I (probably very likely) made a goof in MPL

Jared Pon

July 11, 2021

1 More tracing

Recall the code is as follows.

```
protocol IntTerminal => C =
    IntTerminalGet :: Get(Int | C) => C
    IntTerminalPut :: Put(Int | C) => C
    IntTerminalClose :: TopBot => C

protocol Mem(M|) => S =
    MemPut :: Put(M|S) => S
    MemGet :: Get(M|S) => S
    MemCls :: TopBot => S

protocol Passer(|P) => S =
    Passer :: P (+) (Neg(P) (*) S) => S

proc memory :: A | Mem(A|) => =
    x | ch => -> do
        hcase ch of
            MemPut -> do
                get y on ch
                memory(y | ch => )
            MemGet -> do
                put x on ch
                memory(x | ch => )
            MemCls -> do
                halt ch

proc p1 :: | => Passer(|Mem(Int|)), IntTerminal =
    | => passer, inp -> do
        hput Passer on passer
        split passer into mm,nmpp
        hput MemGet on mm
```

```
        get y on mm
        hput IntTerminalPut on inp
        put y on inp
        hput IntTerminalGet on inp
        get x on inp
        hput MemPut on mm
        put x on mm
        fork nmpp as
            nm with mm -> mm |=| neg nm
            pp with inp -> p1(| => pp, inp)

proc p2 :: | Passer(| Mem(Int|)) => IntTerminal, Mem(Int|) =
    | passer => inp, mem -> do
        hcase passer of
            Passer -> do
                hput MemGet on mem
                get y on mem
                hput IntTerminalPut on inp
                put y on inp
                hput IntTerminalGet on inp
                get x on inp
                hput MemPut on mem
                put x on mem
                fork passer as
                    mm with mem -> do
                        mm |=| mem
                    nmpp with inp -> do
                        split nmpp into nm,pp
                        plug
                            p2( | pp => inp,z)
                            z,nm => -> z |=| neg nm

proc run :: | => IntTerminal , IntTerminal =
    | => _inpterm0, _inpterm1 -> do
        plug
            p1(| => passer, _inpterm0)
            p2(| passer => _inpterm1, mem)
            memory(100 | mem => )
```

First, we execute `run` and open up the service channels. So, the entire system state is:

```
-- Channel manager
( _inpterm0,
    []
    |
    []
```

```
)

( _inpterm1,
    []
    |
    []
)

-- Global translation
_inpterm0 ---> _inpterm0
_inpterm1 ---> _inpterm1

-- Running processes:
proc run :: | => IntTerminal , IntTerminal =
    | => _inpterm0, _inpterm1 -> do
        plug
            p1(| => passer, _inpterm0)
            p2(| passer => _inpterm1, mem)
            memory(100 | mem => )
```

Then, we open up the channels in the "plug" command in `run`

```
-- Channel manager
( _inpterm0,
    []
    |
    []
)

( _inpterm1,
    []
    |
    []
)
( passer,
    []
    |
    []
)

( mem,
    []
    |
    []
)

-- Global translation
```

```
_inpterm0 ---> _inpterm0
_inpterm1 ---> _inpterm1
passer    ---> passer
mem       ---> mem

-- Running processes:
proc run :: | => IntTerminal , IntTerminal =
    | => _inpterm0, _inpterm1 -> do
        plug
            p1(| => passer, _inpterm0)
            p2(| passer => _inpterm1, mem)
            memory(100 | mem => )
```

and we then may execute `p1,p2,memory` as given by the "plug" command (note that the `run` process may terminate now), so we put these all in the running processes (and substitute their arguments as appropriate).

```
-- Channel manager
( _inpterm0,
    []
    |
    []
)

( _inpterm1,
    []
    |
    []
)

( passer,
    []
    |
    []
)

( mem,
    []
    |
    []
)

-- Global translation
_inpterm0 ---> _inpterm0
_inpterm1 ---> _inpterm1
passer    ---> passer
mem       ---> mem
```

```
-- Running processes:
proc memory :: A | Mem(A|) => =
    100 | mem => -> do
        hcase mem of
            MemPut -> do
                get y on mem
                memory(y | mem => )
            MemGet -> do
                put 100 on mem
                memory(100 | mem => )
            MemCls -> do
                halt mem

proc p1 :: | => Passer(|Mem(Int|)), IntTerminal =
    | => passer, _inpterm0 -> do
        hput Passer on passer
        split passer into mm,nmpp
        hput MemGet on mm
        get y on mm
        hput IntTerminalPut on _inpterm0
        put y on _inpterm0
        hput IntTerminalGet on _inpterm0
        get x on _inpterm0
        hput MemPut on mm
        put x on mm
        fork nmpp as
            nm with mm -> mm |=| neg nm
            pp with _inpterm0 -> p1(| => pp, _inpterm0)

proc p2 :: | Passer(| Mem(Int|)) => IntTerminal, Mem(Int|) =
    | passer => _inpterm1, mem -> do
        hcase passer of
            Passer -> do
                hput MemGet on mem
                get y on mem
                hput IntTerminalPut on _inpterm1
                put y on _inpterm1
                hput IntTerminalGet on _inpterm1
                get x on _inpterm1
                hput MemPut on mem
                put x on mem
                fork passer as
                    mm with mem -> do
                        mm |=| mem
                    nmpp with _inpterm1 -> do
```

```
                    split nmpp into nm,pp
                    plug
                        p2( | pp => _inpterm1,z)
                        z,nm => -> z |=| neg nm
```

Now, we execute `memory`. Indeed, we see that it simply will suspend itself on the input polarity queue of `mem ---> mem`.

```
-- Channel manager
( _inpterm0,
    []
    |
    []
)

( _inpterm1,
    []
    |
    []
)
( passer,
    []
    |
    []
)

( mem,
    []
    |
    proc memory :: A | Mem(A|) => =
        100 | mem => -> do
            hcase mem of
                MemPut -> do
                    get y on mem
                    memory(y | mem => )
                MemGet -> do
                    put 100 on mem
                    memory(100 | mem => )
                MemCls -> do
                    halt mem
)

-- Global translation
_inpterm0 ---> _inpterm0
_inpterm1 ---> _inpterm1
passer    ---> passer
```

```
mem       ---> mem

-- Running processes:

proc p1 :: | => Passer(|Mem(Int|)), IntTerminal =
    | => passer, _inpterm0 -> do
        hput Passer on passer
        split passer into mm,nmpp
        hput MemGet on mm
        get y on mm
        hput IntTerminalPut on _inpterm0
        put y on _inpterm0
        hput IntTerminalGet on _inpterm0
        get x on _inpterm0
        hput MemPut on mm
        put x on mm
        fork nmpp as
            nm with mm -> mm |=| neg nm
            pp with _inpterm0 -> p1(| => pp, _inpterm0)

proc p2 :: | Passer(| Mem(Int|)) => IntTerminal, Mem(Int|) =
    | passer => _inpterm1, mem -> do
        hcase passer of
            Passer -> do
                hput MemGet on mem
                get y on mem
                hput IntTerminalPut on _inpterm1
                put y on _inpterm1
                hput IntTerminalGet on _inpterm1
                get x on _inpterm1
                hput MemPut on mem
                put x on mem
                fork passer as
                    mm with mem -> do
                        mm |=| mem
                    nmpp with _inpterm1 -> do
                        split nmpp into nm,pp
                        plug
                            p2( | pp => _inpterm1,z)
                            z,nm => -> z |=| neg nm
```

Now, we execute a step for `p1`.

```
-- Channel manager
( _inpterm0,
    []
```

```
    |
    []
)

( _inpterm1,
    []
    |
    []
)
( passer,
    proc p1 :: | => Passer(|Mem(Int|)), IntTerminal =
        | => passer, _inpterm0 -> do
            hput Passer on passer
    |
    []
)

( mem,
    []
    |
    proc memory :: A | Mem(A|) => =
        100 | mem => -> do
            hcase mem of
                MemPut -> do
                    get y on mem
                    memory(y | mem => )
                MemGet -> do
                    put 100 on mem
                    memory(100 | mem => )
                MemCls -> do
                    halt mem
)

-- Global translation
_inpterm0 ---> _inpterm0
_inpterm1 ---> _inpterm1
passer    ---> passer
mem       ---> mem

-- Running processes:

proc p1 :: | => Passer(|Mem(Int|)), IntTerminal =
    | => passer, _inpterm0 -> do
        split passer into mm,nmpp
        hput MemGet on mm
        get y on mm
```

```
        hput IntTerminalPut on _inpterm0
        put y on _inpterm0
        hput IntTerminalGet on _inpterm0
        get x on _inpterm0
        hput MemPut on mm
        put x on mm
        fork nmpp as
            nm with mm -> mm |=| neg nm
            pp with _inpterm0 -> p1(| => pp, _inpterm0)

proc p2 :: | Passer(| Mem(Int|)) => IntTerminal, Mem(Int|) =
    | passer => _inpterm1, mem -> do
        hcase passer of
            Passer -> do
                hput MemGet on mem
                get y on mem
                hput IntTerminalPut on _inpterm1
                put y on _inpterm1
                hput IntTerminalGet on _inpterm1
                get x on _inpterm1
                hput MemPut on mem
                put x on mem
                fork passer as
                    mm with mem -> do
                        mm |=| mem
                    nmpp with _inpterm1 -> do
                        split nmpp into nm,pp
                        plug
                            p2( | pp => _inpterm1,z)
                            z,nm => -> z |=| neg nm
```

Another step for `p1` (note that "split" opens up two new channels), and `mm` and `nmpp` are both of output polarity (they inherit the polarity from the "split" command)

```
-- Channel manager
( _inpterm0,
    []
    |
    []
)

( _inpterm1,
    []
    |
    []
```

```
)
( passer,
    proc p1 :: | => Passer(|Mem(Int|)), IntTerminal =
        | => passer, _inpterm0 -> do
            split passer into mm,nmpp
    ;
    proc p1 :: | => Passer(|Mem(Int|)), IntTerminal =
        | => passer, _inpterm0 -> do
            hput Passer on passer
    |
    []
)

( mem,
    []
    |
    proc memory :: A | Mem(A|) => =
        100 | mem => -> do
            hcase mem of
                MemPut -> do
                    get y on mem
                    memory(y | mem => )
                MemGet -> do
                    put 100 on mem
                    memory(100 | mem => )
                MemCls -> do
                    halt mem
)

( mm
    []
    |
    []
)

( nmpp
    []
    |
    []
)

-- Global translation
_inpterm0 ---> _inpterm0
_inpterm1 ---> _inpterm1
passer    ---> passer
mem       ---> mem
```

```
mm        ---> mm
nmpp      ---> nmpp

-- Running processes:

proc p1 :: | => Passer(|Mem(Int|)), IntTerminal =
    | => passer, _inpterm0, mm, nmpp -> do
        hput MemGet on mm
        get y on mm
        hput IntTerminalPut on _inpterm0
        put y on _inpterm0
        hput IntTerminalGet on _inpterm0
        get x on _inpterm0
        hput MemPut on mm
        put x on mm
        fork nmpp as
            nm with mm -> mm |=| neg nm
            pp with _inpterm0 -> p1(| => pp, _inpterm0)

proc p2 :: | Passer(| Mem(Int|)) => IntTerminal, Mem(Int|) =
    | passer => _inpterm1, mem -> do
        hcase passer of
            Passer -> do
                hput MemGet on mem
                get y on mem
                hput IntTerminalPut on _inpterm1
                put y on _inpterm1
                hput IntTerminalGet on _inpterm1
                get x on _inpterm1
                hput MemPut on mem
                put x on mem
                fork passer as
                    mm with mem -> do
                        mm |=| mem
                    nmpp with _inpterm1 -> do
                        split nmpp into nm,pp
                        plug
                            p2( | pp => _inpterm1,z)
                            z,nm => -> z |=| neg nm
```

Another step for `p1`.

```
-- Channel manager
( _inpterm0,
    []
    |
```

```
    []
)

( _inpterm1,
    []
    |
    []
)
( passer,
    proc p1 :: | => Passer(|Mem(Int|)), IntTerminal =
        | => passer, _inpterm0 -> do
            split passer into mm,nmpp
    ;
    proc p1 :: | => Passer(|Mem(Int|)), IntTerminal =
        | => passer, _inpterm0 -> do
            hput Passer on passer
    |
    []
)

( mem,
    []
    |
    proc memory :: A | Mem(A|) => =
        100 | mem => -> do
            hcase mem of
                MemPut -> do
                    get y on mem
                    memory(y | mem => )
                MemGet -> do
                    put 100 on mem
                    memory(100 | mem => )
                MemCls -> do
                    halt mem
)

( mm
    proc p1 :: | => Passer(|Mem(Int|)), IntTerminal =
        | => passer, _inpterm0, mm, nmpp -> do
            hput MemGet on mm
    |
    []
)

( nmpp
    []
```

```
    |
    []
)

-- Global translation
_inpterm0 ---> _inpterm0
_inpterm1 ---> _inpterm1
passer    ---> passer
mem       ---> mem
mm        ---> mm
nmpp      ---> nmpp

-- Running processes:

proc p1 :: | => Passer(|Mem(Int|)), IntTerminal =
    | => passer, _inpterm0, mm, nmpp -> do
        get y on mm
        hput IntTerminalPut on _inpterm0
        put y on _inpterm0
        hput IntTerminalGet on _inpterm0
        get x on _inpterm0
        hput MemPut on mm
        put x on mm
        fork nmpp as
            nm with mm -> mm |=| neg nm
            pp with _inpterm0 -> p1(| => pp, _inpterm0)

proc p2 :: | Passer(| Mem(Int|)) => IntTerminal, Mem(Int|) =
    | passer => _inpterm1, mem -> do
        hcase passer of
            Passer -> do
                hput MemGet on mem
                get y on mem
                hput IntTerminalPut on _inpterm1
                put y on _inpterm1
                hput IntTerminalGet on _inpterm1
                get x on _inpterm1
                hput MemPut on mem
                put x on mem
                fork passer as
                    mm with mem -> do
                        mm |=| mem
                    nmpp with _inpterm1 -> do
                        split nmpp into nm,pp
                        plug
                            p2( | pp => _inpterm1,z)
```

```
                        z,nm => -> z |=| neg nm
```

Another step for `p1` (note that the "get" command suspends `p1`).

```
-- Channel manager
( _inpterm0,
    []
    |
    []
)

( _inpterm1,
    []
    |
    []
)
( passer,
    proc p1 :: | => Passer(|Mem(Int|)), IntTerminal =
        | => passer, _inpterm0 -> do
            split passer into mm,nmpp
    ;
    proc p1 :: | => Passer(|Mem(Int|)), IntTerminal =
        | => passer, _inpterm0 -> do
            hput Passer on passer
    |
    []
)

( mem,
    []
    |
    proc memory :: A | Mem(A|) => =
        100 | mem => -> do
            hcase mem of
                MemPut -> do
                    get y on mem
                    memory(y | mem => )
                MemGet -> do
                    put 100 on mem
                    memory(100 | mem => )
                MemCls -> do
                    halt mem
)

( mm
    proc p1 :: | => Passer(|Mem(Int|)), IntTerminal =
```

```
        | => passer, _inpterm0, mm, nmpp -> do
            get y on mm
            hput IntTerminalPut on _inpterm0
            put y on _inpterm0
            hput IntTerminalGet on _inpterm0
            get x on _inpterm0
            hput MemPut on mm
            put x on mm
            fork nmpp as
                nm with mm -> mm |=| neg nm
                pp with _inpterm0 -> p1(| => pp, _inpterm0)
    ;
    proc p1 :: | => Passer(|Mem(Int|)), IntTerminal =
        | => passer, _inpterm0, mm, nmpp -> do
            hput MemGet on mm
    |
    []
)

( nmpp
    []
    |
    []
)

-- Global translation
_inpterm0 ---> _inpterm0
_inpterm1 ---> _inpterm1
passer    ---> passer
mem       ---> mem
mm        ---> mm
nmpp      ---> nmpp

-- Running processes:
proc p2 :: | Passer(| Mem(Int|)) => IntTerminal, Mem(Int|) =
    | passer => _inpterm1, mem -> do
        hcase passer of
            Passer -> do
                hput MemGet on mem
                get y on mem
                hput IntTerminalPut on _inpterm1
                put y on _inpterm1
                hput IntTerminalGet on _inpterm1
                get x on _inpterm1
                hput MemPut on mem
                put x on mem
```

```
                fork passer as
                    mm with mem -> do
                        mm |=| mem
                    nmpp with _inpterm1 -> do
                        split nmpp into nm,pp
                        plug
                            p2( | pp => _inpterm1,z)
                            z,nm => -> z |=| neg nm
```

