Corso: Fondamenti, Linguaggi e Traduttori

Paola Giannini

Commenti su type-checking

Commenti generali

1. Quasi tutti lo avete fatto con il pattern visitor. Spero che capiate come funziona cioè siate in grado di fare un diagramma UML di sequenza della chiamata del metodo

   ```
   nodePr.accept(new TypeCheckingVisitor())
   ```

   su un oggetto `NodeProgram` che è l'AST di un semplice programma, ad esempio

   ```
   int a;
   a = 4 + 7;
   ```

2. Molti hanno sbagliato la conversione, nel senso che o non hanno generato il `NodeConvert` o dopo averlo generato non lo hanno assegnato per cui non serve a niente!
3. C'è anche abbastanza confusione fra cosa fa la visita del `NodeDecl` e quella nel `NodeId`
4. Alcuni hanno settato il tipo di nodi sottoclassi di espressioni a `VOID` e questo NON è corretto! Le espressioni o hanno tipo `ERROR` oppure `FLOAT` o `INT`.

C'è un po' di confusione!

```
1  public abstract class  NodeAST {
2    private TypeDescriptor resType;
3    public TypeDescriptor getResType() {  return resType; }
4    public NodeAST setResType(TypeDescriptor resType) {
5      return this;
6    }
7    protected NodeAST(TypeDescriptor resType) {
8      this.resType = resType;
9    }
10   public abstract void calResType();
11   public abstract void accept(IVisitor visitor);
12 }
```

- Perchè un costruttore **protected**?
- Il type checking o si fa col visitor (riga 11) oppure con il metodo che calcola esplicitamente il resTyepe (riga 10)!

Indice

Quando è corretto un programma?

```
public void visit(NodeProgram node) {
  for(NodeDecSt nodoAttuale : node)
    nodoAttuale.accept(this);
  node.setResType(TypeDescriptor.VOID);
}

public void visit(NodeProgram node) {
  ArrayList<NodeDecSt> arrayNodeDS= node.getDecSt();
  for (NodeDecSt nodeDS : arrayNodeDS) {
    nodeDS.accept(this);
  }
}
```

- Il programma è corretto, cioè ha tipo `VOID` quando tutte le sue dichiarazioni e istruzioni sono corrette!
- Il promo metodo setta a `VOID` qualsiasi programma e il secondo non lo setta per nessuno!

La visita di NodeId e quella di NodeDecl

```
1   public void visit(NodeDecl nodo) {
2     nodo.getId().accept(this);
3     if (nodo.getInit() != null) nodo.getInit().accept(this);
4     if (SymbolTable.lookup(nodo.getId().getNome()) != null)
5       nodo.setResType(TypeDescriptor.Error); // variabile gia' dichiarata
6     else {
7       Attributes att = new Attributes(nodo.getTipo());
8       SymbolTable.enter(nodo.getId().getNome(), att);
9       nodo.getId().setDefinition(att);
10      if (att.getTipo().equals(LangType.INTy)) nodo.setResType(TypeDescriptor.Int);
11      else if (att.getTipo().equals(LangType.FLOATy)) nodo.setResType(TypeDescriptor.Float);
12    }
13  }
14
15  public void visit(NodeId nodo) {
16    if (SymbolTable.lookup(nodo.getNome()) == null)
17      nodo.setResType(TypeDescriptor.Error);
18    else {
19      Attributes att = new Attributes(SymbolTable.lookup(nodo.getNome()).getTipo());
20      if (att.getTipo().equals(LangType.INTy)) nodo.setResType(TypeDescriptor.Int);
21      else if (att.getTipo().equals(LangType.FLOATy))
22      nodo.setResType(TypeDescriptor.Float);
23      nodo.setDefinition(att);
24    }
25  }
```

- La visita di NodeDecl deve inserire l'identificatore nella symbol table (cosa vuol dire la riga 2?) e
- la visita di NodeId che guarda se l'identificatore è definito nella symbol table (cosa vuol dire la riga 19?)

