Static analysis and all that

Martin Steffen

IfI UiO

Spring 2016

Static analysis and all that

Martin Steffen

IfI UiO

Spring 2016

Plan

- approx. 15 lectures, details see web-page
- flexible time-schedule, depending on progress/interest
- covering parts/following the structure of textbook [1], concentrating on
  - overview
  - data-flow
  - control-flow
  - type- and effect systems
- on request, new parts possible
- helpful prior knowledge: having at least heard of
  - typed lambda calculi (especially for CFA)
  - simple type systems
  - operational semantics
  - lattice theory, fixpoints, induction

  but things needed will be covered . . .

Plan

- introduction/motivation into the field
- short survey about the material: 5 main topics
  - data flow analysis
  - control flow analysis/constraint based analysis
  - [Abstract interpretation]
  - type and effect systems
  - [algorithmic issues]
- 2 lessons

SA: why and what?

What:

- static: at “compile time”
- analysis: deduction of program properties
  - automatic/decidable
  - formally, based on semantics

Why:

- error catching
  - enhancing program quality
  - catching common “stupid” errors without bothering the user much
  - spotting errors early
  - certain similarities to model checking
  - examples: type checking, uninitialized variables (potential nil-pointer deref’s), unused code
- optimization: based on analysis, transform the “code”[1], such the the result is “better”
  - examples: precalculation of results, optimized register allocation . . .

success-story for formal methods

---

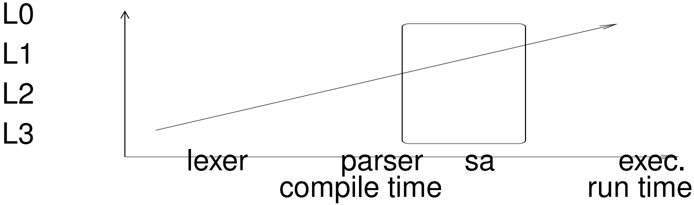
[1] source code, intermediate code at various levels

Nature of SA

- programs have differerent "semantical phases"
- corresponding to Chomsky's hierarchy
- "static" = in principle: before run-time, but in praxis, "*context-free*"[2]
- since: run-time most often: undecidable

⇒ static analysis as approximation

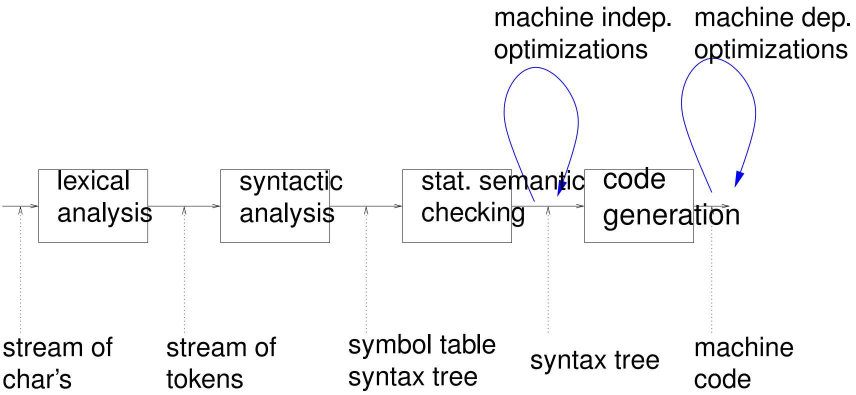
- See [1, Figure 1.1]



---

[2] playing with words, one could call full-scale (hand?) verification "static" analysis, and likewise call lexical analysis a static analysis.

Phases



machine indep. optimizations

machine dep. optimizations

lexical analysis

syntactic analysis

stat. semantic checking

code generation

stream of char's

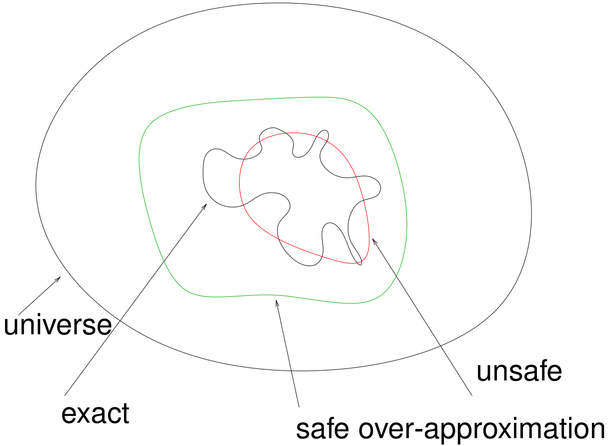
stream of tokens

symbol table syntax tree

syntax tree

machine code

SA as approximation



While-language

- simple, prototypical imperative language:
    - “untyped”
    - simple control structure: while, conditional, sequencing
    - simple data (numerals, booleans)
- abstract syntax $\neq$ concrete syntax
- disambiguation when needed: (...), or { ... } or begin ... end

a	$::=$	$x \mid n \mid a \; \mathtt{op}_a \; a$	arithm. expressions
b	$::=$	$\text{true} \mid \text{false} \mid \text{not } b \mid b \; \mathtt{op}_b \; b \mid a \; \mathtt{op}_r \; a$	boolean expr.
S	$::=$	$x := a \mid \text{skip} \mid S_1 ; S_2$	statements
		$\mathtt{if} \; b \; \mathtt{then} \; S \; \mathtt{else} \; S \mid \mathtt{while} \; b \; \mathtt{do} \; S$	

Table: Abstract syntax

While-language: labelling

- associate flow information

$\Rightarrow$ labels

- elementary block = labelled item
- identify basic building blocks
- unique labelling

$$
\begin{array}{lcll}
a & ::= & x \mid n \mid a \,\mathtt{op}_a\, a & \text{arithm. expression} \\
b & ::= & \text{true} \mid \text{false} \mid \text{not } b \mid b \,\mathtt{op}_b\, b \mid a \,\mathtt{op}_r\, a & \text{boolean expr.} \\
S & ::= & [x := a]^l \mid [\text{skip}]^l \mid S_1 ; S_2 & \text{statements} \\
 & & \mathtt{if}[b]^l \,\mathtt{then}\, S \,\mathtt{else}\, S \mid \mathtt{while}[b]^l \,\mathtt{do}\, S &
\end{array}
$$

