--

-- RoboChart generator version 3.0.0.202401231330

-- Automatically generated on 28-06-2024 12:30:29

--

-- Iterated compression status: true

-- Assertions compression status: false

--

# module pkg0\_M

exports

transparent diamond

transparent sbisim

transparent dbisim

transparent chase

-- declaring channels

-- declaring robotic platform events

channel move1Call: core\_real.core\_int

channel move2Call: core\_real.core\_int

-- declaring module termination channel

channel terminate

-- declaring robotic platform variables

channel get\_pv1, set\_pv1: core\_int

channel get\_a1, set\_a1: {(x0\_\_,x1\_\_) | x0\_\_ <- core\_int,x1\_\_ <- core\_int}

channel get\_a3, set\_a3: core\_int

channel get\_m, set\_m: core\_int

-- channel set with all visible events

sem\_\_events = {|

move1Call,

move2Call

|}

-- declaring controller

## module C1

shared\_variable\_events = {|

set\_EXT\_m,

set\_EXT\_a1,

set\_EXT\_a3

|}

exports

transparent diamond

transparent sbisim

transparent dbisim

transparent chase

-- declaring controller events

channel set\_cv1: core\_int

channel get\_cv1: core\_int

channel set\_m: core\_int

channel get\_m: core\_int

channel set\_a1: {(x0\_\_,x1\_\_) | x0\_\_ <- core\_int,x1\_\_ <- core\_int}

channel get\_a1: {(x0\_\_,x1\_\_) | x0\_\_ <- core\_int,x1\_\_ <- core\_int}

channel set\_a3: core\_int

channel get\_a3: core\_int

channel set\_EXT\_m: core\_int

channel set\_EXT\_a1: {(x0\_\_,x1\_\_) | x0\_\_ <- core\_int,x1\_\_ <- core\_int}

channel set\_EXT\_a3: core\_int

-- declaring call and ret events for undefined operations

channel move1Call: core\_real.core\_int

channel move2Call: core\_real.core\_int

-- declaring controller termination channel

channel terminate

-- channel set with all visible events

sem\_\_events = {|

terminate

, set\_EXT\_m, set\_m,

set\_EXT\_a1, set\_a1,

set\_EXT\_a3, set\_a3

, move1Call,

move2Call

|}

-- defined operations

module OP\_move

exports

datatype TIDS = NULLTRANSITION\_\_

-- These definitions are needed to make the structure of an incompletely defined operation compatible with that of fully defined operations.

MemoryTransitions(id\_\_) = STOP -- this is STOP because MemoryTransitions is put in a choice with the state machine's MemoryTransitions. if it were SKIP, a divergence would be introduced.

MemoryVariablesSyncSet = {}

MemoryVariablesHideSet = {}

MachineMemorySyncSet = {}

MachineMemoryHidingSet = {}

channel internal\_\_

D\_\_(id\_\_,

param\_lv,

param\_a) = let

AUX = SKIP

|~|

STOP

within

AUX

O\_\_(id\_\_,

param\_lv,

param\_a) = D\_\_(id\_\_,

param\_lv,

param\_a)

HUP\_\_(id\_\_,

param\_lv,

param\_a) = D\_\_(id\_\_,

param\_lv,

param\_a)

endmodule

-- declaring machines

## module stm0

exports

transparent diamond

transparent sbisim

transparent dbisim

transparent chase

-- Transition identifiers

-- declaring identifiers of transitions

datatype NIDS =

NID\_i0|

NID\_s0|

NID\_f0

channel internal\_\_ : NIDS

-- Flow channels

channel interrupt

channel exited

channel exit

channel terminate

-- Variable channels

channel get\_l, set\_l, setL\_l, setR\_l: core\_int

channel get\_a, set\_a, setL\_a, setR\_a: core\_int

channel get\_m, set\_m, setL\_m, setR\_m: core\_int

channel get\_a1, set\_a1, setL\_a1, setR\_a1: {(x0\_\_,x1\_\_) | x0\_\_ <- core\_int,x1\_\_ <- core\_int}

channel get\_a3, set\_a3, setL\_a3, setR\_a3: core\_int

-- Shared variable channels

channel set\_EXT\_m: core\_int

channel set\_EXT\_a1: {(x0\_\_,x1\_\_) | x0\_\_ <- core\_int,x1\_\_ <- core\_int}

channel set\_EXT\_a3: core\_int

-- Local variable channels for defined operations that are required by the state machine

-- Declaring state machine events

channel stop\_\_: NIDS.InOut

channel stop: InOut

channel event1\_\_: NIDS.InOut.{(x0\_\_,x1\_\_) | x0\_\_ <- core\_int,x1\_\_ <- core\_int}

channel event1: InOut.{(x0\_\_,x1\_\_) | x0\_\_ <- core\_int,x1\_\_ <- core\_int}

channel event2\_\_: NIDS.InOut.core\_int

channel event2: InOut.core\_int

channel trigger1\_\_: NIDS.InOut.core\_int

channel trigger1: InOut.core\_int

-- Declaring call and ret events for undefined operations

channel move1Call: core\_real.core\_int

channel move2Call: core\_real.core\_int

enterSS = {|

i0::enter,

s0::enter,

f0::enter

|}

enteredSS = {|

s0::entered,

f0::entered

|}

internal\_events = union(enteredSS,union(enterSS,{|interrupt,exited|}))

shared\_variable\_events = {|

set\_EXT\_m,

set\_EXT\_a1,

set\_EXT\_a3

|}

-- channel set with all visible events

sem\_\_events = {|

terminate

, set\_EXT\_m, set\_m,

set\_EXT\_a1, set\_a1,

set\_EXT\_a3, set\_a3

, stop,

event1,

event2,

trigger1

, move1Call,

move2Call

|}

-- Nodes --

-- declaring all nodes

----------------------------------------------------------------------

-- Initial: i0

### module i0

exports

channel enter, interrupt

D\_\_(id\_\_,

const\_pkg0\_M\_C1\_stm0\_c2,

const\_pkg0\_M\_C1\_stm0\_c1) =

dbisim(let

Inactive = share\_\_choice(enter -> SKIP [] terminate -> SKIP) ; Active

Termination = terminate -> SKIP

Active = share\_\_choice(interrupt -> SKIP) ; Inactive

within

Inactive [| {terminate} |> SKIP)

VS\_O\_\_(id\_\_,

const\_pkg0\_M\_C1\_stm0\_c2,

const\_pkg0\_M\_C1\_stm0\_c1) = D\_\_(id\_\_,

const\_pkg0\_M\_C1\_stm0\_c2,

const\_pkg0\_M\_C1\_stm0\_c1)

endmodule

----------------------------------------------------------------------

----------------------------------------------------------------------

-- State: s0

### module s0

enterSS = {}

enteredSS = {}

exports

-- Declarations

channel enter, entered, interrupt

channel enteredL, enteredR

-- Nodes

-- declaring all nodes

-- Rule: behaviours(Node)

-- Note that FDR has problems with efficiently compiling the process below

-- if using a different recursion pattern.

D\_\_(id\_\_,

const\_pkg0\_M\_C1\_stm0\_c2,

const\_pkg0\_M\_C1\_stm0\_c1) =

dbisim(let

-- IMPLEMENTATION NOTE:

-- The following should be defined as: Inactive = share\_\_choice(Activation [] Termination),

-- however FDR struggles with that form in certain cases. So we use the exception operator

-- instead to 'terminate'.

Inactive = share\_\_choice(enter -> SKIP [] terminate -> SKIP) ; Active

Termination = terminate -> SKIP

Active = true&(share\_\_choice(event1.in?a1 -> (SStop /\ set\_a1!a1 -> SKIP)));true&(share\_\_choice(event2.in?a3 -> (SStop /\ set\_a3!a3 -> SKIP)));share\_\_choice(share\_\_choice(get\_m?m -> true & (share\_\_choice(set\_m!Plus(m, const\_pkg0\_M\_C1\_stm0\_c1, core\_int) -> SKIP)))) ;

Behaviour ;

share\_\_choice(exit -> SKIP) ; true&CALL\_\_move(

id\_\_,

const\_pkg0\_M\_C1\_stm0\_c2,

const\_pkg0\_M\_C1\_stm0\_c1,

2,

3

);true&CALL\_\_move1(

id\_\_,

const\_pkg0\_M\_C1\_stm0\_c2,

const\_pkg0\_M\_C1\_stm0\_c1,

4,

5

) ;

share\_\_choice(exited -> SKIP) ; Inactive

Behaviour = entered -> During

During = ((share\_\_choice(share\_\_choice(get\_a3?a3 -> share\_\_choice(get\_l?l -> true & (share\_\_choice(set\_a!Plus(Plus(a3, l, core\_int), 1, core\_int) -> SKIP)))));true&CALL\_\_move2(

id\_\_,

const\_pkg0\_M\_C1\_stm0\_c2,

const\_pkg0\_M\_C1\_stm0\_c1,

4,

5

) ; SStop) /\ interrupt -> SKIP)

within

Inactive [| {terminate} |> SKIP)

VS\_O\_\_(id\_\_,

const\_pkg0\_M\_C1\_stm0\_c2,

const\_pkg0\_M\_C1\_stm0\_c1) =

dbisim(let

Inactive = share\_\_choice(enter -> SKIP [] terminate -> SKIP) ; Active

Termination = terminate -> SKIP

Active = true&(share\_\_choice(event1.in?a1 -> (SStop /\ set\_a1!a1 -> SKIP)));true&(share\_\_choice(event2.in?a3 -> (SStop /\ set\_a3!a3 -> SKIP)));share\_\_choice(share\_\_choice(get\_m?m -> true & (share\_\_choice(set\_m!Plus(m, const\_pkg0\_M\_C1\_stm0\_c1, core\_int) -> SKIP)))) ;