Perchè VOID per NodeID?

```
1  public void visit(NodeId node) {
2    if(SymbolTable.lookup(node.getName()) == null) {
3      node.setResType(TypeDescriptor.Error);
4    }
5    else {
6      node.setResType(TypeDescriptor.Void);
7      node.setDefinitions(SymbolTable.lookup(node.getName()));
8    }
9  }
```

- `NodeId` dovrebbe avere il tipo della sua dichiarazione che si trova negli attributi della symbol table!

Le dichiarazioni possono avere inizializzazioni!!!!!!

```
1  public void visit(NodeDecl node) {
2    NodeId id = node.getId();
3    String name = id.getName();
4    if(SymbolTable.lookup(name) != null)
5      node.setResType(TypeDescriptor.ERROR);
6    else {
7      Attributes att = new Attributes(node.getType());
8      SymbolTable.enter(name, att);
9    }
10 }
```

- Questo type checking è incompleto!

Visita NodeDecl non corretta

```
1   public void visit(NodeDecl node) {
2     if(SymbolTable.lookup(node.getId().getName())!=null) {
3       node.setResType(TypeDescriptor.ERRORtd);
4       log+="Variabile gia' dichiarata\n";
5       return;
6     }
7     else SymbolTable.enter(node.getId().getName(), node.getId().getDefinition());
8     if(node.getId().getResType()!=TypeDescriptor.ERRORtd) {
9       node.setResType(TypeDescriptor.VOIDtd);
10      if(node.getInit()!=null) {
11        node.getInit().accept(this);
12        if(node.getId().getResType()!=node.getInit().getResType()) {
13          if(node.getId().getResType()==TypeDescriptor.INTtd
14                  && node.getInit().getResType()==TypeDescriptor.FLOATtd) {
15            node.setResType(TypeDescriptor.ERRORtd);
16            log+="Impossibile fare ..............+"\n";
17          }
18          else {
19            NodeConvert nodec=new NodeConvert(node.getInit());
20            nodec.accept(this);
21          }
22        }
23      }
24    }
25    else node.setResType(TypeDescriptor.ERRORtd);
26  }
```

- Alla riga 7 assume che nel suo `NodeId` ci sia già una `definition`
- Di nuovo il `nodedec` non viene assegnato (riga 19-10) e poi si rivisita!!!!

Questa NON e' una conversione!!!!!!

```
1   public void visit(NodeAssign nodo) {
2     nodo.getId().accept(this);
3     nodo.getExp().accept(this);
4     if (SymbolTable.lookup(nodo.getId().getNome()) != null) {
5       if ((nodo.getId().getResType()==TypeDescriptor.Int &&
6             nodo.getExp().getResType()==TypeDescriptor.Float)
7           || nodo.getId().getResType() == TypeDescriptor.Error
8           || nodo.getExp().getResType() == TypeDescriptor.Error)
9       nodo.setResType(TypeDescriptor.Error);
10      else {
11        if (nodo.getId().getResType().equals(nodo.getExp().getResType()))
12          nodo.setResType(nodo.getId().getResType());
13        else {
14          conversione(nodo.getExp());
15          nodo.setResType(TypeDescriptor.Float);
16        }
17      }
18    }
19    else nodo.setResType(TypeDescriptor.Error);
20  }
21  public void conversione(NodeExpr nodo) {
22    nodo.setResType(TypeDescriptor.Float);
23  }
```

- Per fare la conversione bisogna creare un `NodeConvert` che contiene il `NodeExpr` da convertire!!!!
- Alla riga 14 si cambia solamente il tipo del nodo!!!!!!