Table: Abstract syntax

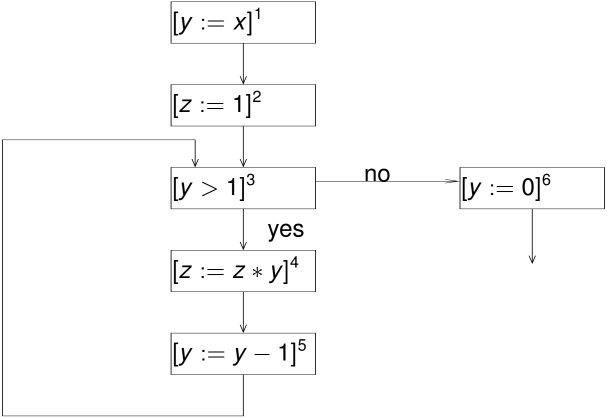
Example: factorial

$$y := x; z := 1; \texttt{while}\, y > 1\, \texttt{do}(z := z * y; y := y - 1); y := 0$$

- input variable: x
- output variable: z

Example: factorial

$[y := x]^1; [z := 1]^2; \texttt{while}[y > 1]^3\ \texttt{do}([z := z{*}y]^4; [y := y{-}1]^5); [y := 0]^6$



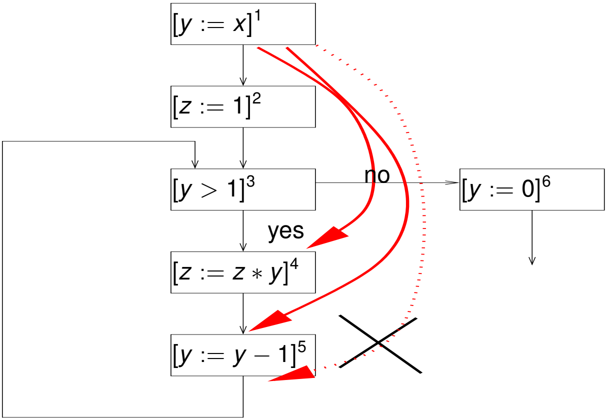
Reaching definitions analysis

- “definition” of x: assignment to x: $x := a$
- better name: reaching assignment analysis
- first, simple example of data flow analysis

Reaching def’s

assignment (= “definition”) $[x := a]^l$ may reach a program point, if there exists an execution where x was last assigned at l, when the mentioned program point is reached.

Factorial: reaching assignment



- $(y, 1)$ (short for $[y := x]^1$) may reach:
  - the entry to 4 (short for $[z := z * y]^4$).
  - the exit to 4 (not in the picture as arrow)
  - the entry to 5
  - but: not the exit to 5

Factorial: reaching assignments

- "points" in the program: entry and exit to elementary blocks/labels
- ?: special label (not occurring otherwise), representing entry to the program, i.e., $(x, ?)$ represents *initial* (uninitialized) value of x
- full information: pair of "functions"

$$\mathrm{RD} = (\mathrm{RD}_{entry}, \mathrm{RD}_{exit}) \tag{1}$$

l	RD_{entry}	RD_{exit}
1	$(x,?),(y,?),(z,?)$	$(x,?),(y,1),(z,?)$
2	$(x,?),(y,1),(z,?)$	$(x,?),(y,1),(z,2)$
3	$(x,?),(y,1),(y,5),(z,2),(z,4)$	$(x,?),(y,1),(y,5),(z,2),(z,4)$
4	$(x,?),(y,1),(y,5),(z,2),(z,4)$	$(x,?),(y,1),(y,5),(z,4)$
5	$(x,?),(y,1),(y,5),(z,4)$	$(x,?),(y,5),(z,4)$
6	$(x,?),(y,1),(y,5),(z,2),(z,4)$	$(x,?),(y,6),(z,2),(z,4)$

Reaching assignments: remarks

- *elementary* blocks of the form
  - $[b]^l$: entry/exit information coincides
  - $[x := a]^l$: entry/exit information (in general) different
- at program exit: $(x, ?)$, x is input variable
- table: “best” information = “smallest”:
  - additional pairs in the table: still safe
  - removing labels: unsafe
- note: still an approximation
  - no real (= run time) data, no real execution, only data flow
  - approximate since
    - in *concrete* runs: at each point in that run, there is exactly one last assignment, not a set
    - label represents (potentially infinitely many) runs
  - e.g.: at program exit in concrete run: either $(z, 2)$ or else $(z, 4)$

Data flow analysis

- standard: representation of program as flow graph
  - nodes: elementary blocks with labels
  - edges: flow of control
- two approaches (both (especially here) quite similar)
  - equational approach
  - constraint-based approach

From flow graphs to equations

- associate an equation system with the flow graph:
  - describing the “flow of information”
  - here:
    - the information related to reaching assignments
    - information imagined to flow forwards
- solution of the equations
  - describe safe approximations
  - not unique, interest in the least (or *largest*) solution
  - here:
    - give back RD of equation (1) on slide 16

Equations for RD and factorial: intra-block

first type: local, “intra-block”:

- flow through each individual block
- relating for each elementary block its exit with its entry

elementary block: $[y := x]^1$

$$\mathrm{RD}_{exit}(1) \quad = \quad \mathrm{RD}_{entry}(1) \setminus \{(y, l) \mid l \in \mathbf{Lab}\} \cup \{(y, 1)\} \qquad (2)$$

Equations for RD and factorial: intra-block

first type: local, "intra-block":

