Outline

Problem: in what sense are Control-Flow Analysis and Type Systems equivalent?

We survey two papers ([1] and [2]) hereafter respectively referred to as Palsberg and Heintze

1. **Palsberg's Closure Analysis**
   Constraint-based control-flow analysis

2. **Palsberg's Safety Analysis**
   Extending CFA to find safe programs

3. **Heintze's Control-Flow Analyses**
   Four equivalences with types

4. **From Control-Flow Information to Types**
   Building type information from CFA

Flow Analysis

- **Imperative** $\rightarrow$ Control Flow Analysis
  Information can be drawn from the syntax
  What elementary blocks may lead to what other elementary blocks?

- **Functional** $\rightarrow$ Closure Analysis
  Arguments of applications must be inferred
  What is the set of λ-abstractions to which a variable can be bound?

```
let f = fun x -> x 1 and
    g = fun y -> y + 2 and
    h = fun z -> z + 3 in
(f g) + (f h)
```

Palsberg's Closure Analysis: Presentation

Problem: For each application point, we look for the set of abstractions that can be applied

Setting the scene

E is a λ-term from the grammar

$$\boxed{E ::= x \mid \lambda x.E \mid E_1 E_2}$$

$[E]$ is a set variable of λ-bindings from Abs(E)

The specification

E subterms yield a constraint system $C(E)$

	Basic constraints
$\lambda x.F$	$\{\lambda x.F\} \subseteq [\lambda x.F]$
	Safety constraints
GH	$[G] \subseteq \text{Abs}(E)$
	Connecting constraints
GH	for each $\lambda x.F$ in E
	$\{\lambda x.F\} \subseteq [G] \Rightarrow [H] \subseteq [x]$
	$\{\lambda x.F\} \subseteq [G] \Rightarrow [F] \subseteq [GH]$

Set Constraint Based Analysis [1/2]

$$\boxed{E ::= \{\lambda x.F\} \mid \alpha \mid \emptyset \mid E_1 \cup E_2 \mid \{\lambda x.F\} \subseteq \alpha \Rightarrow E_1}$$

1. **Constraint generation**

The program text yields a constraint system S

$$S = \bigwedge_i E_i \subseteq \alpha_i$$

2. **Constraint resolution [1/3]**

Applying rewrite rules until closure yields S'

1.Elimination $\boxed{\emptyset \subseteq E}$ $\boxed{\alpha \subseteq \alpha}$ $\rightarrow \square$

2.Distributivity $\boxed{E_1 \cup E_2 \subseteq E_3}$

$\rightarrow$ $\boxed{E_1 \subseteq E_3}$ $\boxed{E_2 \subseteq E_3}$

3.Transitivity $\boxed{E_1 \subseteq \alpha}$ $\boxed{\alpha \subseteq E_2}$

$\rightarrow$ $\boxed{E_1 \subseteq \alpha}$ $\boxed{\alpha \subseteq E_2}$ $\boxed{E_1 \subseteq E_2}$

4.Modus ponens $\boxed{\{\lambda x.F\} \subseteq \alpha \Rightarrow E_1 \subseteq E_2}$

$\boxed{\{\lambda x.F\} \subseteq \alpha}$ $\rightarrow$ $\boxed{E_1 \subseteq E_2}$ $\boxed{\{\lambda x.F\} \subseteq \alpha}$

Set Constraint Based Analysis [2/2]

3. **Constraint resolution [2/3]**
Upper and lower bounds for variables yield S''
$L_i = \bigcup\{y \subseteq \alpha_i \mid y = \{\lambda x.F\} \vee y = \alpha_{i<j}\}$
$U_i = \bigcap\{\alpha_i \subseteq y \mid y = \{\lambda x.F\} \vee y = \alpha_{i<j}\}$

$$S'' = \bigwedge_i L_i \subseteq \alpha_i \subseteq U_i$$

4. **Constraint resolution [3/3]**
A solution is:

$$\begin{array}{rll} \sigma: & \{\alpha_i\}_i & \to 2^{\mathsf{Abs}(E)} \\ & \alpha_i & \mapsto \sigma(\alpha_i) = \{\lambda x.F \mid \lambda x.F \in L_i\} \end{array}$$

Minimality is:

$$\forall \sigma.\sigma' \left(\sigma \prec \sigma' \Leftrightarrow \forall i.\sigma(\alpha_i) \subseteq \sigma'(\alpha_i)\right)$$

A minimal solution is eventually drawn from S''

