Programmazione Visuale*

Davide Rocchesso

Abstract

Si presenta una breve introduzione alla programmazione visuale per mezzo del linguaggio Pure Data (PD)

1 Programmazione Visuale

Un linguaggio di programmazione visuale[1] permette di specificare programmi mediante manipolazione di elementi grafici. I linguaggi più usati in ambito artistico e di produzione multimediale (tra questi Pure Data[2] , MAX/MSP[3] , vvvv[4] , e Quartz Composer[5] sono basati sul paradigma di calcolo dataflow[6] , nel quale i dati (piuttosto che le sequenze di operazioni) sono al centro dell'attenzione. Le operazioni sono "scatole nere" che elaborano l'input, non appena questo è disponibile, producendo un output che può essere dato in input ad altre scatole di elaborazione. Un programma dataflow assomiglia ad una rete di catene di montaggio, esplicita in maniera intrinseca il parallelismo, e non nasconde uno stato. Oltre che prestarsi alla parallelizzazione automatica, i programmi dataflow sono in un certo senso "auto-documentati" e si prestano al debugging mediante sonde.

Pure Data[7] è un linguaggio di programmazione visuale inizialmente concepito da Miller Puckette[8] per la computer music in tempo reale, poi esteso ad opera di una comunità di programmatori ed oggi diffuso anche tra artisti visuali, performer e interaction designer. Pure Data è un software libero, ed ha uno stretto grado di parentela con MAX/MSP[9] . Essendo nato per elaborare e controllare segnale audio, Pure Data supporta la comunicazione dei dati da un operatore all'altro a due rate: il sample rate che per default è di 44100 campioni al secondo, ed il control rate ad un sessantaquattresimo del sample rate. Più precisamente, gli operatori che elaborano segnale audio (caratterizzati dal suffisso ∼) prendono campioni di ingresso in maniera sincrona ad audio rate e allo stesso rate producono segnali di uscita. Tutti gli altri operatori elaborano dati non appena questi sono disponibili (dataflow) ad un rate massimo di circa 690 Hz (per default, un sessantaquattresimo del sample rate). Tutti i calcoli sono effettuati su numeri floating-point a 32 bit. La documentazione di Pure Data è disponibile dalla voce `Help` del suo menu, oppure è consultabile online[10] . Questo modulo è parzialmente basato su tale documentazione e sulle lezioni di Gary Scavone[11] .

*Version 1.7: Oct 27, 2011 4:36 am -0500
†http://creativecommons.org/licenses/by/3.0/
[1]http://en.wikipedia.org/wiki/Visual_programming
[2]http://en.wikipedia.org/wiki/Pure_Data
[3]http://en.wikipedia.org/wiki/Max/MSP
[4]http://vvvv.org/
[5]http://en.wikipedia.org/wiki/Quartz_Composer
[6]http://en.wikipedia.org/wiki/Dataflow_language
[7]http://en.wikipedia.org/wiki/Pure_Data
[8]http://crca.ucsd.edu/∼msp/
[9]http://en.wikipedia.org/wiki/Max_%28software%29
[10]http://crca.ucsd.edu/∼msp/Pd_documentation/
[11]http://www.music.mcgill.ca/∼gary/306/

I programmi Pure Data sono chiamati **patch** e si presentano come grafi di elaborazione di flussi di dati. Ogni programma ha una finestra principale e può avere un numero arbitrario di sottofinestre. Il software Pure Data ha una interfaccia modale[12] , in quanto ci sono due modi distinti di operazione: run mode e edit mode. Si passa dall'uno all'altro con `command-E` e il feedback sul modo corrente è dato dalla forma del puntatore del mouse. I blocchi che si possono connettere in forma di grafo di dataflow sono di tipo: object, message, atom (number o symbol), GUI, e comment.