- flow through each individual block
- relating for each elementary block its exit with its entry

elementary block: $[y > 1]^3$

$$\begin{aligned} RD_{exit}(1) &= RD_{entry}(1) \setminus \{(y, l) \mid l \in \mathbf{Lab}\} \cup \{(y, 1)\} \qquad (2) \\ RD_{exit}(3) &= RD_{entry}(3) \end{aligned}$$

Equations for RD and factorial: intra-block

first type: local, "intra-block":

- flow through each individual block
- relating for each elementary block its exit with its entry

all equations with RD_{exit} as "left-hand side"

$$
\begin{aligned}
\mathrm{RD}_{exit}(1) &= \mathrm{RD}_{entry}(1) \setminus \{(y, l) \mid l \in \mathbf{Lab}\} \cup \{(y, 1)\} \qquad (2)\\
\mathrm{RD}_{exit}(2) &= \mathrm{RD}_{entry}(2) \setminus \{(z, l) \mid l \in \mathbf{Lab}\} \cup \{(z, 2)\}\\
\mathrm{RD}_{exit}(3) &= \mathrm{RD}_{entry}(3)\\
\mathrm{RD}_{exit}(4) &= \mathrm{RD}_{entry}(4) \setminus \{(z, l) \mid l \in \mathbf{Lab}\} \cup \{(z, 4)\}\\
\mathrm{RD}_{exit}(5) &= \mathrm{RD}_{entry}(5) \setminus \{(y, l) \mid l \in \mathbf{Lab}\} \cup \{(y, 5)\}\\
\mathrm{RD}_{exit}(6) &= \mathrm{RD}_{entry}(6) \setminus \{(y, l) \mid l \in \mathbf{Lab}\} \cup \{(y, 6)\}
\end{aligned}
$$

Equations for RD and factorial: inter-block

second type: global, “inter-block”

- reflecting the control flow graph
- flow between the elementary blocks, following the control-flow edges
- relating the entry of each[3] block with the exits of other blocks, that are connected via an edge
- initial block: mark variables as uninitialized

$$
\begin{aligned}
RD_{entry}(2) &= RD_{exit}(1) \qquad (3) \\
\\
RD_{entry}(4) &= RD_{exit}(3) \\
RD_{entry}(5) &= RD_{exit}(4) \\
RD_{entry}(6) &= RD_{exit}(3)
\end{aligned}
$$

---

[3] except (in general) the initial block.

Equations for RD and factorial: inter-block

second type: global, "inter-block"

- reflecting the control flow graph
- flow between the elementary blocks, following the control-flow edges
- relating the entry of each[3] block with the exits of other blocks, that are connected via an edge
- initial block: mark variables as uninitialized

$$
\begin{aligned}
RD_{entry}(2) &= RD_{exit}(1) \\
RD_{entry}(3) &= RD_{exit}(2) \cup RD_{exit}(5) \\
RD_{entry}(4) &= RD_{exit}(3) \\
RD_{entry}(5) &= RD_{exit}(4) \\
RD_{entry}(6) &= RD_{exit}(3)
\end{aligned}
\tag{3}
$$

---

[3] except (in general) the initial block.

Equations for RD and factorial: inter-block

second type: global, “inter-block”

- reflecting the control flow graph
- flow between the elementary blocks, following the control-flow edges
- relating the entry of each[3] block with the exits of other blocks, that are connected via an edge
- initial block: mark variables as uninitialized

$$
\begin{array}{rcl}
RD_{entry}(2) & = & RD_{exit}(1) \\
RD_{entry}(3) & = & RD_{exit}(2) \cup RD_{exit}(5) \\
RD_{entry}(4) & = & RD_{exit}(3) \\
RD_{entry}(5) & = & RD_{exit}(4) \\
RD_{entry}(6) & = & RD_{exit}(3) \\
\\
RD_{entry}(1) & = & \{(x,?),(y,?),(z,?)\}
\end{array}
\tag{3}
$$

---

[3] except (in general) the initial block.

RD: general scheme

Intra: for assignments $[x := a]^l$

$$\mathrm{RD}_{exit}(l) = \mathrm{RD}_{entry}(l) \setminus \{(x, l') \mid l' \in \mathbf{Lab}\} \cup \{(x, l)\} \tag{4}$$

for other blocks $[b]^l$ (side-effect free)

$$\mathrm{RD}_{exit}(l) = \mathrm{RD}_{entry}(l) \tag{5}$$

Inter:

$$\mathrm{RD}_{entry}(l) = \bigcup_{l' \to l} \mathrm{RD}_{exit}(l') \tag{6}$$

Initial: l: label of the initial block[4]

$$\mathrm{RD}_{entry}(l) = \{(x, ?) \mid x \text{ is a program variable}\} \tag{7}$$

---

[4] isolated entry.

The equation system as fix point

- in the example: solution to the equation system = 12 sets

$$\mathrm{RD}_{entry}(1), \ldots, \mathrm{RD}_{exit}(6)$$

- i.e., the $\mathrm{RD}_{entry}(l), \mathrm{RD}_{exit}(l)$ are the variables of the equation system, of "type": "set of (x, l)-pairs"
- $\vec{\mathrm{RD}}$: the mentioned twelve-tuple

⇒ equation system understood as function F

Equations

$$\vec{\mathrm{RD}} = F(\vec{\mathrm{RD}})$$

- more explicitly, broken down to its 12 parts (the "equations")

$$F(\vec{\mathrm{RD}}) = (F_{entry}(1)(\vec{\mathrm{RD}}), F_{exit}(1)(\vec{\mathrm{RD}}), \ldots, F_{exit}(6)(\vec{\mathrm{RD}}))$$

- for instance:

$$F_{entry}(3) = (\ldots, \mathrm{RD}_{exit}(2), \ldots, \mathrm{RD}_{exit}(5), \ldots) = \mathrm{RD}_{exit}(2) \cup \mathrm{RD}_{exit}(5)$$

