Relazione secondo progetto PR2 2017/2018 – Michele Zoncheddu

Si estenda il linguaggio didattico funzionale per permettere la manipolazione di alberi binari di espressioni. Ogni nodo è composto da: identificatore, espressione, figlio sinistro e figlio destro.

1. Estensione informale dei tipi del linguaggio (per quella formale rimando al codice sorgente)
Oltre all'operatore Modulus per semplificare la verifica di parità di un numero, il tipo **exp** è stato esteso da:

- ETree of tree;
- ApplyOver of funzione * albero;
- Update of cammino * funzione * albero;
- Select of cammino * funzione booleana * albero;

dove **tree** è:

- Empty;
- Node of identificatore * espressione * figlio sinistro * figlio destro.

Il tipo **evT** (tipo memorizzabile) è stato esteso con l'albero valutato nel seguente modo:

- Tree of treeVal,

dove **treeVal** è:

- Empty;
- NodeVal of identificatore * evT * treeVal (figlio sinistro) * treeVal (figlio destro).

L'esecuzione della ApplyOver, della Update e della Select restituisce un treeVal, ovvero un albero con le espressioni valutate.
Per ogni operazione è presente un **controllo dinamico** dei tipi, per evitare errori a runtime.

2. Semantica delle operazioni

- **Valutazione standard di alberi**: se il nodo corrente non è vuoto, viene valutata la sua espressione, e il figlio sinistro e destro saranno sostituiti rispettivamente dai risultati della loro stessa valutazione; quest'operazione è di supporto alle successive;
- **ApplyOver**: se il nodo corrente non è vuoto, uso l'espressione come parametro attuale della funzione, e il risultato della chiamata di funzione sarà il nuovo valore del nodo. Figlio sinistro e figlio destro saranno sostituiti dall'applicazione ricorsiva su loro stessi della ApplyOver;
- **Update**: se il nodo corrente non è vuoto e il cammino è vuoto, oppure il cammino non è presente all'interno dell'albero, le espressioni vengono valutate nel modo standard; se il cammino identifica uno o più nodi dell'albero, a questi viene applicata la funzione passata come parametro, mentre i restanti nodi negli eventuali livelli superiori ed inferiori, vengono valutati nel modo standard; nel caso di cammini identici, tutti i nodi identificati dal cammino vengono aggiornati;
- **Select**: se il nodo corrente è vuoto o il cammino è vuoto (condizione vacuamente falsa), restituisce un albero vuoto; il ***primo*** nodo del cammino la cui applicazione della funzione restituisce "true", verrà restituito come risultato, con le informazioni relative ai discendenti: di fatto viene restituito un sottoalbero; nel caso di cammini identici, verrà data la priorità al risultato del sottoalbero sinistro (solo nel caso questo risultato non sia un albero vuoto).

In **tutte** le operazioni aggiuntive, **l'ambiente** nel quale valutare le espressioni **è sempre lo stesso** (simulazione di valutazione parallela).
Nota: nell'esecuzione delle operazioni che utilizzano un cammino, all'atto pratico, il primo tag verrà cercato nel primo livello, il secondo tag nel secondo livello e così via.

3. Test case e miglioramenti possibili
Nella parte finale del codice sono state inserite delle espressioni per testare tutte le operazioni implementate, su due alberi di prova. L'interprete è stato testato sul toplevel di OCaml.
Al momento non è supportato l'annidamento delle operazioni, ovvero la possibilità, ad esempio, di valutare espressioni del tipo Select(ApplyOver(…)): è tuttavia implementabile attraverso una conversione da evT a exp, per ottenere un nuovo albero di espressioni atomiche da fornire come input alle operazioni.