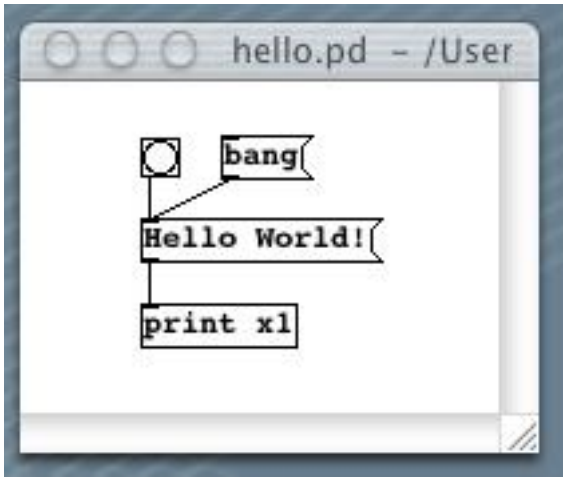

Figure 1

In Figure 1 è illustrato un semplicissimo patch[13] che stampa sulla consolle la scritta `x1: Hello World`. Esso contiene due oggetti message (lato destro concavo) ed un oggetto object (rettangolo con comando `print x1`). L'oggetto object rapresenta un comando con eventuali parametri di default. Il quadrato con cerchio all'interno è uno speciale messaggio senza contenuto, chiamato **bang**, che invia un evento ai blocchi ad esso collegati.

[12] http://en.wikipedia.org/wiki/Mode_%28computer_interface%29
[13] See the file at <http://cnx.org/content/m14602/latest/./hello.pd>

1.1 Idiosincrasie grafiche

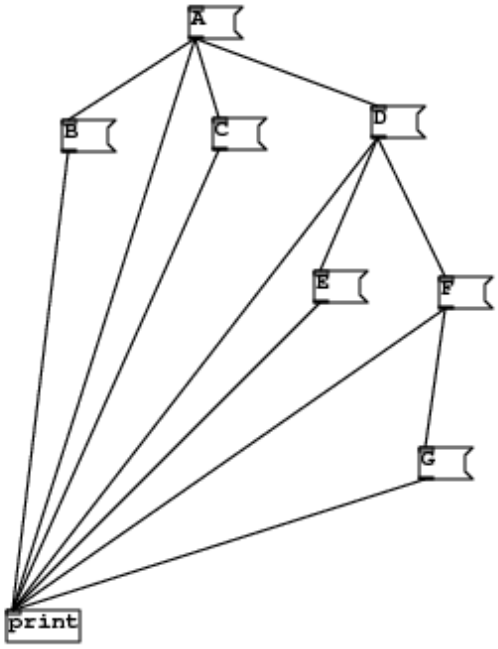

Figure 2

Ogniqualvolta un messaggio è inviato ad un oggetto, questi può diventare mittente a sua volta verso altri oggetti. In altri termini, si configura un albero di ricezioni di messaggi. Secondo il manuale di Pure Data [14] l'ordine di esecuzione utilizzato è depth-first[15] , cioè ogni ramo è esplorato fino alle foglie prima di procedere con gli altri rami. L'ordine di esecuzione tra nodi allo stesso livello dell'albero sarebbe invece da considerarsi indeterminato. In realtà, provando ad eseguire l'esempio[16] di Figure 2 e ridisponendo le connessioni tra i nodi dell'albero con ordini diversi ci si accorge che non si può nemmeno contare sull'ordine depth-first, in quanto esso interferisce con l'ordine di immissione delle connessioni. In Max/MSP la situazione è ancora peggiore perché muovendo messaggi e bang in posti diversi della finestra, a parità di configurazione topologica, si ottengono sequenze di attivazione diverse. L'elaborazione di tipo dataflow può dare luogo a loop non computabili, quale è quello riportato in Figure 3. Pure Data, in questo caso, riporta un messaggio di "stack overflow" sulla consolle. Per rendere il loop computabile è sufficiente inserire un elemento di

[14] http://crca.ucsd.edu/~msp/Pd_documentation/x2.htm#s3.2
[15] http://en.wikipedia.org/wiki/Depth-first_search
[16] See the file at <http://cnx.org/content/m14602/latest/./depth-first.pd>

disaccoppiamento quale è un `pipe` (si veda p. 10). Per quanto sia piccolo il parametro ritardo di questo pipe (al limite anche nullo), esso terrà memoria del dato in ingresso per consumarlo al ciclo successivo di calcolo.

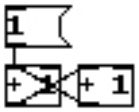

Figure 3

Nella maggior parte degli oggetti, la inlet di sinistra è **hot**, nel senso che messaggi che ad essa arrivano scatenano messaggi in output. E' spesso desiderabile inviare messaggi a due o più inlet di un oggetto, ma il risultato dell'operazione può dipendere dall'ordine con cui si sono fatte le connessioni. Ciò mina l'efficacia semantica del programma visuale, in quanto potremmo avere due patch apparentemente identiche che però si comportano diversamente perché costruite con un ordine diverso. Ad esempio, si provi a fare la somma di un numero con sè stesso secondo la Figure 4.