Palsberg's Closure Analysis: Example

$$\boxed{(\lambda x.x)(\lambda y.y)}$$

Constraint system $C((\lambda x.x)(\lambda y.y))$:

Basic

$\{\lambda x.x\} \subseteq [\lambda x.x]$

$\{\lambda y.y\} \subseteq [\lambda y.y]$

Safety

$[\lambda x.x] \subseteq \{\lambda x.x, \lambda y.y\}$

Connecting

$\{\lambda x.x\} \subseteq [\lambda x.x] \Rightarrow [\lambda y.y] \subseteq [x]$

$\{\lambda x.x\} \subseteq [\lambda x.x] \Rightarrow [x] \subseteq [(\lambda x.x)(\lambda y.y)]$

$\{\lambda y.y\} \subseteq [\lambda x.x] \Rightarrow [\lambda y.y] \subseteq [y]$

$\{\lambda y.y\} \subseteq [\lambda x.x] \Rightarrow [y] \subseteq [(\lambda x.x)(\lambda y.y)]$

Least solution:

$[x] = \{\lambda y.y\}$

$[y] = \emptyset$

$[\lambda x.x] = \{\lambda x.x\}$

$[\lambda y.y] = \{\lambda y.y\}$

$[(\lambda x.x)(\lambda y.y)] = \{\lambda y.y\}$

- $\lambda y.y$ is never applied ($[y] = \emptyset$)
- the program evaluates to $\lambda y.y$

Safety Analysis

Two kinds of execution errors:
- **trapped errors** (immediate effect)
- **untrapped errors** (delayed effect)

$$\boxed{\text{safety} \equiv \text{no untrapped errors}}$$

$$\boxed{\text{good behavior} \equiv \text{safety} + \text{no trapped errors}}$$

Safety may be ensured by:
- **type checking** (based on type inference)
- **safety analysis** (based on closure analysis)

Safety	Typed	Untyped
Safe	ML	Lisp
Unsafe	C	Assembler

Safety analysis accepts more safe programs

Palsberg and Schwartzbach's Safety Analysis: Presentation

E is a λ-term from the grammar

$$\boxed{E ::= x \mid \lambda x.E \mid E_1 E_2 \mid 0 \mid \mathsf{succ}\ E}$$

$[E]$ is a subset of Abs$(E) \cup \{\mathsf{Int}\}$

E subterms yield a constraint system $C(E)$

	Basic constraints
$\lambda x.F$	$\{\lambda x.F\} \subseteq [\lambda x.F]$
0	$\{\mathsf{Int}\} \subseteq [0]$
succ F	$\{\mathsf{Int}\} \subseteq [\mathsf{succ}\ F]$

	Safety constraints
GH	$[G] \subseteq$ Abs(E)
succ F	$[F] \subseteq \{\mathsf{Int}\}$

	Connecting constraints
GH	for each $\lambda x.F$ in E
	$\{\lambda x.F\} \subseteq [G] \Rightarrow [H] \subseteq [x]$
	$\{\lambda x.F\} \subseteq [G] \Rightarrow [F] \subseteq [GH]$

Palsberg and Schwartzbach's Safety Analysis: First Example

$$\boxed{\lambda x.x_1(\mathsf{succ}\; x_2)}$$

Constraint system $C(\lambda x.x_1(\mathsf{succ}\; x_2))$:

Basic

$\{\lambda x.x_1(\mathsf{succ}\; x_2)\} \subseteq [\lambda x.x_1(\mathsf{succ}\; x_2)]$

$\{\mathsf{Int}\} \subseteq [\mathsf{succ}\; x_2]$

Safety

$[x_1] \subseteq \{\lambda x.x_1(\mathsf{succ}\; x_2)\}$

$[x_2] \subseteq \{\mathsf{Int}\}$

Connecting

$\{\lambda x.x_1(\mathsf{succ}\; x_2)\} \subseteq [x_1]$
$\quad \Rightarrow [\mathsf{succ}\; x_2] \subseteq [x]$

$\{\lambda x.x_1(\mathsf{succ}\; x_2)\} \subseteq [x_1]$
$\quad \Rightarrow [x_1(\mathsf{succ}\; x_2)] \subseteq [x_1(\mathsf{succ}\; x_2)]$

Under closure, the constraint system has no
- $\{\mathsf{Int}\} \subseteq \mathsf{Abs}(\lambda x.x_1(\mathsf{succ}\; x_2))$
- $\{\lambda x.F\} \subseteq \{\mathsf{Int}\}$

constraints, therefore the solution is safe.

The program is safe since it is not applied!

