Using Dependence Graphs for Slicing Functional Programs

Dr. Vadim Zaytsev aka @grammarware
IFL 2015
Using Dependence Graphs for Slicing Functional Programs

Dr. Vadim Zaytsev aka @grammarware
IFL 2015
read(text);
read(n);
lines = 1;
chars = 1;
subtext = "";
c = getChar(text);
c = getChar(text);
while (c != '\eof')
    if (c == '\n')
        then lines = lines + 1;
        chars = chars + 1;
    else chars = chars + 1;
    if (n != 0)
        then subtext = subtext ++ c;
        n = n - 1;
    c = getChar(text);
write(lines);
write(chars);
write(subtext);
Slicing

read(text);
read(n);
lines = 1;
chars = 1;
subtext = "";
c = getChar(text);
c = getChar(text);
while (c != '\eof')
    if (c == '\n')
        then lines = lines + 1;
            chars = chars + 1;
    else chars = chars + 1;
    if (n != 0)
        then subtext = subtext ++ c;
            n = n - 1;
    c = getChar(text);
write(lines);
write(chars);
write(subtext);

read(text);
read(n);
lines = 1;
chars = 1;
subtext = ";
c = getChar(text);
c = getChar(text);
while (c != \'\eof\')
    if (c == \'\n\')
        then lines = lines + 1;
        chars = chars + 1;
    else chars = chars + 1;
    if (n != 0)
        then subtext = subtext ++ c;
        n = n - 1;
    c = getChar(text);
write(lines);
write(chars);
write(subtext);

Slicing

module IFL15
where

main = do
  text ← readFile "...
  n ← readLn :: IO Int
  print $ 1 + (length . filter (=='\n')) text
  print $ length text
  print $ take n (filter (/='\n') text)

module IFL15

where

main = do
  text <- readFile "...
  n <- readLn :: IO Int
  putStrLn (1 + (length . filter (=='\n') text)
  putStrLn (length text)
  putStrLn (take n (filter (/='\n') text)

module IFL15
where

main = do
    text <- readFile "...
    n <- readLn :: IO Int
    return $ triple text n

triple :: String -> Int -> (Int, Int, String)
triple text n = (lines, chars, subtext) where
    lines = 1 + (length . filter (=='\n') ) text
    chars = length text
    subtext = take n (filter (/='\n') text)

Slicing

module IFL15
where

main = do
text <- readFile "..."
n <- readLn :: IO Int
return $ triple text n

triple :: String -> Int -> (Int, Int, String)
triple text n = (lines, chars, subtext) where
  lines = 1 + (length . filter (=='\n') text)
  chars = length text
  subtext = take n (filter (/='\n') text)

Slicing

module IFL15
where

main = do
    text <- readFile "...
    n <- readLn :: IO Int
    return $ triple text n

triple :: String -> Int -> (Int, Int, String)
triple text n = (lines, chars, subtext) where
    lines = 1 + (length . filter (=='\n') text)
    chars = length text
    subtext = take n (filter (/='\n') text)

Slicing

module IFL15
where

main = do
  text <- readFile "...
  n <- readLn :: IO Int
  return $ triple text n

triple :: String -> Int -> (Int, Int, String)
triple text n = (lines, chars, subtext) where
  lines = 1 + (length . filter (=='\n')) text
  chars = length text
  subtext = take n (filter (/='\n') text)

Related concepts

✓ Forward/backward slicing
✓ Dynamic/conditioned slicing
✓ constraints on input
✓ Chopping
✓ discover connection between I & O
✓ Amorphous slicing
✓ . . .
Amorphous Slicing

f :: Int -> Int
f cx = add cx 1

inc :: Int -> Int
inc a = add a 1

add :: Int -> Int -> Int
add a b = a + b

Amorphous Slicing

f :: Int -> Int
f cx = add cx 1

inc :: Int -> Int
inc a = add a 1

add :: Int -> Int -> Int
add a b = a + b

Amorphous Slicing

\[ f :: \text{Int} \rightarrow \text{Int} \]
\[ f \ cx = \ cx + 1 \]

\[ \text{inc} :: \text{Int} \rightarrow \text{Int} \]
\[ \text{inc} \ a = \text{add} \ a \ 1 \]

\[ \text{add} :: \text{Int} \rightarrow \text{Int} \rightarrow \text{Int} \]
\[ \text{add} \ a \ b = a + b \]

Forms of Slicing

3.2. Interrelations Between Slicing Techniques

The information in Table I can also be used to identify relations between slicing techniques. We have identified some relations and represented them in the graph of Figure 29, considering that each pair of related slicing techniques relate comparable slicing criteria. There are three kinds of arrows in the graph.

- **Generalization** ($S_1 \rightarrow S_2$): As slicing technique $S_2$ generalizes another technique $S_1$ if and only if all the slicing criteria that can be specified with $S_1$ can also be specified with $S_2$. This answers the question, is slicing technique $A$ a particular case of slicing technique $B$?

- **Superset** ($S_1 \supset S_2$): The slice produced by slicing technique $S_2$ is a superset of the slice produced by another technique $S_1$ if and only if all the statements in $S_1$ also belong to $S_2$. This answers the question, is the slice produced by slicing technique $A$ included in the slice produced by slicing technique $B$?