Now, we can only execute a step for p2, so let's do that.

```
-- Channel manager
( _inpterm0,
    []
    |
    []
)

( _inpterm1,
    []
    |
    []
)
( passer,
    proc p1 :: | => Passer(|Mem(Int|)), IntTerminal =
        | => passer, _inpterm0 -> do
            split passer into mm,nmpp
    ;
    proc p1 :: | => Passer(|Mem(Int|)), IntTerminal =
        | => passer, _inpterm0 -> do
            hput Passer on passer
    |
    proc p2 :: | Passer(| Mem(Int|)) => IntTerminal, Mem(Int|) =
        | passer => _inpterm1, mem -> do
            hcase passer of
                Passer -> do
                    hput MemGet on mem
                    get y on mem
                    hput IntTerminalPut on _inpterm1
                    put y on _inpterm1
                    hput IntTerminalGet on _inpterm1
                    get x on _inpterm1
                    hput MemPut on mem
                    put x on mem
                    fork passer as
                        mm with mem -> do
```

```
                        mm |=| mem
                    nmpp with _inpterm1 -> do
                        split nmpp into nm,pp
                        plug
                            p2( | pp => _inpterm1,z)
                            z,nm => -> z |=| neg nm
    )

( mem,
    []
    |
    proc memory :: A | Mem(A|) => =
        100 | mem => -> do
            hcase mem of
                MemPut -> do
                    get y on mem
                    memory(y | mem => )
                MemGet -> do
                    put 100 on mem
                    memory(100 | mem => )
                MemCls -> do
                    halt mem
)

( mm
    proc p1 :: | => Passer(|Mem(Int|)), IntTerminal =
        | => passer, _inpterm0, mm, nmpp -> do
            get y on mm
            hput IntTerminalPut on _inpterm0
            put y on _inpterm0
            hput IntTerminalGet on _inpterm0
            get x on _inpterm0
            hput MemPut on mm
            put x on mm
            fork nmpp as
                nm with mm -> mm |=| neg nm
                pp with _inpterm0 -> p1(| => pp, _inpterm0)
    ;
    proc p1 :: | => Passer(|Mem(Int|)), IntTerminal =
        | => passer, _inpterm0, mm, nmpp -> do
            hput MemGet on mm
    |
    []
)

( nmpp
```

```
    []
    |
    []
)

-- Global translation
_inpterm0 ---> _inpterm0
_inpterm1 ---> _inpterm1
passer    ---> passer
mem       ---> mem
mm        ---> mm
nmpp      ---> nmpp

-- Running processes:
```

Now, we can execute a channel manager step on `passer`, and note that `p2` may resume execution now.

```
-- Channel manager
( _inpterm0,
    []
    |
    []
)

( _inpterm1,
    []
    |
    []
)
( passer,
    proc p1 :: | => Passer(|Mem(Int|)), IntTerminal =
        | => passer, _inpterm0 -> do
            split passer into mm,nmpp
    |
    []
    )

( mem,
    []
    |
    proc memory :: A | Mem(A|) => =
        100 | mem => -> do
            hcase mem of
                MemPut -> do
                    get y on mem
                    memory(y | mem => )
```

```
                MemGet -> do
                    put 100 on mem
                    memory(100 | mem => )
                MemCls -> do
                    halt mem
)

( mm
    proc p1 :: | => Passer(|Mem(Int|)), IntTerminal =
        | => passer, _inpterm0, mm, nmpp -> do
            get y on mm
            hput IntTerminalPut on _inpterm0
            put y on _inpterm0
            hput IntTerminalGet on _inpterm0
            get x on _inpterm0
            hput MemPut on mm
            put x on mm
            fork nmpp as
                nm with mm -> mm |=| neg nm
                pp with _inpterm0 -> p1(| => pp, _inpterm0)
    ;
    proc p1 :: | => Passer(|Mem(Int|)), IntTerminal =
        | => passer, _inpterm0, mm, nmpp -> do
            hput MemGet on mm
    |
    []
)

( nmpp
    []
    |
    []
)

-- Global translation
_inpterm0 ---> _inpterm0
_inpterm1 ---> _inpterm1
passer    ---> passer
mem       ---> mem
mm        ---> mm
nmpp      ---> nmpp

-- Running processes:
proc p2 :: | Passer(| Mem(Int|)) => IntTerminal, Mem(Int|) =
    | passer => _inpterm1, mem -> do
        hput MemGet on mem
```

```
        get y on mem
        hput IntTerminalPut on _inpterm1
        put y on _inpterm1
        hput IntTerminalGet on _inpterm1
        get x on _inpterm1
        hput MemPut on mem
        put x on mem
        fork passer as
            mm with mem -> do
                mm |=| mem
            nmpp with _inpterm1 -> do
                split nmpp into nm,pp
                plug
                    p2( | pp => _inpterm1,z)
                    z,nm => -> z |=| neg nm
```

A step for p2.

```
-- Channel manager
( _inpterm0,
    []
    |
    []
)

( _inpterm1,
    []
    |
    []
)
( passer,
    proc p1 :: | => Passer(|Mem(Int|)), IntTerminal =
        | => passer, _inpterm0 -> do
            split passer into mm,nmpp
    |
    []
    )

( mem,
    proc p2 :: | Passer(| Mem(Int|)) => IntTerminal, Mem(Int|) =
        | passer => _inpterm1, mem -> do
            hput MemGet on mem
    |
    proc memory :: A | Mem(A|) => =
        100 | mem => -> do
            hcase mem of
                MemPut -> do
```

```
                    get y on mem
                    memory(y | mem => )
                MemGet -> do
                    put 100 on mem
                    memory(100 | mem => )
                MemCls -> do
                    halt mem
)

( mm
    proc p1 :: | => Passer(|Mem(Int|)), IntTerminal =
        | => passer, _inpterm0, mm, nmpp -> do
            get y on mm
            hput IntTerminalPut on _inpterm0
            put y on _inpterm0
            hput IntTerminalGet on _inpterm0
            get x on _inpterm0
            hput MemPut on mm
            put x on mm
            fork nmpp as
                nm with mm -> mm |=| neg nm
                pp with _inpterm0 -> p1(| => pp, _inpterm0)
    ;
    proc p1 :: | => Passer(|Mem(Int|)), IntTerminal =
        | => passer, _inpterm0, mm, nmpp -> do
            hput MemGet on mm
    |
    []
)

( nmpp
    []
    |
    []
)

-- Global translation
_inpterm0 ---> _inpterm0
_inpterm1 ---> _inpterm1
passer    ---> passer
mem       ---> mem
mm        ---> mm
nmpp      ---> nmpp

-- Running processes:
proc p2 :: | Passer(| Mem(Int|)) => IntTerminal, Mem(Int|) =
```

```
    | passer => _inpterm1, mem -> do
        get y on mem
        hput IntTerminalPut on _inpterm1
        put y on _inpterm1
        hput IntTerminalGet on _inpterm1
        get x on _inpterm1
        hput MemPut on mem
        put x on mem
        fork passer as
            mm with mem -> do
                mm |=| mem
            nmpp with _inpterm1 -> do
                split nmpp into nm,pp
                plug
                    p2( | pp => _inpterm1,z)
                    z,nm => -> z |=| neg nm
```

Now, on the channel `mem`, we may execute a step there and resume execution of `memory` while sleeping `p2`.

```
-- Channel manager
( _inpterm0,
    []
    |
    []
)

( _inpterm1,
    []
    |
    []
)
( passer,
    proc p1 :: | => Passer(|Mem(Int|)), IntTerminal =
        | => passer, _inpterm0 -> do
            split passer into mm,nmpp
    |
    []
    )

( mem,
    []
    |
    []
)

( mm
```

```
    proc p1 :: | => Passer(|Mem(Int|)), IntTerminal =
        | => passer, _inpterm0, mm, nmpp -> do
            get y on mm
            hput IntTerminalPut on _inpterm0
            put y on _inpterm0
            hput IntTerminalGet on _inpterm0
            get x on _inpterm0
            hput MemPut on mm
            put x on mm
            fork nmpp as
                nm with mm -> mm |=| neg nm
                pp with _inpterm0 -> p1(| => pp, _inpterm0)
    ;
    proc p1 :: | => Passer(|Mem(Int|)), IntTerminal =
        | => passer, _inpterm0, mm, nmpp -> do
            hput MemGet on mm
    |
    []
)

( nmpp
    []
    |
    []
)

-- Global translation
_inpterm0 ---> _inpterm0
_inpterm1 ---> _inpterm1
passer    ---> passer
mem       ---> mem
mm        ---> mm
nmpp      ---> nmpp

-- Running processes:
proc p2 :: | Passer(| Mem(Int|)) => IntTerminal, Mem(Int|) =
    | passer => _inpterm1, mem -> do
        get y on mem
        hput IntTerminalPut on _inpterm1
        put y on _inpterm1
        hput IntTerminalGet on _inpterm1
        get x on _inpterm1
        hput MemPut on mem
        put x on mem
        fork passer as
            mm with mem -> do
```

```
            mm |=| mem
        nmpp with _inpterm1 -> do
            split nmpp into nm,pp
            plug
                p2( | pp => _inpterm1,z)
                z,nm => -> z |=| neg nm

proc memory :: A | Mem(A|) => =
    100 | mem => -> do
        put 100 on mem
        memory(100 | mem => )
```

Another step for `memory`

```
-- Channel manager
( _inpterm0,
    []
    |
    []
)

( _inpterm1,
    []
    |
    []
)
( passer,
    proc p1 :: | => Passer(|Mem(Int|)), IntTerminal =
        | => passer, _inpterm0 -> do
            split passer into mm,nmpp
    |
    []
    )

( mem,
    []
    |
    proc memory :: A | Mem(A|) => =
        100 | mem => -> do
            put 100 on mem
)

( mm
    proc p1 :: | => Passer(|Mem(Int|)), IntTerminal =
        | => passer, _inpterm0, mm, nmpp -> do
            get y on mm
            hput IntTerminalPut on _inpterm0
```

```
            put y on _inpterm0
            hput IntTerminalGet on _inpterm0
            get x on _inpterm0
            hput MemPut on mm
            put x on mm
            fork nmpp as
                nm with mm -> mm |=| neg nm
                pp with _inpterm0 -> p1(| => pp, _inpterm0)
    ;
    proc p1 :: | => Passer(|Mem(Int|)), IntTerminal =
        | => passer, _inpterm0, mm, nmpp -> do
            hput MemGet on mm
    |
    []
)

( nmpp
    []
    |
    []
)

-- Global translation
_inpterm0 ---> _inpterm0
_inpterm1 ---> _inpterm1
passer    ---> passer
mem       ---> mem
mm        ---> mm
nmpp      ---> nmpp

-- Running processes:
proc p2 :: | Passer(| Mem(Int|)) => IntTerminal, Mem(Int|) =
    | passer => _inpterm1, mem -> do
        get y on mem
        hput IntTerminalPut on _inpterm1
        put y on _inpterm1
        hput IntTerminalGet on _inpterm1
        get x on _inpterm1
        hput MemPut on mem
        put x on mem
        fork passer as
            mm with mem -> do
                mm |=| mem
            nmpp with _inpterm1 -> do
                split nmpp into nm,pp
                plug
```

```
                    p2( | pp => _inpterm1,z)
                    z,nm => -> z |=| neg nm

proc memory :: A | Mem(A|) => =
    100 | mem => -> do
        memory(100 | mem => )
```

Another step for `memory` which is simply a recursive call (so we substitute arguments as necessary).

```
-- Channel manager
( _inpterm0,
    []
    |
    []
)

( _inpterm1,
    []
    |
    []
)
( passer,
    proc p1 :: | => Passer(|Mem(Int|)), IntTerminal =
        | => passer, _inpterm0 -> do
            split passer into mm,nmpp
    |
    []
    )

( mem,
    []
    |
    proc memory :: A | Mem(A|) => =
        100 | mem => -> do
            put 100 on mem
)

( mm
    proc p1 :: | => Passer(|Mem(Int|)), IntTerminal =
        | => passer, _inpterm0, mm, nmpp -> do
            get y on mm
            hput IntTerminalPut on _inpterm0
            put y on _inpterm0
            hput IntTerminalGet on _inpterm0
            get x on _inpterm0
            hput MemPut on mm
```

```
            put x on mm
            fork nmpp as
                nm with mm -> mm |=| neg nm
                pp with _inpterm0 -> p1(| => pp, _inpterm0)
    ;
    proc p1 :: | => Passer(|Mem(Int|)), IntTerminal =
        | => passer, _inpterm0, mm, nmpp -> do
            hput MemGet on mm
    |
    []
)

( nmpp
    []
    |
    []
)

-- Global translation
_inpterm0 ---> _inpterm0
_inpterm1 ---> _inpterm1
passer    ---> passer
mem       ---> mem
mm        ---> mm
nmpp      ---> nmpp

-- Running processes:
proc p2 :: | Passer(| Mem(Int|)) => IntTerminal, Mem(Int|) =
    | passer => _inpterm1, mem -> do
        get y on mem
        hput IntTerminalPut on _inpterm1
        put y on _inpterm1
        hput IntTerminalGet on _inpterm1
        get x on _inpterm1
        hput MemPut on mem
        put x on mem
        fork passer as
            mm with mem -> do
                mm |=| mem
            nmpp with _inpterm1 -> do
                split nmpp into nm,pp
                plug
                    p2( | pp => _inpterm1,z)
                    z,nm => -> z |=| neg nm

proc memory :: A | Mem(A|) => =
```

```
    100 | mem => -> do
        hcase mem of
            MemPut -> do
                get y on mem
                memory(y | mem => )
            MemGet -> do
                put 100 on mem
                memory(100 | mem => )
            MemCls -> do
                halt mem
```

Another step for `memory` where we see that `memory` suspends itself on the input polarity queue of the channel `mem`.

```
-- Channel manager
( _inpterm0,
    []
    |
    []
)

( _inpterm1,
    []
    |
    []
)
( passer,
    proc p1 :: | => Passer(|Mem(Int|)), IntTerminal =
        | => passer, _inpterm0 -> do
            split passer into mm,nmpp
    |
    []
    )

( mem,
    []
    |
    proc memory :: A | Mem(A|) => =
        100 | mem => -> do
            put 100 on mem
    ;
    proc memory :: A | Mem(A|) => =
        100 | mem => -> do
            hcase mem of
                MemPut -> do
                    get y on mem
```

```
                    memory(y | mem => )
                MemGet -> do
                    put 100 on mem
                    memory(100 | mem => )
                MemCls -> do
                    halt mem
)

( mm
    proc p1 :: | => Passer(|Mem(Int|)), IntTerminal =
        | => passer, _inpterm0, mm, nmpp -> do
            get y on mm
            hput IntTerminalPut on _inpterm0
            put y on _inpterm0
            hput IntTerminalGet on _inpterm0
            get x on _inpterm0
            hput MemPut on mm
            put x on mm
            fork nmpp as
                nm with mm -> mm |=| neg nm
                pp with _inpterm0 -> p1(| => pp, _inpterm0)
    ;
    proc p1 :: | => Passer(|Mem(Int|)), IntTerminal =
        | => passer, _inpterm0, mm, nmpp -> do
            hput MemGet on mm
    |
    []
)

( nmpp
    []
    |
    []
)

-- Global translation
_inpterm0 ---> _inpterm0
_inpterm1 ---> _inpterm1
passer    ---> passer
mem       ---> mem
mm        ---> mm
nmpp      ---> nmpp

-- Running processes:
proc p2 :: | Passer(| Mem(Int|)) => IntTerminal, Mem(Int|) =
    | passer => _inpterm1, mem -> do
```

```
        get y on mem
        hput IntTerminalPut on _inpterm1
        put y on _inpterm1
        hput IntTerminalGet on _inpterm1
        get x on _inpterm1
        hput MemPut on mem
        put x on mem
        fork passer as
            mm with mem -> do
                mm |=| mem
            nmpp with _inpterm1 -> do
                split nmpp into nm,pp
                plug
                    p2( | pp => _inpterm1,z)
                    z,nm => -> z |=| neg nm
```