Behaviour ;

share\_\_choice(exit -> SKIP) ; true&CALL\_\_move(

id\_\_,

const\_pkg0\_M\_C1\_stm0\_c2,

const\_pkg0\_M\_C1\_stm0\_c1,

2,

3

);true&CALL\_\_move1(

id\_\_,

const\_pkg0\_M\_C1\_stm0\_c2,

const\_pkg0\_M\_C1\_stm0\_c1,

4,

5

) ;

share\_\_choice(exited -> SKIP) ; Inactive

Behaviour = entered -> During

During = ((share\_\_choice(share\_\_choice(get\_a3?a3 -> share\_\_choice(get\_l?l -> true & (share\_\_choice(set\_a!Plus(Plus(a3, l, core\_int), 1, core\_int) -> SKIP)))));true&CALL\_\_move2(

id\_\_,

const\_pkg0\_M\_C1\_stm0\_c2,

const\_pkg0\_M\_C1\_stm0\_c1,

4,

5

) ; SStop) /\ interrupt -> SKIP)

within

Inactive [| {terminate} |> SKIP)

endmodule

----------------------------------------------------------------------

----------------------------------------------------------------------

-- Final state: f0

### module f0

exports

channel enter, entered, interrupt

channel enteredL, enteredR

-- Rule: behaviours(Node)

D\_\_(id\_\_,

const\_pkg0\_M\_C1\_stm0\_c2,

const\_pkg0\_M\_C1\_stm0\_c1) =

dbisim(let

Inactive = share\_\_choice(enter -> SKIP [] terminate -> SKIP) ; Entering

Entering = entered -> SKIP ; Active

Active = share\_\_choice(terminate -> SKIP [] interrupt -> SKIP) ; Interrupted

Interrupted = share\_\_choice(exit -> exited -> Inactive)

within

Inactive [| {terminate} |> SKIP)

VS\_O\_\_(id\_\_,

const\_pkg0\_M\_C1\_stm0\_c2,

const\_pkg0\_M\_C1\_stm0\_c1) = D\_\_(id\_\_,

const\_pkg0\_M\_C1\_stm0\_c2,

const\_pkg0\_M\_C1\_stm0\_c1)

endmodule

----------------------------------------------------------------------

-- END of Nodes --

### -- Operation calls --

-- Only the undefined operations are declared here.

-- If the state machine is in isolation, all required operations will be undefined.

-- If it is in the context of a controller, the required operations not provided by the

-- controller will be declared here, and the defined operations will be defined in the

-- context of the Controller module, and therefore within scope of the state machine module.

CALL\_\_move1(id\_\_,

Rule 19, let..

const\_pkg0\_M\_C1\_stm0\_c2,

const\_pkg0\_M\_C1\_stm0\_c1,

param\_m,

param\_m1) = move1Call.param\_m.param\_m1 -> SKIP

CALL\_\_move2(id\_\_,

const\_pkg0\_M\_C1\_stm0\_c2,

const\_pkg0\_M\_C1\_stm0\_c1,

param\_lv,

param\_a) = move2Call.param\_lv.param\_a -> SKIP

-- END of Operation calls --

### -- STM processes Rule 25

STM(id\_\_,

const\_pkg0\_M\_C1\_stm0\_c2,

const\_pkg0\_M\_C1\_stm0\_c1) = -- RULE: [[stm:StateMachineDef]]\_STM^nops : CSPProcess

(

(

(

(IteratedStateful(id\_\_,

const\_pkg0\_M\_C1\_stm0\_c2,

const\_pkg0\_M\_C1\_stm0\_c1) \ {terminate} ; share\_\_choice(terminate -> SKIP))

` [[ share\_\_ <- x\_\_ | x\_\_ <- {|set\_EXT\_m,set\_EXT\_a1,set\_EXT\_a3|} ]]

)

[| {share\_\_} |]

SKIP

)

[| union(sharedVarSync,{terminate}) |]

dbisim(sharedVarMemory(id\_\_,

const\_pkg0\_M\_C1\_stm0\_c2,

const\_pkg0\_M\_C1\_stm0\_c1))

)\sharedVarHide

STM\_VS\_O(id\_\_,

const\_pkg0\_M\_C1\_stm0\_c2,

const\_pkg0\_M\_C1\_stm0\_c1) = -- RULE: [[stm:StateMachineDef]]\_STM^nops : CSPProcess

(

(

(

(IteratedStateful\_VS\_O(id\_\_,

const\_pkg0\_M\_C1\_stm0\_c2,

const\_pkg0\_M\_C1\_stm0\_c1) \ {terminate} ; share\_\_choice(terminate -> SKIP))

[[ share\_\_ <- x\_\_ | x\_\_ <- {|set\_EXT\_m,set\_EXT\_a1,set\_EXT\_a3|} ]]

)

[| {share\_\_} |]

SKIP

)

[| union(sharedVarSync,{terminate}) |]

dbisim(sharedVarMemory(id\_\_,

const\_pkg0\_M\_C1\_stm0\_c2,

const\_pkg0\_M\_C1\_stm0\_c1))

)\sharedVarHide

-- Transitions

Transitions(id\_\_,

const\_pkg0\_M\_C1\_stm0\_c2,

const\_pkg0\_M\_C1\_stm0\_c1) = ((let

Trans = share\_\_choice(get\_a3?a3 -> (

((share\_\_ -> SKIP

[] dbisim((true)&(internal\_\_!NID\_i0 -> SKIP ; ((SKIP ; s0::enter -> SKIP))))

[] dbisim((true)&(stop\_\_!NID\_s0.in -> SKIP ; share\_\_choice(exit -> SKIP) ; (share\_\_choice(exited -> SKIP) ; SKIP ; f0::enter -> SKIP)))

[] dbisim((event2\_\_!NID\_s0.in?a:{a|a <- core\_int, (a3>4)} -> share\_\_choice(set\_a!a -> SKIP) ; share\_\_choice(exit -> SKIP) ; (share\_\_choice(exited -> SKIP) ; share\_\_choice(get\_a3?a3 -> true&(share\_\_choice(trigger1.out!(Plus(a3, const\_pkg0\_M\_C1\_stm0\_c2, core\_int)) -> SKIP))) ; s0::enter -> SKIP)))

) ; Trans)

[]

(interrupt -> share\_\_choice(exit -> SKIP) ; share\_\_choice(exited -> terminate -> SKIP))

[]

terminate -> SKIP

)

)

within

Trans

)

)

### -- Stateful Rule 29

-- RULE: Stateful(stm:StateMachineBody) : CSPProcess

-- Named process definitions

#### MachineBody(id\_\_,

const\_pkg0\_M\_C1\_stm0\_c2,

const\_pkg0\_M\_C1\_stm0\_c1) =

dbisim((

let

finalNodesEntered = {|f0::entered|}

within

((let

-- IMPLEMENTATION NOTE:

-- Here the 'enter' channel set is calculated explicitly because of the use of CSPM

-- modules for defining the semantics of each node.

enterSS = {|

i0::enter,

s0::enter,

f0::enter

|}

hideSet = union(enterSS,{|exit,exited,internal\_\_|})

within

((let

-- IMPLEMENTATION NOTE: the channel set 'enter' is calculated explicitly

-- because CSPM modules are used for the semantics of Node.

flowevts = union(enterSS,{|exit,exited,interrupt|})

transSync = {|internal\_\_.NID\_i0,stop\_\_.NID\_s0.in,event2\_\_.NID\_s0.in|}

within

((

(((-- RULE composeNodes(nc:NodeContainer)^nops : CSPProcess

i0::D\_\_(id\_\_,

const\_pkg0\_M\_C1\_stm0\_c2,

const\_pkg0\_M\_C1\_stm0\_c1)

[| { share\_\_, terminate } |] (

s0::D\_\_(id\_\_,

const\_pkg0\_M\_C1\_stm0\_c2,

const\_pkg0\_M\_C1\_stm0\_c1)

[| { share\_\_, terminate } |] (

f0::D\_\_(id\_\_,

const\_pkg0\_M\_C1\_stm0\_c2,

const\_pkg0\_M\_C1\_stm0\_c1)

)

)

)

[[s0::interrupt <- x\_\_ | x\_\_ <- {|interrupt,stop\_\_.NID\_s0.in,event2\_\_.NID\_s0.in|}]]

[[f0::interrupt <- x\_\_ | x\_\_ <- {|interrupt|}]]

[[i0::interrupt <- x\_\_ | x\_\_ <- {|internal\_\_.NID\_i0|}]]

)

[[ share\_\_ <- x\_\_ | x\_\_ <- {| share\_\_,setR\_a |} ]]

[[set\_a3 <- setL\_a3]]

)

[| union(union(union(flowevts,transSync),{terminate}),{|share\_\_

,setL\_a3

,setR\_a

|}) |]

((i0::enter -> Transitions(id\_\_,

const\_pkg0\_M\_C1\_stm0\_c2,

const\_pkg0\_M\_C1\_stm0\_c1))

[[ share\_\_ <- x\_\_ | x\_\_ <- {| share\_\_,setL\_a3 |} ]]

[[set\_a <- setR\_a]]

)

)[[setL\_a3 <- set\_a3]]

[[setR\_a <- set\_a]]

)

)

\ hideSet)

[[

stop\_\_.x\_\_\_\_ <- stop,

event1\_\_.x\_\_\_\_ <- event1,

event2\_\_.x\_\_\_\_ <- event2,

trigger1\_\_.x\_\_\_\_ <- trigger1

| x\_\_\_\_ <- NIDS

]]

)

[| {| interrupt |} |] SKIP)

)

)

Behaviour(id\_\_,

const\_pkg0\_M\_C1\_stm0\_c2,

const\_pkg0\_M\_C1\_stm0\_c1) =

dbisim((MachineBody(id\_\_,

const\_pkg0\_M\_C1\_stm0\_c2,

const\_pkg0\_M\_C1\_stm0\_c1)\ enteredSS)

)

IteratedBehaviour(id\_\_,

const\_pkg0\_M\_C1\_stm0\_c2,

const\_pkg0\_M\_C1\_stm0\_c1) =

dbisim((MachineBody(id\_\_,

const\_pkg0\_M\_C1\_stm0\_c2,

const\_pkg0\_M\_C1\_stm0\_c1)\ enteredSS)

)

Stateful(id\_\_,

const\_pkg0\_M\_C1\_stm0\_c2,

const\_pkg0\_M\_C1\_stm0\_c1) =

((

(Behaviour(id\_\_, const\_pkg0\_M\_C1\_stm0\_c2, const\_pkg0\_M\_C1\_stm0\_c1)