Figure 4

La sequenza di creazione delle connessioni determina la correttezza o meno del risultato. Per imporre chiarezza semantica al patch è opportuno inserire un oggetto `trigger` che forza l'ordine di attivazione delle outlet da destra a sinistra, come indicato in Figure 5.

Figure 5

1.2 Elementi di GUI

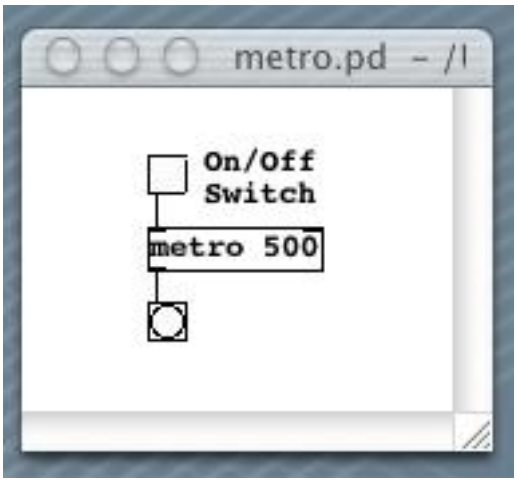

Figure 6

In Figure 6 è illustrata l'utilizzazione di un generatore periodico di bang[17]. Si noti il parametro di `metro` che stabilisce l'intervallo (di 500 millisecondi) tra due bang. Il quadrato in alto è un elemento di GUI detto toggle switch, in pratica un interruttore.

[17] See the file at <http://cnx.org/content/m14602/latest/./metro.pd>

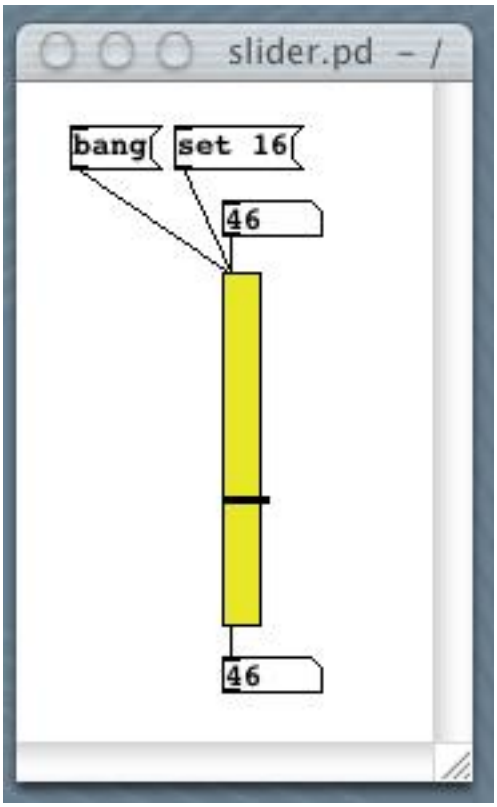

Figure 7

Un altro elemento di GUI, il vslider[18], è illustrato in Figure 7. In questo patch è anche visibile un oggetto number, il quale invia un numero in uscita (**outlet**) ogni volta che ne viene cambiato il valore, per dragging con il mouse o per immissione di numeri dal suo input.

