Fundamentals of Program Analysis

Manuel Oriol
January 9th, 2005
What is Program Analysis?

A set of techniques that consist in analyzing programs in order to prove properties on them.

Several techniques can be distinguished:
- Data flow analysis
- Constraint-based analysis
- Abstract Interpretation
- Types and effects analysis
- ...
Structures

- Abstract Syntax Trees (AST)

- Control-Flow Graphs (CFG)
Abstract Syntax Tree?

```plaintext
a:=1
b:=a+2
if (b>3) then
  Result:=b
else
  Result:=a
end
```
How ASTs are computed? Compilers...

- Use a lexical analyzer that gives you the tokens (e.g.: lex, flex, gelex...)

- Couple it with a parser that groups the tokens and gives the structure (e.g.: yacc, bison, geyacc...)... generally the structure is an AST
Advantages and Disadvantages of ASTs

Advantages:
- No more noise (comments, keywords etc...)
- Not ambiguous
- Exact

Disadvantages:
- Many similar forms: if/switch, for/foreach...
- Expressions may become big and going through them may become costly.

So???
a := 1
b := a + 2
if (b > 3) then
    Result := b
else
    Result := a
end
b := a
How CFGs are computed? Compilers...

- Take the AST
- Group the statements
- Remove the control-flow instructions and keep only jumps and tests
- Transform jumps into transitions
a := 1
b := a + 2
if (b > 3) then
    Result := b
else
    Result := a
    b := a
end
b := a
Advantages and Disadvantages

Advantages:
- no more similar forms
- expressions are easy to understand: only statements

Disadvantages:
- introduce temporaries
- less understandable, regarding to the original code and the general structure
So what can we do with that?

Example:
- how to prove that there is no call on a Void reference in this program?

```plaintext
create b.make(10)
from a:=1 until a>10 loop
  b.test
  a:=a+1
end
```
And what about that program?

create b.make(10)
from a:=1 until a>10 loop
  if (a=10) then b:=Void
  else b.test
  end
  a:=a+1
end
Dataflow Analysis

As we could see in previous examples:

- There is a need for an analysis of data manipulations

- Traditionally, it leads to optimizations (e.g. statements that have no influence may be removed), loops may have statements extracted from them etc...

- Based on all paths through programs

- It is the analysis of labels by excellence
- The set of labels for a node are what really matters as it is what we want to prove.

  **Labels** as the first thing to build and describe

- The rule to build the set of labels is by itself the analysis

  We can use a lot of ways to describe that but people usually use set theory in order to do that and describe it with a transitional set system on statements

  \( \text{In}(S), \text{Out}(S), \text{succ}(S), \text{pred}(S) \)
pred($S_0$)=$\{S_{-1}, S'_{-1}\}$

succ($S_0$)=$\{S_1, S'_1\}$

$In(S_0)$=$\{...\}$

$Out(S_0)$=$\{...\}$
Example: Attachment Analysis

Labels ::= varname=attached | varname=detached

\[ \text{In}(S) = \bigcup_{S_0 \in \text{pred}(S)} \text{Out}(S) \]

\[
\text{Out}(S= \text{create \ varname\.creationFeature}) = \\
\quad \text{In}(S) - \{\text{varname=detached}\} \cup \{\text{varname=attached}\}
\]

\[
\text{Out}(S= \text{varname:=\text{Void}}) = \\
\quad \text{In}(S) - \{\text{varname=attached}\} \cup \{\text{varname=detached}\}
\]

\[
\text{Out}(S= \text{varname:=\text{varname\.FeatureName(...)}}) = \\
\quad \text{Error if } \{\text{varname=detached}\} \in \text{In}(S) \\
\quad \text{In}(S) \text{ otherwise }
\]

What else needs to be explained?
Applications of the algorithm

```
create b.make(10)
from a:=1 until a>10 loop
    b.test
    a:=a+1
end
```

```
create b.make(10)
from a:=1 until a>10 loop
    if (a=10) then b:=Void
    else b.test
    end
    a:=a+1
end
```
An expression is available if the last value that was calculated for it is still the correct value.