Controllo strano e la conversione è persa

```
1   public void visit(NodeAssign node) {
2     node.getInit().accept(this);
3     node.getId().accept(this);
4     node.setResType(TypeDescriptor.VOID);
5     if(node.getId().getResType()==TypeDescriptor.FLOAT &&
6          node.getInit().getResType()==TypeDescriptor.INT) {
7       NodeConverter nodeConv= new NodeConverter(node.getInit());
8       nodeConv.accept(this);
9      }
10     else {
11       node.setResType(TypeDescriptor.ERROR);
12       log+= "Impossibile fare la conversione da float a int ..................".\n";
13    }
14  }
```

- Perchè se hanno lo stesso tipo c'è un errore?
- La variabile nodeConv non è assegnata a niente per cui è persa!!!!

Tipo dell'espressione binaria sbagliato!

```
1   public void visit(NodeBinOp node) {
2     node.getLeft().accept(this);
3     node.getRight().accept(this);
4     if(node.getLeft().getResType()!=node.getRight().getResType()) {
5       if(node.getLeft().getResType()==TypeDescriptor.FLOAT &&
6             node.getRight().getResType()==TypeDescriptor.INT) {
7         NodeConverter nodeConv= new NodeConverter(node.getRight());
8         nodeConv.accept(this);
9       }
10      else {
11        node.setResType(TypeDescriptor.ERROR);
12        log+= "Impossibile fare la conversione da float a int.\n";
13      }
14    }
15    else {
16      node.setResType(TypeDescriptor.VOID);
17    }
18  }
```

- Se i tipi sono uguali alla riga 16 si assegna `VOID`, ma un'espressione NON può avere tipo `VOID`!
- Alla riga 7 si crea un `NodeConvert` che però non viene assegnato a niente (quindi è garbage)
- E se `getRight().getResType()==TypeDescriptor.FLOAT` e `getLeft().getResType()==TypeDescriptor.INT`????

Perchè controllare un descrittore che non c'e'?

```
1   public void visit(NodeBinOp node) {
2     node.getLeft().accept(this);
3     node.getRight().accept(this);
4     if(node.getLeft().getResType().equals(TypeDescriptor.ERROR) ||
5        node.getRight().getResType().equals(TypeDescriptor.VOID))
6       node.setResType(TypeDescriptor.ERROR);
7     else
8       if(node.getLeft().getResType().equals(node.getRight().getResType()))
9         node.setResType(node.getLeft().getResType());
10       else {
11         if(node.getLeft().getResType().equals(TypeDescriptor.INT)) {
12           NodeExpr nodeExpr = this.convert(node.getLeft());
13           node.setLeft(nodeExpr);
14         }
15         else {
16           NodeExpr nodeExpr = this.convert(node.getRight());
17           node.setRight(nodeExpr);
18         }
19       node.setResType(TypeDescriptor.FLOAT);
20       }
21  }
```

- Alla riga 4 si controlla se una espressione ha tipo `VOID` ma questo non dovrebbe mai succcedere

Non ripetere test fatti nella visita di `NodeId`

```
1   public void visit(NodeAssign nodo) {
2     nodo.getId().accept(this);
3     nodo.getExp().accept(this);
4     if (SymbolTable.lookup(nodo.getId().getNome()) != null) {
5         ................
6       }
7       else nodo.setResType(TypeDescriptor.Error);
8   }
9   public void visit(NodeDeref nodo) {
10    nodo.getId().accept(this);
11    if (SymbolTable.lookup(nodo.getId().getNome()) != null) {
12        ................
13      }
14      else nodo.setResType(TypeDescriptor.Error);
```

- È la visita di `NodeId` che guarda se l'identificatore è definito nella symbol table.
- NON ripetere i test già fatti!!!

Cosa fare per NodeConvert

```
1  public class NodeConverter extends NodeExpr {
2    private NodeExpr expression;
3        ..............................
4    public void convert() {
5      expression.setResType(TypeDescriptor.FLOAT);
6    }
7  }
8
9  public void visit(NodeConverter node) {
10   node.convert();
11   node.setResType(TypeDescriptor.VOID);
12 }
```

- Alle righe 9-12 non è una visita del `NodeConvert` perchè:
  - mette il tipo del `NodeConvert` a `VOID` (mentre la conversione deve produrre il tipo `FLOAT`)
  - cambia il tipo dell'espressione contenuta (riga 5) che invece NON deve essere modificato!

