ (\{ s_4 \} \cap \{ s_2, s_4 \}) \cup \ (\{ s_1, s_2, s_3 \} \cap \{ s_1, s_3 \}) \]

\[ = \ \{ s_1, s_3, s_4 \} \]

\[ \text{deadlock or can reach only } s_3, s_4 \]

\[ \text{has a transition to } s_3 \text{ or } s_4 \]
Ex. 3, mu-calculus

\[
\text{fix } \lambda S. \ \left( \{s_4\} \cap \{v \mid \forall w. \ v \rightarrow w \Rightarrow w \in S\} \right) \cup \left( \{s_1, s_2, s_3\} \cap \{v \mid \exists w \in S. \ v \rightarrow w\} \right)
\]

\[
S_0 = \emptyset \quad S_1 = \{s_4\} \quad S_2 = \{s_3, s_4\} \quad S_3 = \{s_1, s_3, s_4\}
\]

\[
S_4 = \left( \{s_4\} \cap \{v \mid \forall w. \ v \rightarrow w \Rightarrow w \in S_3\} \right) \cup \left( \{s_1, s_2, s_3\} \cap \{v \mid \exists w \in S_3. \ v \rightarrow w\} \right)
\]

\[
= \left( \{s_4\} \cap \{v \mid \forall w. \ v \rightarrow w \Rightarrow w \in \{s_1, s_3, s_4\}\} \right) \cup \left( \{s_1, s_2, s_3\} \cap \{v \mid \exists w \in \{s_1, s_3, s_4\}. \ v \rightarrow w\} \right)
\]

\[
= \left( \{s_4\} \cap \{s_2, s_4\} \right) \cup \left( \{s_1, s_2, s_3\} \cap \{s_1, s_3\} \right)
\]

\[
= \{s_1, s_3, s_4\} = S_3 \quad \text{fixpoint reached!}
\]
Ex. 3, mu-calculus

\[ \mu x.((p \land \Box x) \lor (\neg p \land \Diamond x)) \]

\[ \models \{ s_1, s_3, s_4 \} \]
Google Go
[Ex. 4] Write a GoogleGo function that takes one channel \texttt{ini} for receiving integers and one channel \texttt{ins} for receiving strings and returns a channel \texttt{outp} where all the messages received on \texttt{ini} and \texttt{ins} will be paired. 

\textit{Hint: define a \texttt{struct} to form pairs}
package main

import "fmt"

type Pair struct {
    N int
    S string
}
package main

import "fmt"

type Pair struct {
    N int
    S string
}

func pairing(ini chan int, ins chan string) (outp chan Pair) {
    outp = make(chan Pair)
    go func() {
        for {
            i := <-ini
            s := <-ins
            outp <- Pair{i, s}
        }
    }()
    return
}
package main

import "fmt"

type Pair struct {
    N int
    S string
}

func pairing(ini chan int, ins chan string) (outp chan Pair) {
    outp = make(chan Pair)
    return
}
package main

import "fmt"

type Pair struct {
    N int
    S string
}

func pairing(ini chan int, ins chan string) (outp chan Pair) {
    outp = make(chan Pair)
    go func() {
        for {
            i := <-ini
            s := <-ins
            outp <- Pair{i, s}
        }
    }()
    return
}

Ex. 4, pairing in Go
package main

import "fmt"

type Pair struct {
    N int
    S string
}

func pairing(ini chan int, ins chan string) (outp chan Pair) {
    outp = make(chan Pair)
    go func() {
        for {
            i := <-ini
            s := <-ins
            outp <- Pair{i, s}
        }
    }()
    return
}
package main

import "fmt"

