M2048 Compilatori e interpreti

C2060 Compilatori e interpreti

Closures per negati

Una possibile gerarchia di nodi in Funny è

```
Expr (abstract)
    BinaryExpr
    FunExpr
    GetVarExpr
    IfExpr
    InvokeExpr
    PrintExpr
    SeqExpr
    SetVarExpr
    UnaryExpr
    Val (abstract)
        BoolVal
        ClosureVal
        NilVal
        NumVal
        StringVal
    WhileExpr
```

Expr espone il metodo

```
abstract class Expr {
    abstract Val eval(Env env);
    ...
}
```

Valutare (eseguire, interpretare) un'espressione nel contesto dell'ambiente **env** significa ridurla a un valore.

La distinzione tra **Expr** e **Val** è importante nei richiami. Dapprima gli argomenti, che nell'albero ritornato dal compilatore sono nodi generali, vengono ridotti a valori e solo a questo punto viene fatto il richiamo.

Val espone la specializzazione

```
abstract class Val extends Expr {
    Val eval(Env env) {
        return this;
    }
    ...
}
```

perché **Val** già rappresenta un nodo ridotto a valore.

FunExpr rappresenta una funzione come la vede il compilatore, staticamente. Mantiene una lista di identificatori per i parametri, una per le variabili locali e il nodo che costituisce il corpo della funzione. In esecuzione, il risultato di una valutazione di una funzione è un **ClosureVal** (cfr. sotto), cioè una combinazione tra la funzione (componente statica) e l'ambiente (parte dinamica). *Valutare una funzione non significa eseguire il corpo della funzione ma solo preparare una closure.* Il corpo della funzione viene eseguito solo quando più tardi la closure viene applicata agli argomenti.

```
class FunExpr extends Expr {
    Value eval(Env env) {
        return new ClosureVal(env, this);
    }
    ...
}
```

Un **ClosureVal** esiste solo in esecuzione, come risultato della valutazione di una funzione. Una closure mantiene una funzione e l'ambiente in cui la closure viene creata (cfr. **FunExpr**). La finalità di una closure è di essere *applicata* a una lista di valori con lo scopo di eseguire (valutare) il corpo della funzione (un nodo) e ritornarne il valore al chiamante:

```
class ClosureVal extends Val {
    Val apply(List<Val> argVals) {
        return funExpr.code().eval(
            new Env(new Frame(funExpr.params(), funExpr.locals(), argVals), env));
    }
    ...
}
```

Per far ciò, l'ambiente mantenuto dalla closure viene esteso con un nuovo **Frame** inizializzato opportunamente.

Un **Frame** mantiene un'associazione tra identificatori e valori (**Val**). Viene istanziato durante l'applicazione di una closure a una lista di argomenti già ridotti a valori. Gli identificatori vengono presi da quelli della funzione associata alla closure. I parametri vengono inizializzati coi valori degli argomenti, le variabili locali vengono inizializzate a **nil**.

Un ambiente **Env** è costituito da un **Frame** e da un ambiente circostante. Cercare un identificatore in un ambiente, per leggerne o per modificarne il valore, significa cercarlo dapprima nel frame, poi, se necessario, risalire la china cercandolo nel frame dell'ambiente circostante, su su fino a scoprire un'associazione in qualche frame. Questa ricerca non può fallire se il compilatore ha fatto il suo dovere!

La compilazione di un **postfix** della grammatica può produrre un inscatolamento di nodi **InvokeExpr** per ogni gruppo di argomenti **args** presente nel codice. Un nodo **InvokeExpr** appresenta un'invocazione (un richiamo). Mantiene un campo **expr** per la parte a sinistra degli argomenti e un campo **args** per gli argomenti stessi.

P.es., la compilazione del **postfix**

```
m(4, x + y)("a")
```

produce l'albero schematico seguente, dove i nodi per m, la somma, ecc. non sono dettagliati:

```
InvokeExpr
    expr: InvokeExpr
              expr: m
              args: (4, x + y)
    args: ("a")
```

In questo esempio, si spera che valutando il nodo **m** di tipo **GetVarExpr** si ottenga una closure: questa viene quindi applicata al gruppo di argomenti **(4, x + y)**. Ci si aspetta, poi, che il richiamo **m(4, x + y)** torni a sua volta una closure come risultato, che viene quindi applicata al gruppo **("a")**.

Valutare una **InvokeExpr** significa valutare dapprima la parte **expr**, controllare che il risultato sia una closure, valutare tutti gli argomenti (ridurli a valori) e applicare la closure ai valori così ottenuti. Anche in Funny, come nella maggior parte dei linguaggi di programmazione, prima di passare il controllo alla closure si riducono gli argomenti a valori.

Si ha

```
class InvokeExpr extends Expr {
    Val eval(Env env) {
        return expr.eval(env).checkClosure().apply(args.eval(env));
    }
    ...
}
```

La valutazione degli argomenti procede da sinistra a destra, nodo per nodo. Come d'abitudine, la valutazione di un nodo può modificare l'ambiente in cui viene valutato.

Qui **args** è un'istanza di una classe d'appoggio **ExprList** che non mantiene altro che una lista di **Expr**. Il suo unico scopo è di esporre un metodo **eval(Env)** che esegue **eval(Env)** su ogni elemento della lista. È da notare che **ExprList** *non* è un sottotipo di **Expr** e che il suo **eval(Env)** torna una *lista* di valori, non un singolo valore.

Per riassumere:

- Valutare un **`FunExpr`** crea un **`ClosureVal`.**
- Valutare un'**`InvokeExpr`** (un richiamo) applica un **`ClosureVal`** agli argomenti ridotti a **`Val`.**
- Applicare un **`ClosureVal`** significa valutare il corpo della **`FunExpr`** mantenuta dal **`ClosureVal`** nell'ambiente ottenuto estendendo quello mantenuto dal **`ClosureVal`** con un nuovo **`Frame`** inizializzato dal valore degli argomenti per i parametri e da **`nil`** per le variabili locali.

Giunti qui respirate, sgranocchiate qualcosa e rileggete tutto da capo, passando di nuovo da qui fino a raggiungere l'illuminazione.