Labels ::= \{expression\}

\[ \text{In}(S) = \cap_{S_0 \in \text{pred}(S)} \text{Out}(S_0) \]

\[ \text{Out}(S = \text{varname} := \text{expression}_0) = \]
\[ \text{In}(S) \cup \{\text{expression}_0\} - \{e \in \text{In}(S) \mid \text{varname} \in e\} \]

\[ \text{Gen}(S) \]

\[ \text{Kill}(S) \]
Application of the algorithm

\[
\begin{align*}
a & := a - b \\
b & := a + b \\
\text{if } (b > 3) & \text{ then} \\
& \quad \text{Result} := a - b \\
\text{else} & \\
& \quad \text{Result} := a + b \\
\text{end}
\end{align*}
\]
Live Variables Analysis

A variable \( v \) is live at a point \( p \) if it may be used before it is overwritten.

Labels ::= \{varname\}

\[
Out(S) = \bigcup_{S_0 \in \text{succ}(S)} In(S_0)
\]

\[
In(S = \text{varname} := \text{expression}) =
\quad Out(S) - \{\text{varname}\} \cup \{\text{vname} | \text{vname} \in \text{expression}\}
\]

\[
In(S | \text{expression} \in S) =
\quad Out(S) \cup \{\text{vname} | \text{vname} \in \text{expression}\}
\]

\[
\text{Kill}(S)
\]

\[
\text{Gen}(S)
\]
Application of the algorithm

\[
a := 1 \\
b := a + 2 \\
\text{if } (b > 3) \text{ then } \\
\quad \text{Result} := b \\
\text{else} \\
\quad \text{Result} := a \\
\quad b := a \\
\text{end} \\
b := a
\]
Very Busy Expressions Analysis

An expression e is very busy at a point p if all future execution path will need its evaluation before the value of e is changed.

Labels ::= {expressions}
Out(S) = \( \cap_{S0 \in \text{succ}(S)} \) In(S0)
In(S = varname:=…) = 
  \( \text{Out}(S) - \{e_0 \in \text{Out}(S) \mid \text{varname} \in e_0\} \) \text{ (apply first)}
In(S \ni \text{expression}_0) = \text{Out}(S) \cup \{\text{expression}_0\} \text{ (apply last)}
Application of the algorithm

\[
a := 2 \\
\text{if } (b > 3) \text{ then} \\
\quad \text{Result} := a + b \\
\text{else} \\
\quad \text{Result} := a + b \\
\text{end}
\]
Reaching Definition Analysis (also def-use)

A definition (an assignment) of v reaches a point in a program if there is no other definition of v in-between.

Labels ::= \{varname:statementNumber\}
In(S) = \bigcup_{S_0 \in \text{pred}(S)} \text{Out}(S_0)
Out(S= varname:=...) =
\quad \text{In}(S) - \{\text{varname:statementNumber}|0 \in \text{In}(S)\} \cup \
\quad \{\text{varname:statementNumber}(S)\}
Out(S) = \text{In}(S) \text{ otherwise}
Application of the algorithm

\[
\begin{align*}
a &: = 1 \\
b &: = a + 2 \\
\text{if } (b > 3) \text{ then} & \\
& \quad \text{Result} &: = b \\
\text{else} & \\
& \quad \text{Result} &: = a \\
& \quad b &: = a \\
\text{end} \\
b &: = a
\end{align*}
\]
Types of analysis...

**Forward Analysis:** the analysis goes in the same direction as the edges

**Backward Analysis:** the analysis goes in reverse direction comparing to the edges

**Must analysis:** all branches are accounted

**May analysis:** one branch is enough
Terminations of the Dataflow Analyses???

Label values generally form lattices so they are finite.

Most functions are monotonic on these lattices so they are terminating.

It is however needed to verify that they are!!!
Exercise 1

Find applications for each of the analyses we previously described:

- Available Expressions Analysis
- Live Variables Analysis
- Very Busy Expressions Analysis
- Reaching Definition Analysis
Exercise 2: Password leak

How to prove statically that a password is not going to be passed from its original definition to any output?

Setting:
- PASSWORD is a class
- Specify the interface
- Specify the analysis and its limitations
Exercise 3: Access rights for files

How to verify that a file can only be accessed by a thread that acquired the right to do it?

Setting:
- FILE is a class that has a limited number of features
- SECURITY_MANAGER grants accesses
- Specify the analysis and its limitations