type Pair struct {
    N int
    S string
}

func pairing(ini chan int, ins chan string) (outp chan Pair) {
    outp = make(chan Pair)
    go func() {
        for {
            i := <-ini
            s := <-ins
            outp <- Pair{i, s}
        }
    }()
    return
}
func main() {
    chi := make(chan int)
    chs := make(chan string)
    chi <- 1
    chs <- "Alice"
    v := <- chp
    fmt.Println("got", v)
    chi <- 2
    chs <- "Bob"
    v := <- chp
    fmt.Println("got", v)
}
func main() {
    chi := make(chan int)
    chs := make(chan string)
    chp := pairing(chi, chs)

    chi <- 1
    chs <- "Alice"
    v := <- chp
    fmt.Println("got", v)

    chi <- 2
    chs <- "Bob"
    v := <- chp
    fmt.Println("got", v)
}

Ex. 4, pairing in Go
func main() {
    chi := make(chan int)
    chs := make(chan string)
    chp := pairing(chi, chs)
    chi <- 1
    chs <- "Alice"
}

Ex. 4, pairing in Go
func main() {
    chi := make(chan int)
    chs := make(chan string)
    chp := pairing(chi, chs)
    chi <- 1
    chs <- "Alice"
    v := <- chp
    fmt.Println("got", v)
}

Ex. 4, pairing in Go
func main() {
    chi := make(chan int)
    chs := make(chan string)
    chp := pairing(chi, chs)
    chi <- 1
    chs <- "Alice"
    v := <- chp
    fmt.Println("got", v)
    chi <- 2
    chs <- "Bob"
    v = <- chp
    fmt.Println("got", v)
}

Ex. 4, pairing in Go
[Ex. 5] Write a GoogleGo function that takes two channels $f$ and $q$ and tries to send the stream of Fibonacci numbers on $f$ but quits when it receives `true` on channel $q$. Write a `main` program to test the function by printing the first 10 Fibonacci numbers.
Ex. 5, Go Fibonacci

package main

import "fmt"

func fibonacci(f chan int, q chan bool) {

package main

import "fmt"

func fibonacci(f chan int, q chan bool) {
    x, y := 0, 1
    for {
        x, y = y, x+y
        select {
            case <-f:
            case <-q:
                return
        }
    }
}
package main

import "fmt"

func fibonacci(f chan int, q chan bool) {
    x, y := 0, 1
    for {
        select {
            case <-f:
                x, y = y, x+y
            case <-q:
                return
        }
    }
}
Ex. 5, Go Fibonacci

package main

import "fmt"

func fibonacci(f chan int, q chan bool) {
    x, y := 0, 1
    for {
        select {
            case f <- x:
                x, y = y, x+y
        }
    }
}
package main

import "fmt"

func fibonacci(f chan int, q chan bool) {
    x, y := 0, 1
    for {
        select {
            case f <- x:
                x, y = y, x+y
            case <-q:
                return
        }
    }
}
func main() {
    fib := make(chan int)
    quit := make(chan bool)

    fibonacci(fib, quit)
}

Ex. 5, Go Fibonacci
func main() {
    fib := make(chan int)
    quit := make(chan bool)
    go func(){

    }()
    fibonacci(fib, quit)
}
func main() {
    fib := make(chan int)
    quit := make(chan bool)
    go func(){
        for i := 0; i < 10; i++ {
            fmt.Println(<-fib)
        }
    }()
    fibonacci(fib, quit)
}
func main() {
    fib := make(chan int)
    quit := make(chan bool)
    go func() {
        for i := 0; i < 10; i++ {
            fmt.Println(<-fib)
        }
        quit <- true
    }()
    fibonacci(fib, quit)
}
π-calculus
[Ex. 6] The *asynchronous* \(\pi\)-calculus requires that outputs have no continuation:

\[
p ::= \text{nil} \mid \overline{x}(y) \mid x(y).p \mid \tau.p \mid [x = y]p \mid p + p \mid p|p \mid (x)p \mid !p
\]