[| union(getsetLocalChannels,{terminate}) |]

varMemory(id\_\_,

const\_pkg0\_M\_C1\_stm0\_c2,

const\_pkg0\_M\_C1\_stm0\_c1))

\getsetLocalChannels

)

)

#### IteratedStateful(id\_\_,

const\_pkg0\_M\_C1\_stm0\_c2,

const\_pkg0\_M\_C1\_stm0\_c1) =

(dbisim(

sbisim(

dbisim(

sbisim(

Behaviour(id\_\_,

const\_pkg0\_M\_C1\_stm0\_c2,

const\_pkg0\_M\_C1\_stm0\_c1)

[| {|get\_a,set\_a,terminate|} |]

Memory\_a(0)

)\{|get\_a,set\_a|}

)

[| {|get\_l,set\_l,terminate|} |]

Memory\_l(0)

)\{|get\_l,set\_l|}

)

)

-- Visible counterparts

MachineBody\_VS\_O(id\_\_,

const\_pkg0\_M\_C1\_stm0\_c2,

const\_pkg0\_M\_C1\_stm0\_c1) =

dbisim((

let

finalNodesEntered = {|f0::entered|}

within

((let

-- IMPLEMENTATION NOTE:

-- Here the 'enter' channel set is calculated explicitly because of the use of CSPM

-- modules for defining the semantics of each node.

enterSS = {|

i0::enter,

s0::enter,

f0::enter

|}

hideSet = union(enterSS,{|exit,exited,internal\_\_|})

within

((let

-- IMPLEMENTATION NOTE: the channel set 'enter' is calculated explicitly

-- because CSPM modules are used for the semantics of Node.

flowevts = union(enterSS,{|exit,exited,interrupt|})

transSync = {|internal\_\_.NID\_i0,stop\_\_.NID\_s0.in,event2\_\_.NID\_s0.in|}

within

((

(((-- RULE composeNodes(nc:NodeContainer)^nops : CSPProcess

i0::VS\_O\_\_(id\_\_,

const\_pkg0\_M\_C1\_stm0\_c2,

const\_pkg0\_M\_C1\_stm0\_c1)

[| { share\_\_, terminate } |] (

s0::VS\_O\_\_(id\_\_,

const\_pkg0\_M\_C1\_stm0\_c2,

const\_pkg0\_M\_C1\_stm0\_c1)

[| { share\_\_, terminate } |] (

f0::VS\_O\_\_(id\_\_,

const\_pkg0\_M\_C1\_stm0\_c2,

const\_pkg0\_M\_C1\_stm0\_c1)

)

)

)

[[s0::interrupt <- x\_\_ | x\_\_ <- {|interrupt,stop\_\_.NID\_s0.in,event2\_\_.NID\_s0.in|}]]

[[f0::interrupt <- x\_\_ | x\_\_ <- {|interrupt|}]]

[[i0::interrupt <- x\_\_ | x\_\_ <- {|internal\_\_.NID\_i0|}]]

)

[[ share\_\_ <- x\_\_ | x\_\_ <- {| share\_\_,setR\_a |} ]]

[[set\_a3 <- setL\_a3]]

)

[| union(union(union(flowevts,transSync),{terminate}),{|share\_\_

,setL\_a3

,setR\_a

|}) |]

((i0::enter -> Transitions(id\_\_,

const\_pkg0\_M\_C1\_stm0\_c2,

const\_pkg0\_M\_C1\_stm0\_c1))

[[ share\_\_ <- x\_\_ | x\_\_ <- {| share\_\_,setL\_a3 |} ]]

[[set\_a <- setR\_a]]

)

)[[setL\_a3 <- set\_a3]]

[[setR\_a <- set\_a]]

)

)

\ hideSet)

[[

stop\_\_.x\_\_\_\_ <- stop,

event1\_\_.x\_\_\_\_ <- event1,

event2\_\_.x\_\_\_\_ <- event2,

trigger1\_\_.x\_\_\_\_ <- trigger1

| x\_\_\_\_ <- NIDS

]]

)

[| {| interrupt |} |] SKIP)

)

)

Behaviour\_VS\_O(id\_\_,

const\_pkg0\_M\_C1\_stm0\_c2,

const\_pkg0\_M\_C1\_stm0\_c1) =

dbisim((MachineBody\_VS\_O(id\_\_,

const\_pkg0\_M\_C1\_stm0\_c2,

const\_pkg0\_M\_C1\_stm0\_c1))

)

IteratedBehaviour\_VS\_O(id\_\_,

const\_pkg0\_M\_C1\_stm0\_c2,

const\_pkg0\_M\_C1\_stm0\_c1) =

dbisim((MachineBody\_VS\_O(id\_\_,

const\_pkg0\_M\_C1\_stm0\_c2,

const\_pkg0\_M\_C1\_stm0\_c1))

)

Stateful\_VS\_O(id\_\_,

const\_pkg0\_M\_C1\_stm0\_c2,

const\_pkg0\_M\_C1\_stm0\_c1) =

dbisim((

(Behaviour\_VS\_O(id\_\_,

const\_pkg0\_M\_C1\_stm0\_c2,

const\_pkg0\_M\_C1\_stm0\_c1) [| union(getsetLocalChannels,{terminate}) |] varMemory(id\_\_,

const\_pkg0\_M\_C1\_stm0\_c2,

const\_pkg0\_M\_C1\_stm0\_c1))

\getsetLocalChannels

)

)

IteratedStateful\_VS\_O(id\_\_,

const\_pkg0\_M\_C1\_stm0\_c2,

const\_pkg0\_M\_C1\_stm0\_c1) =

(dbisim(

sbisim(

dbisim(

sbisim(

Behaviour\_VS\_O(id\_\_,

const\_pkg0\_M\_C1\_stm0\_c2,

const\_pkg0\_M\_C1\_stm0\_c1)

[| {|get\_a,set\_a,terminate|} |]

Memory\_a(0)

)\{|get\_a,set\_a|}

)

[| {|get\_l,set\_l,terminate|} |]

Memory\_l(0)

)\{|get\_l,set\_l|}

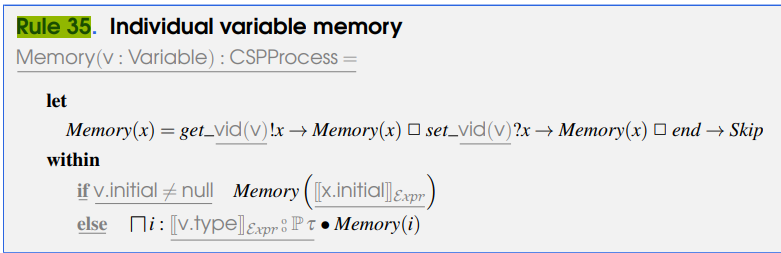
)

)

-- END

-- Memory

### -- Memory variables

 Memory\_l(l) =

get\_l!l -> Memory\_l(l)

[]

set\_l?x\_\_ -> Memory\_l(x\_\_)

[]

terminate -> SKIP

Memory\_a(a) =

get\_a!a -> Memory\_a(a)

[]

set\_a?x\_\_ -> Memory\_a(x\_\_)

[]

terminate -> SKIP

-- varMemory process Rule 31

A screenshot of a computer program

Description automatically generated varMemory(id\_\_,

const\_pkg0\_M\_C1\_stm0\_c2,

const\_pkg0\_M\_C1\_stm0\_c1) =

Memory\_l(0)

[| { terminate } |]

(Memory\_a(0)

)

getsetLocalChannels = {|get\_l,set\_l,get\_a,set\_a|}

-- Definition of the behaviour of state machines (default, optimised, visible, visible and optimised)

FVS\_\_(id\_\_,

const\_pkg0\_M\_C1\_stm0\_c2,

const\_pkg0\_M\_C1\_stm0\_c1) = STM\_VS\_O(id\_\_,

const\_pkg0\_M\_C1\_stm0\_c2,

const\_pkg0\_M\_C1\_stm0\_c1)

D\_\_(id\_\_,

const\_pkg0\_M\_C1\_stm0\_c2,

const\_pkg0\_M\_C1\_stm0\_c1) = STM(id\_\_,

const\_pkg0\_M\_C1\_stm0\_c2,

const\_pkg0\_M\_C1\_stm0\_c1) \ internal\_events

O\_\_(id\_\_,

const\_pkg0\_M\_C1\_stm0\_c2,

const\_pkg0\_M\_C1\_stm0\_c1) = sbisim(diamond(D\_\_(id\_\_,

const\_pkg0\_M\_C1\_stm0\_c2,

const\_pkg0\_M\_C1\_stm0\_c1)))

VS\_\_(id\_\_,

const\_pkg0\_M\_C1\_stm0\_c2,

const\_pkg0\_M\_C1\_stm0\_c1) = FVS\_\_(id\_\_,

const\_pkg0\_M\_C1\_stm0\_c2,

const\_pkg0\_M\_C1\_stm0\_c1)

VS\_O\_\_(id\_\_,

const\_pkg0\_M\_C1\_stm0\_c2,

const\_pkg0\_M\_C1\_stm0\_c1) = sbisim(diamond(FVS\_\_(id\_\_,

const\_pkg0\_M\_C1\_stm0\_c2,

const\_pkg0\_M\_C1\_stm0\_c1)))

HEXT\_\_(id\_\_,

const\_pkg0\_M\_C1\_stm0\_c2,

const\_pkg0\_M\_C1\_stm0\_c1) = O\_\_(id\_\_,

const\_pkg0\_M\_C1\_stm0\_c2,

const\_pkg0\_M\_C1\_stm0\_c1) [|shared\_variable\_events|] SKIP

HUP\_\_(id\_\_,

const\_pkg0\_M\_C1\_stm0\_c2,

const\_pkg0\_M\_C1\_stm0\_c1) = O\_\_(id\_\_,

const\_pkg0\_M\_C1\_stm0\_c2,

const\_pkg0\_M\_C1\_stm0\_c1) [|{share\_\_}|] SKIP

-- Shared memory

-- Shared memory variables

Memory\_m(m) =

get\_m!m -> Memory\_m(m)

[]

set\_m?x\_\_ -> Memory\_m(x\_\_)

[]

set\_EXT\_m?x\_\_ -> Memory\_m(x\_\_)

[]

terminate -> SKIP

Memory\_a1(a1) =

get\_a1!a1 -> Memory\_a1(a1)