[18] See the file at <http://cnx.org/content/m14602/latest/./slider.pd>

1.3 Array

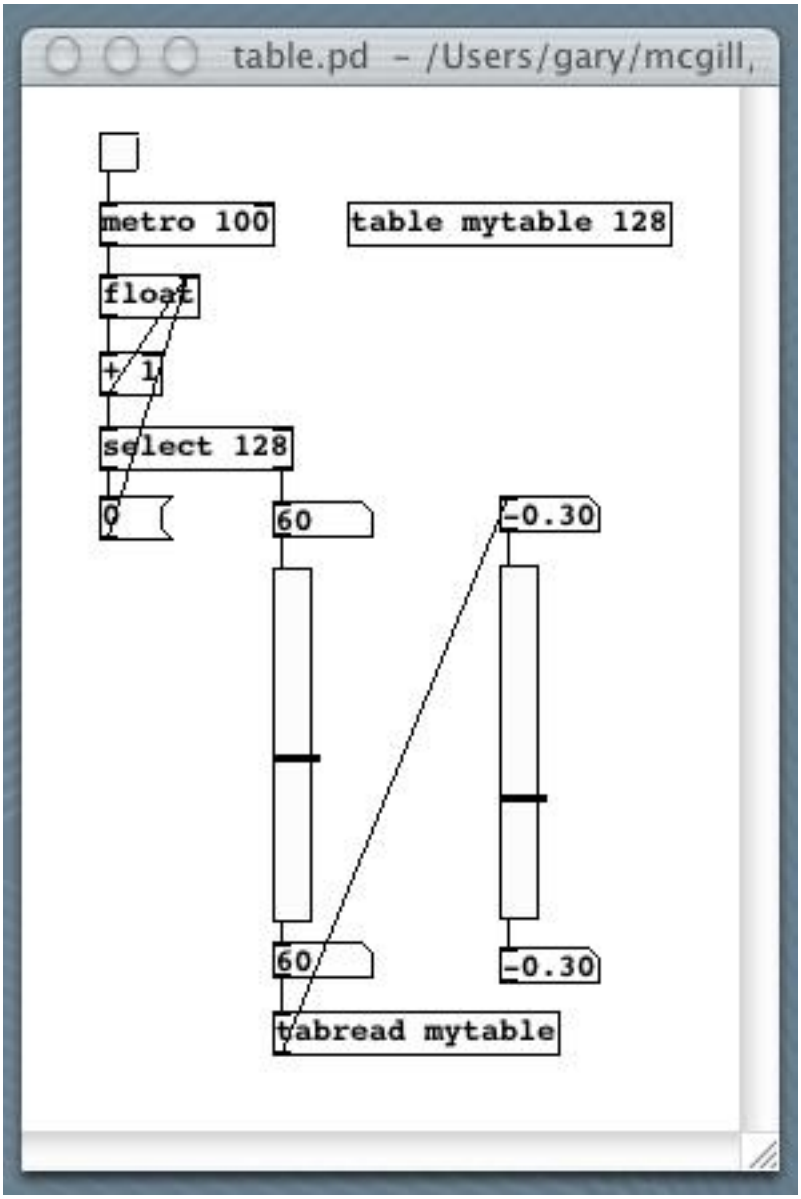

Figure 8

In Pure Data, un array è un contenitore di numeri ed un elemento di visualizzazione al tempo stesso. Infatti, l'oggetto `table` istanzia l'array e crea un subpatch che ne visualizza la rappresentazione grafica. Questa rappresentazione è anche uno strumento di input, in quanto i valori dell'array possono essere "disegnati" con il mouse. Per leggere e scrivere i singoli elementi dell'array si usano gli oggetti `tabread` e `tabwrite`.

L'utilizzazione di `tabread` per scandire un array[19] è illustrata in Figure 8.

1.4 Selezioni, routing, multiplexing

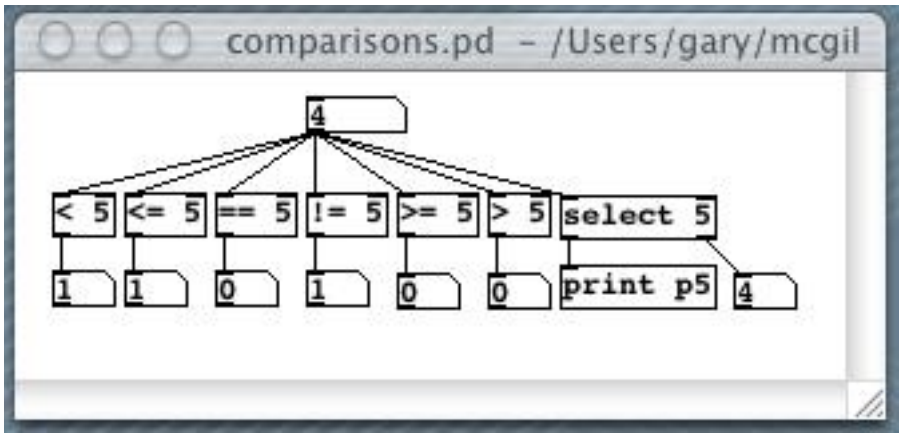

Figure 9

In Figure 9 sono illustrati blocchi di confronto e selezione[20]. In particolare, il blocco `select` attiva l'uscita di sinistra se l'ingresso (**inlet**) è uguale al suo parametro, altrimenti attiva l'uscita di destra.