Show that any process in the original \(\pi\)-calculus can be represented in the asynchronous \(\pi\)-calculus using an extra (fresh) channel to simulate explicit acknowledgement of name transmission.
Ex. 6, async pi

we want to encode ordinary processes in asynchronous ones

\[ A(\text{nil}) \triangleq \text{nil} \]
\[
A(\overline{xy}.p) \triangleq \overline{x}\langle y \rangle | A(p)
\]
\[ A(x(y).p) \triangleq x(y).A(p) \]
\[ A(\tau.p) \triangleq \tau.A(p) \]
\[ A([x = y]p) \triangleq [x = y]A(p) \]
\[ A(p + q) \triangleq A(p) + A(q) \]
\[ A(p|q) \triangleq A(p)|A(q) \]
\[ A((x)p) \triangleq (x)A(p) \]
\[ A(!p) \triangleq !A(p) \]
we want to encode ordinary processes in asynchronous ones

$$A(\overline{xy}.p) \triangleq \overline{x}\langle y\rangle | A(p)$$

$$A((x)(\overline{x}.x.p)) \triangleq (x)(\overline{x} | x.A(p))$$

problem: $A(p)$ can be executed before $\overline{xy}$ is received
Ex. 6, async pi

2nd attempt: wait for an ack

\[ \mathcal{A}(\overline{xy}.p) \triangleq \overline{x}<y> | a . \mathcal{A}(p) \]
\[ \mathcal{A}(x(z).q) \triangleq x(z) . (\overline{a} | \mathcal{A}(q)) \]

\[ \mathcal{A}((x, u)(\overline{x}|x|\overline{u}.p)) \triangleq (x, u)[\overline{x}a.nil | x.(\overline{a}|nil) | \overline{u}|a.A(p)) \]

\[ (x, u)(\text{nil|nil|\overline{u}.p) \quad (x, u)(\text{nil|a.nil | \overline{a} nil | \overline{u}|a.A(p)) \]

deadlock

\[ (x, u)(\text{nil|a.nil | nil|nil | \overline{u}|A(p)) \]

problem: possible interferences on channel \(a\)
Ex. 6, async pi

3rd attempt: wait for a private ack

\[ A(\overline{xy}.p) \triangleq (a)(\overline{x}(a) \parallel \overline{a}(y) \parallel a.A(p)) \quad a, x_a \text{ fresh} \]

\[ A(x(z).q) \triangleq x(x_a) . x_a(z) . (\overline{x_a} \parallel A(q)) \]

problem: possible self-interferences on channels \( a \).
Ex. 6, async pi

4th attempt: use 2 private channels

\[ A(\overline{xy}.p) \triangleq (a)(\overline{x}\langle a \rangle \mid a(x_k) \cdot (\overline{x_k}\langle y \rangle \mid A(p))) \]

\[ A(x(z).q) \triangleq x(x_a) \cdot (k)(\overline{x_a}\langle k \rangle \mid k(z).A(q)) \]

\[ a, k, x_a, x_k \text{ fresh} \]
Ex. 6, async pi

4th attempt: use 2 private channels

\[ \text{A}(\overline{xy}.p) \triangleq (a)(\overline{x}\langle a \rangle | a(x_k) . (\overline{x_k}\langle y \rangle | \text{A}(p))) \]

\[ \text{A}(x(z).q) \triangleq x(x_a) . (k)(\overline{x_a}\langle k \rangle | k(z) \text{A}(q)) \]

\[ a, k, x_a, x_k \text{ fresh} \]

problem: 3 communications involved, can we do better?
Ex. 6, async pi

5th attempt: let the receiver start

$$\mathcal{A}(\overline{xy}.p) \triangleq x(x_{a}) . (\overline{x_{a}}\langle y \rangle \mid \mathcal{A}(p))$$

$$\mathcal{A}(x(z).q) \triangleq (a)(\overline{x} \langle a \rangle \mid a(z).\mathcal{A}(q))$$

$$a, x_{a} \text{ fresh}$$
Ex. 6, async pi

5th attempt: let the receiver start

\[
A(\overline{xy}.p) \triangleq x(x_\alpha) . (\overline{x_\alpha} \langle y \rangle \mid A(p))
\]

\[
A(x(z).q) \triangleq (\alpha)(\overline{x} \langle a \rangle \mid a(z).A(q))
\]