Palsberg and Schwartzbach's Safety Analysis: First Example

Least solution:

$$[x] = \emptyset$$
$$[x_1] = \emptyset$$
$$[x_2] = \emptyset$$
$$[\mathsf{succ}\ x_2] = \{\mathsf{Int}\}$$
$$[x_1(\mathsf{succ}\ x_2)] = \emptyset$$
$$[\lambda x.x_1(\mathsf{succ}\ x_2)] = \{\lambda x.x_1(\mathsf{succ}\ x_2)\}$$

The analysis says:

- this program cannot take any argument,
- yet it is a function, and
- it returns whatever type you need

Palsberg and Schwartzbach's Safety Analysis: Second Example

$$\boxed{(\lambda x.x_1 x_2)(\lambda y.y)}$$

Constraint system $C((\lambda x.xx)(\lambda y.y))$:

Basic

$\{\lambda x.x_1 x_2\} \subseteq [\lambda x.x_1 x_1]$
$\{\lambda y.y\} \subseteq [\lambda y.y]$

Safety

$[\lambda x.x_1 x_2] \subseteq \{\lambda x.x_1 x_2, \lambda y.y\}$
$[x_1] \subseteq \{\lambda x.x_1 x_2, \lambda y.y\}$

Connecting

$\{\lambda x.x_1 x_2\} \subseteq [x_1] \Rightarrow [x_2] \subseteq [x]$
$\{\lambda x.x_1 x_2\} \subseteq [x_1] \Rightarrow [x_1 x_2] \subseteq [x_1 x_2]$
$\{\lambda y.y\} \subseteq [x_1] \Rightarrow [x_2] \subseteq [y]$
$\{\lambda y.y\} \subseteq [x_1] \Rightarrow [y] \subseteq [x_1 x_2]$
$\{\lambda x.x_1 x_2\} \subseteq [\lambda x.x_1 x_2] \Rightarrow [\lambda y.y] \subseteq [x]$
$\{\lambda x.x_1 x_2\} \subseteq [\lambda x.x_1 x_2] \Rightarrow [x_1 x_2] \subseteq [(\lambda x.x_1 x_2)(\lambda y.y)]$
$\{\lambda y.y\} \subseteq [\lambda x.x_1 x_2] \Rightarrow [\lambda y.y] \subseteq [y]$
$\{\lambda y.y\} \subseteq [\lambda x.x_1 x_2] \Rightarrow [y] \subseteq [(\lambda x.x_1 x_2)(\lambda y.y)]$

Palsberg and Schwartzbach's Safety Analysis: Second Example

Least solution:

$[x] = \{\lambda y.y\}$
$[x_1] = \{\lambda y.y\}$
$[x_2] = \{\lambda y.y\}$
$[x_1 x_2] = \{\lambda y.y\}$
$[\lambda x.x_1 x_2] = \{\lambda x.x_1 x_2\}$
$[y] = \{\lambda y.y\}$
$[\lambda y.y] = \{\lambda y.y\}$
$[(\lambda x.x_1 x_2)(\lambda y.y)] = \{\lambda y.y\}$

The analysis says:
- the program evaluates to the identity

Note that the left subterm has a recursive type

Heintze's Formalization

It is basically the same presentation!

z is an annotated term from the grammar

$$\boxed{z ::= z' : annot(z)}$$

$$\boxed{\quad z' ::= x \mid (\lambda^l x.z) \mid (z_1 z_2) \mid 0 \mid (\mathsf{Succ}\ z)}$$

where
- $l \in$ LABELS
- $annot(z) \in 2^{\text{LABELS}} \times$ TYPES

Sets of labels from LABELS are substituted for sets of α-converted abstractions from $\mathsf{Abs}(E)$

Well-annotated versions of terms are ordered

1. $0 : L \triangleright 0 : L'$ if $L \supseteq L'$
2. $x : L \triangleright x : L'$ if $L \supseteq L'$
3. $(\lambda^l x.z) : L \triangleright (\lambda^l x.z') : L'$ if $L \supseteq$ and $z \triangleright z'$
4. $(z_1 z_2) : L \triangleright (z_1' z_2') : L'$ if $L \supseteq L'$, $z_1 \triangleright z_1'$, $z_2 \triangleright z_2'$
5. $(\mathsf{Succ}\ z) : L \triangleright (\mathsf{Succ}\ z') : L'$ if $L \supseteq L'$ and $z \triangleright z'$

