Object-Z

is a specification language extending Z so as to facilitate the specification of systems in an object-oriented style.

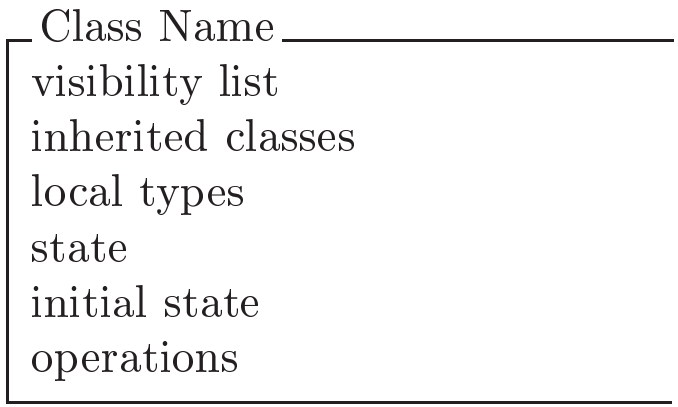
The view is taken that systems are composed of communicating objects.

When specifying a system in Object-Z,

- identify and specify the underlying objects;
- specify the system in terms of the communication between the underlying objects.

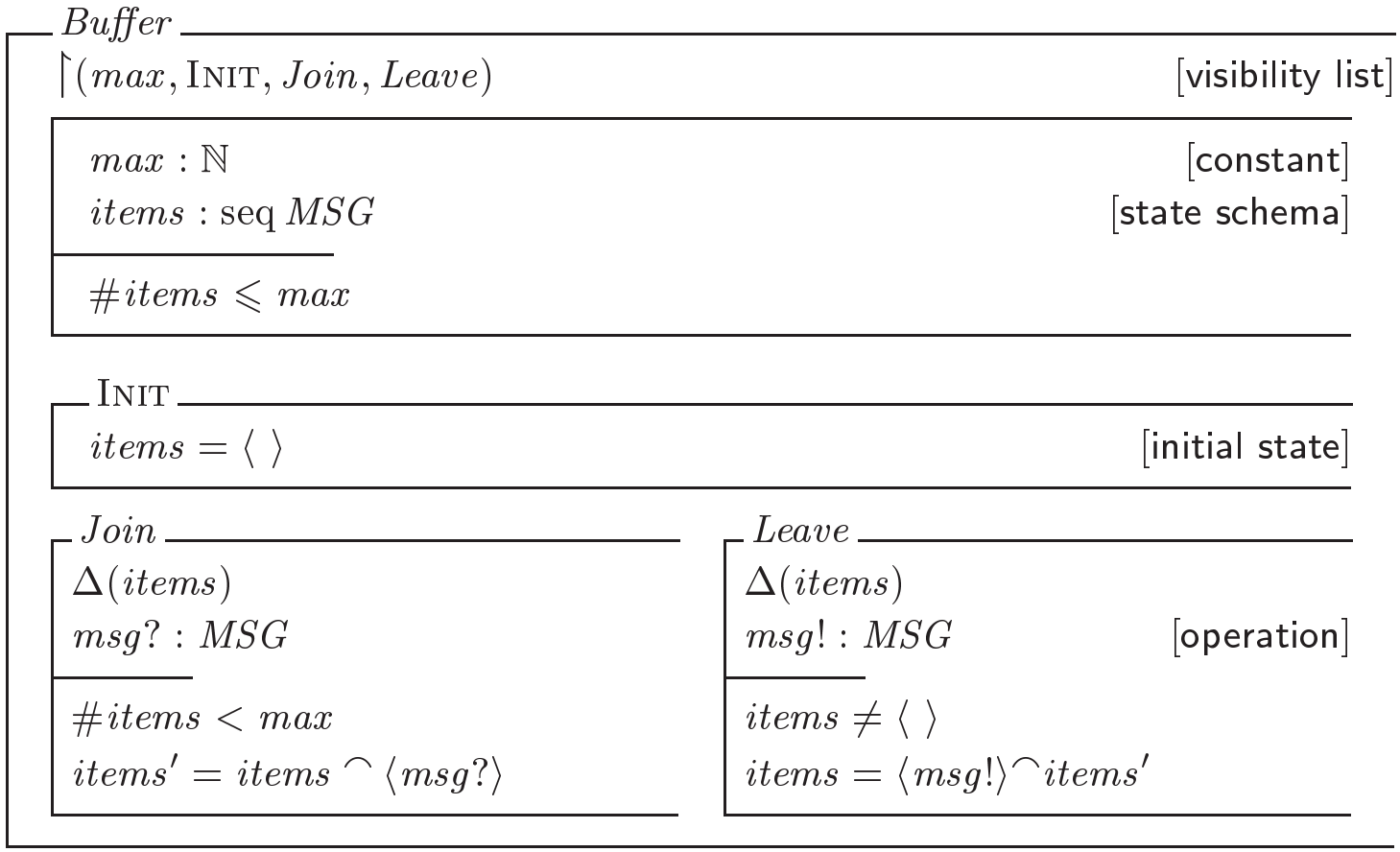
An object may itself be a system of communicating objects.

The Class Construct: Encapsulation



- the class construct encapsulates all relevant features; it is like a template from which objects of the class can be stamped
- the visibility list specifies the interface between object of the class and their environment
- a class incorporates all the features of its inherited classes
- local types have the syntax of Z global types
- the state, initial state and operations have a syntax based on that for Z schemas
- variables declared in the state are called **attributes**
- an instance of a class is an assignment of values to attributes consistent with the state; at any time an object of a class will have an associated value which is some instance of the class

Object-Z Case Study: A Buffer of Messages



- the state is an unnamed schema
- the state schema is implicitly merged into the initial schema

INIT

$$max : \mathbb{N}; \quad items : \operatorname{seq} MSG$$

$$\#items \leqslant max \land items = \langle\ \rangle$$

- the state schema in both primed and unprimed form is implicity merged into each operation schema

Join

$$max, max' : \mathbb{N}; \quad items, items' : \operatorname{seq} MSG$$
$$msg? : MSG$$

$$\#items \leqslant max \land \#items' \leqslant max \land \#items < max$$
$$items' = items \frown \langle msg? \rangle \land max' = max$$

- the Δ convention is modified: only attributes that may change are listed—attributes not listed do not change

Inheritance

LossyBuffer

Buffer

Lose

$\Delta(items)$

$items \neq \langle\,\rangle$

$items' = \text{tail}\ items$

LossyBuffer

$max : \mathbb{N};\quad items : \text{seq}\ MSG$

$\#items \leqslant max$

INIT

$items = \langle\,\rangle$

Join

...