[]

set\_a1?x\_\_ -> Memory\_a1(x\_\_)

[]

set\_EXT\_a1?x\_\_ -> Memory\_a1(x\_\_)

[]

terminate -> SKIP

Memory\_a3(a3) =

get\_a3!a3 -> Memory\_a3(a3)

[]

set\_a3?x\_\_ -> Memory\_a3(x\_\_)

[]

set\_EXT\_a3?x\_\_ -> Memory\_a3(x\_\_)

[]

terminate -> SKIP

### -- sharedVarMemory process

A screenshot of a computer program

Description automatically generated sharedVarMemory(id\_\_,

const\_pkg0\_M\_C1\_stm0\_c2,

const\_pkg0\_M\_C1\_stm0\_c1) =

Memory\_m(0)

[| { terminate } |] (

Memory\_a1((0,0))

[| { terminate } |] (

Memory\_a3(0)

)

)

sharedVarSync = {|get\_m,set\_m,set\_EXT\_m,get\_a1,set\_a1,set\_EXT\_a1,get\_a3,set\_a3,set\_EXT\_a3|}

sharedVarHide = {|get\_m,get\_a1,get\_a3|}

endmodule

## module stm1

exports

transparent diamond

transparent sbisim

transparent dbisim

transparent chase

-- Transition identifiers

-- declaring identifiers of transitions

datatype NIDS =

NID\_i0|

NID\_s0|

NID\_f0

channel internal\_\_ : NIDS

-- Flow channels

channel interrupt

channel exited

channel exit

channel terminate

-- Variable channels

channel get\_l, set\_l, setL\_l, setR\_l: core\_int

channel get\_a, set\_a, setL\_a, setR\_a: core\_int

channel get\_m, set\_m, setL\_m, setR\_m: core\_int

channel get\_a1, set\_a1, setL\_a1, setR\_a1: {(x0\_\_,x1\_\_) | x0\_\_ <- core\_int,x1\_\_ <- core\_int}

channel get\_a3, set\_a3, setL\_a3, setR\_a3: core\_int

-- Shared variable channels

channel set\_EXT\_m: core\_int

channel set\_EXT\_a1: {(x0\_\_,x1\_\_) | x0\_\_ <- core\_int,x1\_\_ <- core\_int}

channel set\_EXT\_a3: core\_int

-- Local variable channels for defined operations that are required by the state machine

-- Declaring state machine events

channel stop\_\_: NIDS.InOut

channel stop: InOut

channel event1\_\_: NIDS.InOut.{(x0\_\_,x1\_\_) | x0\_\_ <- core\_int,x1\_\_ <- core\_int}

channel event1: InOut.{(x0\_\_,x1\_\_) | x0\_\_ <- core\_int,x1\_\_ <- core\_int}

channel event2\_\_: NIDS.InOut.core\_int

channel event2: InOut.core\_int

channel trigger1\_\_: NIDS.InOut.core\_int

channel trigger1: InOut.core\_int

-- Declaring call and ret events for undefined operations

channel move1Call: core\_real.core\_int

channel move2Call: core\_real.core\_int

enterSS = {|

i0::enter,

s0::enter,

f0::enter

|}

enteredSS = {|

s0::entered,

f0::entered

|}

internal\_events = union(enteredSS,union(enterSS,{|interrupt,exited|}))

shared\_variable\_events = {|

set\_EXT\_m,

set\_EXT\_a1,

set\_EXT\_a3

|}

-- channel set with all visible events

sem\_\_events = {|

terminate

, set\_EXT\_m, set\_m,

set\_EXT\_a1, set\_a1,

set\_EXT\_a3, set\_a3

, stop,

event1,

event2,

trigger1

, move1Call,

move2Call

|}

-- Nodes --

-- declaring all nodes

----------------------------------------------------------------------

-- Initial: i0

module i0

exports

channel enter, interrupt

D\_\_(id\_\_,

const\_pkg0\_M\_C1\_stm0\_c2,

const\_pkg0\_M\_C1\_stm0\_c1) =

dbisim(let

Inactive = share\_\_choice(enter -> SKIP [] terminate -> SKIP) ; Active

Termination = terminate -> SKIP

Active = share\_\_choice(interrupt -> SKIP) ; Inactive

within

Inactive [| {terminate} |> SKIP)

VS\_O\_\_(id\_\_,

const\_pkg0\_M\_C1\_stm0\_c2,

const\_pkg0\_M\_C1\_stm0\_c1) = D\_\_(id\_\_,

const\_pkg0\_M\_C1\_stm0\_c2,

const\_pkg0\_M\_C1\_stm0\_c1)

endmodule

----------------------------------------------------------------------

----------------------------------------------------------------------

-- State: s0

module s0

enterSS = {}

enteredSS = {}

exports

-- Declarations

channel enter, entered, interrupt

channel enteredL, enteredR

-- Nodes

-- declaring all nodes

-- Rule: behaviours(Node)

-- Note that FDR has problems with efficiently compiling the process below

-- if using a different recursion pattern.

D\_\_(id\_\_,

const\_pkg0\_M\_C1\_stm0\_c2,

const\_pkg0\_M\_C1\_stm0\_c1) =

dbisim(let

-- IMPLEMENTATION NOTE:

-- The following should be defined as: Inactive = share\_\_choice(Activation [] Termination),

-- however FDR struggles with that form in certain cases. So we use the exception operator

-- instead to 'terminate'.

Inactive = share\_\_choice(enter -> SKIP [] terminate -> SKIP) ; Active

Termination = terminate -> SKIP

Active = true&(share\_\_choice(event1.in?a1 -> (SStop /\ set\_a1!a1 -> SKIP)));true&(share\_\_choice(event2.in?a3 -> (SStop /\ set\_a3!a3 -> SKIP)));share\_\_choice(share\_\_choice(get\_m?m -> true & (share\_\_choice(set\_m!Plus(m, const\_pkg0\_M\_C1\_stm0\_c1, core\_int) -> SKIP)))) ;

Behaviour ;

share\_\_choice(exit -> SKIP) ; true&CALL\_\_move(

id\_\_,

const\_pkg0\_M\_C1\_stm0\_c2,

const\_pkg0\_M\_C1\_stm0\_c1,

2,

3

);true&CALL\_\_move1(

id\_\_,

const\_pkg0\_M\_C1\_stm0\_c2,

const\_pkg0\_M\_C1\_stm0\_c1,

4,

5

) ;

share\_\_choice(exited -> SKIP) ; Inactive

Behaviour = entered -> During

During = ((share\_\_choice(share\_\_choice(get\_a3?a3 -> share\_\_choice(get\_l?l -> true & (share\_\_choice(set\_a!Plus(Plus(a3, l, core\_int), 1, core\_int) -> SKIP)))));true&CALL\_\_move2(

id\_\_,

const\_pkg0\_M\_C1\_stm0\_c2,

const\_pkg0\_M\_C1\_stm0\_c1,

4,

5

) ; SStop) /\ interrupt -> SKIP)

within

Inactive [| {terminate} |> SKIP)

VS\_O\_\_(id\_\_,

const\_pkg0\_M\_C1\_stm0\_c2,

const\_pkg0\_M\_C1\_stm0\_c1) =

dbisim(let

Inactive = share\_\_choice(enter -> SKIP [] terminate -> SKIP) ; Active

Termination = terminate -> SKIP

Active = true&(share\_\_choice(event1.in?a1 -> (SStop /\ set\_a1!a1 -> SKIP)));true&(share\_\_choice(event2.in?a3 -> (SStop /\ set\_a3!a3 -> SKIP)));share\_\_choice(share\_\_choice(get\_m?m -> true & (share\_\_choice(set\_m!Plus(m, const\_pkg0\_M\_C1\_stm0\_c1, core\_int) -> SKIP)))) ;

Behaviour ;

share\_\_choice(exit -> SKIP) ; true&CALL\_\_move(

id\_\_,

const\_pkg0\_M\_C1\_stm0\_c2,

const\_pkg0\_M\_C1\_stm0\_c1,

2,

3

);true&CALL\_\_move1(

id\_\_,

const\_pkg0\_M\_C1\_stm0\_c2,

const\_pkg0\_M\_C1\_stm0\_c1,

4,

5

) ;

share\_\_choice(exited -> SKIP) ; Inactive

Behaviour = entered -> During

During = ((share\_\_choice(share\_\_choice(get\_a3?a3 -> share\_\_choice(get\_l?l -> true & (share\_\_choice(set\_a!Plus(Plus(a3, l, core\_int), 1, core\_int) -> SKIP)))));true&CALL\_\_move2(

id\_\_,

const\_pkg0\_M\_C1\_stm0\_c2,

const\_pkg0\_M\_C1\_stm0\_c1,

4,

5

) ; SStop) /\ interrupt -> SKIP)

within

Inactive [| {terminate} |> SKIP)

endmodule

----------------------------------------------------------------------

----------------------------------------------------------------------

-- Final state: f0

module f0

exports

channel enter, entered, interrupt

channel enteredL, enteredR

-- Rule: behaviours(Node)

D\_\_(id\_\_,

const\_pkg0\_M\_C1\_stm0\_c2,

const\_pkg0\_M\_C1\_stm0\_c1) =

dbisim(let

Inactive = share\_\_choice(enter -> SKIP [] terminate -> SKIP) ; Entering

Entering = entered -> SKIP ; Active

Active = share\_\_choice(terminate -> SKIP [] interrupt -> SKIP) ; Interrupted

Interrupted = share\_\_choice(exit -> exited -> Inactive)

within

Inactive [| {terminate} |> SKIP)

VS\_O\_\_(id\_\_,

const\_pkg0\_M\_C1\_stm0\_c2,

const\_pkg0\_M\_C1\_stm0\_c1) = D\_\_(id\_\_,

const\_pkg0\_M\_C1\_stm0\_c2,

const\_pkg0\_M\_C1\_stm0\_c1)

endmodule

----------------------------------------------------------------------

-- END of Nodes --

-- Operation calls --

-- Only the undefined operations are declared here.

-- If the state machine is in isolation, all required operations will be undefined.

-- If it is in the context of a controller, the required operations not provided by the

-- controller will be declared here, and the defined operations will be defined in the

-- context of the Controller module, and therefore within scope of the state machine module.