A step for p2 (no other choice), where we see that it suspends itself on the output polarity of channel mem.

```
-- Channel manager
( _inpterm0,
    []
    |
    []
)

( _inpterm1,
    []
    |
    []
)
( passer,
    proc p1 :: | => Passer(|Mem(Int|)), IntTerminal =
        | => passer, _inpterm0 -> do
            split passer into mm,nmpp
    |
    []
    )

( mem,
    []
    |
    proc memory :: A | Mem(A|) => =
        100 | mem => -> do
            hcase mem of
                MemPut -> do
```

```
                    get y on mem
                    memory(y | mem => )
                MemGet -> do
                    put 100 on mem
                    memory(100 | mem => )
                MemCls -> do
                    halt mem
)

( mm
    proc p1 :: | => Passer(|Mem(Int|)), IntTerminal =
        | => passer, _inpterm0, mm, nmpp -> do
            get y on mm
            hput IntTerminalPut on _inpterm0
            put y on _inpterm0
            hput IntTerminalGet on _inpterm0
            get x on _inpterm0
            hput MemPut on mm
            put x on mm
            fork nmpp as
                nm with mm -> mm |=| neg nm
                pp with _inpterm0 -> p1(| => pp, _inpterm0)
    ;
    proc p1 :: | => Passer(|Mem(Int|)), IntTerminal =
        | => passer, _inpterm0, mm, nmpp -> do
            hput MemGet on mm
    |
    []
)

( nmpp
    []
    |
    []
)

-- Global translation
_inpterm0 ---> _inpterm0
_inpterm1 ---> _inpterm1
passer    ---> passer
mem       ---> mem
mm        ---> mm
nmpp      ---> nmpp

-- Running processes:
```

Execute a step for the channel `mem`, and substitute arguments appropriately. Note that `p2` may resume.

```
-- Channel manager
( _inpterm0,
    []
    |
    []
)

( _inpterm1,
    []
    |
    []
)
( passer,
    proc p1 :: | => Passer(|Mem(Int|)), IntTerminal =
        | => passer, _inpterm0 -> do
            split passer into mm,nmpp
    |
    []
    )

( mem,
    []
    |
    proc memory :: A | Mem(A|) => =
        100 | mem => -> do
            hcase mem of
                MemPut -> do
                    get y on mem
                    memory(y | mem => )
                MemGet -> do
                    put 100 on mem
                    memory(100 | mem => )
                MemCls -> do
                    halt mem
)

( mm
    proc p1 :: | => Passer(|Mem(Int|)), IntTerminal =
        | => passer, _inpterm0, mm, nmpp -> do
            get y on mm
            hput IntTerminalPut on _inpterm0
            put y on _inpterm0
            hput IntTerminalGet on _inpterm0
```

```
            get x on _inpterm0
            hput MemPut on mm
            put x on mm
            fork nmpp as
                nm with mm -> mm |=| neg nm
                pp with _inpterm0 -> p1(| => pp, _inpterm0)
    ;
    proc p1 :: | => Passer(|Mem(Int|)), IntTerminal =
        | => passer, _inpterm0, mm, nmpp -> do
            hput MemGet on mm
    |
    []
)

( nmpp
    []
    |
    []
)

-- Global translation
_inpterm0 ---> _inpterm0
_inpterm1 ---> _inpterm1
passer    ---> passer
mem       ---> mem
mm        ---> mm
nmpp      ---> nmpp

-- Running processes:
proc p2 :: | Passer(| Mem(Int|)) => IntTerminal, Mem(Int|) =
    | passer => _inpterm1, mem -> do
        hput IntTerminalPut on _inpterm1
        put 100 on _inpterm1
        hput IntTerminalGet on _inpterm1
        get x on _inpterm1
        hput MemPut on mem
        put x on mem
        fork passer as
            mm with mem -> do
                mm |=| mem
            nmpp with _inpterm1 -> do
                split nmpp into nm,pp
                plug
                    p2( | pp => _inpterm1,z)
                    z,nm => -> z |=| neg nm
```

2 Boring service channel steps for p2|

A step for p2 (boring service channel step).

```
-- Channel manager
( _inpterm0,
    []
    |
    []
)

( _inpterm1,
    proc p2 :: | Passer(| Mem(Int|)) => IntTerminal, Mem(Int|) =
        | passer => _inpterm1, mem -> do
            hput IntTerminalPut on _inpterm1
    |
    []
)
( passer,
    proc p1 :: | => Passer(|Mem(Int|)), IntTerminal =
        | => passer, _inpterm0 -> do
            split passer into mm,nmpp
    |
    []
    )

( mem,
    []
    |
    proc memory :: A | Mem(A|) => =
        100 | mem => -> do
            hcase mem of
                MemPut -> do
                    get y on mem
                    memory(y | mem => )
                MemGet -> do
                    put 100 on mem
                    memory(100 | mem => )
                MemCls -> do
                    halt mem
)

( mm
    proc p1 :: | => Passer(|Mem(Int|)), IntTerminal =
        | => passer, _inpterm0, mm, nmpp -> do
            get y on mm
```

```
            hput IntTerminalPut on _inpterm0
            put y on _inpterm0
            hput IntTerminalGet on _inpterm0
            get x on _inpterm0
            hput MemPut on mm
            put x on mm
            fork nmpp as
                nm with mm -> mm |=| neg nm
                pp with _inpterm0 -> p1(| => pp, _inpterm0)
    ;
    proc p1 :: | => Passer(|Mem(Int|)), IntTerminal =
        | => passer, _inpterm0, mm, nmpp -> do
            hput MemGet on mm
    |
    []
)

( nmpp
    []
    |
    []
)

-- Global translation
_inpterm0 ---> _inpterm0
_inpterm1 ---> _inpterm1
passer    ---> passer
mem       ---> mem
mm        ---> mm
nmpp      ---> nmpp

-- Running processes:
proc p2 :: | Passer(| Mem(Int|)) => IntTerminal, Mem(Int|) =
    | passer => _inpterm1, mem -> do
        put 100 on _inpterm1
        hput IntTerminalGet on _inpterm1
        get x on _inpterm1
        hput MemPut on mem
        put x on mem
        fork passer as
            mm with mem -> do
                mm |=| mem
            nmpp with _inpterm1 -> do
                split nmpp into nm,pp
                plug
                    p2( | pp => _inpterm1,z)
```

```
                    z,nm => -> z |=| neg nm
```

Another step for p2 (boring service channel step)

```
-- Channel manager
( _inpterm0,
    []
    |
    []
)

( _inpterm1,
    proc p2 :: | Passer(| Mem(Int|)) => IntTerminal, Mem(Int|) =
        | passer => _inpterm1, mem -> do
            put 100 on _inpterm1
    ;
    proc p2 :: | Passer(| Mem(Int|)) => IntTerminal, Mem(Int|) =
        | passer => _inpterm1, mem -> do
            hput IntTerminalPut on _inpterm1
    |
    []
)
( passer,
    proc p1 :: | => Passer(|Mem(Int|)), IntTerminal =
        | => passer, _inpterm0 -> do
            split passer into mm,nmpp
    |
    []
    )

( mem,
    []
    |
    proc memory :: A | Mem(A|) => =
        100 | mem => -> do
            hcase mem of
                MemPut -> do
                    get y on mem
                    memory(y | mem => )
                MemGet -> do
                    put 100 on mem
                    memory(100 | mem => )
                MemCls -> do
                    halt mem
)
```

```
( mm
    proc p1 :: | => Passer(|Mem(Int|)), IntTerminal =
        | => passer, _inpterm0, mm, nmpp -> do
            get y on mm
            hput IntTerminalPut on _inpterm0
            put y on _inpterm0
            hput IntTerminalGet on _inpterm0
            get x on _inpterm0
            hput MemPut on mm
            put x on mm
            fork nmpp as
                nm with mm -> mm |=| neg nm
                pp with _inpterm0 -> p1(| => pp, _inpterm0)
    ;
    proc p1 :: | => Passer(|Mem(Int|)), IntTerminal =
        | => passer, _inpterm0, mm, nmpp -> do
            hput MemGet on mm
    |
    []
)

( nmpp
    []
    |
    []
)

-- Global translation
_inpterm0 ---> _inpterm0
_inpterm1 ---> _inpterm1
passer    ---> passer
mem       ---> mem
mm        ---> mm
nmpp      ---> nmpp

-- Running processes:
proc p2 :: | Passer(| Mem(Int|)) => IntTerminal, Mem(Int|) =
    | passer => _inpterm1, mem -> do
        hput IntTerminalGet on _inpterm1
        get x on _inpterm1
        hput MemPut on mem
        put x on mem
        fork passer as
            mm with mem -> do
                mm |=| mem
            nmpp with _inpterm1 -> do
```

```
                split nmpp into nm,pp
                plug
                    p2( | pp => _inpterm1,z)
                    z,nm => -> z |=| neg nm
```

Another step for `p2` (boring service channel step)

```
-- Channel manager
( _inpterm0,
    []
    |
    []
)

( _inpterm1,
    proc p2 :: | Passer(| Mem(Int|)) => IntTerminal, Mem(Int|) =
        | passer => _inpterm1, mem -> do
            hput IntTerminalGet on _inpterm1
    ;
    proc p2 :: | Passer(| Mem(Int|)) => IntTerminal, Mem(Int|) =
        | passer => _inpterm1, mem -> do
            put 100 on _inpterm1
    ;
    proc p2 :: | Passer(| Mem(Int|)) => IntTerminal, Mem(Int|) =
        | passer => _inpterm1, mem -> do
            hput IntTerminalPut on _inpterm1
    |
    []
)
( passer,
    proc p1 :: | => Passer(|Mem(Int|)), IntTerminal =
        | => passer, _inpterm0 -> do
            split passer into mm,nmpp
    |
    []
    )

( mem,
    []
    |
    proc memory :: A | Mem(A|) => =
        100 | mem => -> do
            hcase mem of
                MemPut -> do
                    get y on mem
                    memory(y | mem => )
```

```
                MemGet -> do
                    put 100 on mem
                    memory(100 | mem => )
                MemCls -> do
                    halt mem
)

( mm
    proc p1 :: | => Passer(|Mem(Int|)), IntTerminal =
        | => passer, _inpterm0, mm, nmpp -> do
            get y on mm
            hput IntTerminalPut on _inpterm0
            put y on _inpterm0
            hput IntTerminalGet on _inpterm0
            get x on _inpterm0
            hput MemPut on mm
            put x on mm
            fork nmpp as
                nm with mm -> mm |=| neg nm
                pp with _inpterm0 -> p1(| => pp, _inpterm0)
    ;
    proc p1 :: | => Passer(|Mem(Int|)), IntTerminal =
        | => passer, _inpterm0, mm, nmpp -> do
            hput MemGet on mm
    |
    []
)

( nmpp
    []
    |
    []
)

-- Global translation
_inpterm0 ---> _inpterm0
_inpterm1 ---> _inpterm1
passer    ---> passer
mem       ---> mem
mm        ---> mm
nmpp      ---> nmpp

-- Running processes:
proc p2 :: | Passer(| Mem(Int|)) => IntTerminal, Mem(Int|) =
    | passer => _inpterm1, mem -> do
        get x on _inpterm1
```

```
        hput MemPut on mem
        put x on mem
        fork passer as
            mm with mem -> do
                mm |=| mem
            nmpp with _inpterm1 -> do
                split nmpp into nm,pp
                plug
                    p2( | pp => _inpterm1,z)
                    z,nm => -> z |=| neg nm
```

Another step for p2 (boring service channel step)

```
-- Channel manager
( _inpterm0,
    []
    |
    []
)

( _inpterm1,
    proc p2 :: | Passer(| Mem(Int|)) => IntTerminal, Mem(Int|) =
        | passer => _inpterm1, mem -> do
            get x on _inpterm1
            hput MemPut on mem
            put x on mem
            fork passer as
                mm with mem -> do
                    mm |=| mem
                nmpp with _inpterm1 -> do
                    split nmpp into nm,pp
                    plug
                        p2( | pp => _inpterm1,z)
                        z,nm => -> z |=| neg nm
    ;
    proc p2 :: | Passer(| Mem(Int|)) => IntTerminal, Mem(Int|) =
        | passer => _inpterm1, mem -> do
            hput IntTerminalGet on _inpterm1
    ;
    proc p2 :: | Passer(| Mem(Int|)) => IntTerminal, Mem(Int|) =
        | passer => _inpterm1, mem -> do
            put 100 on _inpterm1
    ;
    proc p2 :: | Passer(| Mem(Int|)) => IntTerminal, Mem(Int|) =
        | passer => _inpterm1, mem -> do
            hput IntTerminalPut on _inpterm1
```

```
    |
    []
)
( passer,
    proc p1 :: | => Passer(|Mem(Int|)), IntTerminal =
        | => passer, _inpterm0 -> do
            split passer into mm,nmpp
    |
    []
    )

( mem,
    []
    |
    proc memory :: A | Mem(A|) => =
        100 | mem => -> do
            hcase mem of
                MemPut -> do
                    get y on mem
                    memory(y | mem => )
                MemGet -> do
                    put 100 on mem
                    memory(100 | mem => )
                MemCls -> do
                    halt mem
)

( mm
    proc p1 :: | => Passer(|Mem(Int|)), IntTerminal =
        | => passer, _inpterm0, mm, nmpp -> do
            get y on mm
            hput IntTerminalPut on _inpterm0
            put y on _inpterm0
            hput IntTerminalGet on _inpterm0
            get x on _inpterm0
            hput MemPut on mm
            put x on mm
            fork nmpp as
                nm with mm -> mm |=| neg nm
                pp with _inpterm0 -> p1(| => pp, _inpterm0)
    ;
    proc p1 :: | => Passer(|Mem(Int|)), IntTerminal =
        | => passer, _inpterm0, mm, nmpp -> do
            hput MemGet on mm
    |
    []
```

```
)

( nmpp
    []
    |
    []
)

-- Global translation
_inpterm0 ---> _inpterm0
_inpterm1 ---> _inpterm1
passer    ---> passer
mem       ---> mem
mm        ---> mm
nmpp      ---> nmpp

-- Running processes:
```

Then, we assume the service channel executes and lets say it puts `1` on the system.

```
-- Channel manager
( _inpterm0,
    []
    |
    []
)

( _inpterm1,
    proc p2 :: | Passer(| Mem(Int|)) => IntTerminal, Mem(Int|) =
        | passer => _inpterm1, mem -> do
            get x on _inpterm1
            hput MemPut on mem
            put x on mem
            fork passer as
                mm with mem -> do
                    mm |=| mem
                nmpp with _inpterm1 -> do
                    split nmpp into nm,pp
                    plug
                        p2( | pp => _inpterm1,z)
                        z,nm => -> z |=| neg nm
    |
    put 1 on mem
)
( passer,
    proc p1 :: | => Passer(|Mem(Int|)), IntTerminal =
```

```
        | => passer, _inpterm0 -> do
            split passer into mm,nmpp
    |
    []
    )

( mem,
    []
    |
    proc memory :: A | Mem(A|) => =
        100 | mem => -> do
            hcase mem of
                MemPut -> do
                    get y on mem
                    memory(y | mem => )
                MemGet -> do
                    put 100 on mem
                    memory(100 | mem => )
                MemCls -> do
                    halt mem
)

( mm
    proc p1 :: | => Passer(|Mem(Int|)), IntTerminal =
        | => passer, _inpterm0, mm, nmpp -> do
            get y on mm
            hput IntTerminalPut on _inpterm0
            put y on _inpterm0
            hput IntTerminalGet on _inpterm0
            get x on _inpterm0
            hput MemPut on mm
            put x on mm
            fork nmpp as
                nm with mm -> mm |=| neg nm
                pp with _inpterm0 -> p1(| => pp, _inpterm0)
    ;
    proc p1 :: | => Passer(|Mem(Int|)), IntTerminal =
        | => passer, _inpterm0, mm, nmpp -> do
            hput MemGet on mm
    |
    []
)

( nmpp
    []
    |
```

```
    []
)

-- Global translation
_inpterm0 ---> _inpterm0
_inpterm1 ---> _inpterm1
passer    ---> passer
mem       ---> mem
mm        ---> mm
nmpp      ---> nmpp

-- Running processes:
```

3 End of boring service channel p2|

So, p2 may resume to do some interesting things (of course, we substitute the argument as appropriate).

```
-- Channel manager
( _inpterm0,
    []
    |
    []
)

( _inpterm1,
    []
    |
    []
)
( passer,
    proc p1 :: | => Passer(|Mem(Int|)), IntTerminal =
        | => passer, _inpterm0 -> do
            split passer into mm,nmpp
    |
    []
    )

( mem,
    []
    |
    proc memory :: A | Mem(A|) => =
        100 | mem => -> do
            hcase mem of
                MemPut -> do
                    get y on mem
```

```
                    memory(y | mem => )
                MemGet -> do
                    put 100 on mem
                    memory(100 | mem => )
                MemCls -> do
                    halt mem
)

( mm
    proc p1 :: | => Passer(|Mem(Int|)), IntTerminal =
        | => passer, _inpterm0, mm, nmpp -> do
            get y on mm
            hput IntTerminalPut on _inpterm0
            put y on _inpterm0
            hput IntTerminalGet on _inpterm0
            get x on _inpterm0
            hput MemPut on mm
            put x on mm
            fork nmpp as
                nm with mm -> mm |=| neg nm
                pp with _inpterm0 -> p1(| => pp, _inpterm0)
    ;
    proc p1 :: | => Passer(|Mem(Int|)), IntTerminal =
        | => passer, _inpterm0, mm, nmpp -> do
            hput MemGet on mm
    |
    []
)

( nmpp
    []
    |
    []
)

-- Global translation
_inpterm0 ---> _inpterm0
_inpterm1 ---> _inpterm1
passer    ---> passer
mem       ---> mem
mm        ---> mm
nmpp      ---> nmpp

-- Running processes:
proc p2 :: | Passer(| Mem(Int|)) => IntTerminal, Mem(Int|) =
    | passer => _inpterm1, mem -> do
```

```
        hput MemPut on mem
        put 1 on mem
        fork passer as
            mm with mem -> do
                mm |=| mem
            nmpp with _inpterm1 -> do
                split nmpp into nm,pp
                plug
                    p2( | pp => _inpterm1,z)
                    z,nm => -> z |=| neg nm
```