The least solution

- **Var**$_*$ = variables "of interest" (i.e., occurring), **Lab**$_*$: labels of interest
- here **Var**$_* = \{x, y, z\}$, **Lab**$_* = \{?, 1, \ldots, 6\}$

$$F : (2^{\mathbf{Var}_* \times \mathbf{Lab}_*})^{12} \to (2^{\mathbf{Var}_* \times \mathbf{Lab}_*})^{12} \tag{8}$$

- domain $(2^{\mathbf{Var}_* \times \mathbf{Lab}_*})^{12}$: partially ordered pointwise:

$$\vec{\mathrm{RD}} \sqsubseteq \vec{\mathrm{RD}}' \text{ iff } \forall i.\ \mathrm{RD}_i \subseteq \mathrm{RD}'_i \tag{9}$$

$\Rightarrow$ complete lattice

Constraint-based approach

- here, for DFA: a simple “variant” of the equational approach
- trivial rearrangement of the entry-exit relationships
- instead of equations: inequations (sub-set instead of set-equality)
- in more complex settings: constraints become more complex, no split in exit- and entry-constraints

Factorial program: intra-block constraints

elementary block: $[y := x]^1$

$$\mathrm{RD}_{exit}(1) \supseteq \mathrm{RD}_{entry}(1) \setminus \{(y, l) \mid l \in \mathbf{Lab}\}$$
$$\mathrm{RD}_{exit}(1) \supseteq \{(y, 1)\}$$

Factorial program: intra-block constraints

elementary block: $[y > 1]^3$

$$\mathrm{RD}_{exit}(3) \supseteq \mathrm{RD}_{entry}(3)$$

Factorial program: intra-block constraints

all equations with RD_{exit} as left-hand side

$$
\begin{aligned}
&\mathsf{RD}_{exit}(1) \supseteq \mathsf{RD}_{entry}(1) \setminus \{(y,l) \mid l \in \mathbf{Lab}\}\\
&\mathsf{RD}_{exit}(1) \supseteq \{(y,1)\}\\
&\mathsf{RD}_{exit}(2) \supseteq \mathsf{RD}_{entry}(2) \setminus \{(z,l) \mid l \in \mathbf{Lab}\}\\
&\mathsf{RD}_{exit}(2) \supseteq \{(z,2)\}\\
&\mathsf{RD}_{exit}(3) \supseteq \mathsf{RD}_{entry}(3)\\
&\mathsf{RD}_{exit}(4) \supseteq \mathsf{RD}_{entry}(4) \setminus \{(z,l) \mid l \in \mathbf{Lab}\}\\
&\mathsf{RD}_{exit}(4) \supseteq \{(z,4)\}\\
&\mathsf{RD}_{exit}(5) \supseteq \mathsf{RD}_{entry}(5) \setminus \{(y,l) \mid l \in \mathbf{Lab}\}\\
&\mathsf{RD}_{exit}(5) \supseteq \{(y,5)\}\\
&\mathsf{RD}_{exit}(6) \supseteq \mathsf{RD}_{entry}(6) \setminus \{(y,l) \mid l \in \mathbf{Lab}\}\\
&\mathsf{RD}_{exit}(6) \supseteq \{(y,6)\}
\end{aligned}
$$

Factorial program: inter-block constraints

cf. slide 27 ff.: inter-block equations:

$$\begin{array}{rcl}
RD_{entry}(2) & = & RD_{exit}(1) \\
RD_{entry}(3) & = & RD_{exit}(2) \cup RD_{exit}(5) \\
RD_{entry}(4) & = & RD_{exit}(3) \\
RD_{entry}(5) & = & RD_{exit}(4) \\
RD_{entry}(6) & = & RD_{exit}(3)
\end{array}$$

$$RD_{entry}(1) = \{(x,?),(y,?),(z,?)\}$$

Factorial program: inter-block constraints

splitting of composed right-hand sides + using $\supseteq$ instead of $=$:

$$
\begin{array}{lcl}
\mathsf{RD}_{entry}(2) & \supseteq & \mathsf{RD}_{exit}(1) \\
\mathsf{RD}_{entry}(3) & \supseteq & \mathsf{RD}_{exit}(2) \\
\mathsf{RD}_{entry}(3) & \supseteq & \mathsf{RD}_{exit}(5) \\
\mathsf{RD}_{entry}(4) & \supseteq & \mathsf{RD}_{exit}(3) \\
\mathsf{RD}_{entry}(5) & \supseteq & \mathsf{RD}_{exit}(4) \\
\mathsf{RD}_{entry}(6) & \supseteq & \mathsf{RD}_{exit}(3) \\
\\
\mathsf{RD}_{entry}(1) & \supseteq & \{(x,?),(y,?),(z,?)\}
\end{array}
$$

least solution revisited

- instead of $F(\vec{\mathsf{RD}}) = \vec{\mathsf{RD}}$

$$F(\vec{\mathsf{RD}}) \sqsubseteq \vec{\mathsf{RD}} \tag{10}$$

  for the same F
- clear: solution to the equation system ⇒ solution to the constraint system
- important: least solution coincides!

Control-flow analysis

- goal: which elem. blocks lead to which other elem. blocks
- for while-language: immediate (labelled elem. blocks, resp., graph)
- complex for: more advanced features, higher-order languages, oo languages . . .
- here: prototypical “higher-order” functional language (λ-calc.)
- formulated as constraint based analysis

Simple example

```
let f   = fn x => x 1;
    g   = fn y => y + 2;
    h   = fn z => z + 3;
in (f g) + (f h)
```

- higher-order function *f*:
- for simplicity untyped
- local definitions[5] via let-in

goal (more specific)

for each function application, which function may be applied

- interesting above: $x\ 1$

---

[5] That's something else than assignment. We will not consider it (here) anyway.