CALL\_\_move1(id\_\_,

const\_pkg0\_M\_C1\_stm0\_c2,

const\_pkg0\_M\_C1\_stm0\_c1,

param\_m,

param\_m1) = move1Call.param\_m.param\_m1 -> SKIP

CALL\_\_move2(id\_\_,

const\_pkg0\_M\_C1\_stm0\_c2,

const\_pkg0\_M\_C1\_stm0\_c1,

param\_lv,

param\_a) = move2Call.param\_lv.param\_a -> SKIP

-- END of Operation calls --

-- STM processes

STM(id\_\_,

const\_pkg0\_M\_C1\_stm0\_c2,

const\_pkg0\_M\_C1\_stm0\_c1) = -- RULE: [[stm:StateMachineDef]]\_STM^nops : CSPProcess

(

(

(

(IteratedStateful(id\_\_,

const\_pkg0\_M\_C1\_stm0\_c2,

const\_pkg0\_M\_C1\_stm0\_c1) \ {terminate} ; share\_\_choice(terminate -> SKIP))

[[ share\_\_ <- x\_\_ | x\_\_ <- {|set\_EXT\_m,set\_EXT\_a1,set\_EXT\_a3|} ]]

)

[| {share\_\_} |]

SKIP

)

[| union(sharedVarSync,{terminate}) |]

dbisim(sharedVarMemory(id\_\_,

const\_pkg0\_M\_C1\_stm0\_c2,

const\_pkg0\_M\_C1\_stm0\_c1))

)\sharedVarHide

STM\_VS\_O(id\_\_,

const\_pkg0\_M\_C1\_stm0\_c2,

const\_pkg0\_M\_C1\_stm0\_c1) = -- RULE: [[stm:StateMachineDef]]\_STM^nops : CSPProcess

(

(

(

(IteratedStateful\_VS\_O(id\_\_,

const\_pkg0\_M\_C1\_stm0\_c2,

const\_pkg0\_M\_C1\_stm0\_c1) \ {terminate} ; share\_\_choice(terminate -> SKIP))

[[ share\_\_ <- x\_\_ | x\_\_ <- {|set\_EXT\_m,set\_EXT\_a1,set\_EXT\_a3|} ]]

)

[| {share\_\_} |]

SKIP

)

[| union(sharedVarSync,{terminate}) |]

dbisim(sharedVarMemory(id\_\_,

const\_pkg0\_M\_C1\_stm0\_c2,

const\_pkg0\_M\_C1\_stm0\_c1))

)\sharedVarHide

-- Transitions

Transitions(id\_\_,

const\_pkg0\_M\_C1\_stm0\_c2,

const\_pkg0\_M\_C1\_stm0\_c1) = ((let

Trans = share\_\_choice(get\_a3?a3 -> (

((share\_\_ -> SKIP

[] dbisim((true)&(internal\_\_!NID\_i0 -> SKIP ; ((SKIP ; s0::enter -> SKIP))))

[] dbisim((true)&(stop\_\_!NID\_s0.in -> SKIP ; share\_\_choice(exit -> SKIP) ; (share\_\_choice(exited -> SKIP) ; SKIP ; f0::enter -> SKIP)))

[] dbisim((event2\_\_!NID\_s0.in?a:{a|a <- core\_int, (a3>4)} -> share\_\_choice(set\_a!a -> SKIP) ; share\_\_choice(exit -> SKIP) ; (share\_\_choice(exited -> SKIP) ; share\_\_choice(get\_a3?a3 -> true&(share\_\_choice(trigger1.out!(Plus(a3, const\_pkg0\_M\_C1\_stm0\_c2, core\_int)) -> SKIP))) ; s0::enter -> SKIP)))

) ; Trans)

[]

(interrupt -> share\_\_choice(exit -> SKIP) ; share\_\_choice(exited -> terminate -> SKIP))

[]

terminate -> SKIP

)

)

within

Trans

)

)

-- Stateful

-- RULE: Stateful(stm:StateMachineBody) : CSPProcess

-- Named process definitions

MachineBody(id\_\_,

const\_pkg0\_M\_C1\_stm0\_c2,

const\_pkg0\_M\_C1\_stm0\_c1) =

dbisim((

let

finalNodesEntered = {|f0::entered|}

within

((let

-- IMPLEMENTATION NOTE:

-- Here the 'enter' channel set is calculated explicitly because of the use of CSPM

-- modules for defining the semantics of each node.

enterSS = {|

i0::enter,

s0::enter,

f0::enter

|}

hideSet = union(enterSS,{|exit,exited,internal\_\_|})

within

((let

-- IMPLEMENTATION NOTE: the channel set 'enter' is calculated explicitly

-- because CSPM modules are used for the semantics of Node.

flowevts = union(enterSS,{|exit,exited,interrupt|})

transSync = {|internal\_\_.NID\_i0,stop\_\_.NID\_s0.in,event2\_\_.NID\_s0.in|}

within

((

(((-- RULE composeNodes(nc:NodeContainer)^nops : CSPProcess

i0::D\_\_(id\_\_,

const\_pkg0\_M\_C1\_stm0\_c2,

const\_pkg0\_M\_C1\_stm0\_c1)

[| { share\_\_, terminate } |] (

s0::D\_\_(id\_\_,

const\_pkg0\_M\_C1\_stm0\_c2,

const\_pkg0\_M\_C1\_stm0\_c1)

[| { share\_\_, terminate } |] (

f0::D\_\_(id\_\_,

const\_pkg0\_M\_C1\_stm0\_c2,

const\_pkg0\_M\_C1\_stm0\_c1)

)

)

)

[[s0::interrupt <- x\_\_ | x\_\_ <- {|interrupt,stop\_\_.NID\_s0.in,event2\_\_.NID\_s0.in|}]]

[[f0::interrupt <- x\_\_ | x\_\_ <- {|interrupt|}]]

[[i0::interrupt <- x\_\_ | x\_\_ <- {|internal\_\_.NID\_i0|}]]

)

[[ share\_\_ <- x\_\_ | x\_\_ <- {| share\_\_,setR\_a |} ]]

[[set\_a3 <- setL\_a3]]

)

[| union(union(union(flowevts,transSync),{terminate}),{|share\_\_

,setL\_a3

,setR\_a

|}) |]

((i0::enter -> Transitions(id\_\_,

const\_pkg0\_M\_C1\_stm0\_c2,

const\_pkg0\_M\_C1\_stm0\_c1))

[[ share\_\_ <- x\_\_ | x\_\_ <- {| share\_\_,setL\_a3 |} ]]

[[set\_a <- setR\_a]]

)

)[[setL\_a3 <- set\_a3]]

[[setR\_a <- set\_a]]

)

)

\ hideSet)

[[

stop\_\_.x\_\_\_\_ <- stop,

event1\_\_.x\_\_\_\_ <- event1,

event2\_\_.x\_\_\_\_ <- event2,

trigger1\_\_.x\_\_\_\_ <- trigger1

| x\_\_\_\_ <- NIDS

]]

)

[| {| interrupt |} |] SKIP)

)

)

Behaviour(id\_\_,

const\_pkg0\_M\_C1\_stm0\_c2,

const\_pkg0\_M\_C1\_stm0\_c1) =

dbisim((MachineBody(id\_\_,

const\_pkg0\_M\_C1\_stm0\_c2,

const\_pkg0\_M\_C1\_stm0\_c1)\ enteredSS)

)

IteratedBehaviour(id\_\_,

const\_pkg0\_M\_C1\_stm0\_c2,

const\_pkg0\_M\_C1\_stm0\_c1) =

dbisim((MachineBody(id\_\_,

const\_pkg0\_M\_C1\_stm0\_c2,

const\_pkg0\_M\_C1\_stm0\_c1)\ enteredSS)

)

Stateful(id\_\_,

const\_pkg0\_M\_C1\_stm0\_c2,

const\_pkg0\_M\_C1\_stm0\_c1) =

((

(Behaviour(id\_\_,

const\_pkg0\_M\_C1\_stm0\_c2,

const\_pkg0\_M\_C1\_stm0\_c1) [| union(getsetLocalChannels,{terminate}) |] varMemory(id\_\_,

const\_pkg0\_M\_C1\_stm0\_c2,

const\_pkg0\_M\_C1\_stm0\_c1))

\getsetLocalChannels

)

)

IteratedStateful(id\_\_,

const\_pkg0\_M\_C1\_stm0\_c2,

const\_pkg0\_M\_C1\_stm0\_c1) =

(dbisim(

sbisim(

dbisim(

sbisim(

Behaviour(id\_\_,

const\_pkg0\_M\_C1\_stm0\_c2,

const\_pkg0\_M\_C1\_stm0\_c1)

[| {|get\_a,set\_a,terminate|} |]

Memory\_a(0)

)\{|get\_a,set\_a|}

)

[| {|get\_l,set\_l,terminate|} |]

Memory\_l(0)

)\{|get\_l,set\_l|}

)

)

-- Visible counterparts

MachineBody\_VS\_O(id\_\_,

const\_pkg0\_M\_C1\_stm0\_c2,

const\_pkg0\_M\_C1\_stm0\_c1) =

dbisim((

let

finalNodesEntered = {|f0::entered|}

within

((let

-- IMPLEMENTATION NOTE:

-- Here the 'enter' channel set is calculated explicitly because of the use of CSPM

-- modules for defining the semantics of each node.

enterSS = {|

i0::enter,

s0::enter,

f0::enter

|}

hideSet = union(enterSS,{|exit,exited,internal\_\_|})

within

((let

-- IMPLEMENTATION NOTE: the channel set 'enter' is calculated explicitly

-- because CSPM modules are used for the semantics of Node.

flowevts = union(enterSS,{|exit,exited,interrupt|})

transSync = {|internal\_\_.NID\_i0,stop\_\_.NID\_s0.in,event2\_\_.NID\_s0.in|}

within

((

(((-- RULE composeNodes(nc:NodeContainer)^nops : CSPProcess

i0::VS\_O\_\_(id\_\_,

const\_pkg0\_M\_C1\_stm0\_c2,

const\_pkg0\_M\_C1\_stm0\_c1)