[19] See the file at <http://cnx.org/content/m14602/latest/./table.pd>
[20] See the file at <http://cnx.org/content/m14602/latest/./comparisons.pd>

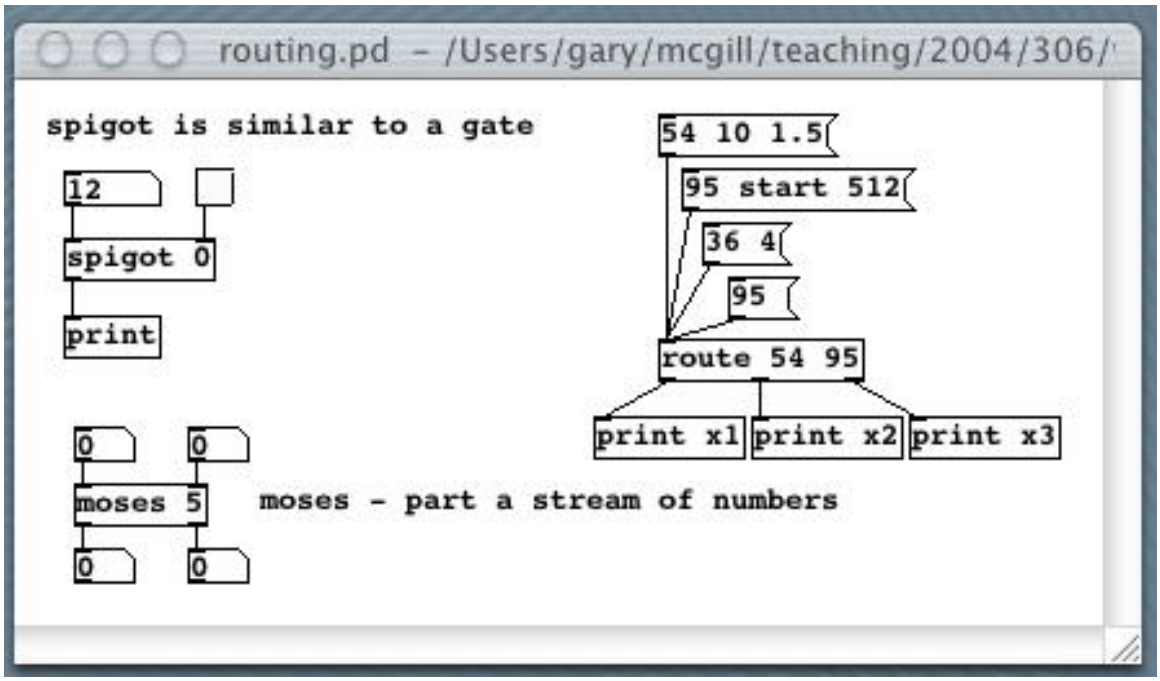

Figure 10

In Figure 10 sono illustrati tre elementi di routing[21]. Il blocco `spigot` trasmette messaggi dall'inlet di sinistra all'outlet in dipendenza dallo stato dell'ingresso (di controllo) di destra. In sostanza si tratta di un **gate** con controllo di apertura e chiusura. Invece, `moses` consente di smistare alle uscite sinistra e destra i valori di ingresso che sono rispettivamente minori o maggiori (o uguali) del valore di controllo passato dalla inlet di destra. Infine, `route` smista dei messaggi ricevuti in ingresso sulla base del loro primo elemento, mettendolo a confronto con gli argomenti. Un multiplexer[22] si può ottenere combinando la `route` con la `pack`.

Exercise 1 ***(Solution on p. 15.)***

Realizzare un multiplexer e un demultiplexer utilizzando gli oggetti `pack`, `route` e `spigot`.