Heintze $\mathrm{CFA}_{\subseteq}$

z is an annotated term from the grammar

$$\boxed{z ::= x : \gamma \mid (\lambda^l x.z) : \gamma \mid (z_1 z_2) : \gamma \mid 0 : \gamma \mid (\mathsf{succ}\ z) : \gamma}$$

where
- $l \in$ LABELS
- $\gamma = L \subseteq \text{LABELS} \cup \{\text{Int}\}$

z is well-annotated if conditions below are met

Basic conditions

$\lambda^l x.z' : \gamma$	$\{l\} \subseteq \gamma$
$0 : \gamma$	$\{\text{Int}\} \subseteq \gamma$
$(\mathsf{succ}\ z') : \gamma$	$\{\text{Int}\} \subseteq \gamma$

Safety conditions

$(z_1 z_2) : \gamma$	$\{\text{Int}\} \not\subseteq annot(z_1)$
$(\mathsf{succ}\ z') : \gamma$	$annot(z') \subseteq \{\text{Int}\}$

Connecting conditions

$(z_1 z_2) : \gamma$	for each $\lambda^l x : \gamma_x.z'$ in z $annot(z_2) \subseteq \gamma_x$ $annot(z') \subseteq \gamma$

Heintze CFA$_{\subseteq,\neg Y}$

A well-annotated versions of $(\lambda^l x.xx)(\lambda^k x.xx)$ is $\left(\lambda^l x.((x : \{k\})(x : \{k\})) : \emptyset\right) : \{l\}$ $\left(\lambda^k x.((x : \{k\})(x : \{k\})) : \{k\}\right) : \emptyset$

We exclude programs involving recursion

z is well-annotated if conditions below are met

	Basic conditions
$\lambda^l x.z' : \gamma$	$\{l\} \subseteq \gamma$
$0 : \gamma$	$\{\mathsf{Int}\} \subseteq \gamma$
$(\mathsf{succ}\ z') : \gamma$	$\{\mathsf{Int}\} \subseteq \gamma$
$\lambda^l x : \gamma_x.z : \gamma$	$(\forall l' \in \gamma_x)(l \succ l')$
	Safety conditions
$(z_1 z_2) : \gamma$	$\{\mathsf{Int}\} \not\subseteq annot(z_1)$
$(\mathsf{succ}\ z') : \gamma$	$annot(z') \subseteq \{\mathsf{Int}\}$
	Connecting conditions
$(z_1 z_2) : \gamma$	for each $\lambda^l x : \gamma_x.z'$ in z
	$annot(z_2) \subseteq \gamma_x$
	$annot(z') \subseteq \gamma$

Heintze CFA$_=$

We restrict the inclusion relation to equality

z is well-annotated if conditions below are met

	Basic conditions
$\lambda^l x.z' : \gamma$	$\{l\} \subseteq \gamma$
$0 : \gamma$	$\{\mathsf{Int}\} = \gamma$
$(\mathsf{succ}\ z') : \gamma$	$\{\mathsf{Int}\} = \gamma$
	Safety conditions
$(z_1 z_2) : \gamma$	$\{\mathsf{Int}\} \not\subseteq annot(z_1)$
$(\mathsf{succ}\ z') : \gamma$	$annot(z') = \{\mathsf{Int}\}$
	Connecting conditions
$(z_1 z_2) : \gamma$	for each $\lambda^l x : \gamma_x.z'$ in z
	$annot(z_2) = \gamma_x$
	$annot(z') = \gamma$

Heintze CFA$_{=,\neg Y}$

We exclude programs involving recursion
&
We restrict the inclusion relation to equality

z is well-annotated if conditions below are met

	Basic conditions
$\lambda^l x.z' : \gamma$	$\{l\} \subseteq \gamma$
$0 : \gamma$	$\{\mathsf{Int}\} = \gamma$
$(\mathsf{succ}\ z') : \gamma$	$\{\mathsf{Int}\} = \gamma$
$\lambda^l x : \gamma_x.z : \gamma$	$(\forall l' \in \gamma_x)(l \succ l')$

	Safety conditions
$(z_1 z_2) : \gamma$	$\{\mathsf{Int}\} \not\subseteq annot(z_1)$
$(\mathsf{succ}\ z') : \gamma$	$annot(z') = \{\mathsf{Int}\}$

	Connecting conditions
$(z_1 z_2) : \gamma$	for each $\lambda^l x : \gamma_x.z'$ in z
	$annot(z_2) = \gamma_x$
	$annot(z') = \gamma$

Control-Flow Information to Types [1/3]

Control-flow analysis gives each subterm a set of potential values it may assume at run-time