[| { share\_\_, terminate } |] (

s0::VS\_O\_\_(id\_\_,

const\_pkg0\_M\_C1\_stm0\_c2,

const\_pkg0\_M\_C1\_stm0\_c1)

[| { share\_\_, terminate } |] (

f0::VS\_O\_\_(id\_\_,

const\_pkg0\_M\_C1\_stm0\_c2,

const\_pkg0\_M\_C1\_stm0\_c1)

)

)

)

[[s0::interrupt <- x\_\_ | x\_\_ <- {|interrupt,stop\_\_.NID\_s0.in,event2\_\_.NID\_s0.in|}]]

[[f0::interrupt <- x\_\_ | x\_\_ <- {|interrupt|}]]

[[i0::interrupt <- x\_\_ | x\_\_ <- {|internal\_\_.NID\_i0|}]]

)

[[ share\_\_ <- x\_\_ | x\_\_ <- {| share\_\_,setR\_a |} ]]

[[set\_a3 <- setL\_a3]]

)

[| union(union(union(flowevts,transSync),{terminate}),{|share\_\_

,setL\_a3

,setR\_a

|}) |]

((i0::enter -> Transitions(id\_\_,

const\_pkg0\_M\_C1\_stm0\_c2,

const\_pkg0\_M\_C1\_stm0\_c1))

[[ share\_\_ <- x\_\_ | x\_\_ <- {| share\_\_,setL\_a3 |} ]]

[[set\_a <- setR\_a]]

)

)[[setL\_a3 <- set\_a3]]

[[setR\_a <- set\_a]]

)

)

\ hideSet)

[[

stop\_\_.x\_\_\_\_ <- stop,

event1\_\_.x\_\_\_\_ <- event1,

event2\_\_.x\_\_\_\_ <- event2,

trigger1\_\_.x\_\_\_\_ <- trigger1

| x\_\_\_\_ <- NIDS

]]

)

[| {| interrupt |} |] SKIP)

)

)

Behaviour\_VS\_O(id\_\_,

const\_pkg0\_M\_C1\_stm0\_c2,

const\_pkg0\_M\_C1\_stm0\_c1) =

dbisim((MachineBody\_VS\_O(id\_\_,

const\_pkg0\_M\_C1\_stm0\_c2,

const\_pkg0\_M\_C1\_stm0\_c1))

)

IteratedBehaviour\_VS\_O(id\_\_,

const\_pkg0\_M\_C1\_stm0\_c2,

const\_pkg0\_M\_C1\_stm0\_c1) =

dbisim((MachineBody\_VS\_O(id\_\_,

const\_pkg0\_M\_C1\_stm0\_c2,

const\_pkg0\_M\_C1\_stm0\_c1))

)

Stateful\_VS\_O(id\_\_,

const\_pkg0\_M\_C1\_stm0\_c2,

const\_pkg0\_M\_C1\_stm0\_c1) =

dbisim((

(Behaviour\_VS\_O(id\_\_,

const\_pkg0\_M\_C1\_stm0\_c2,

const\_pkg0\_M\_C1\_stm0\_c1) [| union(getsetLocalChannels,{terminate}) |] varMemory(id\_\_,

const\_pkg0\_M\_C1\_stm0\_c2,

const\_pkg0\_M\_C1\_stm0\_c1))

\getsetLocalChannels

)

)

IteratedStateful\_VS\_O(id\_\_,

const\_pkg0\_M\_C1\_stm0\_c2,

const\_pkg0\_M\_C1\_stm0\_c1) =

(dbisim(

sbisim(

dbisim(

sbisim(

Behaviour\_VS\_O(id\_\_,

const\_pkg0\_M\_C1\_stm0\_c2,

const\_pkg0\_M\_C1\_stm0\_c1)

[| {|get\_a,set\_a,terminate|} |]

Memory\_a(0)

)\{|get\_a,set\_a|}

)

[| {|get\_l,set\_l,terminate|} |]

Memory\_l(0)

)\{|get\_l,set\_l|}

)

)

-- END

-- Memory

-- Memory variables

Memory\_l(l) =

get\_l!l -> Memory\_l(l)

[]

set\_l?x\_\_ -> Memory\_l(x\_\_)

[]

terminate -> SKIP

Memory\_a(a) =

get\_a!a -> Memory\_a(a)

[]

set\_a?x\_\_ -> Memory\_a(x\_\_)

[]

terminate -> SKIP

-- varMemory process

varMemory(id\_\_,

const\_pkg0\_M\_C1\_stm0\_c2,

const\_pkg0\_M\_C1\_stm0\_c1) = Memory\_l(0)

[| { terminate } |] (

Memory\_a(0)

)

getsetLocalChannels = {|get\_l,set\_l,get\_a,set\_a|}

-- Definition of the behaviour of state machines (default, optimised, visible, visible and optimised)

FVS\_\_(id\_\_,

const\_pkg0\_M\_C1\_stm0\_c2,

const\_pkg0\_M\_C1\_stm0\_c1) = STM\_VS\_O(id\_\_,

const\_pkg0\_M\_C1\_stm0\_c2,

const\_pkg0\_M\_C1\_stm0\_c1)

D\_\_(id\_\_,

const\_pkg0\_M\_C1\_stm0\_c2,

const\_pkg0\_M\_C1\_stm0\_c1) = STM(id\_\_,

const\_pkg0\_M\_C1\_stm0\_c2,

const\_pkg0\_M\_C1\_stm0\_c1) \ internal\_events

O\_\_(id\_\_,

const\_pkg0\_M\_C1\_stm0\_c2,

const\_pkg0\_M\_C1\_stm0\_c1) = sbisim(diamond(D\_\_(id\_\_,

const\_pkg0\_M\_C1\_stm0\_c2,

const\_pkg0\_M\_C1\_stm0\_c1)))

VS\_\_(id\_\_,

const\_pkg0\_M\_C1\_stm0\_c2,

const\_pkg0\_M\_C1\_stm0\_c1) = FVS\_\_(id\_\_,

const\_pkg0\_M\_C1\_stm0\_c2,

const\_pkg0\_M\_C1\_stm0\_c1)

VS\_O\_\_(id\_\_,

const\_pkg0\_M\_C1\_stm0\_c2,

const\_pkg0\_M\_C1\_stm0\_c1) = sbisim(diamond(FVS\_\_(id\_\_,

const\_pkg0\_M\_C1\_stm0\_c2,

const\_pkg0\_M\_C1\_stm0\_c1)))

HEXT\_\_(id\_\_,

const\_pkg0\_M\_C1\_stm0\_c2,

const\_pkg0\_M\_C1\_stm0\_c1) = O\_\_(id\_\_,

const\_pkg0\_M\_C1\_stm0\_c2,

const\_pkg0\_M\_C1\_stm0\_c1) [|shared\_variable\_events|] SKIP

HUP\_\_(id\_\_,

const\_pkg0\_M\_C1\_stm0\_c2,

const\_pkg0\_M\_C1\_stm0\_c1) = O\_\_(id\_\_,

const\_pkg0\_M\_C1\_stm0\_c2,

const\_pkg0\_M\_C1\_stm0\_c1) [|{share\_\_}|] SKIP

-- Shared memory

-- Shared memory variables

Memory\_m(m) =

get\_m!m -> Memory\_m(m)

[]

set\_m?x\_\_ -> Memory\_m(x\_\_)

[]

set\_EXT\_m?x\_\_ -> Memory\_m(x\_\_)

[]

terminate -> SKIP

Memory\_a1(a1) =

get\_a1!a1 -> Memory\_a1(a1)

[]

set\_a1?x\_\_ -> Memory\_a1(x\_\_)

[]

set\_EXT\_a1?x\_\_ -> Memory\_a1(x\_\_)

[]

terminate -> SKIP

Memory\_a3(a3) =

get\_a3!a3 -> Memory\_a3(a3)

[]

set\_a3?x\_\_ -> Memory\_a3(x\_\_)

[]

set\_EXT\_a3?x\_\_ -> Memory\_a3(x\_\_)

[]

terminate -> SKIP

-- sharedVarMemory process

sharedVarMemory(id\_\_,

const\_pkg0\_M\_C1\_stm0\_c2,

const\_pkg0\_M\_C1\_stm0\_c1) = Memory\_m(0)

[| { terminate } |] (

Memory\_a1((0,0))

[| { terminate } |] (

Memory\_a3(0)

)

)

sharedVarSync = {|get\_m,set\_m,set\_EXT\_m,get\_a1,set\_a1,set\_EXT\_a1,get\_a3,set\_a3,set\_EXT\_a3|}

sharedVarHide = {|get\_m,get\_a1,get\_a3|}

endmodule

## undefined operations

CALL\_\_move1(id\_\_,

param\_m,

param\_m1) = move1Call.param\_m.param\_m1 -> SKIP

CALL\_\_move2(id\_\_,

param\_lv,

param\_a) = move2Call.param\_lv.param\_a -> SKIP

## -- declaring controller memory Rule 20

Memory(id\_\_, cv1, m, a1, a3, c2) = (

set\_cv1?x\_\_ -> Memory(id\_\_,x\_\_,m,a1,a3,c2)

[]

set\_EXT\_m?x\_\_ -> stm0::set\_EXT\_m!x\_\_ -> stm1::set\_EXT\_m!x\_\_ ->

Memory(id\_\_,cv1,x\_\_,a1,a3,c2)

[]

set\_EXT\_a1?x\_\_ -> stm0::set\_EXT\_a1!x\_\_ -> stm1::set\_EXT\_a1!x\_\_ ->

Memory(id\_\_,cv1,m,x\_\_,a3,c2)

[]

set\_EXT\_a3?x\_\_ -> stm0::set\_EXT\_a3!x\_\_ -> stm1::set\_EXT\_a3!x\_\_ ->

Memory(id\_\_,cv1,m,a1,x\_\_,c2)

)

A screenshot of a computer program

Description automatically generated

## Controller process?

D\_\_(id\_\_,

const\_pkg0\_M\_C1\_c2,

const\_pkg0\_M\_C1\_stm0\_c1,

const\_pkg0\_M\_C1\_stm1\_c1) = ((

(

let

const\_pkg0\_M\_C1\_stm0\_c2 = const\_pkg0\_M\_C1\_c2

within

stm0::D\_\_(id\_\_,

const\_pkg0\_M\_C1\_stm0\_c2,

const\_pkg0\_M\_C1\_stm0\_c1)

