Una breve introduzione a STEL (STGraph Expression Language)

Luca Mari, versione 2.3.16

Premessa

STEL (STGraph Expression Language) è il *domain-specific language* con cui è definita la componente quantitativa dei modelli di STGraph.

Modelli come grafi; variabili come nodi; frecce come relazioni di dipendenza funzionale

Un modello in STGraph è descritto qualitativamente come un grafo orientato, in cui i nodi sono variabili (o, come caso particolare, costanti) e le cui frecce descrivono relazioni di dipendenza funzionale.

Figura 1 - Una freccia dal nodo `x` *al nodo y specifica che* `y` *dipende funzionalmente da* `x`.

La forma analitica di tale dipendenza è scritta come un'espressione in STEL.

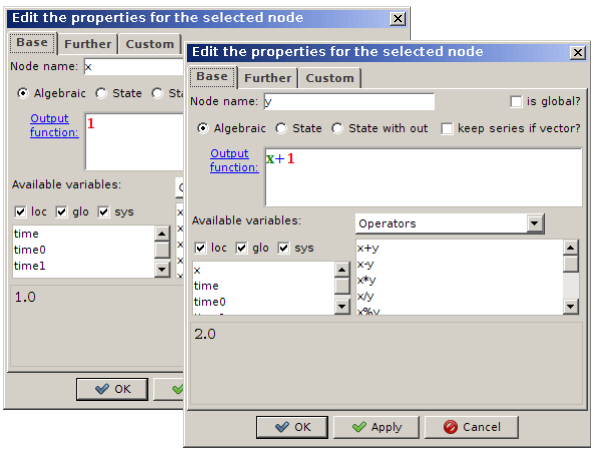

Figura 2 - Le finestre di configurazione dei nodi della Figura 1.
Il contenuto corrisponde alle assegnazioni `x←1`*;* `y←x+1`.

Il fatto che STGraph ammetta tre tipi di nodi (algebrici, di stato, di stato con output) non ha conseguenze, e quindi si tratterà qui genericamente di *espressioni in STEL*, che nei casi specifici potrebbero essere espressioni corrispondenti a una funzione di output, a uno stato iniziale, o a una funzione di transizione di stato locale.

Le caratteristiche principali di STEL in sintesi

Linguaggio funzionale

STEL è un linguaggio funzionale puro (dunque non contiene "istruzioni" né "comandi", ma appunto solo funzioni), ed è basato sull'ipotesi che le uniche relazioni ammesse tra variabili sono quelle descritte mediante le frecce del grafo (nell'esempio delle Figure 1 e 2, la funzione che definisce `y` include un riferimento a `x`, ma il viceversa non potrebbe essere data l'assenza di una freccia da `y` a `x`).

Assenza di istruzioni / funzioni di assegnazione di valore a variabili

Per rendere l'ambiente di esecuzione perfettamente immune da effetti collaterali (*side effect*), STEL non include alcuna funzione per l'assegnazione di valore a variabili (ma vedi sotto a proposito di sottoespressioni): il valore di una variabile è sempre e solo assegnato mediante l'espressione associata al nodo corrispondente alla variabile stessa. In questo modo, ogni nodo del grafo corrisponde a un micro-ambiente computazionale, aperto in input solo attraverso le frecce in ingresso al nodo e in output solo attraverso il valore prodotto dalla valutazione dell'espressione.

Scope delle variabili

Ciò definisce lo scope di default delle variabili, ognuna delle quali può essere comunque resa visibile globalmente al grafo (STGraph gestisce anche modelli strutturati in sottomodelli, corrispondenti a grafi in cui uno o più nodi sono associati non a variabili ma ad altri grafi: lo scope rimane sempre a livello di sottomodello, e ogni supermodello interagisce con i suoi sottomodelli solo attraverso le variabili di input e di output dei sottomodelli stessi – tutte e sole le variabili senza frecce entranti sono di input; ogni variabile può essere dichiarata di output).

Ordine di valutazione delle variabili

L'ordine di valutazione delle espressioni nei diversi nodi è inferito da STGraph in base alla topologia del grafo: questo fa sì che, in linea di principio, sia ammessa la valutazione parallela di nodi non reciprocamente dipendenti (in presenza di variabili di stato, un grafo può contenere anche loop).