Another step for `p2`

```
-- Channel manager
( _inpterm0,
    []
    |
    []
)

( _inpterm1,
    []
    |
    []
)
( passer,
    proc p1 :: | => Passer(|Mem(Int|)), IntTerminal =
        | => passer, _inpterm0 -> do
            split passer into mm,nmpp
    |
    []
    )

( mem,
    proc p2 :: | Passer(| Mem(Int|)) => IntTerminal, Mem(Int|) =
        | passer => _inpterm1, mem -> do
            hput MemPut on mem
    |
    proc memory :: A | Mem(A|) => =
        100 | mem => -> do
            hcase mem of
                MemPut -> do
                    get y on mem
                    memory(y | mem => )
                MemGet -> do
                    put 100 on mem
                    memory(100 | mem => )
```

```
                MemCls -> do
                    halt mem
)

( mm
    proc p1 :: | => Passer(|Mem(Int|)), IntTerminal =
        | => passer, _inpterm0, mm, nmpp -> do
            get y on mm
            hput IntTerminalPut on _inpterm0
            put y on _inpterm0
            hput IntTerminalGet on _inpterm0
            get x on _inpterm0
            hput MemPut on mm
            put x on mm
            fork nmpp as
                nm with mm -> mm |=| neg nm
                pp with _inpterm0 -> p1(| => pp, _inpterm0)
    ;
    proc p1 :: | => Passer(|Mem(Int|)), IntTerminal =
        | => passer, _inpterm0, mm, nmpp -> do
            hput MemGet on mm
    |
    []
)

( nmpp
    []
    |
    []
)

-- Global translation
_inpterm0 ---> _inpterm0
_inpterm1 ---> _inpterm1
passer    ---> passer
mem       ---> mem
mm        ---> mm
nmpp      ---> nmpp

-- Running processes:
proc p2 :: | Passer(| Mem(Int|)) => IntTerminal, Mem(Int|) =
    | passer => _inpterm1, mem -> do
        put 1 on mem
        fork passer as
            mm with mem -> do
                mm |=| mem
```

```
            nmpp with _inpterm1 -> do
                split nmpp into nm,pp
                plug
                    p2( | pp => _inpterm1,z)
                    z,nm => -> z |=| neg nm
```

Another step for `p2`

```
-- Channel manager
( _inpterm0,
    []
    |
    []
)

( _inpterm1,
    []
    |
    []
)
( passer,
    proc p1 :: | => Passer(|Mem(Int|)), IntTerminal =
        | => passer, _inpterm0 -> do
            split passer into mm,nmpp
    |
    []
    )

( mem,
    proc p2 :: | Passer(| Mem(Int|)) => IntTerminal, Mem(Int|) =
        | passer => _inpterm1, mem -> do
            put 1 on mem
    ;
    proc p2 :: | Passer(| Mem(Int|)) => IntTerminal, Mem(Int|) =
        | passer => _inpterm1, mem -> do
            hput MemPut on mem
    |
    proc memory :: A | Mem(A|) => =
        100 | mem => -> do
            hcase mem of
                MemPut -> do
                    get y on mem
                    memory(y | mem => )
                MemGet -> do
                    put 100 on mem
                    memory(100 | mem => )
                MemCls -> do
```

```
                    halt mem
)

( mm
    proc p1 :: | => Passer(|Mem(Int|)), IntTerminal =
        | => passer, _inpterm0, mm, nmpp -> do
            get y on mm
            hput IntTerminalPut on _inpterm0
            put y on _inpterm0
            hput IntTerminalGet on _inpterm0
            get x on _inpterm0
            hput MemPut on mm
            put x on mm
            fork nmpp as
                nm with mm -> mm |=| neg nm
                pp with _inpterm0 -> p1(| => pp, _inpterm0)
    ;
    proc p1 :: | => Passer(|Mem(Int|)), IntTerminal =
        | => passer, _inpterm0, mm, nmpp -> do
            hput MemGet on mm
    |
    []
)

( nmpp
    []
    |
    []
)

-- Global translation
_inpterm0 ---> _inpterm0
_inpterm1 ---> _inpterm1
passer    ---> passer
mem       ---> mem
mm        ---> mm
nmpp      ---> nmpp

-- Running processes:
proc p2 :: | Passer(| Mem(Int|)) => IntTerminal, Mem(Int|) =
    | passer => _inpterm1, mem -> do
        fork passer as
            mm with mem -> do
                mm |=| mem
            nmpp with _inpterm1 -> do
                split nmpp into nm,pp
```

```
                plug
                    p2( | pp => _inpterm1,z)
                    z,nm => -> z |=| neg nm
```

Another step for p2, where we note that p2 suspends itself on the input polarity of the channel passer.

```
-- Channel manager
( _inpterm0,
    []
    |
    []
)

( _inpterm1,
    []
    |
    []
)
( passer,
    proc p1 :: | => Passer(|Mem(Int|)), IntTerminal =
        | => passer, _inpterm0 -> do
            split passer into mm,nmpp
    |
    proc p2 :: | Passer(| Mem(Int|)) => IntTerminal, Mem(Int|) =
        | passer => _inpterm1, mem -> do
            fork passer as
                mm with mem -> do
                    mm |=| mem
                nmpp with _inpterm1 -> do
                    split nmpp into nm,pp
                    plug
                        p2( | pp => _inpterm1,z)
                        z,nm => -> z |=| neg nm
    )

( mem,
    proc p2 :: | Passer(| Mem(Int|)) => IntTerminal, Mem(Int|) =
        | passer => _inpterm1, mem -> do
            put 1 on mem
    ;
    proc p2 :: | Passer(| Mem(Int|)) => IntTerminal, Mem(Int|) =
        | passer => _inpterm1, mem -> do
            hput MemPut on mem
    |
    proc memory :: A | Mem(A|) => =
        100 | mem => -> do
```

```
            hcase mem of
                MemPut -> do
                    get y on mem
                    memory(y | mem => )
                MemGet -> do
                    put 100 on mem
                    memory(100 | mem => )
                MemCls -> do
                    halt mem
)

( mm
    proc p1 :: | => Passer(|Mem(Int|)), IntTerminal =
        | => passer, _inpterm0, mm, nmpp -> do
            get y on mm
            hput IntTerminalPut on _inpterm0
            put y on _inpterm0
            hput IntTerminalGet on _inpterm0
            get x on _inpterm0
            hput MemPut on mm
            put x on mm
            fork nmpp as
                nm with mm -> mm |=| neg nm
                pp with _inpterm0 -> p1(| => pp, _inpterm0)
    ;
    proc p1 :: | => Passer(|Mem(Int|)), IntTerminal =
        | => passer, _inpterm0, mm, nmpp -> do
            hput MemGet on mm
    |
    []
)

( nmpp
    []
    |
    []
)

-- Global translation
_inpterm0 ---> _inpterm0
_inpterm1 ---> _inpterm1
passer    ---> passer
mem       ---> mem
mm        ---> mm
nmpp      ---> nmpp
```

```
-- Running processes:
```

Now, we let the channel `mem` continue, so the process `memory` may continue.

```
-- Channel manager
( _inpterm0,
    []
    |
    []
)

( _inpterm1,
    []
    |
    []
)
( passer,
    proc p1 :: | => Passer(|Mem(Int|)), IntTerminal =
        | => passer, _inpterm0 -> do
            split passer into mm,nmpp
    |
    proc p2 :: | Passer(| Mem(Int|)) => IntTerminal, Mem(Int|) =
        | passer => _inpterm1, mem -> do
            fork passer as
                mm with mem -> do
                    mm |=| mem
                nmpp with _inpterm1 -> do
                    split nmpp into nm,pp
                    plug
                        p2( | pp => _inpterm1,z)
                        z,nm => -> z |=| neg nm
    )

( mem,
    proc p2 :: | Passer(| Mem(Int|)) => IntTerminal, Mem(Int|) =
        | passer => _inpterm1, mem -> do
            put 1 on mem
    |
    []
)

( mm
    proc p1 :: | => Passer(|Mem(Int|)), IntTerminal =
        | => passer, _inpterm0, mm, nmpp -> do
            get y on mm
            hput IntTerminalPut on _inpterm0
            put y on _inpterm0
```

```
            hput IntTerminalGet on _inpterm0
            get x on _inpterm0
            hput MemPut on mm
            put x on mm
            fork nmpp as
                nm with mm -> mm |=| neg nm
                pp with _inpterm0 -> p1(| => pp, _inpterm0)
    ;
    proc p1 :: | => Passer(|Mem(Int|)), IntTerminal =
        | => passer, _inpterm0, mm, nmpp -> do
            hput MemGet on mm
    |
    []
)

( nmpp
    []
    |
    []
)

-- Global translation
_inpterm0 ---> _inpterm0
_inpterm1 ---> _inpterm1
passer    ---> passer
mem       ---> mem
mm        ---> mm
nmpp      ---> nmpp

-- Running processes:
proc memory :: A | Mem(A|) => =
    100 | mem => -> do
        get y on mem
        memory(y | mem => )
```

Another step for `memory`.

```
-- Channel manager
( _inpterm0,
    []
    |
    []
)

( _inpterm1,
    []
    |
```

```
    []
)
( passer,
    proc p1 :: | => Passer(|Mem(Int|)), IntTerminal =
        | => passer, _inpterm0 -> do
            split passer into mm,nmpp
    |
    proc p2 :: | Passer(| Mem(Int|)) => IntTerminal, Mem(Int|) =
        | passer => _inpterm1, mem -> do
            fork passer as
                mm with mem -> do
                    mm |=| mem
                nmpp with _inpterm1 -> do
                    split nmpp into nm,pp
                    plug
                        p2( | pp => _inpterm1,z)
                        z,nm => -> z |=| neg nm
    )

( mem,
    proc p2 :: | Passer(| Mem(Int|)) => IntTerminal, Mem(Int|) =
        | passer => _inpterm1, mem -> do
            put 1 on mem
    |
    proc memory :: A | Mem(A|) => =
        100 | mem => -> do
            get y on mem
            memory(y | mem => )
)

( mm
    proc p1 :: | => Passer(|Mem(Int|)), IntTerminal =
        | => passer, _inpterm0, mm, nmpp -> do
            get y on mm
            hput IntTerminalPut on _inpterm0
            put y on _inpterm0
            hput IntTerminalGet on _inpterm0
            get x on _inpterm0
            hput MemPut on mm
            put x on mm
            fork nmpp as
                nm with mm -> mm |=| neg nm
                pp with _inpterm0 -> p1(| => pp, _inpterm0)
    ;
    proc p1 :: | => Passer(|Mem(Int|)), IntTerminal =
        | => passer, _inpterm0, mm, nmpp -> do
```

```
            hput MemGet on mm
    |
    []
)

( nmpp
    []
    |
    []
)

-- Global translation
_inpterm0 ---> _inpterm0
_inpterm1 ---> _inpterm1
passer    ---> passer
mem       ---> mem
mm        ---> mm
nmpp      ---> nmpp

-- Running processes:
```

Another step for the channel `mem`, and continute `memory`.

```
-- Channel manager
( _inpterm0,
    []
    |
    []
)

( _inpterm1,
    []
    |
    []
)
( passer,
    proc p1 :: | => Passer(|Mem(Int|)), IntTerminal =
        | => passer, _inpterm0 -> do
            split passer into mm,nmpp
    |
    proc p2 :: | Passer(| Mem(Int|)) => IntTerminal, Mem(Int|) =
        | passer => _inpterm1, mem -> do
            fork passer as
                mm with mem -> do
                    mm |=| mem
                nmpp with _inpterm1 -> do
```

```
                    split nmpp into nm,pp
                    plug
                        p2( | pp => _inpterm1,z)
                        z,nm => -> z |=| neg nm
    )

( mem,
    []
    |
    []
)

( mm
    proc p1 :: | => Passer(|Mem(Int|)), IntTerminal =
        | => passer, _inpterm0, mm, nmpp -> do
            get y on mm
            hput IntTerminalPut on _inpterm0
            put y on _inpterm0
            hput IntTerminalGet on _inpterm0
            get x on _inpterm0
            hput MemPut on mm
            put x on mm
            fork nmpp as
                nm with mm -> mm |=| neg nm
                pp with _inpterm0 -> p1(| => pp, _inpterm0)
    ;
    proc p1 :: | => Passer(|Mem(Int|)), IntTerminal =
        | => passer, _inpterm0, mm, nmpp -> do
            hput MemGet on mm
    |
    []
)

( nmpp
    []
    |
    []
)

-- Global translation
_inpterm0 ---> _inpterm0
_inpterm1 ---> _inpterm1
passer    ---> passer
mem       ---> mem
mm        ---> mm
nmpp      ---> nmpp
```

```
-- Running processes:
proc memory :: A | Mem(A|) => =
    100 | mem => -> do
        memory(1 | mem => )
```

Another step for `memory` (recursive call, substitute arguments).

```
-- Channel manager
( _inpterm0,
    []
    |
    []
)

( _inpterm1,
    []
    |
    []
)
( passer,
    proc p1 :: | => Passer(|Mem(Int|)), IntTerminal =
        | => passer, _inpterm0 -> do
            split passer into mm,nmpp
    |
    proc p2 :: | Passer(| Mem(Int|)) => IntTerminal, Mem(Int|) =
        | passer => _inpterm1, mem -> do
            fork passer as
                mm with mem -> do
                    mm |=| mem
                nmpp with _inpterm1 -> do
                    split nmpp into nm,pp
                    plug
                        p2( | pp => _inpterm1,z)
                        z,nm => -> z |=| neg nm
    )

( mem,
    []
    |
    []
)

( mm
    proc p1 :: | => Passer(|Mem(Int|)), IntTerminal =
        | => passer, _inpterm0, mm, nmpp -> do
            get y on mm
```

```
            hput IntTerminalPut on _inpterm0
            put y on _inpterm0
            hput IntTerminalGet on _inpterm0
            get x on _inpterm0
            hput MemPut on mm
            put x on mm
            fork nmpp as
                nm with mm -> mm |=| neg nm
                pp with _inpterm0 -> p1(| => pp, _inpterm0)
    ;
    proc p1 :: | => Passer(|Mem(Int|)), IntTerminal =
        | => passer, _inpterm0, mm, nmpp -> do
            hput MemGet on mm
    |
    []
)

( nmpp
    []
    |
    []
)

-- Global translation
_inpterm0 ---> _inpterm0
_inpterm1 ---> _inpterm1
passer    ---> passer
mem       ---> mem
mm        ---> mm
nmpp      ---> nmpp

-- Running processes:
proc memory :: A | Mem(A|) => =
    1 | mem => -> do
        hcase mem of
            MemPut -> do
                get y on mem
                memory(y | mem => )
            MemGet -> do
                put 1 on mem
                memory(1 | mem => )
            MemCls -> do
                halt mem
```