[[

stm0::terminate <- terminate,

stm0::set\_m <- set\_m,

stm0::get\_m <- get\_m,

stm0::set\_a1 <- set\_a1,

stm0::get\_a1 <- get\_a1,

stm0::set\_a3 <- set\_a3,

stm0::get\_a3 <- get\_a3,

stm0::move1Call <- move1Call,

stm0::move2Call <- move2Call

]]

\ {|

stm0::stop,

stm0::event1,

stm0::event2,

stm0::trigger1

|}

)

[|{|terminate|}|]

(

let

const\_pkg0\_M\_C1\_stm0\_c1 = const\_pkg0\_M\_C1\_stm1\_c1

const\_pkg0\_M\_C1\_stm0\_c2 = const\_pkg0\_M\_C1\_c2

within

stm1::D\_\_(id\_\_,

const\_pkg0\_M\_C1\_stm0\_c2,

const\_pkg0\_M\_C1\_stm0\_c1)[[

stm1::terminate <- terminate,

stm1::set\_m <- set\_m,

stm1::get\_m <- get\_m,

stm1::set\_a1 <- set\_a1,

stm1::get\_a1 <- get\_a1,

stm1::set\_a3 <- set\_a3,

stm1::get\_a3 <- get\_a3,

stm1::move1Call <- move1Call,

stm1::move2Call <- move2Call

]]

\ {|

stm1::stop,

stm1::event1,

stm1::event2,

stm1::trigger1

|}

)

)

\diff(

{|terminate|},

{|terminate|}

)

[|

union(

{|

set\_cv1

|},

{|

stm0::set\_EXT\_m,stm0::set\_EXT\_a1,stm0::set\_EXT\_a3,

stm1::set\_EXT\_m,stm1::set\_EXT\_a1,stm1::set\_EXT\_a3

|}

) |]

Memory(id\_\_, 0, (0,0), 0, const\_pkg0\_M\_C1\_c2)

)

\ union(

{|set\_cv1, get\_cv1 |},

{|

stm0::set\_EXT\_m,stm0::set\_EXT\_a1,stm0::set\_EXT\_a3,

stm1::set\_EXT\_m,stm1::set\_EXT\_a1,stm1::set\_EXT\_a3

A close-up of a word

Description automatically generated |}

)

[|{|terminate|}|>SKIP

-- VS version

VS\_\_(id\_\_,

const\_pkg0\_M\_C1\_c2,

const\_pkg0\_M\_C1\_stm0\_c1,

const\_pkg0\_M\_C1\_stm1\_c1) = ((

(

let

const\_pkg0\_M\_C1\_stm0\_c2 = const\_pkg0\_M\_C1\_c2

within

stm0::VS\_\_(id\_\_,

const\_pkg0\_M\_C1\_stm0\_c2,

const\_pkg0\_M\_C1\_stm0\_c1)[[

stm0::terminate <- terminate,

stm0::set\_m <- set\_m,

stm0::get\_m <- get\_m,

stm0::set\_a1 <- set\_a1,

stm0::get\_a1 <- get\_a1,

stm0::set\_a3 <- set\_a3,

stm0::get\_a3 <- get\_a3,

stm0::move1Call <- move1Call,

stm0::move2Call <- move2Call

]]

\ {|

stm0::stop,

stm0::event1,

stm0::event2,

stm0::trigger1

|}

)

[|{|terminate|}|]

(

let

const\_pkg0\_M\_C1\_stm0\_c1 = const\_pkg0\_M\_C1\_stm1\_c1

const\_pkg0\_M\_C1\_stm0\_c2 = const\_pkg0\_M\_C1\_c2

within

stm1::VS\_\_(id\_\_,

const\_pkg0\_M\_C1\_stm0\_c2,

const\_pkg0\_M\_C1\_stm0\_c1)[[

stm1::terminate <- terminate,

stm1::set\_m <- set\_m,

stm1::get\_m <- get\_m,

stm1::set\_a1 <- set\_a1,

stm1::get\_a1 <- get\_a1,

stm1::set\_a3 <- set\_a3,

stm1::get\_a3 <- get\_a3,

stm1::move1Call <- move1Call,

stm1::move2Call <- move2Call

]]

\ {|

stm1::stop,

stm1::event1,

stm1::event2,

stm1::trigger1

|}

)

)

\diff(

{|terminate|},

{|terminate|}

)

[|

union(

{|

|},

{|

stm0::set\_EXT\_m,stm0::set\_EXT\_a1,stm0::set\_EXT\_a3,

stm1::set\_EXT\_m,stm1::set\_EXT\_a1,stm1::set\_EXT\_a3

|}

)

|]

Memory(id\_\_, 0, (0,0), 0, const\_pkg0\_M\_C1\_c2)

)

\ union(

{|

|},

{|

stm0::set\_EXT\_m,stm0::set\_EXT\_a1,stm0::set\_EXT\_a3,

stm1::set\_EXT\_m,stm1::set\_EXT\_a1,stm1::set\_EXT\_a3

|}

)

[|{|terminate|}|>SKIP

-- O version (optimised) THESE TWO PROCESSES SHOULD BE KEPT IN SYNC WITH THE ABOVE TWO

O\_\_(id\_\_,

const\_pkg0\_M\_C1\_c2,

const\_pkg0\_M\_C1\_stm0\_c1,

const\_pkg0\_M\_C1\_stm1\_c1) = sbisim(diamond( ((

(

let

const\_pkg0\_M\_C1\_stm0\_c2 = const\_pkg0\_M\_C1\_c2

within

stm0::O\_\_(id\_\_,

const\_pkg0\_M\_C1\_stm0\_c2,

const\_pkg0\_M\_C1\_stm0\_c1)[[

stm0::terminate <- terminate,

stm0::set\_m <- set\_m,

stm0::get\_m <- get\_m,

stm0::set\_a1 <- set\_a1,

stm0::get\_a1 <- get\_a1,

stm0::set\_a3 <- set\_a3,

stm0::get\_a3 <- get\_a3,

stm0::move1Call <- move1Call,

stm0::move2Call <- move2Call

]]

\ {|

stm0::stop,

stm0::event1,

stm0::event2,

stm0::trigger1

|}

)

[|{|terminate|}|]

(

let

const\_pkg0\_M\_C1\_stm0\_c1 = const\_pkg0\_M\_C1\_stm1\_c1

const\_pkg0\_M\_C1\_stm0\_c2 = const\_pkg0\_M\_C1\_c2

within

stm1::O\_\_(id\_\_,

const\_pkg0\_M\_C1\_stm0\_c2,

const\_pkg0\_M\_C1\_stm0\_c1)[[

stm1::terminate <- terminate,

stm1::set\_m <- set\_m,

stm1::get\_m <- get\_m,

stm1::set\_a1 <- set\_a1,

stm1::get\_a1 <- get\_a1,

stm1::set\_a3 <- set\_a3,

stm1::get\_a3 <- get\_a3,

stm1::move1Call <- move1Call,

stm1::move2Call <- move2Call

]]

\ {|

stm1::stop,

stm1::event1,

stm1::event2,

stm1::trigger1

|}

)

)

\diff(

{|terminate|},

{|terminate|}

)

[|

union(

{|

|},

{|

stm0::set\_EXT\_m,stm0::set\_EXT\_a1,stm0::set\_EXT\_a3,

stm1::set\_EXT\_m,stm1::set\_EXT\_a1,stm1::set\_EXT\_a3

|}

)

|]

dbisim(Memory(id\_\_, 0, (0,0), 0, const\_pkg0\_M\_C1\_c2))

)

\ union(

{|

|},

{|

stm0::set\_EXT\_m,stm0::set\_EXT\_a1,stm0::set\_EXT\_a3,

stm1::set\_EXT\_m,stm1::set\_EXT\_a1,stm1::set\_EXT\_a3

|}

)

[|{|terminate|}|>SKIP

))

-- VS\_O version

VS\_O\_\_(id\_\_,

const\_pkg0\_M\_C1\_c2,

const\_pkg0\_M\_C1\_stm0\_c1,

const\_pkg0\_M\_C1\_stm1\_c1) = sbisim(diamond( ((

(

let

const\_pkg0\_M\_C1\_stm0\_c2 = const\_pkg0\_M\_C1\_c2

within

stm0::VS\_O\_\_(id\_\_,

const\_pkg0\_M\_C1\_stm0\_c2,

const\_pkg0\_M\_C1\_stm0\_c1)[[

stm0::terminate <- terminate,

stm0::set\_m <- set\_m,

stm0::get\_m <- get\_m,

stm0::set\_a1 <- set\_a1,

stm0::get\_a1 <- get\_a1,

stm0::set\_a3 <- set\_a3,

stm0::get\_a3 <- get\_a3,

stm0::move1Call <- move1Call,

stm0::move2Call <- move2Call

]]

\ {|

stm0::stop,

stm0::event1,

stm0::event2,

stm0::trigger1

|}

)

[|{|terminate|}|]

(

let

const\_pkg0\_M\_C1\_stm0\_c1 = const\_pkg0\_M\_C1\_stm1\_c1

const\_pkg0\_M\_C1\_stm0\_c2 = const\_pkg0\_M\_C1\_c2

within

stm1::VS\_O\_\_(id\_\_,

const\_pkg0\_M\_C1\_stm0\_c2,

const\_pkg0\_M\_C1\_stm0\_c1)[[

stm1::terminate <- terminate,

stm1::set\_m <- set\_m,

stm1::get\_m <- get\_m,

stm1::set\_a1 <- set\_a1,

stm1::get\_a1 <- get\_a1,

stm1::set\_a3 <- set\_a3,

stm1::get\_a3 <- get\_a3,

stm1::move1Call <- move1Call,

stm1::move2Call <- move2Call

]]

\ {|

stm1::stop,

stm1::event1,

stm1::event2,

stm1::trigger1

|}

)

)

\diff(

{|terminate|},

{|terminate|}

)

[|

union(

{|

|},

{|

stm0::set\_EXT\_m,stm0::set\_EXT\_a1,stm0::set\_EXT\_a3,

stm1::set\_EXT\_m,stm1::set\_EXT\_a1,stm1::set\_EXT\_a3

|}

)

|]

dbisim(Memory(id\_\_, 0, (0,0), 0, const\_pkg0\_M\_C1\_c2))