Leave

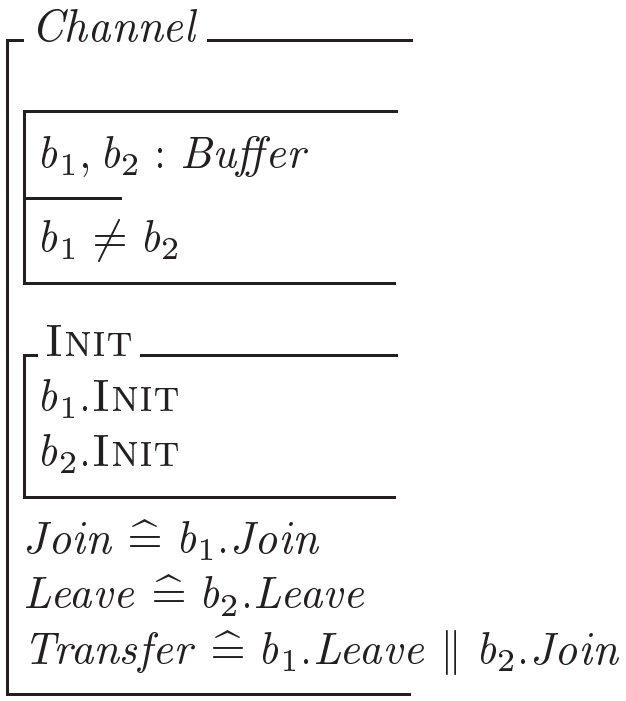
...

Lose

$\Delta(items)$

$items \neq \langle\,\rangle \land items' = \text{tail}\ items$

Instantiation and Communication



- an object is a variable of class type — objects are instantiations of classes
- objects have **integrity** — change state via class operations only
- objects have **persistence** — exist from creation to deallocation
- objects communicate by message passing — engage in cooperative operations

The initial schema of *Channel* is equivalent to:

INIT

$$b_1.items = \langle\,\rangle \land b_2.items = \langle\,\rangle$$

The operations of *Channel* are equivalent to:

Join

$$msg? : MSG$$

$$open(b_1)$$
$$\#b_1.items < b_1.max$$
$$b_1.items' = b_1.items \frown \langle msg? \rangle$$
$$b_1.max' = b_1.max$$

Leave

$$msg! : MSG$$

$$open(b_2)$$
$$b_2.items \neq \langle\,\rangle$$
$$b_2.items = \langle msg! \rangle \frown b_2.items'$$
$$b_2.max' = b_2.max$$

Transfer

$$\exists\, msg : MSG \bullet b_1.Leave[msg/msg!] \parallel b_2.Join[msg/msg?]$$

or

Transfer

$$open(b_1, b_2)$$
$$b_1.items \neq \langle\,\rangle$$
$$\#b_2.items < b_2.max$$
$$\exists\, msg : MSG \bullet b_1.items = \langle msg \rangle \frown b_1.items' \wedge b_2.items' = b_2.items \frown \langle msg \rangle$$
$$b_1.max' = b_1.max$$
$$b_2.max' = b_2.max$$

In general

- $a.op$
  denotes the operation op performed upon object a; the operation op must be one of the operations specified in the class of a

- $a.op_1 \parallel b.op_2$
  denotes the operation op_1 performed upon object a, in parallel with the operation op_2 performed upon object b; inputs and outputs having the same base name (i.e. apart from the '?' and '!') are identified (equated) and hidden

Aggregation and Identity

BufferSystem

$$buffers : \mathbb{P}\, Buffer$$

INIT

$$buffers = \varnothing$$

AddBuffer

$$\Delta(buffers)$$
$$b? : Buffer$$
$$b? \notin buffers \land b?.\text{INIT}$$
$$buffers' = buffers \cup \{b?\}$$

RemoveBuffer

$$\Delta(buffers)$$
$$b? : Buffer$$
$$b? \in buffers$$
$$buffers' = buffers - \{b?\}$$

SelectBuffer

$$b? : Buffer$$
$$b? \in buffers$$

Select2Buffers

$$b_1?, b_2? : Buffer$$
$$\{b_1?, b_2?\} \subseteq buffers \land b_1? \neq b_2?$$

$$Join \mathrel{\widehat=} SelectBuffer \bullet b?.Join$$
$$Leave \mathrel{\widehat=} SelectBuffer \bullet b?.Leave$$
$$Transfer \mathrel{\widehat=} Select2Buffers \bullet b_1?.Leave \parallel b_2?.Join$$

The operations *Join* and *Transfer* are equivalent to:

Join

$b? : Buffer$

$msg? : MSG$

$open(b?)$

$b? \in buffers$

$\#b?.items < b?.max$

$b?.items' = b?.items \frown \langle msg? \rangle$

$b?.max' = b?.max$

Transfer

$b_1?, b_2? : Buffer$

$open(b_1?, b_2?)$

$\{b_1?, b_2?\} \subseteq buffers$

$b_1? \neq b_2?$

$b_1?.items \neq \langle\ \rangle$

$\#b_2?.items < b_2?.max$

$\exists\, msg : MSG \bullet$

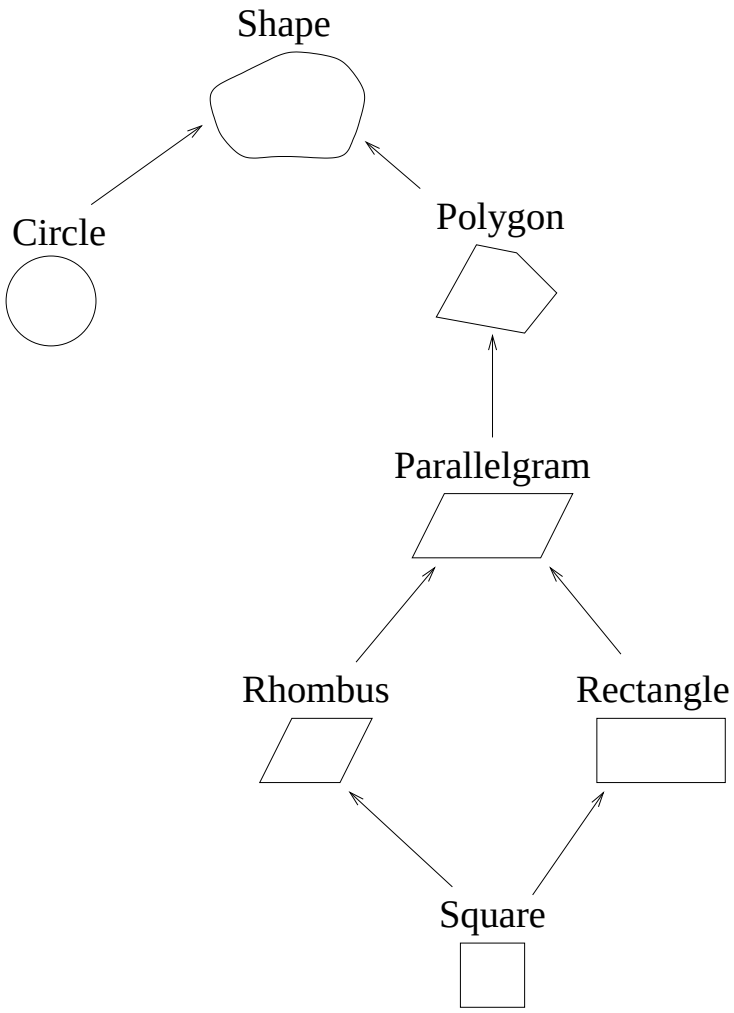
$\quad b_1?.items = \langle msg \rangle \frown b_1?.items'$

$\quad b_2?.items' = b_2?.items \frown \langle msg \rangle$

$b_1?.max' = b_1?.max$

$b_2?.max' = b_2?.max$

Object-Z Case Study: A Shapes Hierarchy



$$Vector == \mathbb{R} \times \mathbb{R}$$

$$\_ + \_ : Vector \times Vector \rightarrow Vector$$

$$|\_| : Vector \rightarrow \mathbb{R}$$

$$\_ \perp \_ : Vector \leftrightarrow Vector$$

Shape

$$refpoint : Vector$$
$$perim : \mathbb{R}$$

$$perim > 0$$

Translate

$$\Delta(refpoint)$$
$$v? : Vector$$

$$refpoint' = refpoint + v?$$

Circle

$Shape[centre/refpoint, circum/perim]$

$radius : \mathbb{R}$

$circum = 2\,\pi\ radius$

i.e.

Circle

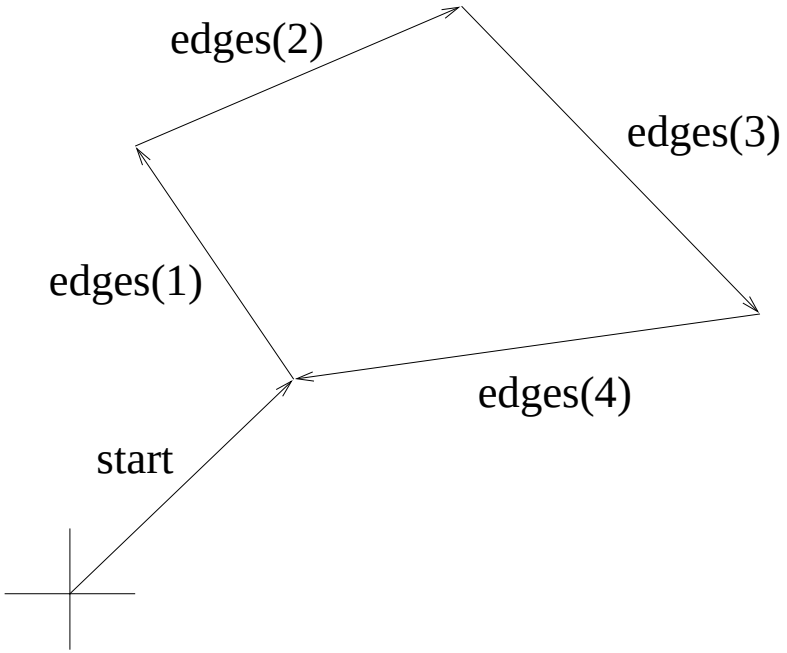
$centre : Vector$
$circum : \mathbb{R}$
$radius : \mathbb{R}$

$circum > 0$
$circum = 2\,\pi\ radius$

Translate

$\Delta(centre)$
$v? : Vector$

$centre' = centre + v?$



Polygon

Shape[*start*/*refpoint*]

$edges : \text{seq}\ Vector$

$\#edges \geqslant 3$

$\mathbf{O} \notin \text{ran}\ edges$

$(\sum i : \text{dom}\ edges \bullet edges(i)) = \mathbf{O}$

$perim = \sum i : \text{dom}\ edges \bullet |\ edges(i)\ |$

... [connectivity ...]

Parallelogram

Polygon

$\#edges = 4$

$edges(1) + edges(3) = \mathbf{O}$

Rhombus

Parallelogram

$|\, edges(1)\,| = |\, edges(2)\,|$

Rectangle

Parallelogram

$edges(1) \perp edges(2)$

Square

Rhombus

Rectangle

Expanding *Square* gives

Square

$$
\begin{array}{l}
start : Vector \\
perim : \mathbb{R} \\
edges : \operatorname{seq} Vector \\
\hline
\#edges = 4 \land \mathbf{O} \notin \operatorname{ran} edges \\
(\sum i : \operatorname{dom} edges \bullet edges(i)) = \mathbf{O} \\
perim = \sum i : \operatorname{dom} edges \bullet |\, edges(i) \,| \\
edges(1) + edges(3) = \mathbf{O} \\
|\, edges(1) \,| = |\, edges(2) \,| \land edges(1) \perp edges(2)
\end{array}
$$

Translate

$$
\begin{array}{l}
\Delta(start) \\
v? : Vector \\
\hline
start' = start + v?
\end{array}
$$

Figure

$shapes : \mathbb{P} \downarrow Shape$
$totalperim : \mathbb{R}$

$totalperim = \sum s : shapes \bullet s.perim$

SelectShape

$s? : \downarrow Shape$

$s? \in shapes$

$ShapeTranslate \mathrel{\widehat=} SelectShape \bullet s?.Translate$
$FigureTranslate \mathrel{\widehat=} \parallel s : shapes \bullet s.Translate$

i.e.