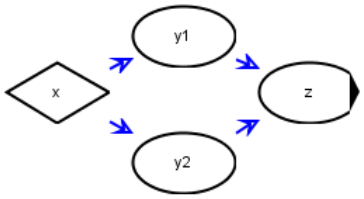

Figura 3 – Grafo in cui le variabili `y1` *e* `y2` *potrebbero essere valutate in parallelo.*

Variabili di un unico tipo

Dato il suo dominio applicativo e per ragioni di semplicità, le variabili in STEL hanno tutte un unico e implicito (dunque non dichiarato) tipo di base, numero, che in base al contesto è interpretato come numero intero, numero decimale, o costante logica (i numeri strettamente positivi sono associati alla costante 'vero').

Variabili come array

Le variabili in STEL sono tutte di un unico tipo strutturato, array, che include come casi particolari scalari, vettori, matrici, e che ammette anche strutture tensoriali di dimensione superiore. Gli array sono dinamici, e non devono essere predefiniti.

Funzioni predefinite e funzioni definite dall'utente

STEL è dotato di un insieme di funzioni predefinite (scritte in Java come STGraph) (alcune di queste scritte nell'usuale notazione operatoriale infissa), e di una funzione per la definizione in STEL di nuove funzioni, anche in forma ricorsiva, sia in nodi di un grafo (in tal caso lo scope è il grafo stesso) sia in file XML esterni (in tal caso le funzioni sono utilizzabili da qualsiasi grafo).

Polimorfismo per funzioni a un argomento

Le funzioni predefinite sono generalmente polimorfe rispetto alla dimensione di array dei loro argomenti. Nel caso di funzioni a un argomento, ciò corrisponde, per esempio, al fatto che la funzione `sqrt(x)` ha il seguente comportamento (in questi esempi '`=`' è il simbolo metalinguistico per 'ha valore'):

se `x=4` allora `sqrt(x)=2`

se `x=[4,9]` allora `sqrt(x)=[2,3]`

se `x=[[4,9],[16,25]]` allora `sqrt(x)=[[2,3],[4,5]]`

e così via.

Polimorfismo per funzioni a due argomenti

Nel caso di funzioni a due argomenti, il polimorfismo è garantito in tutti i casi in cui la valutazione sia ammessa. Per esempio:

se `x=1` e `y=2` allora `x+y=3`

se `x=1` e `y=[1:3]` allora `x+y=[2,3,4]`

se `x=[[1,2],[3,4]]` e `y=[5,6]` allora `x+y=[[6,7],[9,10]]`

dove quest'ultimo caso è da intendere come:

$$\begin{bmatrix}1 & 2\\ 3 & 4\end{bmatrix}+\begin{bmatrix}5\\ 6\end{bmatrix}=\begin{bmatrix}6 & 7\\ 9 & 10\end{bmatrix}$$

cosa che mostra che i vettori sono interpretati come vettori colonna, che gli array a più dimensioni sono linearizzati in modo da mantenere come ultima dimensione quella più innestata, e che, quando si applica, è sull'ultima dimensione degli array che le funzioni operano.

Non includendo istruzioni, STEL non ha strutture di controllo tradizionali.

If

E' invece dotato di una funzione condizionale generalizzata, `if(`c_1`,`v_1`,...,`c_n`,`v_n`,`v_{n+1}`)`, a numero dispari ≥3 di argomenti e tale che gli argomenti in posizione dispari salvo l'ultimo sono condizioni c_i e gli altri sono valori v_j: la prima condizione vera seleziona il valore

immediatamente successivo, e se tutte le condizioni sono false è selezionato l'ultimo valore. Per esempio:

se `x=if(1,2,3)` allora `x=2`

dato che tale funzione corrisponde alla struttura di controllo in Java:

```
if(1) {
      x=2;
} else {
      x=3;
}
```

ma anche:

se `x=if(0,2,0,3,4)` allora `x=4`

che infatti corrisponde a:

```
if(0) {
      x=2;
} else if(0){
      x=3;
} else {
      x=4;
}
```

e anche, in modo polimorfo:

```
if([1,0],2,3)=[2,3]
```

Loop temporali

Nei modelli di STGraph non è generalmente necessario introdurre in modo esplicito dei loop. Ciò è principalmente dovuto al fatto che nel corso della simulazione il tempo viene incrementato a passi discreti e quindi crea le condizioni perché, in linea di principio, l'espressione di ogni variabile sia valutata in loop (in particolare nella funzione di transizione di stato locale di un nodo di stato, per esempio un'espressione come:

```
this+1
```

– dove `this` denota lo stato attuale – incrementa di 1 lo stato a ogni istante della simulazione).