[21] See the file at <http://cnx.org/content/m14602/latest/./routing.pd>
[22] http://en.wikipedia.org/wiki/Multiplexer

1.5 Ritardi, code e gestione del tempo

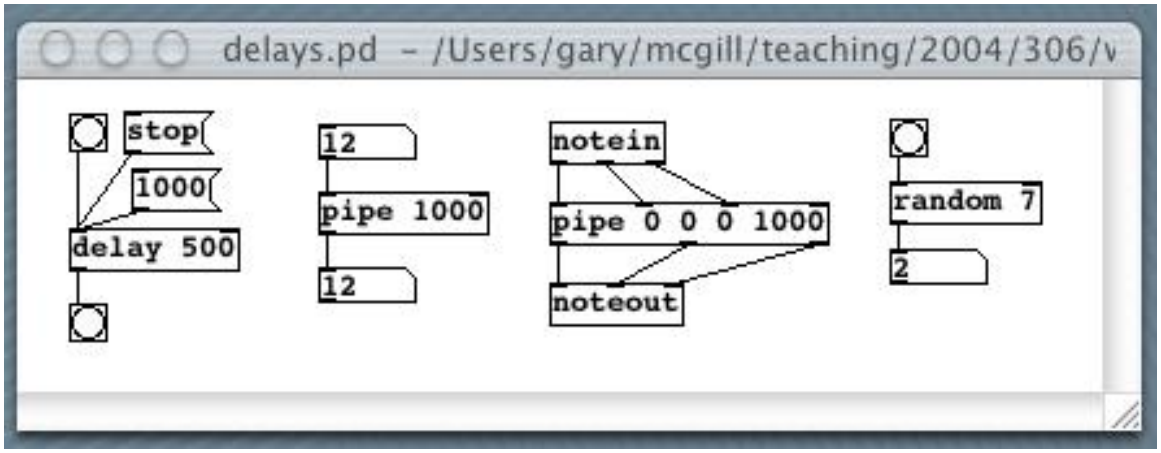

Figure 11

In Figure 11 è illustrata l'utilizzazione di ritardi e code[23]. La `delay` ritarda un bang in ingresso di un numero di millisecondi pari al suo argomento. Se si vuole ritardare un flusso di dati bisogna usare la `pipe`, la quale è realizzata come coda a buffer circolare[24].

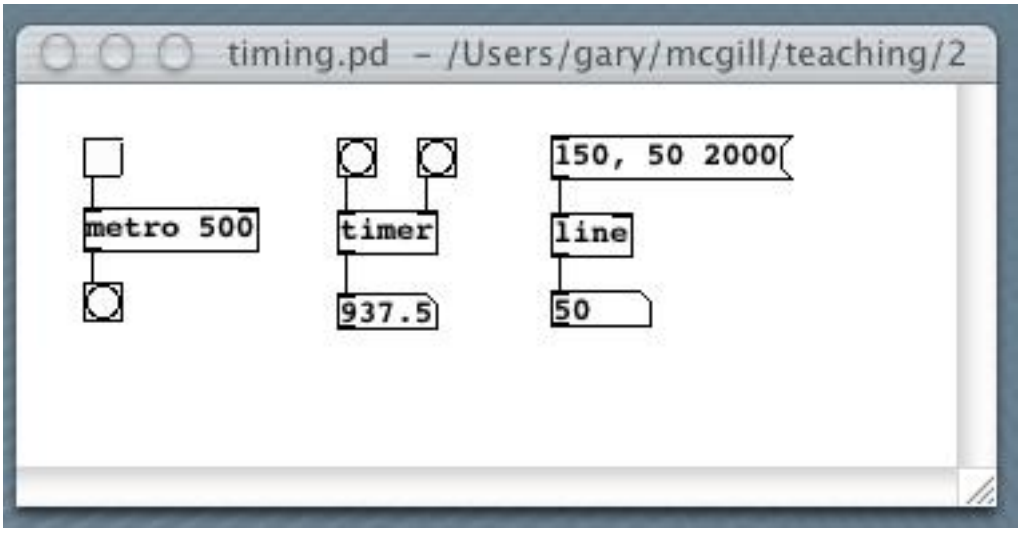

Figure 12

[23] See the file at <http://cnx.org/content/m14602/latest/./delays.pd>
[24] "Liste, pile e code" <http://cnx.org/content/m14556/latest/#codep>

In Figure 12 si vedono alcuni oggetti che consentono di scandire, misurare, e avanzare[25] il tempo. Rispettivamente, ciò si ottiene con `metro`, `timer`, e `line`. Quest'ultimo oggetto genera una rampa lineare, da un valore iniziale a uno finale, in un certo tempo.