FigureTranslate

$v? : Vector$

$\forall s : shapes \bullet s.Translate$

FigureTranslate

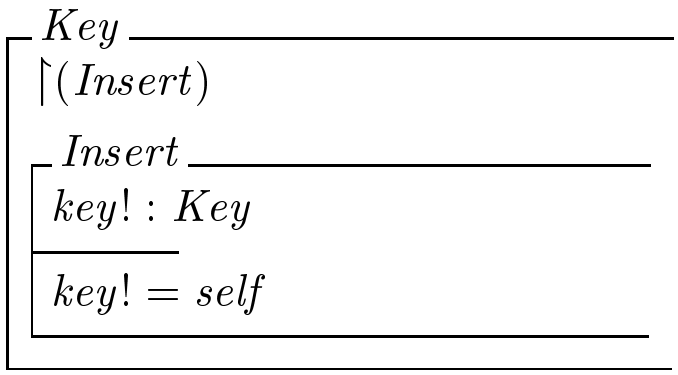
$v? : Vector$

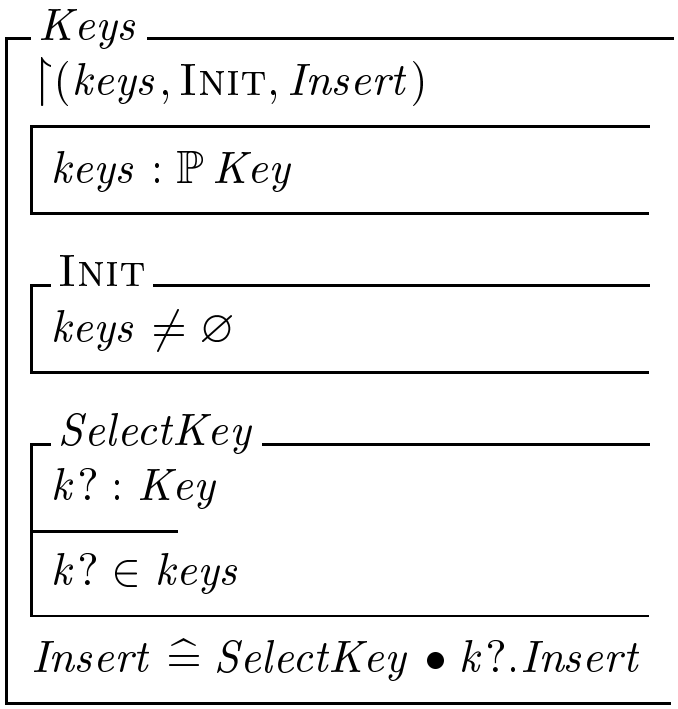
$\forall s : shapes \bullet$
$\quad open(s)$
$\quad s.refpoint' = s.refpoint + v?$
$\quad s.perim' = s.perim$

Object-Z Case Study: An Electronic Key System

Informal Description

- there is a fixed set of magnetic keys
- there is a fixed set of rooms
- each key has access to a subset of these rooms
- a room may be added to the set accessed by a key
- a room may be removed from the set accessed by a key


Key
↾(Insert)
Insert
key! : Key
key! = self


Keys
↾(keys, INIT, Insert)
keys : ℙ Key
INIT
keys ≠ ∅
SelectKey
k? : Key
k? ∈ keys
Insert ≙ SelectKey • k?.Insert

Room

$\restriction(\textsc{Init}, InsertedAndUnlock, Lock)$

$Status ::= locked \mid unlocked$

$status : Status$

Init

$status = locked$

Inserted

$room! : Room$

$room! = self$

Unlock

$\Delta(status)$

$status = locked \land status' = unlocked$

$InsertedAndUnlock \mathrel{\widehat=} Inserted \land Unlock$

Lock

$\Delta(status)$

$status = unlocked \land status' = locked$

Rooms

$\upharpoonright(rooms, \textsc{Init}, Unlock, Lock)$

$rooms : \mathbb{P}\, Room$

INIT

$rooms \neq \varnothing$

$\forall\, r : rooms \bullet r.\textsc{Init}$

SelectRoom

$r? : Room$

$r? \in rooms$

$Unlock \mathrel{\widehat=} SelectRoom \bullet r?.InsertedAndUnlock$

$Lock \mathrel{\widehat=} SelectRoom \bullet r?.Lock$

DataBase

$\restriction (access, \textsc{Init}, AuthorizeAccess, RescindAccess, CheckAccess)$

$$access : Key \leftrightarrow Room$$

Init

$$access = \varnothing$$

AuthorizeAccess

$$\Delta(access)$$
$$key? : Key$$
$$room? : Room$$

$$\neg\ (key?\ \underline{access}\ room?)$$
$$access' = access \cup \{(key?, room?)\}$$

RescindAccess

$$\Delta(access)$$
$$key? : Key$$
$$room? : Room$$

$$key?\ \underline{access}\ room?$$
$$access' = access - \{(key?, room?)\}$$

CheckAccess

$$key? : Key$$
$$room? : Room$$

$$key?\ \underline{access}\ room?$$

KeySystem

$keys : Keys$
$rooms : Rooms$
$database : DataBase$

$database.access \subseteq keys.keys \times rooms.rooms$

INIT

$keys.\text{INIT} \land rooms.\text{INIT} \land database.\text{INIT}$

$AuthorizeAccess \mathrel{\widehat{=}} database.AuthorizeAccess$
$RescindAccess \mathrel{\widehat{=}} database.RescindAccess$
$Unlock \mathrel{\widehat{=}} (keys.Insert \land rooms.Unlock)$
$\quad\quad\quad\quad \parallel$
$\quad\quad\quad\quad database.CheckAccess$
$Lock \mathrel{\widehat{=}} rooms.Lock$