Example

- more complex language ⇒ more complex labelling
- “elem. blocks” can be nested
- *all* syntactic constructs (expressions) are labeled
- consider:

$$(\mathtt{fn}\, x \Rightarrow x)\ (\mathtt{fn}\, y \Rightarrow y)$$

Example

- more complex language ⇒ more complex labelling
- “elem. blocks” can be nested
- *all* syntactic constructs (expressions) are labeled
- consider:

$$[\,[\mathtt{fn}\, x \Rightarrow [x]^1]^2\ [\mathtt{fn}\, y \Rightarrow [y]^3]^4\,]^5$$

- functional language: side effect free

⇒ no need to distinguish entry and exit of labelled blocks.

- data of the analysis: $(\hat{C}, \hat{\rho})$, pair of functions

  abstract cache: $\hat{C}(l)$: set of values/function abstractions, the subexpression labelled l may evaluate to

  abstract env.: $\hat{\rho}$: values, x may be bound to

The constraint system

- ignoring "`let`" here: three syntactic constructs ⇒ three kinds of constraints

1. function abstraction: $[\texttt{fn}\, x \Rightarrow x]^l$
2. variables: $[x]^l$
3. application: $[f\ g]^l$

- relating $\hat{C}$, $\hat{\rho}$, and the program in form of constraints (subsets, order-relation)

The constraint system

- ignoring "`let`" here: three syntactic constructs ⇒ three kinds of constraints

1. function abstraction: $[\texttt{fn}\, x \Rightarrow x]^l$
2. variables: $[x]^l$
3. application: $[f\ g]^l$

- relating $\hat{C}$, $\hat{\rho}$, and the program in form of constraints (subsets, order-relation)

The constraint system

- ignoring “`let`” here: three syntactic constructs ⇒ three kinds of constraints

1. function abstraction: $[\mathtt{fn}\, x \Rightarrow x]^l$
2. variables: $[x]^l$
3. application: $[f\ g]^l$

- relating $\hat{C}$, $\hat{\rho}$, and the program in form of constraints (subsets, order-relation)
- function abstractions

$$\begin{array}{rcl} \{\mathtt{fn}\, x \Rightarrow [x]^1\} & \subseteq & \hat{C}(2) \\ \{\mathtt{fn}\, y \Rightarrow [y]^3\} & \subseteq & \hat{C}(4) \end{array}$$

The constraint system

- ignoring "`let`" here: three syntactic constructs ⇒ three kinds of constraints

1. function abstraction: $[\texttt{fn}\, x \Rightarrow x]^l$
2. variables: $[x]^l$
3. application: $[f\ g]^l$

- relating $\hat{C}$, $\hat{\rho}$, and the program in form of constraints (subsets, order-relation)
- variables

$$\begin{array}{rcl} \hat{\rho}(x) & \subseteq & \hat{C}(1) \\ \hat{\rho}(y) & \subseteq & \hat{C}(3) \end{array}$$

The constraint system

- ignoring "`let`" here: three syntactic constructs ⇒ three kinds of constraints

1. function abstraction: $[\texttt{fn}\, x \Rightarrow x]^l$
2. variables: $[x]^l$
3. application: $[f\ g]^l$

- relating $\hat{C}$, $\hat{\rho}$, and the program in form of constraints (subsets, order-relation)
- application: connecting function entry and (body) exit with the argument

$$\begin{array}{lcl} \hat{C}(4) & \subseteq & \hat{\rho}(x) \\ \hat{C}(1) & \subseteq & \hat{C}(5) \end{array}$$

The constraint system

- ignoring “`let`” here: three syntactic constructs ⇒ three kinds of constraints

1. function abstraction: $[\mathtt{fn}\, x \Rightarrow x]^l$
2. variables: $[x]^l$
3. application: $[f\ g]^l$

- relating $\hat{C}$, $\hat{\rho}$, and the program in form of constraints (subsets, order-relation)
- application: connecting function entry and (body) exit with the argument but:
- also $[\mathtt{fn}\, y \Rightarrow [y]^3]^4$ is a candidate at 2! (according to $\hat{C}(2)$)

$$\begin{array}{rcl}
\hat{C}(4) & \subseteq & \hat{\rho}(x) \\
\hat{C}(1) & \subseteq & \hat{C}(5) \\
\mathbf{\hat{C}(4)} & \subseteq & \hat{\rho}(y) \\
\hat{C}(3) & \subseteq & \mathbf{\hat{C}(5)}
\end{array}$$

The constraint system

- ignoring "`let`" here: three syntactic constructs ⇒ three kinds of constraints

1. function abstraction: $[\texttt{fn}\, x \Rightarrow x]^l$
2. variables: $[x]^l$
3. application: $[f\ g]^l$

- relating $\hat{C}$, $\hat{\rho}$, and the program in form of constraints (subsets, order-relation)

$$
\begin{array}{lcrcl}
\{\texttt{fn}\, x \Rightarrow [x]^1\} \subseteq \hat{C}(2) & \Rightarrow & \hat{C}(4) & \subseteq & \hat{\rho}(x) \\
\{\texttt{fn}\, x \Rightarrow [x]^1\} \subseteq \hat{C}(2) & \Rightarrow & \hat{C}(1) & \subseteq & \hat{C}(5) \\
\{\texttt{fn}\, y \Rightarrow [y]^3\} \subseteq \hat{C}(2) & \Rightarrow & \hat{\mathbf{C}}(\mathbf{4}) & \subseteq & \hat{\rho}(y) \\
\{\texttt{fn}\, y \Rightarrow [y]^3\} \subseteq \hat{C}(2) & \Rightarrow & \hat{C}(3) & \subseteq & \hat{\mathbf{C}}(\mathbf{5})
\end{array}
$$

The least solution

$$
\begin{array}{rcl}
\hat{C}(1) & = & \{\texttt{fn}\, y \Rightarrow [y]^3\} \\
\hat{C}(2) & = & \{\texttt{fn}\, x \Rightarrow [x]^1\} \\
\hat{C}(3) & = & \emptyset \\
\hat{C}(4) & = & \{\texttt{fn}\, y \Rightarrow [y]^3\} \\
\hat{C}(5) & = & \{\texttt{fn}\, y \Rightarrow [y]^3\} \\
\hline
\hat{\rho}(x) & = & \{\texttt{fn}\, y \Rightarrow [y]^3\} \\
\hat{\rho}(y) & = & \emptyset
\end{array}
$$

Effects: Intro

- type system: "classical" static analysis:

$$t : T$$

- judgment: "term/program phrase has type T"
- in general: context-sensitive judgments[6]

$$\Gamma \vdash t : T$$

  Γ: assumptions/context
- here: "non-standard" type systems: effects and annotations
- natural setting: typed languages, here: trivial! setting (while-language)

---

[6] remember Chomsky ...