)

\ union(

{|

|},

{|

stm0::set\_EXT\_m,stm0::set\_EXT\_a1,stm0::set\_EXT\_a3,

stm1::set\_EXT\_m,stm1::set\_EXT\_a1,stm1::set\_EXT\_a3

|}

)

[|{|terminate|}|>SKIP

))

HEXT(id\_\_,

const\_pkg0\_M\_C1\_c2,

const\_pkg0\_M\_C1\_stm0\_c1,

const\_pkg0\_M\_C1\_stm1\_c1) = O\_\_(id\_\_,

const\_pkg0\_M\_C1\_c2,

const\_pkg0\_M\_C1\_stm0\_c1,

const\_pkg0\_M\_C1\_stm1\_c1) [|shared\_variable\_events|] SKIP

endmodule

## -- declaring module memory Rule 12

Memory(id\_\_, pv1, a1, a3, m, c2) = (

set\_pv1?x\_\_ ->

Memory(id\_\_,x\_\_,a1,a3,m,c2)

[]

set\_a1?x\_\_ ->

C1::set\_EXT\_a1!x\_\_ ->

Memory(id\_\_,pv1,x\_\_,a3,m,c2)

[]

set\_a3?x\_\_ ->

C1::set\_EXT\_a3!x\_\_ ->

Memory(id\_\_,pv1,a1,x\_\_,m,c2)

[]

set\_m?x\_\_ ->

C1::set\_EXT\_m!x\_\_ ->

Memory(id\_\_,pv1,a1,a3,x\_\_,c2)

)

## Module process RULe 1

D\_\_(id\_\_,

const\_pkg0\_M\_P1\_c2,

const\_pkg0\_M\_C1\_stm0\_c1,

const\_pkg0\_M\_C1\_stm1\_c1) = (

(

(SKIP)

[|{||}|]

(

(

let

const\_pkg0\_M\_C1\_c2 = const\_pkg0\_M\_P1\_c2

within

C1::D\_\_(id\_\_,

const\_pkg0\_M\_C1\_c2,

const\_pkg0\_M\_C1\_stm0\_c1,

const\_pkg0\_M\_C1\_stm1\_c1)

[[

C1::terminate <- terminate,

C1::set\_a1 <- set\_a1,

C1::get\_a1 <- get\_a1,

C1::set\_a3 <- set\_a3,

C1::get\_a3 <- get\_a3,

C1::set\_m <- set\_m,

C1::get\_m <- get\_m,

C1::move1Call <- move1Call,

C1::move2Call <- move2Call

]]

)

[|

union(

{|

set\_pv1,

set\_a1,

set\_a3,

set\_m

|},

{|

C1::set\_EXT\_m,C1::set\_EXT\_a1,C1::set\_EXT\_a3

|}

)

|]

Memory(id\_\_, 0, (0,0), 0, 0, const\_pkg0\_M\_P1\_c2)

)

)

\ Union({

{|

|},

{|

C1::set\_EXT\_m,C1::set\_EXT\_a1,C1::set\_EXT\_a3

|},

{|

get\_pv1, set\_pv1,

get\_a1, set\_a1,

get\_a3, set\_a3,

get\_m, set\_m

|}

})

[|{|terminate|}|>SKIP

)

\{|terminate|}

-- visible state equivalent

VS\_\_(id\_\_,

const\_pkg0\_M\_P1\_c2,

const\_pkg0\_M\_C1\_stm0\_c1,

const\_pkg0\_M\_C1\_stm1\_c1) = (

(

(SKIP)

[|{||}|]

(

(

let

const\_pkg0\_M\_C1\_c2 = const\_pkg0\_M\_P1\_c2

within

C1::VS\_\_(id\_\_,

const\_pkg0\_M\_C1\_c2,

const\_pkg0\_M\_C1\_stm0\_c1,

const\_pkg0\_M\_C1\_stm1\_c1)[[

C1::terminate <- terminate,

C1::set\_a1 <- set\_a1,

C1::get\_a1 <- get\_a1,

C1::set\_a3 <- set\_a3,

C1::get\_a3 <- get\_a3,

C1::set\_m <- set\_m,

C1::get\_m <- get\_m,

C1::move1Call <- move1Call,

C1::move2Call <- move2Call

]]

)

[|

union(

{|

set\_pv1,

set\_a1,

set\_a3,

set\_m

|},

{|

C1::set\_EXT\_m,C1::set\_EXT\_a1,C1::set\_EXT\_a3

|}

)

|]

Memory(id\_\_, 0, (0,0), 0, 0, const\_pkg0\_M\_P1\_c2)

)

)

\ Union({

{|

|},

{|

C1::set\_EXT\_m,C1::set\_EXT\_a1,C1::set\_EXT\_a3

|},

{|

get\_pv1, set\_pv1,

get\_a1, set\_a1,

get\_a3, set\_a3,

get\_m, set\_m

|}

})

[|{|terminate|}|>SKIP

)

\{|terminate|}

-- O version (optimised)

O\_\_(id\_\_,

const\_pkg0\_M\_P1\_c2,

const\_pkg0\_M\_C1\_stm0\_c1,

const\_pkg0\_M\_C1\_stm1\_c1) = sbisim(diamond( (

(

(SKIP)

[|{||}|]

(

(

let

const\_pkg0\_M\_C1\_c2 = const\_pkg0\_M\_P1\_c2

within

C1::O\_\_(id\_\_,

const\_pkg0\_M\_C1\_c2,

const\_pkg0\_M\_C1\_stm0\_c1,

const\_pkg0\_M\_C1\_stm1\_c1)[[

C1::terminate <- terminate,

C1::set\_a1 <- set\_a1,

C1::get\_a1 <- get\_a1,

C1::set\_a3 <- set\_a3,

C1::get\_a3 <- get\_a3,

C1::set\_m <- set\_m,

C1::get\_m <- get\_m,

C1::move1Call <- move1Call,

C1::move2Call <- move2Call

]]

)

[|

union(

{|

set\_pv1,

set\_a1,

set\_a3,

set\_m

|},

{|

C1::set\_EXT\_m,C1::set\_EXT\_a1,C1::set\_EXT\_a3

|}

)

|]

dbisim(Memory(id\_\_, 0, (0,0), 0, 0, const\_pkg0\_M\_P1\_c2))

)

)

\ Union({

{|

|},

{|

C1::set\_EXT\_m,C1::set\_EXT\_a1,C1::set\_EXT\_a3

|},

{|

get\_pv1, set\_pv1,

get\_a1, set\_a1,

get\_a3, set\_a3,

get\_m, set\_m

|}

})

[|{|terminate|}|>SKIP

)

\{|terminate|}

))

-- visible state optimised equivalent

VS\_O\_\_(id\_\_,

const\_pkg0\_M\_P1\_c2,

const\_pkg0\_M\_C1\_stm0\_c1,

const\_pkg0\_M\_C1\_stm1\_c1) = sbisim(diamond( (

(

(SKIP)

[|{||}|]

(

(

let

const\_pkg0\_M\_C1\_c2 = const\_pkg0\_M\_P1\_c2

within

C1::VS\_O\_\_(id\_\_,

const\_pkg0\_M\_C1\_c2,

const\_pkg0\_M\_C1\_stm0\_c1,

const\_pkg0\_M\_C1\_stm1\_c1)[[

C1::terminate <- terminate,

C1::set\_a1 <- set\_a1,

C1::get\_a1 <- get\_a1,

C1::set\_a3 <- set\_a3,

C1::get\_a3 <- get\_a3,

C1::set\_m <- set\_m,

C1::get\_m <- get\_m,

C1::move1Call <- move1Call,

C1::move2Call <- move2Call

]]

)

[|

union(

{|

set\_pv1,

set\_a1,

set\_a3,

set\_m

|},

{|

C1::set\_EXT\_m,C1::set\_EXT\_a1,C1::set\_EXT\_a3

|}

)

|]

dbisim(Memory(id\_\_, 0, (0,0), 0, 0, const\_pkg0\_M\_P1\_c2))

)

)

\ Union({

{|

|},

{|

C1::set\_EXT\_m,C1::set\_EXT\_a1,C1::set\_EXT\_a3

|},

{|

get\_pv1, set\_pv1,

get\_a1, set\_a1,

get\_a3, set\_a3,

get\_m, set\_m

|}

})

[|{|terminate|}|>SKIP

)

\{|terminate|}

))

-- O version (optimised) with visible assignments

AS\_O\_\_(id\_\_,

const\_pkg0\_M\_P1\_c2,

const\_pkg0\_M\_C1\_stm0\_c1,

const\_pkg0\_M\_C1\_stm1\_c1) = sbisim(diamond( (

(

(SKIP)

[|{||}|]

(

(

let

const\_pkg0\_M\_C1\_c2 = const\_pkg0\_M\_P1\_c2

within

C1::O\_\_(id\_\_,

const\_pkg0\_M\_C1\_c2,

const\_pkg0\_M\_C1\_stm0\_c1,

const\_pkg0\_M\_C1\_stm1\_c1)[[

C1::terminate <- terminate,

C1::set\_a1 <- set\_a1,

C1::get\_a1 <- get\_a1,

C1::set\_a3 <- set\_a3,

C1::get\_a3 <- get\_a3,

C1::set\_m <- set\_m,

C1::get\_m <- get\_m,

C1::move1Call <- move1Call,

C1::move2Call <- move2Call

]]

)

[|

union(

{|

set\_pv1,

set\_a1,

set\_a3,

set\_m

|},

{|

C1::set\_EXT\_m,C1::set\_EXT\_a1,C1::set\_EXT\_a3

|}

)

|]

dbisim(Memory(id\_\_, 0, (0,0), 0, 0, const\_pkg0\_M\_P1\_c2))

)

)

\ Union({

{|

|},

{|

C1::set\_EXT\_m,C1::set\_EXT\_a1,C1::set\_EXT\_a3

|},

{|

get\_pv1,

get\_a1,

get\_a3,

get\_m

|}

})

[|{|terminate|}|>SKIP

)

\{|terminate|}

))

Endmodule