\(\alpha, x_\alpha\) fresh
[Ex. 7] The polyadic $\pi$-calculus allows communicating more than one name in a single action, i.e., its action prefixes are of the form:

$$\pi ::= \tau \mid \overline{x}(z_1, \ldots z_n) \mid x(z_1, \ldots z_n)$$

The polyadic extension is useful especially when studying types for name passing processes. Show that the polyadic $\pi$-calculus can be encoded in the ordinary (monadic) $\pi$-calculus by passing the name of a private channel through which the multiple arguments are then passed in a sequence.
Ex. 7, polyadic pi

we want to encode **polyadic processes** in ordinary ones

\[
\begin{align*}
\mathcal{M}(\text{nil}) & \triangleq \text{nil} \\
\mathcal{M}(\overline{x}\langle y_1, \ldots, y_n \rangle.p) & \triangleq \overline{xy}_1 \cdots \overline{xy}_n.\mathcal{M}(p) \\
\mathcal{M}(x(z_1, \ldots, z_n).q) & \triangleq x(z_1) \cdots x(z_n).\mathcal{M}(q) \\
\mathcal{M}(\tau.p) & \triangleq \tau.\mathcal{M}(p) \\
\mathcal{M}([x = y]p) & \triangleq [x = y]\mathcal{M}(p) \\
\mathcal{M}(p + q) & \triangleq \mathcal{M}(p) + \mathcal{M}(q) \\
\mathcal{M}(p|q) & \triangleq \mathcal{M}(p)|\mathcal{M}(q) \\
\mathcal{M}((x)p) & \triangleq (x)\mathcal{M}(p) \\
\mathcal{M}(!p) & \triangleq !\mathcal{M}(p)
\end{align*}
\]
Ex. 7, polyadic pi

we want to encode polyadic processes in ordinary ones

\[
\begin{align*}
\mathcal{M}(\overline{x}(y_1, \ldots, y_n).p) & \triangleq \overline{x}y_1 \cdots \overline{x}y_n.\mathcal{M}(p) \\
\mathcal{M}(x(z_1, \ldots, z_n).q) & \triangleq x(z_1) \cdots x(z_n).\mathcal{M}(q)
\end{align*}
\]

\[
\mathcal{M}((x)(\overline{a}, b) \mid x(y_1, y_2).p \mid x(z_1, z_2).q)) \triangleq (x)(\overline{x}a, \overline{x}b \mid x(y_1).x(y_2).\mathcal{M}(p) \mid x(z_1).x(z_2).\mathcal{M}(q))
\]

\[
\begin{align*}
p' & = p[^a/y_1, ^b/y_2] \\
q' & = q[^a/z_1, ^b/z_2]
\end{align*}
\]

deadlock

\[
\begin{align*}
p'' & = \mathcal{M}(p)[^a/y_1] \\
q'' & = \mathcal{M}(q)[^b/z_1]
\end{align*}
\]

problem: interference between multiple senders / receivers
Ex. 7, polyadic πi

fix a private channel and send all data on it

\[
\begin{align*}
\mathcal{M}(\bar{x}\langle y_1, \ldots, y_n \rangle.p) & \triangleq (a)\bar{x}a.\bar{a}y_1. \cdots .\bar{a}y_n.\mathcal{M}(p) \\
\mathcal{M}(x(z_1, \ldots, z_n).q) & \triangleq x(x_a).x_a(z_1). \cdots .x_a(z_n).\mathcal{M}(q)
\end{align*}
\]

\[\alpha, x_\alpha \text{ fresh}\]