Secondo progetto intermedio A.A. 2020-2021

Nicola Vetrini, matricola 600199

Il progetto consiste in un'estensione del linguaggio funzionale didattico presentato durante il corso con il tipo di dato Set (insieme omogeneo, con annotazione di tipo) e le relative operazioni, la definizione delle regole della semantica operazionale per la sua interpretazione e conseguentemente l'estensione dell'interprete e del typechecker dinamico. Ho implementato anche l'operazione Size su Set, che ritorna la cardinalità del Set fornito come argomento, il tipo di dato stringa e l'operazione di concatenazione, per avere una maggiore varietà di tipi primitivi nel linguaggio.

Espressioni e loro semantica

La definizione delle espressioni, dei tipi esprimibili e la definizione dell'ambiente sono nel file *linguaggio.ml*. Le regole della semantica operazionale per la valutazione degli insiemi e le operazioni su di essi, che hanno guidato l'estensione dell'interprete del linguaggio, sono contenute nel file *semantica.pdf* e comprendono le regole per la valutazione dei costruttori di Set (EmptySet, Singleton e Set con lista di elementi) al tipo esprimibile SetVal, anch'esso aggiunto ai tipi esprimibili già presenti nel linguaggio. In realtà i tipi di Set accettati dall'interprete sono limitati a *int*, *bool* e *string* in quanto sono i soli tipi per i quali si possa definire agevolmente una nozione di ordinamento totale, per cui ha senso valutare SetMin e SetMax su di essi. Ho aggiunto anche i denotabili UnboundInt, UnboundBool e UnboundString ad evT per restituire un valore specifico per il tipo nel caso in cui si valutino set vuoti in SetMin e SetMax.
Infine ho esteso exp con il costruttore Estring ed il corrispondente tipo esprimibile String a evT per introdurre le stringhe, così come il costruttore Concat ad exp per la loro concatenazione.

Interprete e typechecker

L'interprete del linguaggio è contenuto nel file *interprete.ml*, che contiene anche la funzione che implementa il typechecking dinamico (typecheck). Entrambe estendono le funzionalità di quelle da lei fornite, anche se nel caso del typechecker ho invertito il parametro sul quale fare pattern matching per evitare di dover scrivere una funzione separata per il typechecking dei Set, ma l'idea alla base non cambia (maggiori dettagli nel codice).
Per l'interprete ho seguito invece la struttura usata nell'implementazione delle operazioni già fornite: ho definito delle funzioni esterne ad eval, le cui chiamate sono effettualte dopo aver individuato con pattern matching l'operazione (o il costruttore), ed eseguono internamente il typechecking oltre alla valutazione. Nel caso in cui siano riscontrati errori viene lanciata un'eccezione (con failwith(…)) oppure viene stampato un warning su standard output (ad esempio se l'elemento non è presente nel Set, allora non può essere rimosso).
Infine ho inserito due funzioni utili per il testing: string_of_evT e print_exp con le seguenti signature:

- `let rec string_of_evT (obj : evT) : string`
- `let print_exp (e : exp) (r : evT env) : unit`

string_of_evT, analogamente alle funzioni disponibili in OcaML (string_of_*), restituisce la stringa corrispondente ad alcuni denotabili (in particolare ho escluso le funzioni, in quanto non è semplice stampare una chiusura). print_exp utilizza la funzione sopracitata per stampare su standard output l'espressione valutata nell'ambiente dato e va a capo.

Testing

I testcase sono nel file *tests.ml*. Assumendo che Ocaml e il programma make siano installati, per compilare e testare il codice è sufficiente digitare il seguente comando in una shell

```
make
```

che eseguirà i comandi contenuti in *Makefile* producendo il file *test*, contenente bytecode, e lo esegue. Altrimenti è equivalente la sequenza di comandi

```
ocamlc linguaggio.ml interprete.ml tests.ml -o test; ocamlrun ./test
```