Another step for `memory`

```
-- Channel manager
```

```
( _inpterm0,
    []
    |
    []
)

( _inpterm1,
    []
    |
    []
)
( passer,
    proc p1 :: | => Passer(|Mem(Int|)), IntTerminal =
        | => passer, _inpterm0 -> do
            split passer into mm,nmpp
    |
    proc p2 :: | Passer(| Mem(Int|)) => IntTerminal, Mem(Int|) =
        | passer => _inpterm1, mem -> do
            fork passer as
                mm with mem -> do
                    mm |=| mem
                nmpp with _inpterm1 -> do
                    split nmpp into nm,pp
                    plug
                        p2( | pp => _inpterm1,z)
                        z,nm => -> z |=| neg nm
    )

( mem,
    []
    |
    proc memory :: A | Mem(A|) => =
        1 | mem => -> do
            hcase mem of
                MemPut -> do
                    get y on mem
                    memory(y | mem => )
                MemGet -> do
                    put 1 on mem
                    memory(1 | mem => )
                MemCls -> do
                    halt mem
)

( mm
    proc p1 :: | => Passer(|Mem(Int|)), IntTerminal =
```

```
        | => passer, _inpterm0, mm, nmpp -> do
            get y on mm
            hput IntTerminalPut on _inpterm0
            put y on _inpterm0
            hput IntTerminalGet on _inpterm0
            get x on _inpterm0
            hput MemPut on mm
            put x on mm
            fork nmpp as
                nm with mm -> mm |=| neg nm
                pp with _inpterm0 -> p1(| => pp, _inpterm0)
    ;
    proc p1 :: | => Passer(|Mem(Int|)), IntTerminal =
        | => passer, _inpterm0, mm, nmpp -> do
            hput MemGet on mm
    |
    []
)

( nmpp
    []
    |
    []
)

-- Global translation
_inpterm0 ---> _inpterm0
_inpterm1 ---> _inpterm1
passer    ---> passer
mem       ---> mem
mm        ---> mm
nmpp      ---> nmpp

-- Running processes:
```

Now, we execute a step on the channel `passer`. So, the process `p2` may continue, and we note that `mm,nmpp` are of input polarity for `p2`.

```
-- Channel manager
( _inpterm0,
    []
    |
    []
)

( _inpterm1,
    []
```

```
    |
    []
)
( passer,
    []
    |
    []
    )

( mem,
    []
    |
    proc memory :: A | Mem(A|) => =
        1 | mem => -> do
            hcase mem of
                MemPut -> do
                    get y on mem
                    memory(y | mem => )
                MemGet -> do
                    put 1 on mem
                    memory(1 | mem => )
                MemCls -> do
                    halt mem
)

( mm
    proc p1 :: | => Passer(|Mem(Int|)), IntTerminal =
        | => passer, _inpterm0, mm, nmpp -> do
            get y on mm
            hput IntTerminalPut on _inpterm0
            put y on _inpterm0
            hput IntTerminalGet on _inpterm0
            get x on _inpterm0
            hput MemPut on mm
            put x on mm
            fork nmpp as
                nm with mm -> mm |=| neg nm
                pp with _inpterm0 -> p1(| => pp, _inpterm0)
    ;
    proc p1 :: | => Passer(|Mem(Int|)), IntTerminal =
        | => passer, _inpterm0, mm, nmpp -> do
            hput MemGet on mm
    |
    []
)
```

```
( nmpp
    []
    |
    []
)

-- Global translation
_inpterm0 ---> _inpterm0
_inpterm1 ---> _inpterm1
passer    ---> passer
mem       ---> mem
mm        ---> mm
nmpp      ---> nmpp

-- Running processes:
proc p2_top :: | Passer(| Mem(Int|)) => IntTerminal, Mem(Int|) =
    | passer, mm  => mem -> do
        mm |=| mem
proc p2_bot :: | Passer(| Mem(Int|)) => IntTerminal, Mem(Int|) =
    | passer,nmpp  => _inpterm1, mem -> do
        nmpp with _inpterm1 -> do
            split nmpp into nm,pp
            plug
                p2( | pp => _inpterm1,z)
                z,nm => -> z |=| neg nm
```

Now, we execute `p2_top`.

```
-- Channel manager
( _inpterm0,
    []
    |
    []
)

( _inpterm1,
    []
    |
    []
)
( passer,
    []
    |
    []
    )

( mem,
```

```
    []
    |
    proc memory :: A | Mem(A|) => =
        1 | mem => -> do
            hcase mem of
                MemPut -> do
                    get y on mem
                    memory(y | mem => )
                MemGet -> do
                    put 1 on mem
                    memory(1 | mem => )
                MemCls -> do
                    halt mem
)

( mm
    proc p1 :: | => Passer(|Mem(Int|)), IntTerminal =
        | => passer, _inpterm0, mm, nmpp -> do
            get y on mm
            hput IntTerminalPut on _inpterm0
            put y on _inpterm0
            hput IntTerminalGet on _inpterm0
            get x on _inpterm0
            hput MemPut on mm
            put x on mm
            fork nmpp as
                nm with mm -> mm |=| neg nm
                pp with _inpterm0 -> p1(| => pp, _inpterm0)
    ;
    proc p1 :: | => Passer(|Mem(Int|)), IntTerminal =
        | => passer, _inpterm0, mm, nmpp -> do
            hput MemGet on mm
    |
    proc p2_top :: | Passer(| Mem(Int|)) => IntTerminal, Mem(Int|) =
        | passer, mm  => mem -> do
            mm |=| mem
)

( nmpp
    []
    |
    []
)

-- Global translation
_inpterm0 ---> _inpterm0
```

```
_inpterm1 ---> _inpterm1
passer    ---> passer
mem       ---> mem
mm        ---> mm
nmpp      ---> nmpp

-- Running processes:
proc p2_bot :: | Passer(| Mem(Int|)) => IntTerminal, Mem(Int|) =
    | passer,nmpp  => _inpterm1, mem -> do
        split nmpp into nm,pp
        plug
            p2( | pp => _inpterm1,z)
            z,nm => -> z |=| neg nm
```

Now, we execute a step for `p2_bot`, where we note that the "split" command opens new channels `nm,pp` which are added to the translation of `p2_bot` (and are of input polariy since `nmpp` is of input polarity).

```
-- Channel manager
( _inpterm0,
    []
    |
    []
)

( _inpterm1,
    []
    |
    []
)
( passer,
    []
    |
    []
    )

( mem,
    []
    |
    proc memory :: A | Mem(A|) => =
        1 | mem => -> do
            hcase mem of
                MemPut -> do
                    get y on mem
                    memory(y | mem => )
                MemGet -> do
                    put 1 on mem
```

```
                    memory(1 | mem => )
                MemCls -> do
                    halt mem
)

( mm
    proc p1 :: | => Passer(|Mem(Int|)), IntTerminal =
        | => passer, _inpterm0, mm, nmpp -> do
            get y on mm
            hput IntTerminalPut on _inpterm0
            put y on _inpterm0
            hput IntTerminalGet on _inpterm0
            get x on _inpterm0
            hput MemPut on mm
            put x on mm
            fork nmpp as
                nm with mm -> mm |=| neg nm
                pp with _inpterm0 -> p1(| => pp, _inpterm0)
    ;
    proc p1 :: | => Passer(|Mem(Int|)), IntTerminal =
        | => passer, _inpterm0, mm, nmpp -> do
            hput MemGet on mm
    |
    proc p2_top :: | Passer(| Mem(Int|)) => IntTerminal, Mem(Int|) =
        | passer, mm  => mem -> do
            mm |=| mem
)

( nmpp
    []
    |
    proc p2_bot :: | Passer(| Mem(Int|)) => IntTerminal, Mem(Int|) =
        | passer,nmpp  => _inpterm1, mem -> do
            split nmpp into nm,pp
)

( nm
    []
    |
    []
)

( pp
    []
    |
    []
```

```
)

-- Global translation
_inpterm0 ---> _inpterm0
_inpterm1 ---> _inpterm1
passer    ---> passer
mem       ---> mem
mm        ---> mm
nmpp      ---> nmpp
nm        ---> nm
pp        ---> pp

-- Running processes:
proc p2_bot :: | Passer(| Mem(Int|)) => IntTerminal, Mem(Int|) =
    | passer,nmpp,nm,pp => _inpterm1, mem -> do
        plug
            p2( | pp => _inpterm1,z)
            z,nm => -> z |=| neg nm
```

We execute another step for `p2_bot`. Note that this is a "plug" command, so we first open a new channel `z`

```
-- Channel manager
( _inpterm0,
    []
    |
    []
)

( _inpterm1,
    []
    |
    []
)
( passer,
    []
    |
    []
    )

( mem,
    []
    |
    proc memory :: A | Mem(A|) => =
        1 | mem => -> do
            hcase mem of
                MemPut -> do
```

```
                    get y on mem
                    memory(y | mem => )
                MemGet -> do
                    put 1 on mem
                    memory(1 | mem => )
                MemCls -> do
                    halt mem
)

( mm
    proc p1 :: | => Passer(|Mem(Int|)), IntTerminal =
        | => passer, _inpterm0, mm, nmpp -> do
            get y on mm
            hput IntTerminalPut on _inpterm0
            put y on _inpterm0
            hput IntTerminalGet on _inpterm0
            get x on _inpterm0
            hput MemPut on mm
            put x on mm
            fork nmpp as
                nm with mm -> mm |=| neg nm
                pp with _inpterm0 -> p1(| => pp, _inpterm0)
    ;
    proc p1 :: | => Passer(|Mem(Int|)), IntTerminal =
        | => passer, _inpterm0, mm, nmpp -> do
            hput MemGet on mm
    |
    proc p2_top :: | Passer(| Mem(Int|)) => IntTerminal, Mem(Int|) =
        | passer, mm  => mem -> do
            mm |=| mem
)

( nmpp
    []
    |
    proc p2_bot :: | Passer(| Mem(Int|)) => IntTerminal, Mem(Int|) =
        | passer,nmpp  => _inpterm1, mem -> do
            split nmpp into nm,pp
)

( nm
    []
    |
    []
)
```

```
( pp
    []
    |
    []
)

( z
    []
    |
    []
)

-- Global translation
_inpterm0 ---> _inpterm0
_inpterm1 ---> _inpterm1
passer    ---> passer
mem       ---> mem
mm        ---> mm
nmpp      ---> nmpp
nm        ---> nm
pp        ---> pp

-- Running processes:
proc p2_bot :: | Passer(| Mem(Int|)) => IntTerminal, Mem(Int|) =
    | passer,nmpp,nm,pp => _inpterm1, mem -> do
        plug
            p2( | pp => _inpterm1,z)
            z,nm => -> z |=| neg nm
```

Now, we do another step for `p2_bot` which includes actually running the processes in the plug command (substituting arguments as needed, and creating a new process `p2_bot_bot` for the bottom plug phrase).

```
-- Channel manager
( _inpterm0,
    []
    |
    []
)

( _inpterm1,
    []
    |
    []
)
( passer,
    []
```

```
    |
    []
    )

( mem,
    []
    |
    proc memory :: A | Mem(A|) => =
        1 | mem => -> do
            hcase mem of
                MemPut -> do
                    get y on mem
                    memory(y | mem => )
                MemGet -> do
                    put 1 on mem
                    memory(1 | mem => )
                MemCls -> do
                    halt mem
)

( mm
    proc p1 :: | => Passer(|Mem(Int|)), IntTerminal =
        | => passer, _inpterm0, mm, nmpp -> do
            get y on mm
            hput IntTerminalPut on _inpterm0
            put y on _inpterm0
            hput IntTerminalGet on _inpterm0
            get x on _inpterm0
            hput MemPut on mm
            put x on mm
            fork nmpp as
                nm with mm -> mm |=| neg nm
                pp with _inpterm0 -> p1(| => pp, _inpterm0)
    ;
    proc p1 :: | => Passer(|Mem(Int|)), IntTerminal =
        | => passer, _inpterm0, mm, nmpp -> do
            hput MemGet on mm
    |
    proc p2_top :: | Passer(| Mem(Int|)) => IntTerminal, Mem(Int|) =
        | passer, mm  => mem -> do
            mm |=| mem
)

( nmpp
    []
    |
```

```
    proc p2_bot :: | Passer(| Mem(Int|)) => IntTerminal, Mem(Int|) =
        | passer,nmpp  => _inpterm1, mem -> do
            split nmpp into nm,pp
)

( nm
    []
    |
    []
)

( pp
    []
    |
    []
)

( z
    []
    |
    []
)

-- Global translation
_inpterm0 ---> _inpterm0
_inpterm1 ---> _inpterm1
passer    ---> passer
mem       ---> mem
mm        ---> mm
nmpp      ---> nmpp
nm        ---> nm
pp        ---> pp

-- Running processes:
proc p2 :: | Passer(| Mem(Int|)) => IntTerminal, Mem(Int|) =
    | pp => _inpterm1, z -> do
        hcase pp of
            Passer -> do
                hput MemGet on z
                get y on z
                hput IntTerminalPut on _inpterm1
                put y on _inpterm1
                hput IntTerminalGet on _inpterm1
                get x on _inpterm1
                hput MemPut on z
                put x on z
```

```
                fork pp as
                    mm with z -> do
                        mm |=| z
                    nmpp with _inpterm1 -> do
                        split nmpp into nm,pp
                        plug
                            p2( | pp => _inpterm1,z)
                            z,nm => -> z |=| neg nm

proc p2_bot_bot =
    z,nm => -> z |=| neg nm
```

Now, let's pause the execution of `p2`, and execute the id comand on `mm` which connects it to `mem` (note we update the global translations as well)

```
-- Channel manager
( _inpterm0,
    []
    |
    []
)

( _inpterm1,
    []
    |
    []
)
( passer,
    []
    |
    []
    )

( mem,
    proc p1 :: | => Passer(|Mem(Int|)), IntTerminal =
        | => passer, _inpterm0, mm, nmpp -> do
            get y on mm
            hput IntTerminalPut on _inpterm0
            put y on _inpterm0
            hput IntTerminalGet on _inpterm0
            get x on _inpterm0
            hput MemPut on mm
            put x on mm
            fork nmpp as
                nm with mm -> mm |=| neg nm
                pp with _inpterm0 -> p1(| => pp, _inpterm0)
    ;
```

```
    proc p1 :: | => Passer(|Mem(Int|)), IntTerminal =
        | => passer, _inpterm0, mm, nmpp -> do
            hput MemGet on mm
    |
    proc memory :: A | Mem(A|) => =
        1 | mem => -> do
            hcase mem of
                MemPut -> do
                    get y on mem
                    memory(y | mem => )
                MemGet -> do
                    put 1 on mem
                    memory(1 | mem => )
                MemCls -> do
                    halt mem
)

( nmpp
    []
    |
    proc p2_bot :: | Passer(| Mem(Int|)) => IntTerminal, Mem(Int|) =
        | passer,nmpp  => _inpterm1, mem -> do
            split nmpp into nm,pp
)

( nm
    []
    |
    []
)

( pp
    []
    |
    []
)

( z
    []
    |
    []
)

-- Global translation
_inpterm0 ---> _inpterm0
_inpterm1 ---> _inpterm1
```

```
passer    ---> passer
mem       ---> mem
mm        ---> mem
nmpp      ---> nmpp
nm        ---> nm
pp        ---> pp

-- Running processes:
proc p2 :: | Passer(| Mem(Int|)) => IntTerminal, Mem(Int|) =
    | pp => _inpterm1, z -> do
        hcase pp of
            Passer -> do
                hput MemGet on z
                get y on z
                hput IntTerminalPut on _inpterm1
                put y on _inpterm1
                hput IntTerminalGet on _inpterm1
                get x on _inpterm1
                hput MemPut on z
                put x on z
                fork pp as
                    mm with z -> do
                        mm |=| z
                    nmpp with _inpterm1 -> do
                        split nmpp into nm,pp
                        plug
                            p2( | pp => _inpterm1,z)
                            z,nm => -> z |=| neg nm

proc p2_bot_bot =
    z,nm => -> z |=| neg nm
```

Now, to simplify things a bit, we're just gonna apply the substitution from the global translation i.e., we replace all `mm` with `mem`, so there's a little less book keeping to do in this trace (note: this does NOT include the `mm` in `p2`, since that `mm` is scoped differently and will open a new channel from some other split).

```
-- Channel manager
( _inpterm0,
    []
    |
    []
)

( _inpterm1,
    []
    |
```

```
    []
)
( passer,
    []
    |
    []
    )

( mem,
    proc p1 :: | => Passer(|Mem(Int|)), IntTerminal =
        | => passer, _inpterm0, mem, nmpp -> do
            get y on mem
            hput IntTerminalPut on _inpterm0
            put y on _inpterm0
            hput IntTerminalGet on _inpterm0
            get x on _inpterm0
            hput MemPut on mem
            put x on mem
            fork nmpp as
                nm with mem -> mem |=| neg nm
                pp with _inpterm0 -> p1(| => pp, _inpterm0)
    ;
    proc p1 :: | => Passer(|Mem(Int|)), IntTerminal =
        | => passer, _inpterm0, mem, nmpp -> do
            hput MemGet on mem
    |
    proc memory :: A | Mem(A|) => =
        1 | mem => -> do
            hcase mem of
                MemPut -> do
                    get y on mem
                    memory(y | mem => )
                MemGet -> do
                    put 1 on mem
                    memory(1 | mem => )
                MemCls -> do
                    halt mem
)

( nmpp
    []
    |
    proc p2_bot :: | Passer(| Mem(Int|)) => IntTerminal, Mem(Int|) =
        | passer,nmpp  => _inpterm1, mem -> do
            split nmpp into nm,pp
)
```

```
( nm
    []
    |
    []
)

( pp
    []
    |
    []
)

( z
    []
    |
    []
)

-- Global translation
_inpterm0 ---> _inpterm0
_inpterm1 ---> _inpterm1
passer    ---> passer
mem       ---> mem
mm        ---> mem
nmpp      ---> nmpp
nm        ---> nm
pp        ---> pp

-- Running processes:
proc p2 :: | Passer(| Mem(Int|)) => IntTerminal, Mem(Int|) =
    | pp => _inpterm1, z -> do
        hcase pp of
            Passer -> do
                hput MemGet on z
                get y on z
                hput IntTerminalPut on _inpterm1
                put y on _inpterm1
                hput IntTerminalGet on _inpterm1
                get x on _inpterm1
                hput MemPut on z
                put x on z
                fork pp as
                    mm with z -> do
                        mm |=| z
                    nmpp with _inpterm1 -> do
```

```
                        split nmpp into nm,pp
                        plug
                            p2( | pp => _inpterm1,z)
                            z,nm => -> z |=| neg nm

proc p2_bot_bot =
    z,nm => -> z |=| neg nm
```