Per il caso in cui sia necessario operare in loop sincronicamente sugli elementi di un array `x` (dunque con un loop "spaziale" invece che temporale), se la funzione /operatore `f` da applicare è a due argomenti (come si è visto, funzioni a un argomento sono applicabili direttamente agli array) STEL dispone di meta-operatori che iterano la valutazione della funzione sull'array secondo logiche diverse:

- l'espressione `f/x`, dove '`/`' denota il meta-operatore *reduction*, corrisponde, nel caso `x` sia un vettore, allo scalare `f(f(x[0],x[1]),x[2])...`; per esempio:

```
+/[1:4]=((1+2)+3)+4
```

- l'espressione `f\x`, dove '`\`' denota il meta-operatore *scan*, corrisponde, nel caso `x` sia un vettore, al vettore `[x[0],f(x[0],x[1]),f(f(x[0],x[1]),x[2]),...]`; per esempio:

```
+\[1:3]=[1,1+2,(1+2)+3]
```

- l'espressione `f|x`, dove '`|`' denota il meta-operatore *pairscan*, corrisponde, nel caso `x` sia un vettore, al vettore `[f(x[0],x[1]),f(x[1],x[2]),f(x[2],x[3]),...]`; per esempio:

```
+|[1:4]=[1+2,2+3,3+4]
```

Se applicati a matrici o array di dimensione superiore, questi meta-operatori valutano la funzione `f` lungo l'ultima dimensione. Per esempio:

```
+/[[1,2,3],[4,5,6],[7,8,9]]=[1+2+3,4+5+6,7+8+9]
```

Loop spaziali

Loop spaziali su array `x` possono essere computati più in generale mediante la meta-funzione `iter(x,e,z)`, in cui `e` è l'espressione da valutare iterativamente sugli elementi (dell'ultima dimensione) di `x` e `z` è lo "zero" di tale espressione. La valutazione è ripetuta per un numero di volte pari al numero di elementi (dell'ultima dimensione) di `x`. L'espressione `e` può contenere le variabili di sistema:

- `$i`, che a ogni iterazione ha come valore l'indice dell'iterazione;
- `$0`, che inizialmente ha valore `z` e a ogni iterazione ha come valore il risultato ottenuto valutando `e`;
- `$1`, che a ogni iterazione ha come valore `x[$i]`.

Per esempio, `iter(x,$0+$1,0)` valuta la somma degli elementi (dell'ultima dimensione) di `x`, e `iter(x,$1#$0,[])` ('`#`' denota l'operatore di concatenazione) produce un vettore a elementi invertiti.

Sottoespressioni

Pur non includendo funzioni per l'assegnazione di valore a variabili, STEL consente di identificare una o più sottoespressioni (al momento fino a 4) dell'espressione in corso di valutazione, delimitandole mediante parentesi graffe: `{`*`sottoespressione`*`}`. Tali sottoespressioni vengono valutate secondo l'ordine di valutazione dell'espressione complessiva, e il valore di ognuna di esse viene assegnato alla variabile di sistema `$w`*`i`*, *`i`*=0,...,3, e quindi può essere richiamato in punti successivi dell'espressione stessa.
Per esempio:

```
if({order(x)}==0,1,$w0==1,@x,*/@x)
```

calcola il numero di elementi dell'array `x`, secondo la logica:
- se `x` è uno scalare, cioè un array di ordine 0, produci il valore `1`; calcola nel frattempo il valore della sottoespressione `order(x)` e assegnalo alla variabile di sistema `$w0`;
- se invece `x` è un vettore, cioè un array di ordine 1, produci il valore `@x`, cioè la dimensione di `x`;
- altrimenti, cioè se `x` è una matrice o un array di dimensione superiore, produci il valore `*/@x`, cioè il prodotto delle dimensioni di `x`.

Qualche esempio

Prodotto vettoriale di due matrici `x` e `y`

```
array([numRows(x),numCols(y)],+/(x[[$i0],[]]*y[[],[$i1]]))
```

Prodotto scalare di due vettori `x` e `y`

```
+/(x*y)
```

Range del vettore `x`

```
max/(x)-min/(x)
```

Media del vettore `x`

```
(+/x)/@x
```

Mediana del vettore `x`

```
if(isEven({@x}),(get({sort(x)},$w0/2-1)+get($w1,$w0/2))/2,
get(sort(x),($w0-1)/2))
```

Deviazione standard del vettore `x`

```
sqrt((+/((x-mean(x))^2))/(lastDim(x)-1))
```

Vettore ottenuto rimuovendo gli `0` dal vettore `x`

```
iter(x,if($1!=0,$0#$1,$0),[])
```