Lecture 13
Effect systems

Motivation

We have so far seen many analyses which deal with control- and data-flow properties of pure languages.

However, many languages contain operations with *side-effects*, so we must also be able to analyse and safely transform these impure programs.

Effect systems, a form of inference-based analysis, are often used for this purpose.

Side-effects

A side-effect is some event — typically a *change of state* — which occurs as a result of evaluating an expression.

- "`x++`" changes the value of variable `x`.

- "`malloc(42)`" allocates some memory.

- "`print 42`" outputs a value to a stream.

Side-effects

As an example language, we will use the lambda calculus extended with *read* and *write* operations on "channels".

$$e ::= x \mid \lambda x.e \mid e_1\ e_2 \mid \xi?x.e \mid \xi!e_1.e_2$$

- ξ represents some channel name.

- $\xi?x.e$ reads an integer from the channel named ξ, binds it to x, and returns the result of evaluating e.

- $\xi!e_1.e_2$ evaluates e_1, writes the resulting integer to channel ξ, and returns the result of evaluating e_2.

Side-effects

Some example expressions:

$$\xi?x.\ x$$

read an integer from channel ξ and return it

$$\xi!x.\ y$$

write the (integer) value of x to channel ξ and return the value of y

$$\xi?x.\ \zeta!x.\ x$$

read an integer from channel ξ, write it to channel ζ and return it

Side-effects

Ignoring their side-effects, the typing rules for these new operations are straightforward.

Side-effects

$$\frac{\Gamma[x : int] \vdash e : t}{\Gamma \vdash \xi?x.e : t} \quad (\text{READ})$$

$$\frac{\Gamma \vdash e_1 : int \quad \Gamma \vdash e_2 : t}{\Gamma \vdash \xi!e_1.e_2 : t} \quad (\text{WRITE})$$

Effect systems

However, in order to perform any transformations on a program in this language it would be necessary to pay attention to its potential side-effects.

For example, we might need to devise an analysis to tell us which channels may be read or written during evaluation of an expression.

We can do this by modifying our existing type system to create an *effect system* (or "type and effect system").
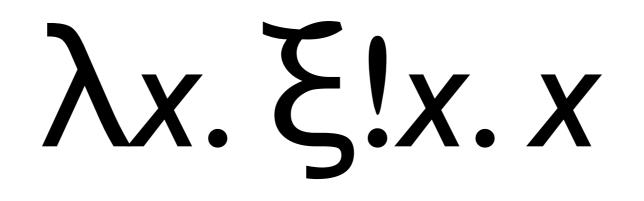
Effect systems

First we must formally define our *effects*:

An expression has effects F.
F is a set containing elements of the form

$$R\xi \qquad \text{read from channel } \xi$$

$$W\xi \qquad \text{write to channel } \xi$$

Effect systems

For example:

$$\xi?x.\ x \qquad F = \{\ R_\xi\ \}$$

$$\xi!x.\ y \qquad F = \{\ W_\xi\ \}$$

$$\xi?x.\ \zeta!x.\ x \qquad F = \{\ R_\xi, W_\zeta\ \}$$

Effect systems

But we also need to be able to handle expressions like

$$\lambda x.\ \xi!x.\ x$$

whose evaluation doesn't have any *immediate* effects.

In this case, the effect W_ξ may occur *later*, whenever this newly-created function is applied.

Effect systems

To handle these *latent effects* we extend the syntax of types so that function types are annotated with the effects that may occur when a function is applied:
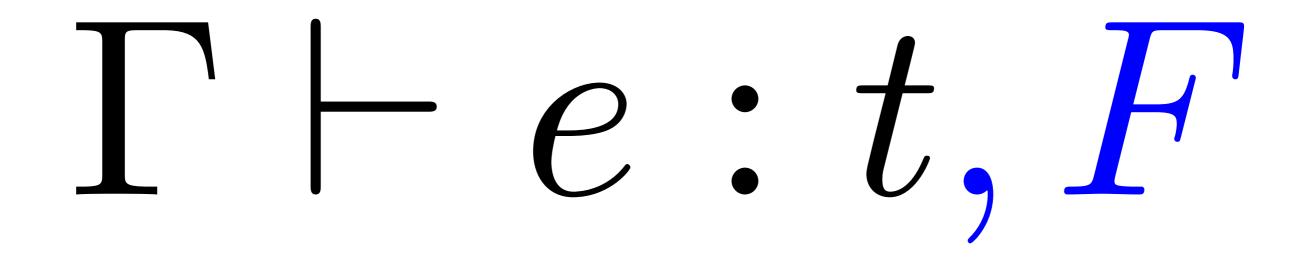
$$t ::= int \mid t_1 \xrightarrow{F} t_2$$