Now, we execute a step on the channel `mem`, so `memory` may resume.

```
-- Channel manager
( _inpterm0,
    []
    |
    []
)

( _inpterm1,
    []
    |
    []
)
( passer,
    []
    |
    []
    )

( mem,
    proc p1 :: | => Passer(|Mem(Int|)), IntTerminal =
        | => passer, _inpterm0, mem, nmpp -> do
            get y on mem
            hput IntTerminalPut on _inpterm0
            put y on _inpterm0
            hput IntTerminalGet on _inpterm0
            get x on _inpterm0
            hput MemPut on mem
            put x on mem
            fork nmpp as
                nm with mem -> mem |=| neg nm
                pp with _inpterm0 -> p1(| => pp, _inpterm0)
    |
    []
)

( nmpp
    []
```

```
    |
    proc p2_bot :: | Passer(| Mem(Int|)) => IntTerminal, Mem(Int|) =
        | passer,nmpp  => _inpterm1, mem -> do
            split nmpp into nm,pp
)

( nm
    []
    |
    []
)

( pp
    []
    |
    []
)

( z
    []
    |
    []
)

-- Global translation
_inpterm0 ---> _inpterm0
_inpterm1 ---> _inpterm1
passer    ---> passer
mem       ---> mem
mm        ---> mem
nmpp      ---> nmpp
nm        ---> nm
pp        ---> pp

-- Running processes:
proc p2 :: | Passer(| Mem(Int|)) => IntTerminal, Mem(Int|) =
    | pp => _inpterm1, z -> do
        hcase pp of
            Passer -> do
                hput MemGet on z
                get y on z
                hput IntTerminalPut on _inpterm1
                put y on _inpterm1
                hput IntTerminalGet on _inpterm1
                get x on _inpterm1
                hput MemPut on z
```

```
                put x on z
                fork pp as
                    mm with z -> do
                        mm |=| z
                    nmpp with _inpterm1 -> do
                        split nmpp into nm,pp
                        plug
                            p2( | pp => _inpterm1,z)
                            z,nm => -> z |=| neg nm

proc p2_bot_bot =
    z,nm => -> z |=| neg nm

proc memory :: A | Mem(A|) => =
    1 | mem => -> do
        put 1 on mem
        memory(1 | mem => )
```

Continue a step for `memory`.

```
-- Channel manager
( _inpterm0,
    []
    |
    []
)

( _inpterm1,
    []
    |
    []
)
( passer,
    []
    |
    []
    )

( mem,
    proc p1 :: | => Passer(|Mem(Int|)), IntTerminal =
        | => passer, _inpterm0, mem, nmpp -> do
            get y on mem
            hput IntTerminalPut on _inpterm0
            put y on _inpterm0
            hput IntTerminalGet on _inpterm0
            get x on _inpterm0
            hput MemPut on mem
```

```
            put x on mem
            fork nmpp as
                nm with mem -> mem |=| neg nm
                pp with _inpterm0 -> p1(| => pp, _inpterm0)
    |
    proc memory :: A | Mem(A|) => =
        1 | mem => -> do
            put 1 on mem
)

( nmpp
    []
    |
    proc p2_bot :: | Passer(| Mem(Int|)) => IntTerminal, Mem(Int|) =
        | passer,nmpp  => _inpterm1, mem -> do
            split nmpp into nm,pp
)

( nm
    []
    |
    []
)

( pp
    []
    |
    []
)

( z
    []
    |
    []
)

-- Global translation
_inpterm0 ---> _inpterm0
_inpterm1 ---> _inpterm1
passer    ---> passer
mem       ---> mem
mm        ---> mem
nmpp      ---> nmpp
nm        ---> nm
pp        ---> pp
```

```
-- Running processes:
proc p2 :: | Passer(| Mem(Int|)) => IntTerminal, Mem(Int|) =
    | pp => _inpterm1, z -> do
        hcase pp of
            Passer -> do
                hput MemGet on z
                get y on z
                hput IntTerminalPut on _inpterm1
                put y on _inpterm1
                hput IntTerminalGet on _inpterm1
                get x on _inpterm1
                hput MemPut on z
                put x on z
                fork pp as
                    mm with z -> do
                        mm |=| z
                    nmpp with _inpterm1 -> do
                        split nmpp into nm,pp
                        plug
                            p2( | pp => _inpterm1,z)
                            z,nm => -> z |=| neg nm

proc p2_bot_bot =
    z,nm => -> z |=| neg nm

proc memory :: A | Mem(A|) => =
    1 | mem => -> do
        memory(1 | mem => )
```

Another step for `memory` (substitue agumetns as needed since it is a "run" call).

```
-- Channel manager
( _inpterm0,
    []
    |
    []
)

( _inpterm1,
    []
    |
    []
)
( passer,
    []
    |
    []
```

```
    )

( mem,
    proc p1 :: | => Passer(|Mem(Int|)), IntTerminal =
        | => passer, _inpterm0, mem, nmpp -> do
            get y on mem
            hput IntTerminalPut on _inpterm0
            put y on _inpterm0
            hput IntTerminalGet on _inpterm0
            get x on _inpterm0
            hput MemPut on mem
            put x on mem
            fork nmpp as
                nm with mem -> mem |=| neg nm
                pp with _inpterm0 -> p1(| => pp, _inpterm0)
    |
    proc memory :: A | Mem(A|) => =
        1 | mem => -> do
            put 1 on mem
)

( nmpp
    []
    |
    proc p2_bot :: | Passer(| Mem(Int|)) => IntTerminal, Mem(Int|) =
        | passer,nmpp  => _inpterm1, mem -> do
            split nmpp into nm,pp
)

( nm
    []
    |
    []
)

( pp
    []
    |
    []
)

( z
    []
    |
    []
)
```

```
-- Global translation
_inpterm0 ---> _inpterm0
_inpterm1 ---> _inpterm1
passer    ---> passer
mem       ---> mem
mm        ---> mem
nmpp      ---> nmpp
nm        ---> nm
pp        ---> pp

-- Running processes:
proc p2 :: | Passer(| Mem(Int|)) => IntTerminal, Mem(Int|) =
    | pp => _inpterm1, z -> do
        hcase pp of
            Passer -> do
                hput MemGet on z
                get y on z
                hput IntTerminalPut on _inpterm1
                put y on _inpterm1
                hput IntTerminalGet on _inpterm1
                get x on _inpterm1
                hput MemPut on z
                put x on z
                fork pp as
                    mm with z -> do
                        mm |=| z
                    nmpp with _inpterm1 -> do
                        split nmpp into nm,pp
                        plug
                            p2( | pp => _inpterm1,z)
                            z,nm => -> z |=| neg nm

proc p2_bot_bot =
    z,nm => -> z |=| neg nm

proc memory :: A | Mem(A|) => =
    1 | mem => -> do
        hcase mem of
            MemPut -> do
                get y on mem
                memory(y | mem => )
            MemGet -> do
                put 1 on mem
                memory(1 | mem => )
            MemCls -> do
```

```
                halt mem
```

Another step for `memory` (suspends on `mem` again).

```
-- Channel manager
( _inpterm0,
    []
    |
    []
)

( _inpterm1,
    []
    |
    []
)
( passer,
    []
    |
    []
    )

( mem,
    proc p1 :: | => Passer(|Mem(Int|)), IntTerminal =
        | => passer, _inpterm0, mem, nmpp -> do
            get y on mem
            hput IntTerminalPut on _inpterm0
            put y on _inpterm0
            hput IntTerminalGet on _inpterm0
            get x on _inpterm0
            hput MemPut on mem
            put x on mem
            fork nmpp as
                nm with mem -> mem |=| neg nm
                pp with _inpterm0 -> p1(| => pp, _inpterm0)
    |
    proc memory :: A | Mem(A|) => =
        1 | mem => -> do
            put 1 on mem
    ;
    proc memory :: A | Mem(A|) => =
        1 | mem => -> do
            hcase mem of
                MemPut -> do
                    get y on mem
                    memory(y | mem => )
```

```
                MemGet -> do
                    put 1 on mem
                    memory(1 | mem => )
                MemCls -> do
                    halt mem
)

( nmpp
    []
    |
    proc p2_bot :: | Passer(| Mem(Int|)) => IntTerminal, Mem(Int|) =
        | passer,nmpp  => _inpterm1, mem -> do
            split nmpp into nm,pp
)

( nm
    []
    |
    []
)

( pp
    []
    |
    []
)

( z
    []
    |
    []
)

-- Global translation
_inpterm0 ---> _inpterm0
_inpterm1 ---> _inpterm1
passer    ---> passer
mem       ---> mem
mm        ---> mem
nmpp      ---> nmpp
nm        ---> nm
pp        ---> pp

-- Running processes:
proc p2 :: | Passer(| Mem(Int|)) => IntTerminal, Mem(Int|) =
    | pp => _inpterm1, z -> do
```

```
        hcase pp of
            Passer -> do
                hput MemGet on z
                get y on z
                hput IntTerminalPut on _inpterm1
                put y on _inpterm1
                hput IntTerminalGet on _inpterm1
                get x on _inpterm1
                hput MemPut on z
                put x on z
                fork pp as
                    mm with z -> do
                        mm |=| z
                    nmpp with _inpterm1 -> do
                        split nmpp into nm,pp
                        plug
                            p2( | pp => _inpterm1,z)
                            z,nm => -> z |=| neg nm

proc p2_bot_bot =
    z,nm => -> z |=| neg nm
```

Execute the channel `mem`, so `p1` may resume (substitue arguments as needed).

```
-- Channel manager
( _inpterm0,
    []
    |
    []
)

( _inpterm1,
    []
    |
    []
)
( passer,
    []
    |
    []
    )

( mem,
    []
    |
    proc memory :: A | Mem(A|) => =
```

```
        1 | mem => -> do
            hcase mem of
                MemPut -> do
                    get y on mem
                    memory(y | mem => )
                MemGet -> do
                    put 1 on mem
                    memory(1 | mem => )
                MemCls -> do
                    halt mem
)

( nmpp
    []
    |
    proc p2_bot :: | Passer(| Mem(Int|)) => IntTerminal, Mem(Int|) =
        | passer,nmpp  => _inpterm1, mem -> do
            split nmpp into nm,pp
)

( nm
    []
    |
    []
)

( pp
    []
    |
    []
)

( z
    []
    |
    []
)

-- Global translation
_inpterm0 ---> _inpterm0
_inpterm1 ---> _inpterm1
passer    ---> passer
mem       ---> mem
mm        ---> mem
nmpp      ---> nmpp
nm        ---> nm
```

```
pp        ---> pp

-- Running processes:
proc p2 :: | Passer(| Mem(Int|)) => IntTerminal, Mem(Int|) =
    | pp => _inpterm1, z -> do
        hcase pp of
            Passer -> do
                hput MemGet on z
                get y on z
                hput IntTerminalPut on _inpterm1
                put y on _inpterm1
                hput IntTerminalGet on _inpterm1
                get x on _inpterm1
                hput MemPut on z
                put x on z
                fork pp as
                    mm with z -> do
                        mm |=| z
                    nmpp with _inpterm1 -> do
                        split nmpp into nm,pp
                        plug
                            p2( | pp => _inpterm1,z)
                            z,nm => -> z |=| neg nm

proc p2_bot_bot =
    z,nm => -> z |=| neg nm

proc p1 :: | => Passer(|Mem(Int|)), IntTerminal =
    | => passer, _inpterm0, mem, nmpp -> do
        get 1 on mem
        hput IntTerminalPut on _inpterm0
        put 1 on _inpterm0
        hput IntTerminalGet on _inpterm0
        get x on _inpterm0
        hput MemPut on mem
        put x on mem
        fork nmpp as
            nm with mem -> mem |=| neg nm
            pp with _inpterm0 -> p1(| => pp, _inpterm0)
```

4 boring p1| service step

Step p1 (boring service step).

```
-- Channel manager
( _inpterm0,
```

```
    proc p1 :: | => Passer(|Mem(Int|)), IntTerminal =
        | => passer, _inpterm0, mem, nmpp -> do
            hput IntTerminalPut on _inpterm0
    |
    []
)

( _inpterm1,
    []
    |
    []
)
( passer,
    []
    |
    []
    )

( mem,
    []
    |
    proc memory :: A | Mem(A|) => =
        1 | mem => -> do
            hcase mem of
                MemPut -> do
                    get y on mem
                    memory(y | mem => )
                MemGet -> do
                    put 1 on mem
                    memory(1 | mem => )
                MemCls -> do
                    halt mem
)

( nmpp
    []
    |
    proc p2_bot :: | Passer(| Mem(Int|)) => IntTerminal, Mem(Int|) =
        | passer,nmpp  => _inpterm1, mem -> do
            split nmpp into nm,pp
)

( nm
    []
    |
    []
```

```
)

( pp
    []
    |
    []
)

( z
    []
    |
    []
)

-- Global translation
_inpterm0 ---> _inpterm0
_inpterm1 ---> _inpterm1
passer    ---> passer
mem       ---> mem
mm        ---> mem
nmpp      ---> nmpp
nm        ---> nm
pp        ---> pp

-- Running processes:
proc p2 :: | Passer(| Mem(Int|)) => IntTerminal, Mem(Int|) =
    | pp => _inpterm1, z -> do
        hcase pp of
            Passer -> do
                hput MemGet on z
                get y on z
                hput IntTerminalPut on _inpterm1
                put y on _inpterm1
                hput IntTerminalGet on _inpterm1
                get x on _inpterm1
                hput MemPut on z
                put x on z
                fork pp as
                    mm with z -> do
                        mm |=| z
                    nmpp with _inpterm1 -> do
                        split nmpp into nm,pp
                        plug
                            p2( | pp => _inpterm1,z)
                            z,nm => -> z |=| neg nm
```

```
proc p2_bot_bot =
    z,nm => -> z |=| neg nm

proc p1 :: | => Passer(|Mem(Int|)), IntTerminal =
    | => passer, _inpterm0, mem, nmpp -> do
        put 1 on _inpterm0
        hput IntTerminalGet on _inpterm0
        get x on _inpterm0
        hput MemPut on mem
        put x on mem
        fork nmpp as
            nm with mem -> mem |=| neg nm
            pp with _inpterm0 -> p1(| => pp, _inpterm0)
```