“Trival” type system

- setting: while-language
- each statement maps: state to states
- Σ: type of states

judgment

$$S : \Sigma \to \Sigma \tag{11}$$

- specified as a derivation system
- note: partial correctness assertion

“Trival” type system: rules

$$[x := a]^l : \Sigma \to \Sigma \qquad \textsc{Ass}$$

$$[\text{skip}]^l : \Sigma \to \Sigma \qquad \textsc{Skip}$$

$$\frac{S_1 : \Sigma \to \Sigma \qquad S_2 : \Sigma \to \Sigma}{S_1 ; S_2 : \Sigma \to \Sigma}\ \textsc{Seq}$$

$$\frac{S : \Sigma \to \Sigma}{\texttt{while}[b]^l\,\texttt{do}\,S : \Sigma \to \Sigma}\ \textsc{While}$$

$$\frac{S_1 : \Sigma \to \Sigma \qquad S_2 : \Sigma \to \Sigma}{\texttt{if}[b]^l\,\texttt{then}\,S_1\,\texttt{else}\,S_2 : \Sigma \to \Sigma}\ \textsc{Cond}$$

Types, effects, and annotations

annotated type system

$$\vdash S : \Sigma_1 \rightarrow \Sigma_2 \tag{12}$$

effect system

$$\vdash S : \Sigma \xrightarrow{\varphi} \Sigma \tag{13}$$

type and effect system (TES)

- often effect system + annotated type system (border fuzzy)
- annotated type system
  - Σ_i: property of state ("$\Sigma_i \subseteq \Sigma$")
  - "abstract" properties: invariants, a variable is positive, etc.
- effect system

  "statement S maps state to state, with (potential ...) effect φ"
  - effect φ: e.g.: errors, exceptions, file/resource access, ...

Annotated type systems

- example: reaching definitions/assignments in While-lang.
- 2 flavors
    1. annotated base types: $S : \mathrm{RD}_1 \to \mathrm{RD}_2$
    2. annotated type constructors: $S : \Sigma \xrightarrow[\mathrm{RD}]{X} \Sigma$

Annotated base types

- judgment

$$S : \mathrm{RD}_1 \to \mathrm{RD}_2 \tag{14}$$

- $\mathrm{RD} \subseteq 2^{\mathbf{Var} \times \mathbf{Lab}}$
- auxiliary functions
  - note: every S has one “initial” elementary block, potentially more than one “at the end”
  - $init(S)$: the (unique) label at the entry of S
  - $final(S)$: the set of labels at the exits of S

“meaning” of judgment $S : \mathrm{RD}_1 \to \mathrm{RD}_2$:

“RD_1 is the set of var/label reaching the entry of S and RD_2 the corresponding set at the exit(s) of S”:

$$\begin{aligned} \mathrm{RD}_1 &= \mathrm{RD}_{entry}(init(S)) \\ \mathrm{RD}_2 &= \bigcup\{\mathrm{RD}_{exit}(l) \mid l \in final(S)\} \end{aligned}$$

$$[x := a]^{l'} : \mathrm{RD} \to \mathrm{RD} \setminus \{(x,l) \mid l \in \mathbf{Lab}\} \cup \{(x,l')\} \qquad \textsc{ass}$$

$$[\mathrm{skip}]^{l} : \mathrm{RD} \to \mathrm{RD} \qquad \textsc{skip}$$

$$\frac{S_1 : \mathrm{RD}_1 \to \mathrm{RD}_2 \qquad S_2 : \mathrm{RD}_2 \to \mathrm{RD}_3}{S_1; S_2 : \mathrm{RD}_1 \to \mathrm{RD}_3}\ \textsc{Seq}$$

$$\frac{S_1 : \mathrm{RD}_1 \to \mathrm{RD}_2 \qquad S_2 : \mathrm{RD}_1 \to \mathrm{RD}_2}{\texttt{if}[b]^{l}\,\texttt{then}\,S_1\,\texttt{else}\,S_2 : \mathrm{RD}_1 \to \mathrm{RD}_2}\ \textsc{If}$$

$$\frac{S : \mathrm{RD} \to \mathrm{RD}}{\texttt{while}[b]^{l}\,\texttt{do}\,S : \mathrm{RD} \to \mathrm{RD}}\ \textsc{While}$$

$$\frac{S : \mathrm{RD}_1' \to \mathrm{RD}_2' \qquad \mathrm{RD}_1 \subseteq \mathrm{RD}_1' \qquad \mathrm{RD}_2' \subseteq \mathrm{RD}_2}{S : \mathrm{RD}_1 \to \mathrm{RD}_2}\ \textsc{Sub}$$

Meaning of annotated judgment

"Meaning" of judgment $S : \mathrm{RD}_1 \to \mathrm{RD}_2$:

"RD_1 is *the* set of var/label reaching the entry of S and RD_2 the corresponding set at the exit(s) of S":

$$\begin{array}{rcl} \mathrm{RD}_1 & = & \mathrm{RD}_{entry}(init(S)) \\ \mathrm{RD}_2 & = & \bigcup\{\mathrm{RD}_{exit}\, l \mid l \in final(S)\} \end{array}$$