Case Study: The Game of Tic Tac Toe

An Informal View:

- there are two players and a board
- the board consists of 9 positions in a 3×3 array
- initially all positions are unoccupied
- the players take it in turns to occupy unoccupied positions
- the first player to occupy three positions in a horizontal, vertical or diagonal row is the winner

A Formal Description – Initial Abstractions

A Data Structure for the Board

$$Posn == 1 \,..\, 3 \times 1 \,..\, 3$$

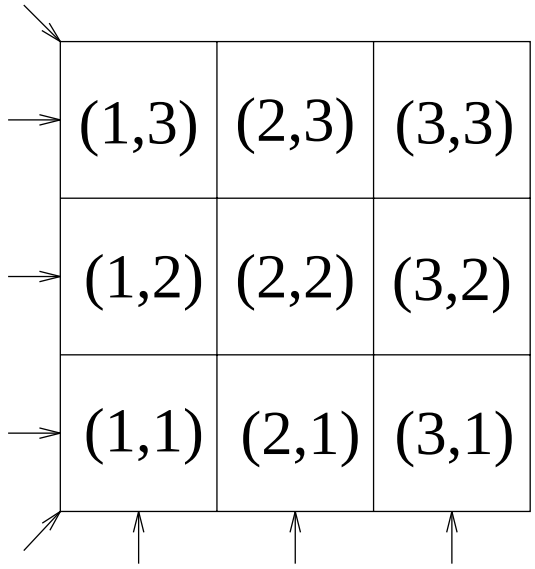
the abstraction:

(1,3)	(2,3)	(3,3)
(1,2)	(2,2)	(3,2)
(1,1)	(2,1)	(3,1)

Three in a Row (1st Abstraction)

$$3InRow : \mathbb{P}\, Posn \to \mathbb{B}$$

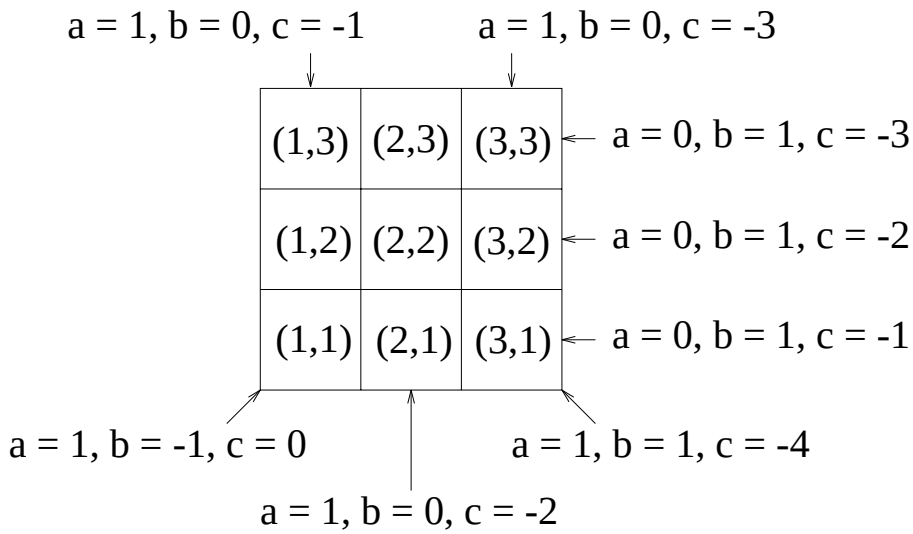
$$\begin{aligned}
&\forall\, ps : \mathbb{P}\, Posn \bullet \\
&\quad 3InRow(ps) \Leftrightarrow \\
&\qquad \{(1,1),(1,2),(1,3)\} \subseteq ps \\
&\qquad \lor \{(2,1),(2,2),(2,3)\} \subseteq ps \\
&\qquad \lor \{(3,1),(3,2),(3,3)\} \subseteq ps \\
&\qquad \lor \{(1,1),(2,1),(3,1)\} \subseteq ps \\
&\qquad \lor \{(1,2),(2,2),(3,2)\} \subseteq ps \\
&\qquad \lor \{(1,3),(2,3),(3,3)\} \subseteq ps \\
&\qquad \lor \{(1,1),(2,2),(3,3)\} \subseteq ps \\
&\qquad \lor \{(1,3),(2,2),(3,1)\} \subseteq ps
\end{aligned}$$



Three in a Row (2nd Abstraction)

$$3InRow : \mathbb{P}\, Posn \rightarrow \mathbb{B}$$

$$\forall\, ps : \mathbb{P}\, Posn \bullet$$
$$\quad 3InRow(ps) \;\Leftrightarrow\; \exists\, a, b, c : \mathbb{Z} \bullet$$
$$\qquad \{a, b, c\} \neq \{0\}$$
$$\qquad \#\{(x, y) : ps \mid ax + by + c = 0\} = 3$$



$$Colour \;::=\; black \mid white$$

Board

$$bposn, wposn : \mathbb{P}\, Posn$$
$$turn : Colour$$
$$bposn \cap wposn = \varnothing$$

INIT

$$bposn = \varnothing$$
$$wposn = \varnothing$$
$$turn = black$$

BlackMove

$$\Delta(bposn, turn); \quad p? : Posn$$
$$\neg\ 3InRow(wposn)$$
$$p? \notin bposn \cup wposn$$
$$bposn' = bposn \cup \{p?\}$$
$$turn = black \land turn' = white$$

WhiteMove

$$\Delta(wposn, turn); \quad p? : Posn$$
$$\neg\ 3InRow(bposn)$$
$$p? \notin bposn \cup wposn$$
$$wposn' = wposn \cup \{p?\}$$
$$turn = white \land turn' = black$$