Step `p1` (boring service step).

```
-- Channel manager
( _inpterm0,
    proc p1 :: | => Passer(|Mem(Int|)), IntTerminal =
        | => passer, _inpterm0, mem, nmpp -> do
            put 1 on _inpterm0
    ;
    proc p1 :: | => Passer(|Mem(Int|)), IntTerminal =
        | => passer, _inpterm0, mem, nmpp -> do
            hput IntTerminalPut on _inpterm0
    |
    []
)

( _inpterm1,
    []
    |
    []
)
( passer,
    []
    |
    []
    )

( mem,
    []
    |
    proc memory :: A | Mem(A|) => =
        1 | mem => -> do
            hcase mem of
                MemPut -> do
```

```
                    get y on mem
                    memory(y | mem => )
                MemGet -> do
                    put 1 on mem
                    memory(1 | mem => )
                MemCls -> do
                    halt mem
)

( nmpp
    []
    |
    proc p2_bot :: | Passer(| Mem(Int|)) => IntTerminal, Mem(Int|) =
        | passer,nmpp  => _inpterm1, mem -> do
            split nmpp into nm,pp
)

( nm
    []
    |
    []
)

( pp
    []
    |
    []
)

( z
    []
    |
    []
)

-- Global translation
_inpterm0 ---> _inpterm0
_inpterm1 ---> _inpterm1
passer    ---> passer
mem       ---> mem
mm        ---> mem
nmpp      ---> nmpp
nm        ---> nm
pp        ---> pp

-- Running processes:
```

```
proc p2 :: | Passer(| Mem(Int|)) => IntTerminal, Mem(Int|) =
    | pp => _inpterm1, z -> do
        hcase pp of
            Passer -> do
                hput MemGet on z
                get y on z
                hput IntTerminalPut on _inpterm1
                put y on _inpterm1
                hput IntTerminalGet on _inpterm1
                get x on _inpterm1
                hput MemPut on z
                put x on z
                fork pp as
                    mm with z -> do
                        mm |=| z
                    nmpp with _inpterm1 -> do
                        split nmpp into nm,pp
                        plug
                            p2( | pp => _inpterm1,z)
                            z,nm => -> z |=| neg nm

proc p2_bot_bot =
    z,nm => -> z |=| neg nm

proc p1 :: | => Passer(|Mem(Int|)), IntTerminal =
    | => passer, _inpterm0, mem, nmpp -> do
        hput IntTerminalGet on _inpterm0
        get x on _inpterm0
        hput MemPut on mem
        put x on mem
        fork nmpp as
            nm with mem -> mem |=| neg nm
            pp with _inpterm0 -> p1(| => pp, _inpterm0)
```

Step `p1` (boring service step).

```
-- Channel manager
( _inpterm0,
    proc p1 :: | => Passer(|Mem(Int|)), IntTerminal =
        | => passer, _inpterm0, mem, nmpp -> do
            hput IntTerminalGet on _inpterm0
    ;
    proc p1 :: | => Passer(|Mem(Int|)), IntTerminal =
        | => passer, _inpterm0, mem, nmpp -> do
            put 1 on _inpterm0
    ;
```

```
    proc p1 :: | => Passer(|Mem(Int|)), IntTerminal =
        | => passer, _inpterm0, mem, nmpp -> do
            hput IntTerminalPut on _inpterm0
    |
    []
)

( _inpterm1,
    []
    |
    []
)
( passer,
    []
    |
    []
    )

( mem,
    []
    |
    proc memory :: A | Mem(A|) => =
        1 | mem => -> do
            hcase mem of
                MemPut -> do
                    get y on mem
                    memory(y | mem => )
                MemGet -> do
                    put 1 on mem
                    memory(1 | mem => )
                MemCls -> do
                    halt mem
)

( nmpp
    []
    |
    proc p2_bot :: | Passer(| Mem(Int|)) => IntTerminal, Mem(Int|) =
        | passer,nmpp  => _inpterm1, mem -> do
            split nmpp into nm,pp
)

( nm
    []
    |
    []
```

```
)

( pp
    []
    |
    []
)

( z
    []
    |
    []
)

-- Global translation
_inpterm0 ---> _inpterm0
_inpterm1 ---> _inpterm1
passer    ---> passer
mem       ---> mem
mm        ---> mem
nmpp      ---> nmpp
nm        ---> nm
pp        ---> pp

-- Running processes:
proc p2 :: | Passer(| Mem(Int|)) => IntTerminal, Mem(Int|) =
    | pp => _inpterm1, z -> do
        hcase pp of
            Passer -> do
                hput MemGet on z
                get y on z
                hput IntTerminalPut on _inpterm1
                put y on _inpterm1
                hput IntTerminalGet on _inpterm1
                get x on _inpterm1
                hput MemPut on z
                put x on z
                fork pp as
                    mm with z -> do
                        mm |=| z
                    nmpp with _inpterm1 -> do
                        split nmpp into nm,pp
                        plug
                            p2( | pp => _inpterm1,z)
                            z,nm => -> z |=| neg nm
```

```
proc p2_bot_bot =
    z,nm => -> z |=| neg nm

proc p1 :: | => Passer(|Mem(Int|)), IntTerminal =
    | => passer, _inpterm0, mem, nmpp -> do
        get x on _inpterm0
        hput MemPut on mem
        put x on mem
        fork nmpp as
            nm with mem -> mem |=| neg nm
            pp with _inpterm0 -> p1(| => pp, _inpterm0)
```

Step `p1` (boring service step).

```
-- Channel manager
( _inpterm0,
    proc p1 :: | => Passer(|Mem(Int|)), IntTerminal =
        | => passer, _inpterm0, mem, nmpp -> do
            get x on _inpterm0
            hput MemPut on mem
            put x on mem
            fork nmpp as
                nm with mem -> mem |=| neg nm
                pp with _inpterm0 -> p1(| => pp, _inpterm0)
    ;
    proc p1 :: | => Passer(|Mem(Int|)), IntTerminal =
        | => passer, _inpterm0, mem, nmpp -> do
            hput IntTerminalGet on _inpterm0
    ;
    proc p1 :: | => Passer(|Mem(Int|)), IntTerminal =
        | => passer, _inpterm0, mem, nmpp -> do
            put 1 on _inpterm0
    ;
    proc p1 :: | => Passer(|Mem(Int|)), IntTerminal =
        | => passer, _inpterm0, mem, nmpp -> do
            hput IntTerminalPut on _inpterm0
    |
    []
)

( _inpterm1,
    []
    |
    []
)
( passer,
```

```
    []
    |
    []
    )

( mem,
    []
    |
    proc memory :: A | Mem(A|) => =
        1 | mem => -> do
            hcase mem of
                MemPut -> do
                    get y on mem
                    memory(y | mem => )
                MemGet -> do
                    put 1 on mem
                    memory(1 | mem => )
                MemCls -> do
                    halt mem
)

( nmpp
    []
    |
    proc p2_bot :: | Passer(| Mem(Int|)) => IntTerminal, Mem(Int|) =
        | passer,nmpp  => _inpterm1, mem -> do
            split nmpp into nm,pp
)

( nm
    []
    |
    []
)

( pp
    []
    |
    []
)

( z
    []
    |
    []
)
```

```
-- Global translation
_inpterm0 ---> _inpterm0
_inpterm1 ---> _inpterm1
passer    ---> passer
mem       ---> mem
mm        ---> mem
nmpp      ---> nmpp
nm        ---> nm
pp        ---> pp

-- Running processes:
proc p2 :: | Passer(| Mem(Int|)) => IntTerminal, Mem(Int|) =
    | pp => _inpterm1, z -> do
        hcase pp of
            Passer -> do
                hput MemGet on z
                get y on z
                hput IntTerminalPut on _inpterm1
                put y on _inpterm1
                hput IntTerminalGet on _inpterm1
                get x on _inpterm1
                hput MemPut on z
                put x on z
                fork pp as
                    mm with z -> do
                        mm |=| z
                    nmpp with _inpterm1 -> do
                        split nmpp into nm,pp
                        plug
                            p2( | pp => _inpterm1,z)
                            z,nm => -> z |=| neg nm

proc p2_bot_bot =
    z,nm => -> z |=| neg nm
```

Again, doesn't really matter what the service does, we just assume that `2` is inputted, so eventually `p1` may resume and we substitue the arguments as needed

```
-- Channel manager
( _inpterm0,
    []
    |
    []
)
```

```
( _inpterm1,
    []
    |
    []
)
( passer,
    []
    |
    []
    )

( mem,
    []
    |
    proc memory :: A | Mem(A|) => =
        1 | mem => -> do
            hcase mem of
                MemPut -> do
                    get y on mem
                    memory(y | mem => )
                MemGet -> do
                    put 1 on mem
                    memory(1 | mem => )
                MemCls -> do
                    halt mem
)

( nmpp
    []
    |
    proc p2_bot :: | Passer(| Mem(Int|)) => IntTerminal, Mem(Int|) =
        | passer,nmpp  => _inpterm1, mem -> do
            split nmpp into nm,pp
)

( nm
    []
    |
    []
)

( pp
    []
    |
    []
```

```
)

( z
    []
    |
    []
)

-- Global translation
_inpterm0 ---> _inpterm0
_inpterm1 ---> _inpterm1
passer    ---> passer
mem       ---> mem
mm        ---> mem
nmpp      ---> nmpp
nm        ---> nm
pp        ---> pp

-- Running processes:
proc p2 :: | Passer(| Mem(Int|)) => IntTerminal, Mem(Int|) =
    | pp => _inpterm1, z -> do
        hcase pp of
            Passer -> do
                hput MemGet on z
                get y on z
                hput IntTerminalPut on _inpterm1
                put y on _inpterm1
                hput IntTerminalGet on _inpterm1
                get x on _inpterm1
                hput MemPut on z
                put x on z
                fork pp as
                    mm with z -> do
                        mm |=| z
                    nmpp with _inpterm1 -> do
                        split nmpp into nm,pp
                        plug
                            p2( | pp => _inpterm1,z)
                            z,nm => -> z |=| neg nm

proc p2_bot_bot =
    z,nm => -> z |=| neg nm

proc p1 :: | => Passer(|Mem(Int|)), IntTerminal =
    | => passer, _inpterm0, mem, nmpp -> do
        hput MemPut on mem
```

```
        put 2 on mem
        fork nmpp as
            nm with mem -> mem |=| neg nm
            pp with _inpterm0 -> p1(| => pp, _inpterm0)
```

5 endof boring service step

So, we execute `p1` again.

```
-- Channel manager
( _inpterm0,
    []
    |
    []
)

( _inpterm1,
    []
    |
    []
)
( passer,
    []
    |
    []
    )

( mem,
    proc p1 :: | => Passer(|Mem(Int|)), IntTerminal =
        | => passer, _inpterm0, mem, nmpp -> do
            hput MemPut on mem
    |
    proc memory :: A | Mem(A|) => =
        1 | mem => -> do
            hcase mem of
                MemPut -> do
                    get y on mem
                    memory(y | mem => )
                MemGet -> do
                    put 1 on mem
                    memory(1 | mem => )
                MemCls -> do
                    halt mem
)
```

```
( nmpp
    []
    |
    proc p2_bot :: | Passer(| Mem(Int|)) => IntTerminal, Mem(Int|) =
        | passer,nmpp  => _inpterm1, mem -> do
            split nmpp into nm,pp
)

( nm
    []
    |
    []
)

( pp
    []
    |
    []
)

( z
    []
    |
    []
)

-- Global translation
_inpterm0 ---> _inpterm0
_inpterm1 ---> _inpterm1
passer    ---> passer
mem       ---> mem
mm        ---> mem
nmpp      ---> nmpp
nm        ---> nm
pp        ---> pp

-- Running processes:
proc p2 :: | Passer(| Mem(Int|)) => IntTerminal, Mem(Int|) =
    | pp => _inpterm1, z -> do
        hcase pp of
            Passer -> do
                hput MemGet on z
                get y on z
                hput IntTerminalPut on _inpterm1
                put y on _inpterm1
```

```
                hput IntTerminalGet on _inpterm1
                get x on _inpterm1
                hput MemPut on z
                put x on z
                fork pp as
                    mm with z -> do
                        mm |=| z
                    nmpp with _inpterm1 -> do
                        split nmpp into nm,pp
                        plug
                            p2( | pp => _inpterm1,z)
                            z,nm => -> z |=| neg nm

proc p2_bot_bot =
    z,nm => -> z |=| neg nm

proc p1 :: | => Passer(|Mem(Int|)), IntTerminal =
    | => passer, _inpterm0, mem, nmpp -> do
        put 2 on mem
        fork nmpp as
            nm with mem -> mem |=| neg nm
            pp with _inpterm0 -> p1(| => pp, _inpterm0)
```

So, we execute `p1` again.

```
-- Channel manager
( _inpterm0,
    []
    |
    []
)

( _inpterm1,
    []
    |
    []
)
( passer,
    []
    |
    []
    )

( mem,
    proc p1 :: | => Passer(|Mem(Int|)), IntTerminal =
        | => passer, _inpterm0, mem, nmpp -> do
```

```
            put 2 on mem
    ;
    proc p1 :: | => Passer(|Mem(Int|)), IntTerminal =
        | => passer, _inpterm0, mem, nmpp -> do
            hput MemPut on mem
    |
    proc memory :: A | Mem(A|) => =
        1 | mem => -> do
            hcase mem of
                MemPut -> do
                    get y on mem
                    memory(y | mem => )
                MemGet -> do
                    put 1 on mem
                    memory(1 | mem => )
                MemCls -> do
                    halt mem
)

( nmpp
    []
    |
    proc p2_bot :: | Passer(| Mem(Int|)) => IntTerminal, Mem(Int|) =
        | passer,nmpp  => _inpterm1, mem -> do
            split nmpp into nm,pp
)

( nm
    []
    |
    []
)

( pp
    []
    |
    []
)

( z
    []
    |
    []
)

-- Global translation
```

```
_inpterm0 ---> _inpterm0
_inpterm1 ---> _inpterm1
passer    ---> passer
mem       ---> mem
mm        ---> mem
nmpp      ---> nmpp
nm        ---> nm
pp        ---> pp

-- Running processes:
proc p2 :: | Passer(| Mem(Int|)) => IntTerminal, Mem(Int|) =
    | pp => _inpterm1, z -> do
        hcase pp of
            Passer -> do
                hput MemGet on z
                get y on z
                hput IntTerminalPut on _inpterm1
                put y on _inpterm1
                hput IntTerminalGet on _inpterm1
                get x on _inpterm1
                hput MemPut on z
                put x on z
                fork pp as
                    mm with z -> do
                        mm |=| z
                    nmpp with _inpterm1 -> do
                        split nmpp into nm,pp
                        plug
                            p2( | pp => _inpterm1,z)
                            z,nm => -> z |=| neg nm

proc p2_bot_bot =
    z,nm => -> z |=| neg nm

proc p1 :: | => Passer(|Mem(Int|)), IntTerminal =
    | => passer, _inpterm0, mem, nmpp -> do
        fork nmpp as
            nm with mem -> mem |=| neg nm
            pp with _inpterm0 -> p1(| => pp, _inpterm0)
```

So, we execute `p1` again where it suspends since it uses the "fork" instruction

```
-- Channel manager
( _inpterm0,
    []
    |
```

```
    []
)

( _inpterm1,
    []
    |
    []
)
( passer,
    []
    |
    []
    )

( mem,
    proc p1 :: | => Passer(|Mem(Int|)), IntTerminal =
        | => passer, _inpterm0, mem, nmpp -> do
            put 2 on mem
    ;
    proc p1 :: | => Passer(|Mem(Int|)), IntTerminal =
        | => passer, _inpterm0, mem, nmpp -> do
            hput MemPut on mem
    |
    proc memory :: A | Mem(A|) => =
        1 | mem => -> do
            hcase mem of
                MemPut -> do
                    get y on mem
                    memory(y | mem => )
                MemGet -> do
                    put 1 on mem
                    memory(1 | mem => )
                MemCls -> do
                    halt mem
)

( nmpp
    proc p1 :: | => Passer(|Mem(Int|)), IntTerminal =
        | => passer, _inpterm0, mem, nmpp -> do
            fork nmpp as
                nm with mem -> mem |=| neg nm
                pp with _inpterm0 -> p1(| => pp, _inpterm0)
    |
    proc p2_bot :: | Passer(| Mem(Int|)) => IntTerminal, Mem(Int|) =
        | passer,nmpp  => _inpterm1, mem -> do
            split nmpp into nm,pp
```

```
)

( nm
    []
    |
    []
)

( pp
    []
    |
    []
)

( z
    []
    |
    []
)

-- Global translation
_inpterm0 ---> _inpterm0
_inpterm1 ---> _inpterm1
passer    ---> passer
mem       ---> mem
mm        ---> mem
nmpp      ---> nmpp
nm        ---> nm
pp        ---> pp

-- Running processes:
proc p2 :: | Passer(| Mem(Int|)) => IntTerminal, Mem(Int|) =
    | pp => _inpterm1, z -> do
        hcase pp of
            Passer -> do
                hput MemGet on z
                get y on z
                hput IntTerminalPut on _inpterm1
                put y on _inpterm1
                hput IntTerminalGet on _inpterm1
                get x on _inpterm1
                hput MemPut on z
                put x on z
                fork pp as
                    mm with z -> do
                        mm |=| z
```

```
                    nmpp with _inpterm1 -> do
                        split nmpp into nm,pp
                        plug
                            p2( | pp => _inpterm1,z)
                            z,nm => -> z |=| neg nm

proc p2_bot_bot =
    z,nm => -> z |=| neg nm
```