- Be careful:

$$\mathtt{if}[b]^l\,\mathtt{then}\,S_1\,\mathtt{else}\,S_2$$

- more concretely

$$\mathtt{if}[b]^l\,\mathtt{then}[x := y]^{l_1}\,\mathtt{else}[y := x]^{l_2}$$

Meaning of annotated judgment

Once again: “Meaning” of judgment $S : \mathrm{RD}_1 \rightarrow \mathrm{RD}_2$:

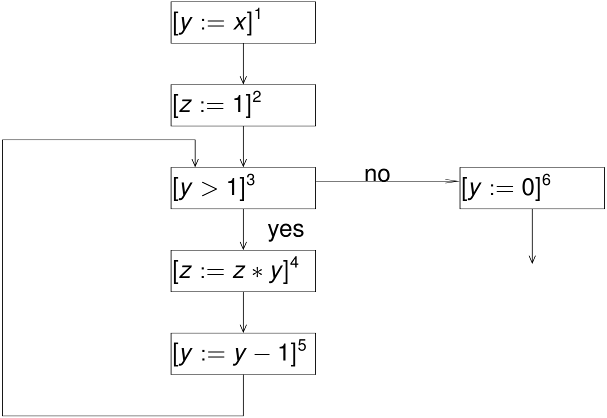
“RD_1 is the set of var/label reaching the entry of S and RD_2 the corresponding set at the exit(s) of S”:

$$\begin{array}{rrcl} & \text{if} \quad \mathrm{RD}_1 & \subseteq & \mathrm{RD}_{entry}(init(S)) \\ \text{then} & \forall l \in final(S).\ \mathrm{RD}_{exit}(l) & \subseteq & \mathrm{RD}_2 \end{array}$$

- compare subsumption rule SUB
- subsumption adds necessary slack
- similar to the contraint formulation
- Remember: data flow equations and their (possible/minimal) solution

Example: factorial

$$[y := x]^1; [z := 1]^2; \texttt{while}[y > 1]^3\ \texttt{do}([z := z{*}y]^4; [y := y{-}1]^5); [y := 0]^6$$



$[y := x]^1$

$[z := 1]^2$

$[y > 1]^3$

no

$[y := 0]^6$

yes

$[z := z * y]^4$

$[y := y - 1]^5$

$$\dfrac{[y := x]^1 : \mathrm{RD}_0 \to \{?_x, 1, ?_z\} \qquad \dfrac{[z := 1]^2 : \{?_x, 1, ?_z\} \to \{?_x, 1, 2\} \quad f_3 : \{?_x, 1, 2\} \to \mathrm{RD}_{final}}{f_2 : \{?_x, 1, ?_z\} \to \mathrm{RD}_{final}}}{f : \mathrm{RD}_0 \to \mathrm{RD}_{final}}$$

$$\mathrm{RD}_0 = \{?_x, ?_y, ?_z\} \quad \mathrm{RD}_{final} = \{?_x, 6, 2, 4\}$$

type sub-derivation for the rest $f_3 = \texttt{while}\ldots; [y := 0]^6$
loop invariant

$$\mathrm{RD}_{body} = \{?_x, 1, 5, 2, 4\}$$

$$\dfrac{\dfrac{\dfrac{\dfrac{\begin{array}{c}[z := \_]^4 : \mathrm{RD}_{body} \to \{?_x, 1, 5, 4\} \\ [y := \_]^5 : \{?_x, 1, 5, 4\} \to \{?_x, 5, 4\}\end{array}}{f_{body} : \mathrm{RD}_{body} \to \{?_x, 5, 4\}}\ \textsc{Sub}}{f_{body} : \mathrm{RD}_{body} \to \mathrm{RD}_{body}}}{f_{while} : \mathrm{RD}_{body} \to \mathrm{RD}_{body}}\ \textsc{Sub}}{f_{while} : \{?_x, 1, 2\} \to \mathrm{RD}_{body}} \qquad [y := 0]^6 : \mathrm{RD}_{body} \to \mathrm{RD}_{final}$$

$$\overline{\hspace{20em}}$$

$$f_3 : \{?_x, 1, 2\} \to \mathrm{RD}_{final}$$

Annotated type constructors

- alternative approach of annotated type systems
- arrow constructor itself annotated
- annotion of →: flavor of effect system
- judgment

$$S : \Sigma \underset{\text{RD}}{\longrightarrow} \Sigma$$

- annotation with RD (corresponding to the post-condition from above) alone is not enough

Annotated type constructors

- alternative approach of annotated type systems
- arrow constructor itself annotated
- annotion of →: flavor of effect system
- judgment

$$S : \Sigma \xrightarrow[\mathrm{RD}]{X} \Sigma$$

- annotation with RD (corresponding to the post-condition from above) alone is not enough
- also need: the variables "being" changed
- Meaning

“S maps states to states, where RD is the set of reaching definition, S may produce and X the set of var's S must (= unavoidably) assign

$$[x := a]^l : \Sigma \xrightarrow[\{(x,l)\}]{\{x\}} \Sigma \qquad \textsc{Ass}$$

$$[\mathrm{skip}]^l : \Sigma \xrightarrow[\emptyset]{\emptyset} \Sigma \qquad \textsc{Skip}$$

$$\frac{S_1 : \Sigma \xrightarrow[\mathrm{RD}_1]{X_1} \Sigma \qquad S_2 : \Sigma \xrightarrow[\mathrm{RD}_2]{X_2} \Sigma}{S_1 ; S_2 : \Sigma \xrightarrow[\mathrm{RD}_1 \setminus X_2 \cup \mathrm{RD}_2]{X_1 \cup X_2} \Sigma} \textsc{Seq}$$

$$\frac{S_1 : \Sigma \xrightarrow[\mathrm{RD}]{X} \Sigma \qquad S_2 : \Sigma \xrightarrow[\mathrm{RD}]{X} \Sigma}{\texttt{if}[b]^l\, \texttt{then}\, S_1\, \texttt{else}\, S_2 : \Sigma \xrightarrow[\mathrm{RD}]{X} \Sigma} \textsc{If}$$

$$\frac{S : \Sigma \xrightarrow[\mathrm{RD}]{X} \Sigma}{\texttt{while}[b]^l\, \texttt{do}\, S : \Sigma \xrightarrow[\mathrm{RD}]{\emptyset} \Sigma} \textsc{While}$$

$$\frac{S : \Sigma \xrightarrow[\mathrm{RD}']{X'} \Sigma \qquad X \subseteq X' \qquad \mathrm{RD}' \subseteq \mathrm{RD}}{S : \Sigma \xrightarrow[\mathrm{RD}]{X} \Sigma} \textsc{Sub}$$