Exercise 2 *(Solution on p. 15.)*

Nel patch di Figure 8, si provi a rimuovere e ripristinare i collegamenti che arrivano alla inlet di destra dell'oggetto `float`, e si verifichi che il reset dell'indice una volta raggiunto il limite di 128 può non avere luogo. Perché? Come si può introdurre in maniera deterministica il reset dell'indice?

1.6 Message Passing

Il funzionamento dataflow di Pure Data avviene mediante **pipe** che corrispondono alle connessioni tra oggetti. E' anche possibile operare in regime di message passing[26] , cioè ricevere (receive[27]) e spedire (send[28]) dati tra diverse parti di un patch o tra patch diversi. Ciò è illustrato in Figure 13. Invece, l'oggetto `value` realizza una sorta di variabile globale.

[25] See the file at <http://cnx.org/content/m14602/latest/./timing.pd>
[26] http://en.wikipedia.org/wiki/Message_passing
[27] See the file at <http://cnx.org/content/m14602/latest/./receive.pd>
[28] See the file at <http://cnx.org/content/m14602/latest/./send.pd>

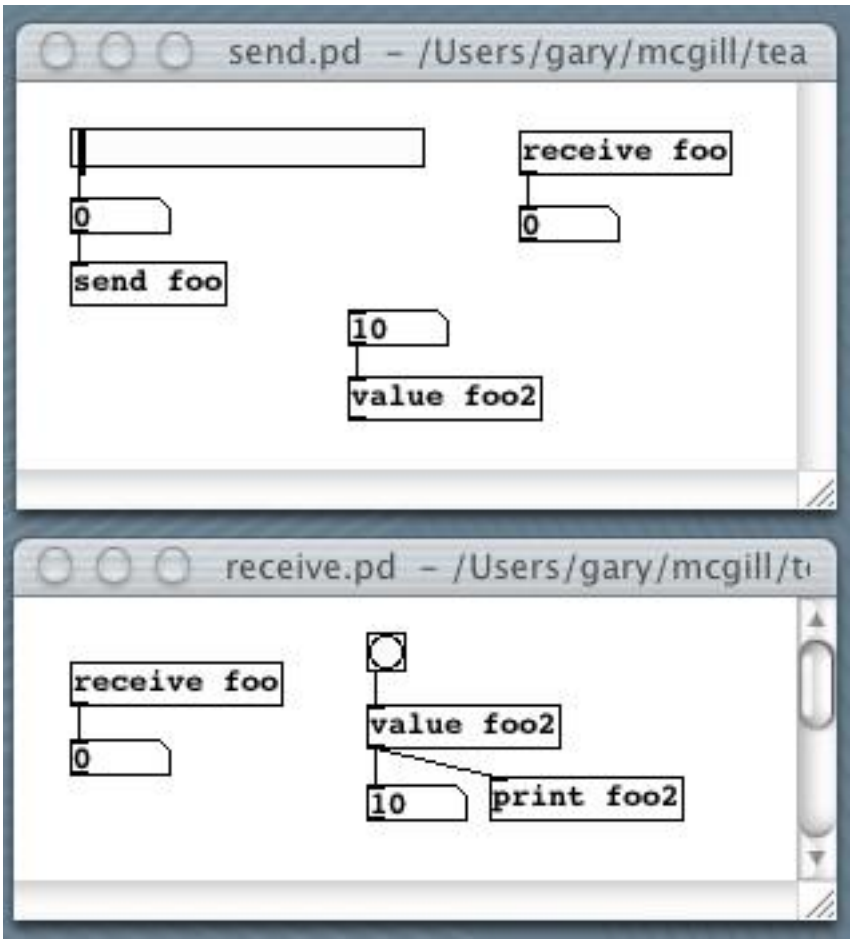

Figure 13

1.7 Modularità

Nella programmazione visuale la modularità si ottiene con i meccanismi di **subpatching** che, in Pure Data, sono di due tipi:

- Si utilizza l'oggetto `pd` per dichiarare un subpatch. In quest'ultimo, si specificano le `inlet` e `outlet`. Si veda Figure 14.
- Si costruisce un patch come file separato, che a questo punto diventa uno degli oggetti a disposizione di Pure Data. Si veda Figure 15.

