Analizzatore di Liveness in Java

UNIVERSITÀ
di **VERONA**

Astolfi Riccardo - Ferro Giacomo

Indice

- Background Liveness
- Definizione linguaggio e CFG
- Parser
- Classe Live
- Demo

Background Liveness

- studio delle variabili vive in un programma
- variabile è "viva" se si trova tra la sua definizione e l'uso
- una variabile è live su un cammino se il cammino non contiene ulteriori definizioni ed in mezzo c'è almeno un uso
- backward e possible

$$
\begin{aligned}
LiveOut(n) &= \begin{cases} \emptyset & \text{se } n = exit \\ \bigcup_{m \in Succ(n)} LiveIn(m) & \text{altrimenti} \end{cases} \\
LiveIn(n) &= Use(n) \cup (LiveOut(n) \backslash VarKill(n)) \\
LiveOut(n) &= \bigcup_{m \in Succ(n)} Use(m) \cup (LiveOut(m) \backslash VarKill(m))
\end{aligned}
$$

Definizione linguaggio e CFG

Linguaggio accettato dal parser:

- guardie come condizioni semplici e non composte
- assegnamenti liberi
- costrutti condizionali ed iterativi non annidati

variabili:	x
espressioni:	e
assegnamenti:	x=e;
costrutti condizionali:	if(x op e){ ... }
costrutti iterativi (1):	while(x op e){...}
costrutti iterativi (2):	for(x=e; x op e; x=x+1;){...}

- CFG costruito con JgraphT
- nodi re-implementati come classi Vertexgraph.java

Parser

JavaCC per la costruzione del parser

- classe principale Analisi.jj
- token nella sezione dell' analizzatore lessicale
- 5 metodi nella sezione dell'analizzatore sintattico
  - one_line() : riga per riga chiama il metodo appropriato
  - simpleassign() : identifica le componenti (variabili, operatori, …)
  - whilecycle(), forcycle() e ifcondition() costruiti a partire da simpleassign()

```
SKIP :
{" "|"\r"|"\t"|"\n"|"int"}
TOKEN :{
< AND : "&&" > | < OR : "||" > | < NOT : "!" > | < DIMENSOPER : ">"
    | "<" | ">=" | "<=" >
|
< OPERATOR : "+" | "-" | "*" | "%" | "/" > | < ogr : "{" > | < cgr
    : "}" >
|
< FOR : "for" > | < WHILE :"while" | "While" | "WHILE" > | < IF
    :"if">| < NULL:"NULL" > | < EXIT:"EXIT" >|
< INTEGER : ([ "0"-"9" ])+ >
|
< END : ";" > | < OPENPAR : "(" > |< CLOSEPAR : ")" > | < EQUAL :
    "=" >| < COMMA : "," >|
< ANNCLASS : [ "A"-"Z","a"-"z" ]
| (["A"-"Z","a"-"z","_"](["A"-"Z","a"-"z","0"-"9","_" ])*) >}
```

```
String one_line() :
{
  Vertexgraph s;
  String u= "";
  String risultato="";
}

{
  (
    (
      s = simpleassign()
      {
        g.addVertex(s);
        g.addEdge(nodoPrec,s);
        nodoPrec = s;
        }
      |
      u = whilecycle()
      |
      u = ifcondition()
      |
      u = forcycle()
    )
  )*
   {
     risultato = "Insieme dei nodi: \n" +
         g.vertexSet().toString()+"\n"+
              "Insieme degli archi: \n" + g.edgeSet().toString()+"\n"+
              "Insieme delle var generate: \n"+gen.toString()+"\n"+
              "Insieme delle var killate: \n"+kill.toString()+"\n";

     Live obj = new Live(i, kill, gen, g);

     String results = obj.compute_LiveOut(iterazioni);

     return risultato+"\n"+results;
   }
}
```

Live.java

- 2 ArrayList<Set<String>> per costruire la griglia di LiveOut
- aggiornamento iterativo LiveOut(i) e LiveOut(i+1)
- utilizzo set variabili killed e generated creati dal parser
- complessità: $O(|V|^2) = K * |V| * (|V|-1)$
  - K = for(numero iterazioni)
  - |V| = while(VertexSet(CFG))
  - O(|V|-1) = while(SuccessorList(node))

Link risorse

https://jgrapht.org/guide/UserOverview

https://javacc.org

www.eclipse.org

https://docs.oracle.com/javase/8/docs/api/javax/swing

Link progetto

https://github.com/GiacomoFerro/LivenessProject