Alcuni di voi hanno fatto parte della visita giusta ma poi settato il tipo a `VOID`!

Indice

Cosa si deve fare per `NodeDecl` e `NodeId` (implementazione standard)

```
1  public class NodeDecl extends NodeDecSt {
2    private NodeId id; private LangType type; private NodeExpr init;
3    public void calResType() {
4      id.calResType();
5      boolean exprResTypeErr = false;
6      if(init != null) {
7        init.calResType();
8        if(init.getResType() == TypeDescriptor.Error)  exprResTypeErr = true;
9      }
10     if(id.getResType() == TypeDescriptor.Error || exprResTypeErr) {
11       this.setResType(TypeDescriptor.Error); return;
12     }
13     this.setResType(TypeDescriptor.Void);
14   }
15 }
16 public class NodeId extends NodeAST {
17    ............
18   public void calResType() { }
19 }
```

- Perché `calResType()` di `NodeId` non fa niente? Invece dovrebbe cercare l'identificatore nella symbol table, se lo trova assegnare al nodo il suo tipo e se non lo trova assegnare al nodo il descrittore di errore. Inoltre dovrebbe essere assegnato al campo `declaration` l'attributo associato nella symbol table.
- Nel metodo `calResType()` di `NodeDecl` non si visita il `NodeId` perchè qua è dove si inserisce nella symbol table.
- Se il tipo della variabile dichiarata è `FLOAT` e l'espressione ha tipo `INT` deve essere introdotta una conversione di tipo

Cosa fa il metodo `calResType()` di `NodePrint`

```
1   public class NodePrint extends NodeStm {
2     private NodeId id;
3        ............
4     public void calResType() {
5       id.calResType();
6       if(id.getResType() != TypeDescriptor.Error)
7         this.setResType(TypeDescriptor.Void);
8       this.setResType(TypeDescriptor.Error);
9     }
10  }
11  public class NodeId extends NodeAST {
12       .........
13    public void calResType() { }
14  }
```

- Va bene ma solo se il metodo corrispondente di `NodeId` fa i controlli necessari!

Non inserisce la conversione e codice non bello

```
1   public class NodeAssign extends NodeStm {
2     private NodeId id;
3     private NodeExpr expr;
4        ......................
5     public void calResType() {
6       id.calResType();
7       expr.calResType();
8       if(id.getResType() == TypeDescriptor.Error || expr.getResType() == TypeDescriptor.Error) {
9         this.setResType(TypeDescriptor.Error);
10        return;
11      }
12      if(id.getResType() == expr.getResType()) {
13        this.setResType(id.getResType());
14        return;
15      }
16      if(id.getResType() == TypeDescriptor.Float && expr.getResType() == TypeDescriptor.Int) {
17        this.setResType(TypeDescriptor.Float);
18        return;
19      }
20      this.setResType(TypeDescriptor.Error);
21    }
22  }
```

- Alla riga 17 si modifica solo il tipo ma non si inserisce il nodo di conversione!!!
- Meglio usare **`else if`** invece che il **`return`** alla fine del ramo del **`if`**.
- Gli stessi problemi ci sono nel metodo del `NodeBinOp`!

Uso else if

```
1   public class NodeAssign extends NodeStm {
2     private NodeId id;
3     private NodeExpr expr;
4        ......................
5     public void calResType() {
6       id.calResType();
7       expr.calResType();
8       if(id.getResType()==TypeDescriptor.Error || expr.getResType()==TypeDescriptor.Error) {
9         this.setResType(TypeDescriptor.Error);
10      } else if(id.getResType()==expr.getResType()) {
11        this.setResType(id.getResType());
12      } else if(id.getResType()==TypeDescriptor.Float&&expr.getResType()==TypeDescriptor.Int) {
13        this.setResType(TypeDescriptor.Float);
14      } else this.setResType(TypeDescriptor.Error);
15    }
16  }
```