Relazione del secondo progetto intermedio per il corso di *Programmazione 2* del CdL in Informatica dell’Università di Pisa, a cura di **Salvatore Correnti**.

**Introduzione**

Il progetto prevede la progettazione e realizzazione di un’estensione del linguaggio didattico funzionale presentato durante il corso al fine di manipolare insiemi (Set), ovvero collezioni di valori dello stesso tipo, non ordinati, e che non contengono duplicati.

Il sistema dei tipi del linguaggio prevede i seguenti tipi: *int, bool, string, closure, rec-closure, set* e *unbound*, dove il tipo *set* è parametrico rispetto ai tipi per esso ammissibili. In particolare, si è deciso di restringere l’insieme dei tipi ammissibili per *set* ai soli *int, bool* e *string*: la motivazione di questa scelta risiede nel fatto che questi tipi di dato sono naturalmente confrontabili, mentre non è ben definita una nozione di confronto fra insiemi e funzioni (né ha chiaramente senso per il tipo *unbound*), e dunque per questi ultimi non sarebbe possibile definire alcune delle operazioni primitive di *set*.

Come scelta di implementazione, si è deciso di usare la keyword *type* di OCaml per creare due nuove categorie sintattiche per il linguaggio:

* type tsub = TInt | TBool | TString;
* type tname = TInt | TBool | TString | TSet of tsub | TClosure | TRecClosure | TUnbound;

dove i costruttori TInt, TBool e TString sono ripetuti in modo da essere utilizzabili (previa annotazione nel corpo di una funzione di OCaml ad esempio) sia come tname sia come tsub, per esprimere l’idea che tsub è come se fosse un “sottotipo” di tname che rappresenta esattamente i tipi ammissibili per set. Per permettere la conversione da un tipo a un altro, sono state aggiunte due funzioni e .

I tipi tname e tsub sono utilizzati nel typechecker al posto delle annotazioni “int”, “bool” etc. È stata presa questa scelta per esprimere più efficacemente il fatto che set è un tipo polimorfo e per delimitare l’insieme dei tipi per esso ammissibili.

Di conseguenza, un insieme è stato definito come “Set of tsub\*(evT list)”, dove list è una lista di elementi dello stesso tipo senza ripetizioni (i controlli opportuni sono effettuati in tutte le funzioni che creano o modificano insiemi).

**Eccezioni ridefinite:** È stata introdotta l’eccezione *RuntimeError of string* per segnalare un qualunque errore avvenuto nel runtime dell’interprete.

**Note sulle operazioni**

* *Of*: come scelta di progetto, si è deciso che in caso di elementi duplicati nella lista passata come secondo argomento, verrà creato un insieme dove tali duplicati sono eliminati.
* *Insert*: qualora l’elemento da inserire fosse già presente non succede nulla, mentre se il tipo è diverso da quello dell’insieme dato viene lanciata una RuntimeError.
* *Remove*: operazione che permette di rimuovere un elemento se presente nell’insieme, in caso contrario o se venisse chiesto di eliminare un elemento di un tipo diverso l’interprete provvederà a lanciare una RuntimeError.
* *Union, Intersection, Difference*: in caso di tipi di dato diversi contenuti dai due insiemi, viene lanciata una RuntimeError.
* *GetMin* e *GetMax*: per ottenere il minimo / massimo degli elementi dell’insieme, si è fatto uso della funzione built-in “compare” di OCaml.
* *For\_all, Filter, Exists*: si è scelto uno stile *lazy* di controllo del tipo del predicato: se l’insieme è vuoto allora vengono automaticamente ritornati (rispettivamente): *Bool(true), Set(t,[]), Bool(false)*, dove *t* è il tipo degli elementi dell’insieme di partenza; il tipo della funzione è determinato dinamicamente solo se questa viene effettivamente eseguita, ovvero se l’insieme ha almeno un elemento: se non venisse restituito un booleano allora certamente la funzione passata non è un predicato e viene lanciato un errore. Si osservi che in assenza di controlli di tipo statici sul corpo della funzione, in caso di funzioni “polimorfe” possono esserci comportamenti diversi a seconda del tipo dei dati in input: ad esempio, l’identità filtra correttamente un insieme di booleani ma dà errore se applicata a un insieme di interi.
* Map: si è deciso in tal caso di lanciare un errore se l’insieme passato è vuoto.
* Empty: se non venisse inserito un tipo tsub viene lanciata una RuntimeError.
* Contains: se viene richiesta la verifica della presenza di un elemento su un Set che contiene oggetti con tipo diverso viene lanciata una RuntimeError.
* IsSubset: se viene richiesta una verifica di contenimento su due Set di diverso tipo viene lanciata una RuntimeError.