Effect systems

So, although it has no immediate effects, the type of

$$\lambda x.\ \xi!x.\ x$$

is

$$\text{int} \xrightarrow{\{\ W_\xi\ \}} \text{int}$$

Effect systems

We can now modify the existing type system to make an effect system — an inference system which produces judgements about the type *and effects* of an expression:

$$\Gamma \vdash e : t, \textcolor{blue}{F}$$

Effect systems

$$\frac{\Gamma[x:int] \vdash e : t}{\Gamma \vdash \xi?x.e : t} \quad (\text{Read})$$

$$\frac{\Gamma \vdash e_1 : int \quad \Gamma \vdash e_2 : t}{\Gamma \vdash \xi!e_1.e_2 : t} \quad (\text{Write})$$

Effect systems

$$\frac{\Gamma[x : int] \vdash e : t, F}{\Gamma \vdash \xi?x.e : t, \{R_\xi\} \cup F} \quad (\text{READ})$$

$$\frac{\Gamma \vdash e_1 : int, F \quad \Gamma \vdash e_2 : t, F'}{\Gamma \vdash \xi!e_1.e_2 : t, F \cup \{W_\xi\} \cup F'} \quad (\text{WRITE})$$

Effect systems

$$\frac{}{\Gamma[x:t] \vdash x:t,\{\}} \quad (\text{Var})$$

$$\frac{\Gamma[x:t] \vdash e:t',F}{\Gamma \vdash \lambda x.e : t \xrightarrow{F} t',\{\}} \quad (\text{Lam})$$

$$\frac{\Gamma \vdash e_1 : t \xrightarrow{F''} t',F \quad \Gamma \vdash e_2 : t,F'}{\Gamma \vdash e_1 e_2 : t', F \cup F' \cup F''} \quad (\text{App})$$

Effect systems

$$\frac{\dfrac{}{\{x : int, y : int\} \vdash x : int, \{\}}}{\{x : int, y : int\} \vdash \xi!x.\, x : int, \{W_\xi\}}$$

$$\frac{\{x : int, y : int\} \vdash \xi!x.\, x : int, \{W_\xi\}}{\{y : int\} \vdash \lambda x.\, \xi!x.\, x : int \xrightarrow{\{W_\xi\}} int, \{\} \qquad \dfrac{}{\{y : int\} \vdash y : int, \{\}}}{\{y : int\} \vdash (\lambda x.\, \xi!x.\, x)\, y : int, \{W_\xi\}}$$
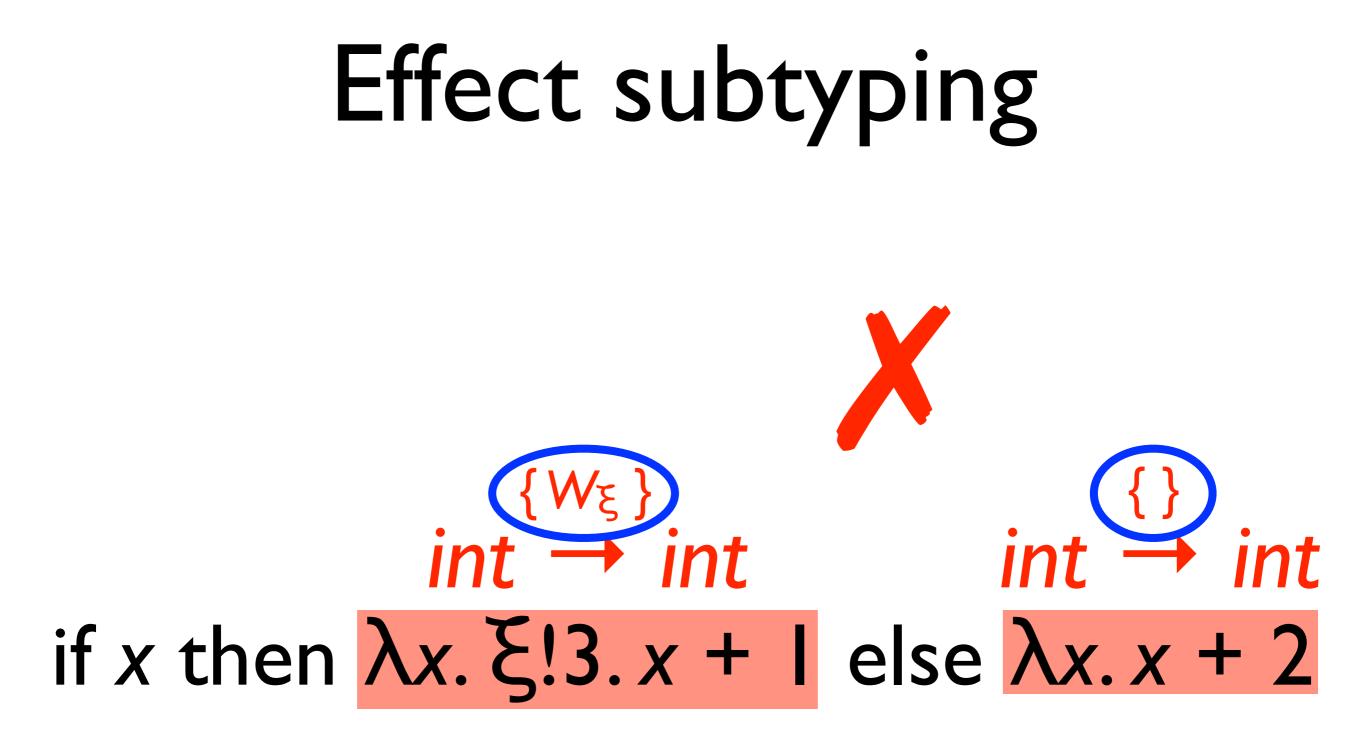
Effect subtyping

We would probably want more expressive control structure in a real programming language.