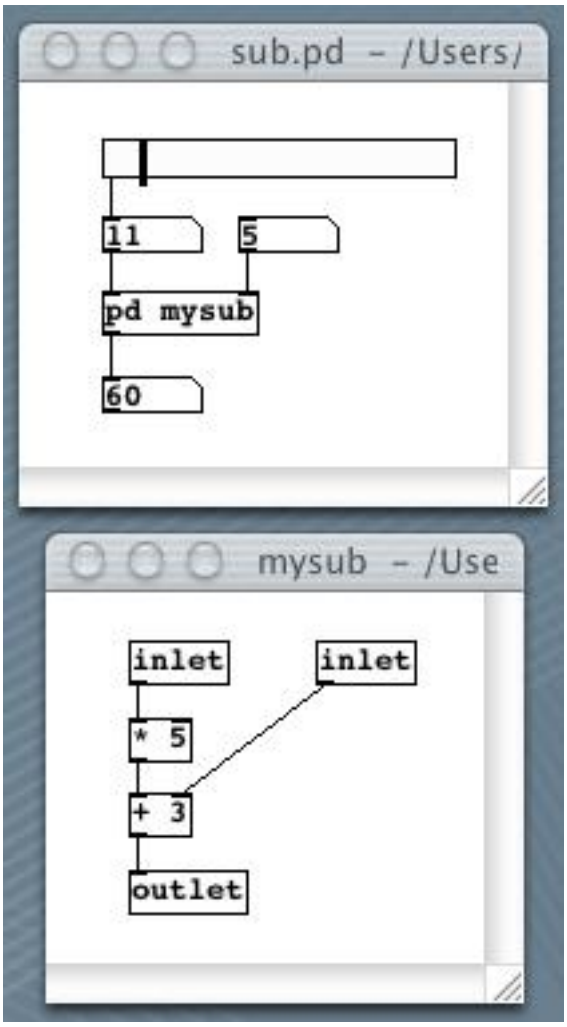

Figure 14

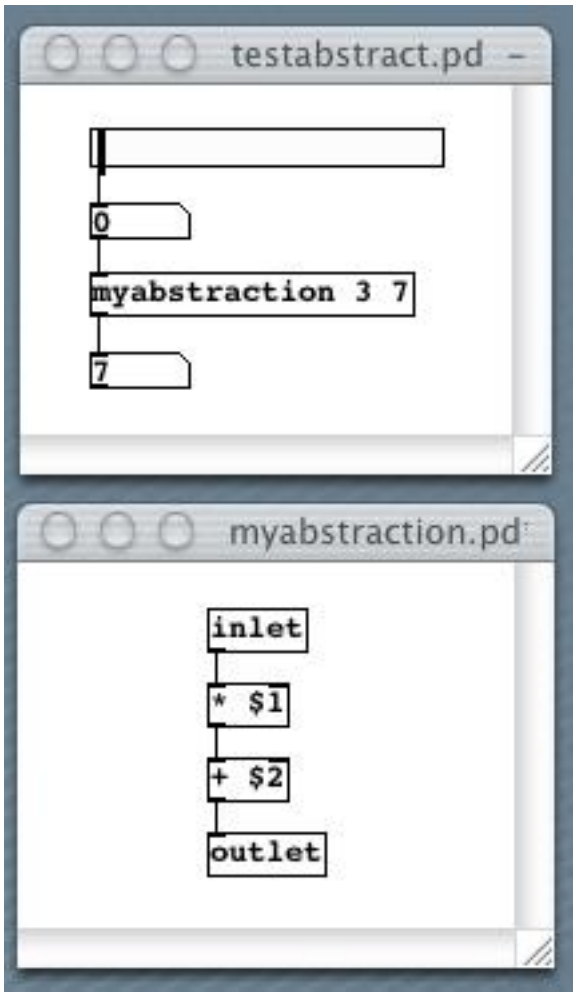
testabstract.pd
0
myabstraction 3 7
7
myabstraction.pd
inlet
* $1
+ $2
outlet

Figure 15

Solutions to Exercises in this Module

Solution to Exercise (p. 9)
Pure Data patch[29].
Solution to Exercise (p. 11)
E' sufficiente introdurre un elemento di disaccoppiamento (ad esempio, una `pipe 0`) che imponga un ordine tra l'immissione del numero incrementato e l'immissione di 0 nella inlet di destra.

[29] See the file at <http://cnx.org/content/m14602/latest/./muxdemux.pd>