**Espressioni aggiuntive**

Sono stati aggiunti alcuni costruttori al tipo exp per supportare i confronti fra interi, per aiutare a scrivere i predicati da usare in fase di test: *LessThan of exp\*exp,* che accetta due *EInt* e viene valutata a *true* sse il primo argomento è valutato a un intero minore del secondo, e *GreaterThan of exp\*exp*, che accetta due *EInt* e viene valutata a *true* sse il primo argomento è valutato a un intero maggiore del secondo.

**Funzioni aggiuntive**

Per implementare correttamente tutte le funzionalità richieste e migliorare la leggibilità del codice le funzioni primitive richiamate dalla *eval* si appoggiano a una serie di funzioni di supporto che operano su liste:

* list\_union, list\_intersection, list\_difference: creano rispettivamente la “lista unione”, “lista intersezione”, “lista differenza” delle due liste passate come parametri;
* checkNotEquals: controlla che una lista non abbia elementi ripetuti;
* list\_contains: controlla se una lista contiene un dato elemento;
* is\_contained: controlla se tutti gli elementi della prima lista appartengono anche alla seconda;
* list\_remove: rimuove un elemento da una lista;
* list\_max e list\_min: restituiscono il massimo e il minimo elemento di una lista.

Nei casi degli operatori forall, exists, filter e map le analoghe funzioni di supporto (forall\_list, exists\_list, filter\_list, map\_list) sono definite *dentro* la eval, in quanto servono entrambi i nomi nello scope locale.

**Typechecker dinamico**

Si è realizzato un typechecker dinamico che permetta di controllare a run-time tutti i tipi di evT avendo come parametri il tipo tname e l’elemento, in particolare nel caso dell’insieme si verifica che il tipo in ingresso sia lo stesso del tipo di dato che l’insieme può contenere.

**Test**

È stata realizzata una batteria di test che permetta di controllare il corretto funzionamento dell’interprete a partire dalle funzioni primitive aggiunte, viene inoltre verificato il corretto scoping statico delle variabili. Per completezza nella batteria è stata aggiunta anche una sezione “Errori” dove vengono verificate e catturate tutte le tipologie di errore che possono presentarsi.

Per maggiore chiarezza il file con i test è un sorgente a parte, in modo da tenere separati l’interprete del linguaggio e i test su di esso.

**Avvio**

È possibile compilare direttamente in codice nativo (almeno su Windows e Unix) i due file con qualunque fra:

* ocamlopt –o <nomeEseguibile> MiniCaml.ml testMiniCaml.ml
* ocamlc –o <nomeEseguibile> MiniCaml.ml testMiniCaml.ml

e visualizzare l’output da terminale (nel file di test è definita una funzione che formatta la stampa per ogni tipo di dato e una per valutare e stampare a schermo.

In caso di problemi, su:

<http://ocsigen.org/js_of_ocaml/dev/manual/files/toplevel/index.html>

è disponibile un ambiente REPL in cui possono essere inseriti i codici dei due sorgenti (rimuovendo “*open MiniCaml;;”*) e visualizzare l’output di ogni istruzione.

**Regole Operazionali**

Le regole operazionali per l’introduzione del tipo di dato set e le operazioni su di esso sono nell’allegato “Regole Operazionali.pdf”.