Now, looking at the channel `mem`, execute that so the process `memory` may continue.

```
-- Channel manager
( _inpterm0,
    []
    |
    []
)

( _inpterm1,
    []
    |
    []
)
( passer,
    []
    |
    []
    )

( mem,
    proc p1 :: | => Passer(|Mem(Int|)), IntTerminal =
        | => passer, _inpterm0, mem, nmpp -> do
            put 2 on mem
    |
    []
)

( nmpp
    proc p1 :: | => Passer(|Mem(Int|)), IntTerminal =
        | => passer, _inpterm0, mem, nmpp -> do
            fork nmpp as
                nm with mem -> mem |=| neg nm
                pp with _inpterm0 -> p1(| => pp, _inpterm0)
    |
    proc p2_bot :: | Passer(| Mem(Int|)) => IntTerminal, Mem(Int|) =
```

```
        | passer,nmpp  => _inpterm1, mem -> do
            split nmpp into nm,pp
)

( nm
    []
    |
    []
)

( pp
    []
    |
    []
)

( z
    []
    |
    []
)

-- Global translation
_inpterm0 ---> _inpterm0
_inpterm1 ---> _inpterm1
passer    ---> passer
mem       ---> mem
mm        ---> mem
nmpp      ---> nmpp
nm        ---> nm
pp        ---> pp

-- Running processes:
proc p2 :: | Passer(| Mem(Int|)) => IntTerminal, Mem(Int|) =
    | pp => _inpterm1, z -> do
        hcase pp of
            Passer -> do
                hput MemGet on z
                get y on z
                hput IntTerminalPut on _inpterm1
                put y on _inpterm1
                hput IntTerminalGet on _inpterm1
                get x on _inpterm1
                hput MemPut on z
                put x on z
                fork pp as
```

```
                    mm with z -> do
                        mm |=| z
                    nmpp with _inpterm1 -> do
                        split nmpp into nm,pp
                        plug
                            p2( | pp => _inpterm1,z)
                            z,nm => -> z |=| neg nm

proc p2_bot_bot =
    z,nm => -> z |=| neg nm

proc memory :: A | Mem(A|) => =
    1 | mem => -> do
        get y on mem
        memory(y | mem => )
```

Now, step `memory` again.

```
-- Channel manager
( _inpterm0,
    []
    |
    []
)

( _inpterm1,
    []
    |
    []
)
( passer,
    []
    |
    []
    )

( mem,
    proc p1 :: | => Passer(|Mem(Int|)), IntTerminal =
        | => passer, _inpterm0, mem, nmpp -> do
            put 2 on mem
    |
    proc memory :: A | Mem(A|) => =
        1 | mem => -> do
            get y on mem
            memory(y | mem => )
)
```

```
( nmpp
    proc p1 :: | => Passer(|Mem(Int|)), IntTerminal =
        | => passer, _inpterm0, mem, nmpp -> do
            fork nmpp as
                nm with mem -> mem |=| neg nm
                pp with _inpterm0 -> p1(| => pp, _inpterm0)
    |
    proc p2_bot :: | Passer(| Mem(Int|)) => IntTerminal, Mem(Int|) =
        | passer,nmpp  => _inpterm1, mem -> do
            split nmpp into nm,pp
)

( nm
    []
    |
    []
)

( pp
    []
    |
    []
)

( z
    []
    |
    []
)

-- Global translation
_inpterm0 ---> _inpterm0
_inpterm1 ---> _inpterm1
passer    ---> passer
mem       ---> mem
mm        ---> mem
nmpp      ---> nmpp
nm        ---> nm
pp        ---> pp

-- Running processes:
proc p2 :: | Passer(| Mem(Int|)) => IntTerminal, Mem(Int|) =
    | pp => _inpterm1, z -> do
        hcase pp of
            Passer -> do
                hput MemGet on z
```

```
                get y on z
                hput IntTerminalPut on _inpterm1
                put y on _inpterm1
                hput IntTerminalGet on _inpterm1
                get x on _inpterm1
                hput MemPut on z
                put x on z
                fork pp as
                    mm with z -> do
                        mm |=| z
                    nmpp with _inpterm1 -> do
                        split nmpp into nm,pp
                        plug
                            p2( | pp => _inpterm1,z)
                            z,nm => -> z |=| neg nm

proc p2_bot_bot =
    z,nm => -> z |=| neg nm
```

Do a step on the channel `mem`, so `memory` may resume.

```
-- Channel manager
( _inpterm0,
    []
    |
    []
)

( _inpterm1,
    []
    |
    []
)
( passer,
    []
    |
    []
    )

( mem,
    []
    |
    []
)

( nmpp
```

```
    proc p1 :: | => Passer(|Mem(Int|)), IntTerminal =
        | => passer, _inpterm0, mem, nmpp -> do
            fork nmpp as
                nm with mem -> mem |=| neg nm
                pp with _inpterm0 -> p1(| => pp, _inpterm0)
    |
    proc p2_bot :: | Passer(| Mem(Int|)) => IntTerminal, Mem(Int|) =
        | passer,nmpp  => _inpterm1, mem -> do
            split nmpp into nm,pp
)

( nm
    []
    |
    []
)

( pp
    []
    |
    []
)

( z
    []
    |
    []
)

-- Global translation
_inpterm0 ---> _inpterm0
_inpterm1 ---> _inpterm1
passer    ---> passer
mem       ---> mem
mm        ---> mem
nmpp      ---> nmpp
nm        ---> nm
pp        ---> pp

-- Running processes:
proc p2 :: | Passer(| Mem(Int|)) => IntTerminal, Mem(Int|) =
    | pp => _inpterm1, z -> do
        hcase pp of
            Passer -> do
                hput MemGet on z
                get y on z
```

```
                hput IntTerminalPut on _inpterm1
                put y on _inpterm1
                hput IntTerminalGet on _inpterm1
                get x on _inpterm1
                hput MemPut on z
                put x on z
                fork pp as
                    mm with z -> do
                        mm |=| z
                    nmpp with _inpterm1 -> do
                        split nmpp into nm,pp
                        plug
                            p2( | pp => _inpterm1,z)
                            z,nm => -> z |=| neg nm

proc p2_bot_bot =
    z,nm => -> z |=| neg nm

proc memory :: A | Mem(A|) => =
    1 | mem => -> do
        memory(2 | mem => )
```

Step the process `memory` again, substituting arguments as needed.

```
-- Channel manager
( _inpterm0,
    []
    |
    []
)

( _inpterm1,
    []
    |
    []
)
( passer,
    []
    |
    []
    )

( mem,
    []
    |
    []
```

```
)

( nmpp
    proc p1 :: | => Passer(|Mem(Int|)), IntTerminal =
        | => passer, _inpterm0, mem, nmpp -> do
            fork nmpp as
                nm with mem -> mem |=| neg nm
                pp with _inpterm0 -> p1(| => pp, _inpterm0)
    |
    proc p2_bot :: | Passer(| Mem(Int|)) => IntTerminal, Mem(Int|) =
        | passer,nmpp  => _inpterm1, mem -> do
            split nmpp into nm,pp
)

( nm
    []
    |
    []
)

( pp
    []
    |
    []
)

( z
    []
    |
    []
)

-- Global translation
_inpterm0 ---> _inpterm0
_inpterm1 ---> _inpterm1
passer    ---> passer
mem       ---> mem
mm        ---> mem
nmpp      ---> nmpp
nm        ---> nm
pp        ---> pp

-- Running processes:
proc p2 :: | Passer(| Mem(Int|)) => IntTerminal, Mem(Int|) =
    | pp => _inpterm1, z -> do
        hcase pp of
```

```
            Passer -> do
                hput MemGet on z
                get y on z
                hput IntTerminalPut on _inpterm1
                put y on _inpterm1
                hput IntTerminalGet on _inpterm1
                get x on _inpterm1
                hput MemPut on z
                put x on z
                fork pp as
                    mm with z -> do
                        mm |=| z
                    nmpp with _inpterm1 -> do
                        split nmpp into nm,pp
                        plug
                            p2( | pp => _inpterm1,z)
                            z,nm => -> z |=| neg nm

proc p2_bot_bot =
    z,nm => -> z |=| neg nm

proc memory :: A | Mem(A|) => =
    2 | mem => -> do
        hcase mem of
            MemPut -> do
                get y on mem
                memory(y | mem => )
            MemGet -> do
                put 2 on mem
                memory(2 | mem => )
            MemCls -> do
                halt mem
```

Step the process `memory` again, where it suspends itself on `mem`.

```
-- Channel manager
( _inpterm0,
    []
    |
    []
)

( _inpterm1,
    []
    |
    []
```

```
)
( passer,
    []
    |
    []
    )

( mem,
    []
    |
    proc memory :: A | Mem(A|) => =
        2 | mem => -> do
            hcase mem of
                MemPut -> do
                    get y on mem
                    memory(y | mem => )
                MemGet -> do
                    put 2 on mem
                    memory(2 | mem => )
                MemCls -> do
                    halt mem
)

( nmpp
    proc p1 :: | => Passer(|Mem(Int|)), IntTerminal =
        | => passer, _inpterm0, mem, nmpp -> do
            fork nmpp as
                nm with mem -> mem |=| neg nm
                pp with _inpterm0 -> p1(| => pp, _inpterm0)
    |
    proc p2_bot :: | Passer(| Mem(Int|)) => IntTerminal, Mem(Int|) =
        | passer,nmpp  => _inpterm1, mem -> do
            split nmpp into nm,pp
)

( nm
    []
    |
    []
)

( pp
    []
    |
    []
)
```

```
( z
    []
    |
    []
)

-- Global translation
_inpterm0 ---> _inpterm0
_inpterm1 ---> _inpterm1
passer    ---> passer
mem       ---> mem
mm        ---> mem
nmpp      ---> nmpp
nm        ---> nm
pp        ---> pp

-- Running processes:
proc p2 :: | Passer(| Mem(Int|)) => IntTerminal, Mem(Int|) =
    | pp => _inpterm1, z -> do
        hcase pp of
            Passer -> do
                hput MemGet on z
                get y on z
                hput IntTerminalPut on _inpterm1
                put y on _inpterm1
                hput IntTerminalGet on _inpterm1
                get x on _inpterm1
                hput MemPut on z
                put x on z
                fork pp as
                    mm with z -> do
                        mm |=| z
                    nmpp with _inpterm1 -> do
                        split nmpp into nm,pp
                        plug
                            p2( | pp => _inpterm1,z)
                            z,nm => -> z |=| neg nm

proc p2_bot_bot =
    z,nm => -> z |=| neg nm
```

Now, looking at the channel `nmpp`, we may execute a step there. Note we name the top fork phrase `p1_top` and the bottom `p1_bot`.

```
-- Channel manager
```

```
( _inpterm0,
    []
    |
    []
)

( _inpterm1,
    []
    |
    []
)
( passer,
    []
    |
    []
    )

( mem,
    []
    |
    proc memory :: A | Mem(A|) => =
        2 | mem => -> do
            hcase mem of
                MemPut -> do
                    get y on mem
                    memory(y | mem => )
                MemGet -> do
                    put 2 on mem
                    memory(2 | mem => )
                MemCls -> do
                    halt mem
)

( nmpp
    []
    |
    []
)

( nm
    []
    |
    []
)

( pp
```

```
    []
    |
    []
)

( z
    []
    |
    []
)

-- Global translation
_inpterm0 ---> _inpterm0
_inpterm1 ---> _inpterm1
passer    ---> passer
mem       ---> mem
mm        ---> mem
nmpp      ---> nmpp
nm        ---> nm
pp        ---> pp

-- Running processes:
proc p2 :: | Passer(| Mem(Int|)) => IntTerminal, Mem(Int|) =
    | pp => _inpterm1, z -> do
        hcase pp of
            Passer -> do
                hput MemGet on z
                get y on z
                hput IntTerminalPut on _inpterm1
                put y on _inpterm1
                hput IntTerminalGet on _inpterm1
                get x on _inpterm1
                hput MemPut on z
                put x on z
                fork pp as
                    mm with z -> do
                        mm |=| z
                    nmpp with _inpterm1 -> do
                        split nmpp into nm,pp
                        plug
                            p2( | pp => _inpterm1,z)
                            z,nm => -> z |=| neg nm

proc p2_bot_bot =
    z,nm => -> z |=| neg nm
```

```
proc p1_top :: | => Passer(|Mem(Int|)), IntTerminal =
    | => passer, _inpterm0, mem, nmpp, nm -> do
        mem |=| neg nm

proc p1_bot :: | => Passer(|Mem(Int|)), IntTerminal =
    | => passer, _inpterm0, mem, nmpp, nm,pp -> do
        p1(| => pp, _inpterm0)
```

Now, we step `p1_top` – recalling that "neg" is the same as id. **resuming of problem number 1 – the first negation that is executed** (observe that in strictly following the rules, the input of `mem` should be empty for the "neg" on `nm` to execute, but it will never be empty.)

```
-- Channel manager
( _inpterm0,
    []
    |
    []
)

( _inpterm1,
    []
    |
    []
)
( passer,
    []
    |
    []
    )

( mem,
    proc p1_top :: | => Passer(|Mem(Int|)), IntTerminal =
        | => passer, _inpterm0, mem, nmpp, nm -> do
            mem |=| neg nm
    |
    proc memory :: A | Mem(A|) => =
        2 | mem => -> do
            hcase mem of
                MemPut -> do
                    get y on mem
                    memory(y | mem => )
                MemGet -> do
                    put 2 on mem
                    memory(2 | mem => )
                MemCls -> do
```

```
                    halt mem
)

( nmpp
    []
    |
    []
)

( nm
    []
    |
    []
)

( pp
    []
    |
    []
)

( z
    []
    |
    []
)

-- Global translation
_inpterm0 ---> _inpterm0
_inpterm1 ---> _inpterm1
passer    ---> passer
mem       ---> mem
mm        ---> mem
nmpp      ---> nmpp
nm        ---> nm
pp        ---> pp

-- Running processes:
proc p2 :: | Passer(| Mem(Int|)) => IntTerminal, Mem(Int|) =
    | pp => _inpterm1, z -> do
        hcase pp of
            Passer -> do
                hput MemGet on z
                get y on z
                hput IntTerminalPut on _inpterm1
                put y on _inpterm1
```

```
                hput IntTerminalGet on _inpterm1
                get x on _inpterm1
                hput MemPut on z
                put x on z
                fork pp as
                    mm with z -> do
                        mm |=| z
                    nmpp with _inpterm1 -> do
                        split nmpp into nm,pp
                        plug
                            p2( | pp => _inpterm1,z)
                            z,nm => -> z |=| neg nm

proc p2_bot_bot =
    z,nm => -> z |=| neg nm

proc p1_bot :: | => Passer(|Mem(Int|)), IntTerminal =
    | => passer, _inpterm0, mem, nmpp, nm,pp -> do
        p1(| => pp, _inpterm0)
```

Step p2_bot_bot.

```
-- Channel manager
( _inpterm0,
    []
    |
    []
)

( _inpterm1,
    []
    |
    []
)
( passer,
    []
    |
    []
    )

( mem,
    proc p1_top :: | => Passer(|Mem(Int|)), IntTerminal =
        | => passer, _inpterm0, mem, nmpp, nm -> do
            mem |=| neg nm
    |
    proc memory :: A | Mem(A|) => =
        2 | mem => -> do
```

```
            hcase mem of
                MemPut -> do
                    get y on mem
                    memory(y | mem => )
                MemGet -> do
                    put 2 on mem
                    memory(2 | mem => )
                MemCls -> do
                    halt mem
)

( nmpp
    []
    |
    []
)

( nm
    []
    |
    []
)

( pp
    []
    |
    []
)

( z
    []
    |
    proc p2_bot_bot =
        z,nm => -> z |=| neg nm
)

-- Global translation
_inpterm0 ---> _inpterm0
_inpterm1 ---> _inpterm1
passer    ---> passer
mem       ---> mem
mm        ---> mem
nmpp      ---> nmpp
nm        ---> nm
pp        ---> pp
```

```
-- Running processes:
proc p2 :: | Passer(| Mem(Int|)) => IntTerminal, Mem(Int|) =
    | pp => _inpterm1, z -> do
        hcase pp of
            Passer -> do
                hput MemGet on z
                get y on z
                hput IntTerminalPut on _inpterm1
                put y on _inpterm1
                hput IntTerminalGet on _inpterm1
                get x on _inpterm1
                hput MemPut on z
                put x on z
                fork pp as
                    mm with z -> do
                        mm |=| z
                    nmpp with _inpterm1 -> do
                        split nmpp into nm,pp
                        plug
                            p2( | pp => _inpterm1,z)
                            z,nm => -> z |=| neg nm

proc p1_bot :: | => Passer(|Mem(Int|)), IntTerminal =
    | => passer, _inpterm0, mem, nmpp, nm,pp -> do
        p1(| => pp, _inpterm0)
```

This clearly (probably) works so much truth in this, frankly haven't seen so much truth in my entire life.