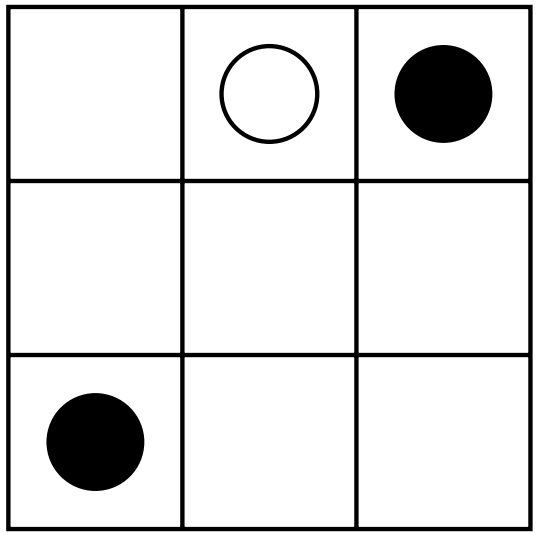
TheWinner

$$winner! : Colour$$
$$3InRow(bposn) \ \lor \ 3InRow(wposn)$$
$$3InRow(bposn) \ \Rightarrow \ winner! = black \land 3InRow(wposn) \ \Rightarrow \ winner! = white$$

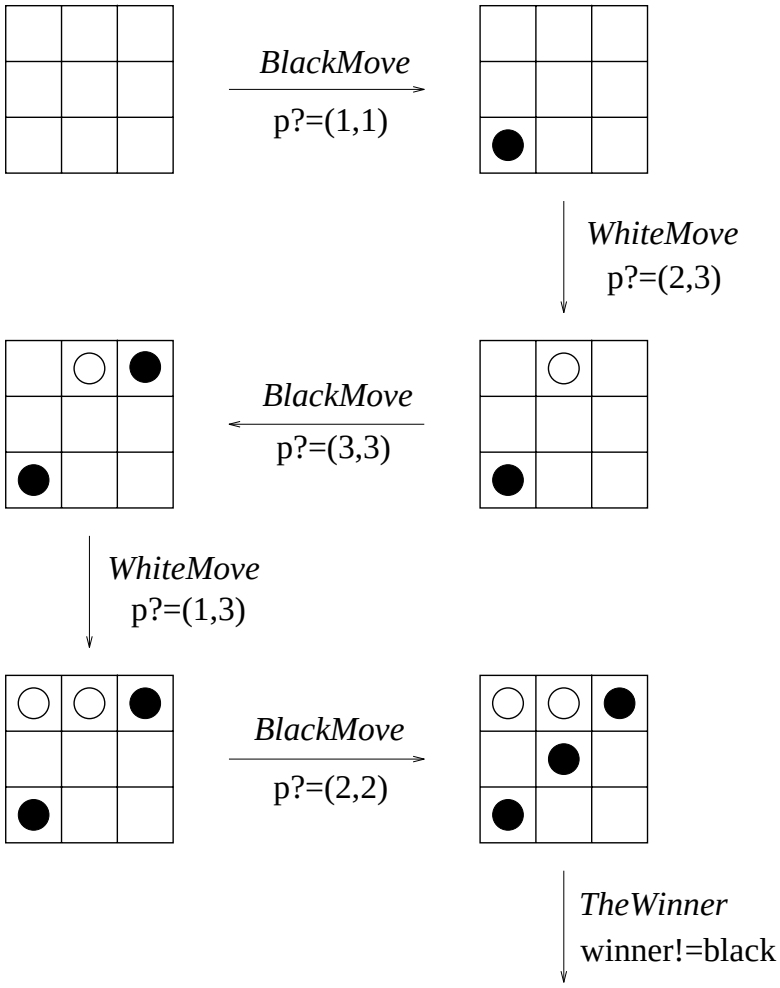
Board Example: A Static View

$bposn = \{(1,1),(3,3)\}$
$wposn = \{(2,3)\}$
$turn = white$

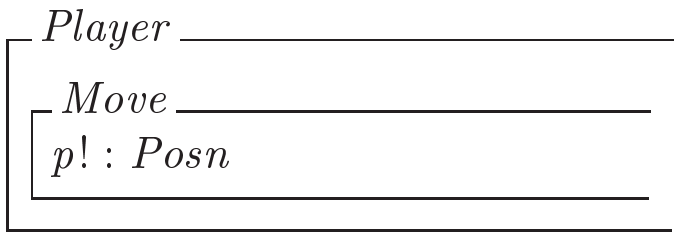
$3InRow(bposn) = false$
$3InRow(wposn) = false$
$bposn \cap wposn = \varnothing$
$bposn \cup wposn \neq Posn$



Board Example: A Dynamic View



The Player Class



The Game

TicTacToe

$bplayer, wplayer : Player$
$board : Board$

INIT
$board.\text{INIT}$

$BlackMove \mathrel{\widehat=} bplayer.Move \parallel board.BlackMove$
$WhiteMove \mathrel{\widehat=} wplayer.Move \parallel board.WhiteMove$
$TheWinner \mathrel{\widehat=} board.TheWinner$

The Game Communication

	bplayer	*board*	*wplayer*
BlackMove	$p! : Posn$	$p? : Posn$	
WhiteMove		$p? : Posn$	$p! : Posn$
TheWinner		$winner! : Colour$	

Operation Expressions

promotion

$$object.operation$$

indicates that the named object undergoes the named operation. This operation must be specified in the class of the object.