Effect systems

- this time: functional language[7]
- starting point: simple type system
- judgment:

$$\Gamma \vdash e : \tau$$

- Γ: type environment, “mapping” from var’s to types
- types: bool, int, and $\tau \to \tau$

---

[7] same as for constraint-based cfa.

$$\frac{\Gamma(x) = \tau}{\Gamma \vdash x : \tau}\ \textsc{Var}$$

$$\frac{\Gamma, x{:}\tau_1 \vdash e : \tau_2}{\Gamma \vdash \underset{\pi}{\texttt{fn}}\, x \Rightarrow e : \tau_1 \to \tau_2}\ \textsc{Abs}$$

$$\frac{\Gamma \vdash e_1 : \tau_1 \to \tau_2 \qquad \Gamma \vdash e_2 : \tau_1}{\Gamma \vdash e_1\ e_2 : \tau_2}\ \textsc{App}$$

Effect: Call tracking analysis

call tracking analysis:

Determine: for each subexpression, which function abstractions may be applied *during* its evaluation.

⇒ set of function names

- annotate: function type with latent effect

⇒ annotated types: $\hat{\tau}$: base types as before, arrow types:

$$\hat{\tau}_1 \xrightarrow{\varphi} \hat{\tau}_2 \tag{15}$$

- functions from τ_1 to τ_2, where in the execution, functions from set φ are called.
- judgment

$$\hat{\Gamma} \vdash e : \hat{\tau} \;\&\; \varphi \tag{16}$$

$$\frac{\hat{\Gamma}(x) = \hat{\tau}}{\hat{\Gamma} \vdash x : \hat{\tau} \mathbin{\&} \emptyset}\ \textsc{Var}$$

$$\frac{\Gamma, x{:}\hat{\tau}_1 \vdash e : \hat{\tau}_2 \mathbin{\&}\ \varphi}{\Gamma \vdash \underset{\pi}{\texttt{fn}}\, x \Rightarrow e : \hat{\tau}_1 \overset{\varphi \cup \{\pi\}}{\rightarrow} \hat{\tau}_2 \mathbin{\&} \emptyset}\ \textsc{Abs}$$

$$\frac{\hat{\Gamma} \vdash e_1 : \hat{\tau}_1 \overset{\varphi}{\rightarrow} \hat{\tau}_2 \mathbin{\&} \varphi_1 \qquad \hat{\Gamma} \vdash e_2 : \hat{\tau}_1 \mathbin{\&} \varphi_2}{\hat{\Gamma} \vdash e_1\ e_2 : \hat{\tau}_2 \mathbin{\&} \varphi \cup \varphi_1 \cup \varphi_2}\ \textsc{App}$$

Call tracking: example

$$\dfrac{\dfrac{x{:}\mathrm{int} \overset{\{Y\}}{\to} \mathrm{int} \vdash x{:}\mathrm{int} \overset{\{Y\}}{\to} \mathrm{int} \mathbin{\&} \emptyset}{\vdash (\underset{X}{\mathtt{fn}}\; x \Rightarrow x) : (\mathrm{int} \overset{\{Y\}}{\to} \mathrm{int}) \overset{\{X\}}{\to} (\mathrm{int} \overset{\{Y\}}{\to} \mathrm{int}) \mathbin{\&} \emptyset} \qquad \vdash (\underset{Y}{\mathtt{fn}}\; y \Rightarrow y) : \mathrm{int} \overset{\{Y\}}{\to} \mathrm{int} \mathbin{\&} \emptyset}{\vdash (\underset{X}{\mathtt{fn}}\; x \Rightarrow x)\;(\underset{Y}{\mathtt{fn}}\; y \Rightarrow y) : \mathrm{int} \overset{\{Y\}}{\to} \mathrm{int} \mathbin{\&} \{X\}}$$

Chaotic iteration

- back to Data flow/reaching def's
- goal: solve

$$\vec{RD} = F(\mathrm{RD}) \quad \text{or} \quad \vec{RD} \sqsubseteq F(\mathrm{RD})$$

- F: monotone, finite domain
- straightforward/naive approach

$$\begin{aligned} \text{init: } & \vec{\mathrm{RD}}_0 = F^0(\emptyset) \\ \text{iterate: } & \vec{\mathrm{RD}}_{n+1} = F(\vec{\mathrm{RD}}_n) = F^{n+1}(\emptyset) \text{ until stabilization} \end{aligned}$$

- approach to implement that: chaotic iteration
- abbrev:

$$\begin{aligned} \vec{\mathrm{RD}} &= (\mathrm{RD}_1, \ldots, \mathrm{RD}_{12}) \\ F(\vec{\mathrm{RD}}) &= F(\vec{\mathrm{RD}}, \ldots, \vec{\mathrm{RD}}) \end{aligned}$$

Chaotic iteration (for RD)

```
Input:    example equations for reaching definitions

Output:   least solution: $\vec{RD} = (RD_1, \ldots, RD_{12})$

Method: step 1:  initialization
                 $RD_1 := \emptyset; \ldots; RD_{12} := \emptyset$

        step 2:  iteration

                 while $RD_j \neq F_j(RD_1, \ldots, RD_{12})$ for some $j$
                 do
                       $RD_j := F_j(RD_1, \ldots, RD_{12})$
```

References I

[1] F. Nielson, H.-R. Nielson, and C. L. Hankin.
Principles of Program Analysis.
Springer-Verlag, 1999.