- **Composed of** ($S_1 \cdots \lesssim S_2$): As slicing technique $S_1$ is composed of the technique $S_2$ if and only if the slicing criterion of $S_1$ contains the slicing criterion of $S_2$. For example, chopping is composed of two slicing techniques (forward slicing and backward slicing).
Uses for slicing

✓ Debugging
  ✓ cf. Weiser CACM 1982
✓ Cohesion measurement
  ✓ cf. Ott&Bieman IST 1998
✓ Comprehension
  ✓ cf. De Lucia&Fasolino&Munro IWPC 1996
✓ Maintenance
  ✓ e.g. reuse
✓ Re-engineering
  ✓ e.g. clone detection

http://www0.cs.ucl.ac.uk/staff/mharman/sf.html
Goal: Library

http://github.io/grammarware/pdg
Using Dependence Graphs for Slicing Functional Programs

Dr. Vadim Zaytsev aka @grammarware
IFL 2015
Control Dependence

ack :: Int -> Int -> Int
ack 0 n = n + 1
ack m 0 = ack (m-1) 1
ack m n = ack (m-1) (ack m (n-1))

✓ Control flow
✓ execution path
✓ Exclude domination
✓ inevitable

J. de Bakker, E. de Vink, Control Flow Semantics, MIT Press, 1996.
ack :: Int -> Int -> Int
ack 0 n = n + 1
ack m 0 = ack (m-1) 1
ack m n = ack (m-1) (ack m (n-1))

✓ Control flow
✓ execution path
✓ Exclude domination
✓ inevitable
Data Dependence

ack :: Int -> Int -> Int
ack 0 n = n + 1
ack m 0 = ack (m-1) 1
ack m n = ack (m-1) (ack m (n-1))

✓ “Define” locations
✓ change
✓ “Use” locations
✓ access
ack :: Int -> Int -> Int
ack 0 n = n + 1
ack m 0 = ack (m-1) 1
ack m n = ack (m-1) (ack m (n-1))

✓ Merge
✓ CDG
✓ DDG
✓ into one multigraph

System Dependence

ack :: Int -> Int -> Int
ack 0 n = n + 1
ack m 0 = ack (m-1) 1
ack m n = ack (m-1) (ack m (n-1))

✓ Same as PDG
✓ interprocedural support

Variations

✓ Dynamic + Unified
✓ runtime info
✓ Probabilistic + Weighted
✓ trace coverage
✓ Values
✓ lazy eval
✓ OO
✓ methods, fields, classes

F–Statements Dep.

✓ m, f, dt, c, d
✓ “Functional Statements”
✓ Very high level
✓ architectural

Behaviours Dependence

✓ Edges
  ✓ data, control, behaviour
✓ Can handle
  ✓ pattern-driven dispatch
  ✓ expression decomposition
✓ But
  ✓ extremely Erlang-specific

RefactorErl

http://plc.inf.elte.hu/erlang/
Term Dependence

✓ “Program positions”
✓ Left and right hand sides
✓ S-edges inside terms
  ✓ structural
✓ C-edges link uses & defs
  ✓ control
✓ No higher order

“Erlang Dependence”

✓ (almost) all of the above
✓ Edges
  ✓ control, input, output, data, summary
✓ no deep decomposition
✓ no concurrency

W.I.P.

✓ CDG/DDG work
✓ not a serious challenge
✓ PDG works
✓ SDG works
✓ next?
Questions/Advice?

✓ @grammarware
✓ http://grammarware.net
✓ http://grammarware.github.io
✓ http://twitter.com/grammarware
✓ ...

...
Feedback

✓ slice modules
  ✓ comprehension: see where to start
  ✓ optimisation before deployment
✓ slicing across modules
  ✓ and versions of modules
✓ Clemens: general program trafo ± symbolic computation ± slicing
✓ Rinus: slicing modules for doc
✓ чувак в очках — оптимизация
✓ Philip: version slicing
✓ чувак из Брюсселя: слайс яваскрипта в VUB
PERSON: Vadim Zaytsev

DBLP: Zaytsev:Vadim

Facilitated 7 volumes:

- PubCh
- SciCo
- PrCo
- PrCo
- PubCh
- PubCh
- WebCh

Contributed to:

- 2014
- 2013
- 2013
- 2012
- 2011
- 2010
- 2009

Wrote 13 papers:

- CSMR-WCRE-2014-BaggeZ
  International workshop on open and original problems in software language engineering (AHB, VZ), p. 478.
- CSMR-WCRE-2014-Zaytsev
- MODELS-2014-Zaytsev
  Parsing in a Broad Sense (VZ, AHB), pp. 50–67.
- SLE-2013-Zaytsev
- WCRE-2013-BaggeZ
  Workshop on open and original problems in software language engineering (AHB, VZ), pp. 493–494.
- LDTA-2012-Zaytsev
  Notation-parametric grammar recovery (VZ), p. 9.
- SAC-2012-Zaytsev
  BNF was here: what have we done about the unnecessary diversity of notation for syntactic definitions (VZ), pp. 1910–1915.
- SLE-2011-FischerLZ
- SLE-2010-ZaytsevL
- GTTSE-2009-Zaytsev
  Language Convergence Infrastructure (VZ), pp. 481–497.
- IFM-2009-LammelZ
- SCAM-2009-J-LammelZ11