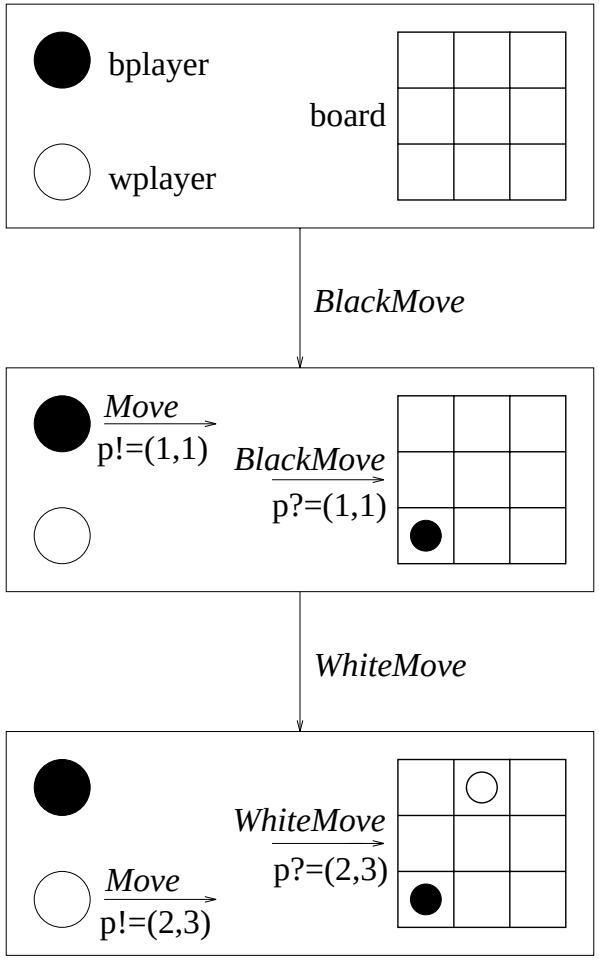
parallel operator

$$object_1.operation_1 \parallel object_2.operation_2$$

indicates that the objects synchronise, with output from the first component being identified with the same named input (apart from the ? and ! decorations) to the second component; this communication is hidden (internal).

All other input/output is with the environment.

Game Example: A Dynamic View



Extending Player by Inheritance

$Player_+$

$Player$

$board : Board$

$Move$

$p! \notin board.bposn \cup board.wposn$

which expands to

$Player_+$

$board : Board$

$Move$

$p! : Posn$

$p! \notin board.bposn \cup board.wposn$

The Game Revisited

TicTacToe

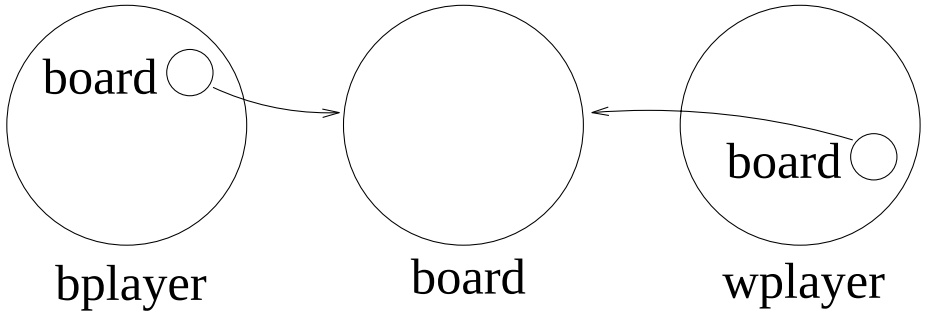
$bplayer, wplayer : Player_+$
$board : Board$

$bplayer.board = board = wplayer.board$

INIT
$board.\text{INIT}$

$BlackMove \mathrel{\widehat{=}} bplayer.Move \parallel board.BlackMove$
$WhiteMove \mathrel{\widehat{=}} wplayer.Move \parallel board.WhiteMove$
$TheWinner \mathrel{\widehat{=}} board.TheWinner$

The Object Structure



Specializing Player by Inheritance

Bplayer

$Player_+$

Move

$\neg\ 3InRow(board.wposn)$

$board.turn = black$

Wplayer

$Player_+$

Move

$\neg\ 3InRow(board.bposn)$

$board.turn = white$

e.g. *Wplayer* expands to

Wplayer

$board : Board$

Move

$p! : Posn$

$\neg\ 3InRow(board.bposn)$

$p! \notin board.bposn \cup board.wposn$

$board.turn = white$

The Game Again

TicTacToe

$bplayer : Bplayer$
$wplayer : Wplayer$
$board : Board$

$bplayer.board = board$
$wplayer.board = board$

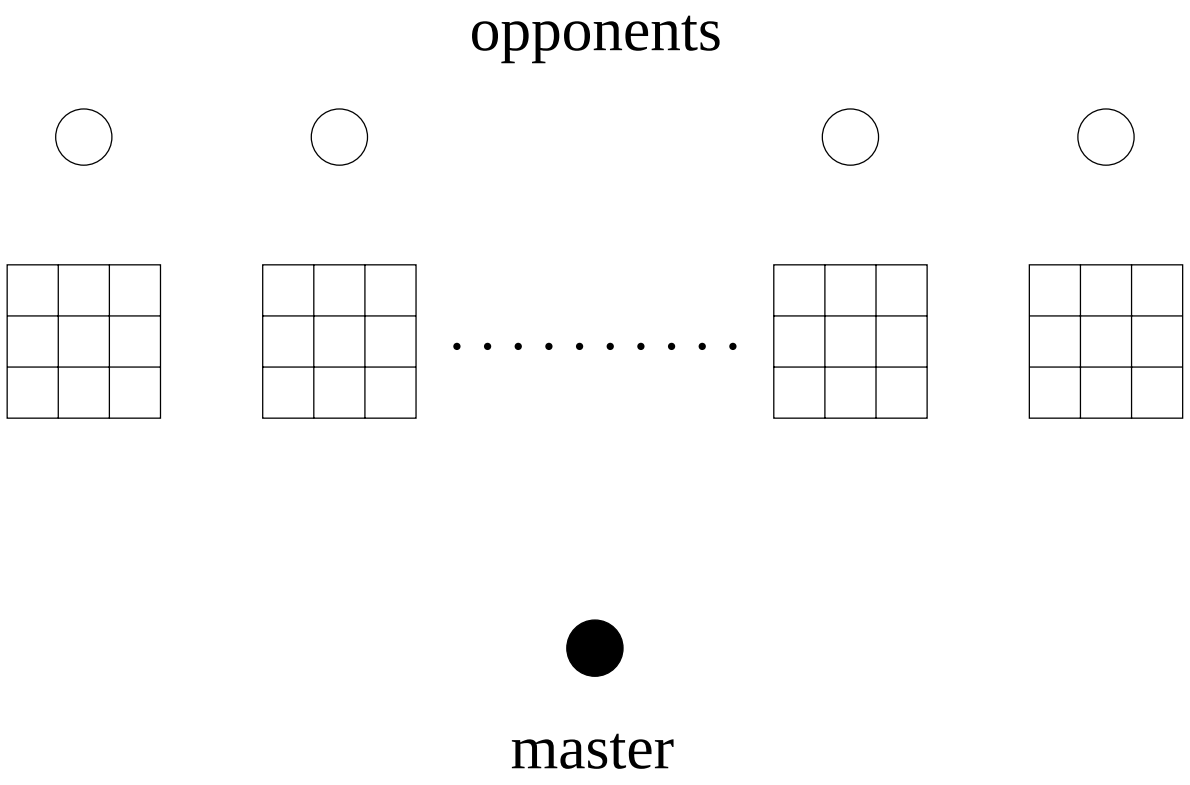
INIT

$board.\text{INIT}$

$BlackMove \mathrel{\widehat{=}} bplayer.Move \parallel board.BlackMove$
$WhiteMove \mathrel{\widehat{=}} wplayer.Move \parallel board.WhiteMove$
$TheWinner \mathrel{\widehat{=}} board.TheWinner$

A Tic Tac Toe Tournament

A tic tac toe master is prepared to play any number of opponents concurrently.