A set variable/of labels can be:

- $\{\mathsf{Int}\}$
  The subterm is an integer
- a subset of abstractions ($\subseteq \mathsf{Abs}(E)$/LABELS)
  The subterm is a function
  domain: the intersection of all the domains
  codomain: the union of all the codomains
- any other kind of set
  The subterm is an integer or a function
  But is is only passed during reduction
- $\emptyset$
  The subterm is neither an integer nor a function
  Anyway, it is not touched during reduction

Control-Flow Information to Types [2/3]

A type of a variable is an upper bound of the range of values it can assume during execution

- Control-Flow Analysis:
each subterm $\rightarrow$ a set of values
- Type inference:
each subterm $\rightarrow$ a collection of values

Let us assume a function that maps

Control Flow Analysis	Type
{Int}	Int
a subset of abstractions	Type $\rightarrow$ Type
any other kind of set	$\top$
$\emptyset$	$\bot$

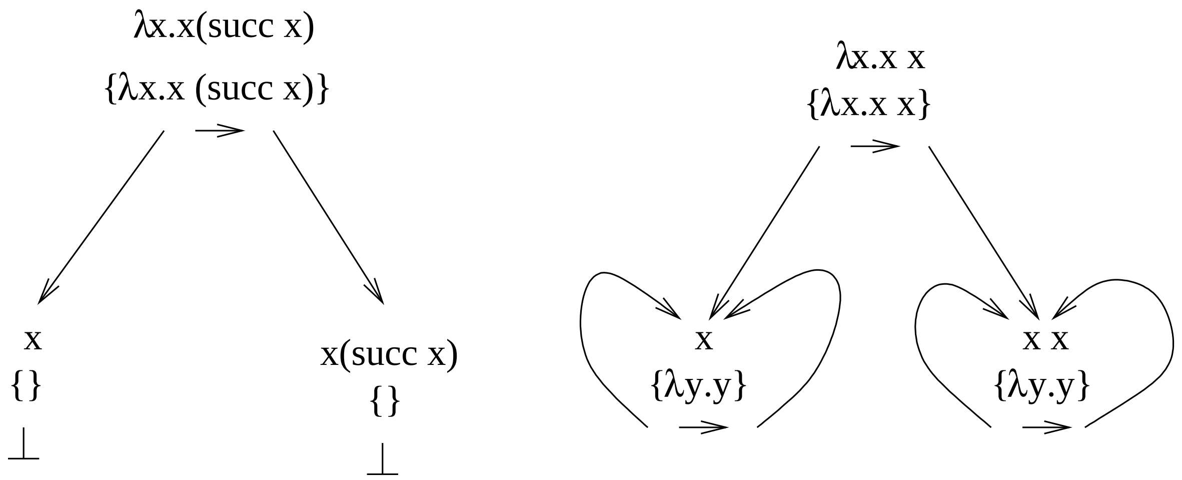
Control-Flow Information to Types: Examples

$$\boxed{\lambda x.x_1(\mathsf{succ}\ x_2)}$$

$[x] = [x_1] = [x_2] = [x_1(\mathsf{succ}\ x_2)] = \emptyset$
$[\mathsf{succ}\ x_2] = \{\mathsf{Int}\}$
$[\lambda x.x_1(\mathsf{succ}\ x_2)] = \{\lambda x.x_1(\mathsf{succ}\ x_2)\}$

$$\boxed{(\lambda x.x_1 x_2)(\lambda y.y)}$$

$[x] = [x_1] = [x_2] = [x_1 x_2] = [y] = [\lambda y.y] =$
$\quad [(\lambda x.x_1 x_2)(\lambda y.y)] = \{\lambda y.y\}$
$[\lambda x.x_1 x_2] = \{\lambda x.x_1 x_2\}$



Control-Flow Information to Types [3/3]

Heintze:
- defines four control-flow analyses, and
- finds the corresponding type systems

Analysis	subsumption	recursion	types
$\mathrm{CFA}_{\subseteq}$	$\circ$	$\circ$	?
$\mathrm{CFA}_{\subseteq,\neg Y}$	$\circ$	$\times$	?
$\mathrm{CFA}_{=}$	$\times$	$\circ$	?
$\mathrm{CFA}_{=,\neg Y}$	$\times$	$\times$	?

Bibliography

[1] Jens Palsberg, Patrick O'Keefe, *A Type System Equivalent to Flow Analysis*, POPL, 1995.

[2] Nevin Heintze, *Control-Flow Analysis and Type Systems*, SAS, 1995.