An Opponent

Opponent

$Wplayer$

Move

$b! : Board$

$b! = board$

Opponent

$board : Board$

Move

$p! : Posn$

$b! : Board$

$\neg\, 3InRow(board.bposn)$

$p! \notin board.bposn \cup board.wposn$

$board.turn = white$

$b! = board$

The Master

Master

$$boards : \mathbb{P}\, Board$$

Move

$$p! : Posn$$
$$b! : Board$$

$$\neg\, 3InRow(b!.wposn)$$
$$p! \notin b!.bposn \cup b!.wposn$$
$$b!.turn = black$$
$$b! \in boards$$

The Tournament

Tournament

$master : Master$
$opponents : \mathbb{P}\, Opponent$
$boards : \mathbb{P}\, Board$

$master.boards = boards$
$\{op : opponents \bullet op.board\} = boards$

INIT

$\forall\, b : boards \bullet b.\text{INIT}$

$MasterMove \mathrel{\widehat=} master.move \parallel ([b? : boards] \bullet b?.BlackMove)$

$OpponentMove \mathrel{\widehat=} [op : opponents] \bullet op.move$
$\parallel$
$[b? : boards] \bullet b?.WhiteMove$

$Awinner \mathrel{\widehat=} [b! : boards] \bullet b!.TheWinner$

	$master$	$boards$	$opponents$
$MasterMove$	$p! : Posn$ $b! : Board$	$p? : Posn$ $b? : Board$	
$OpponentMove$		$p? : Posn$ $b? : Board$	$p! : Posn$ $b! : Board$
$Awinner$		$b! : Board$ $winner! : Colour$	

Notation

Set Abstraction

$$\{a : A \bullet f(a)\}$$

is the set of all elements $f(a)$ where $a \in A$. e.g.

$$\{n : \mathbb{N} \bullet 2n\}$$

is the set of even natural numbers.

$$\{op : opponents \bullet op.board\}$$

is the set of boards associated with the set of opponents.

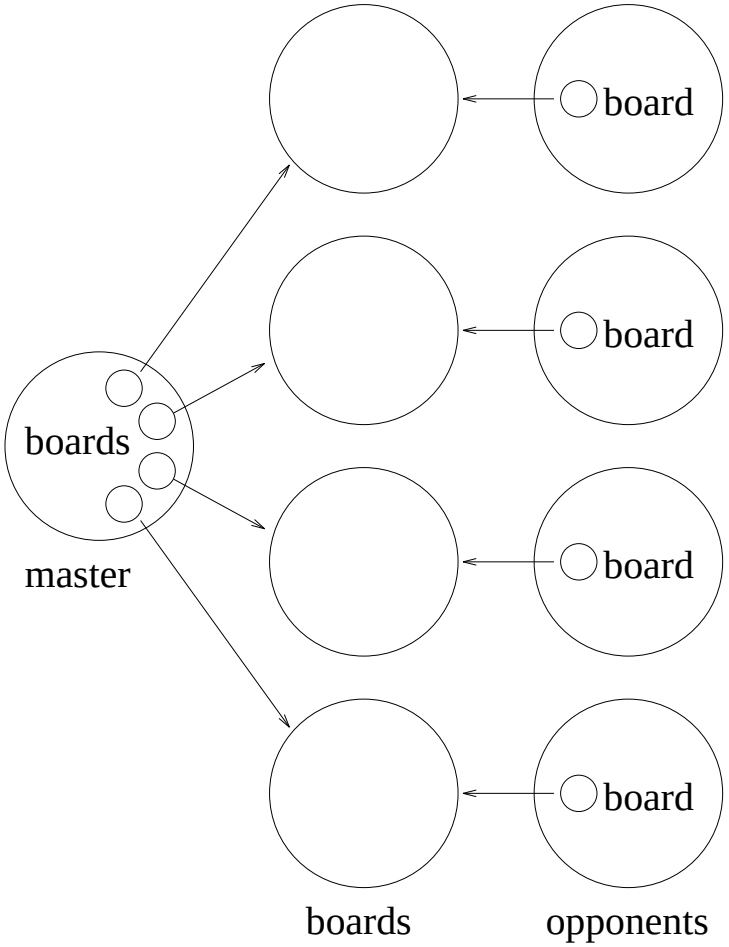
Object Selection

$$[ob : \text{set of objects}] \bullet ob.operation$$

indicates that object ob is selected from the given set of objects and performs $operation$.

Object selection may be via input/output communication.

The Object Structure



An Alternative Informal View

- there are two players and nine marbles
- each marble is uniquely labelled with a number between 1 and 9
- initially the players have no marbles
- the players take it in turns to select a marble from those not already selected
- the first player to have three marbles whose labels add to 15 is the winner

Initial Abstractions

$$Posn == 1 \,..\, 9$$

The Abstraction

6	1	8
7	5	3
2	9	4

Three in a Row

$$3InRow : \mathbb{P}\, Posn \to \mathbb{B}$$

$$\begin{array}{l} \forall\, ps : \mathbb{P}\, Posn \bullet \\ \quad 3InRow(ps) \Leftrightarrow \\ \qquad \exists\, p, q, r : ps \bullet \\ \qquad\quad \#\{p, q, r\} = 3 \\ \qquad\quad p + q + r = 15 \end{array}$$