For example, we could add *if-then-else*:

$e ::= x \mid \lambda x.e \mid e_1\ e_2 \mid \xi?x.e \mid \xi!e_1.e_2 \mid$ if e_1 then e_2 else e_3

Effect subtyping

$$\frac{\Gamma \vdash e_1 : int, F \quad \Gamma \vdash e_2 : t, F' \quad \Gamma \vdash e_3 : t, F''}{\Gamma \vdash \texttt{if } e_1 \texttt{ then } e_2 \texttt{ else } e_3 : t, F \cup F' \cup F''} \quad (\text{Cond})$$
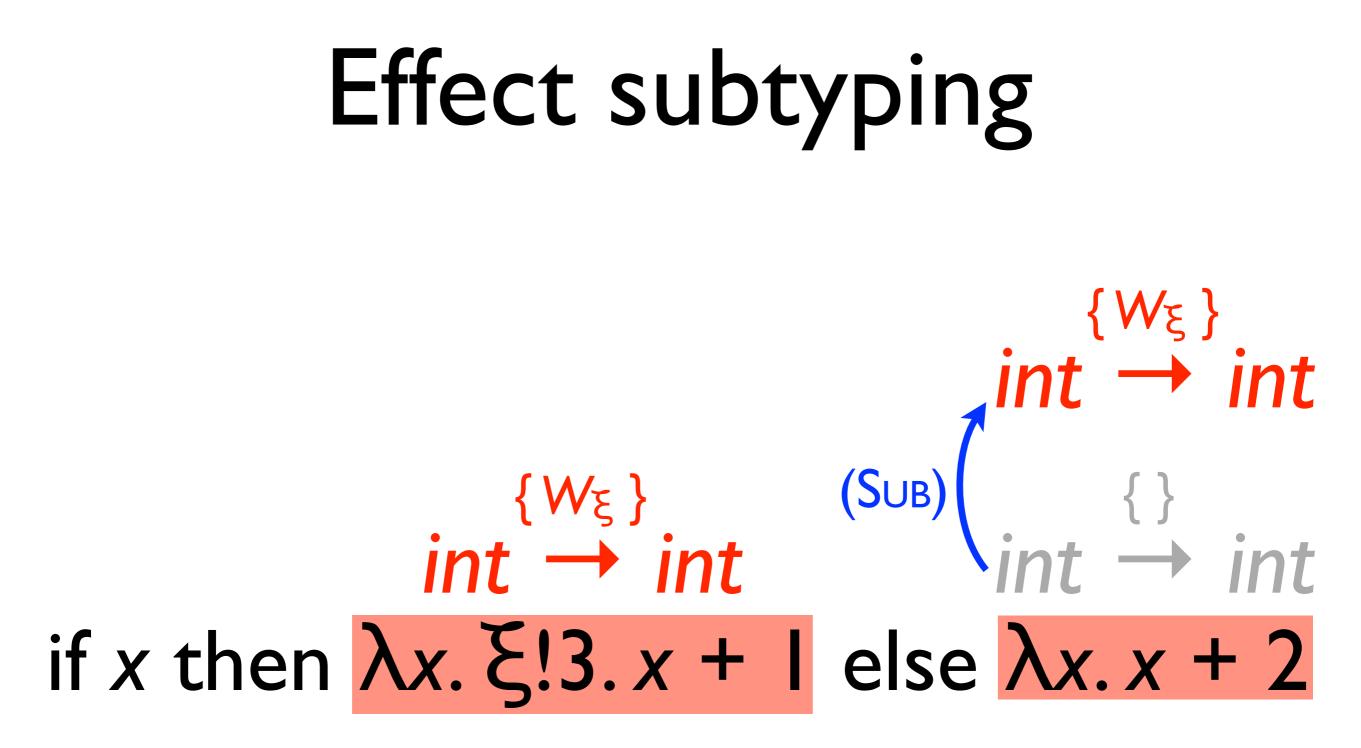
Effect subtyping

However, there are some valid uses of *if-then-else* which this rule cannot handle by itself.

Effect subtyping

$$\{ W_\xi \}$$
$$int \longrightarrow int$$

$$\{ \}$$
$$int \longrightarrow int$$

if x then $\lambda x.\ \xi!3.\ x + 1$ else $\lambda x.\ x + 2$

$$\frac{\Gamma \vdash e_1 : int, F \quad \Gamma \vdash e_2 : t, F' \quad \Gamma \vdash e_3 : t, F''}{\Gamma \vdash \texttt{if } e_1 \texttt{ then } e_2 \texttt{ else } e_3 : t, F \cup F' \cup F''} \quad (\textsc{Cond})$$

Effect subtyping

if *x* then $\lambda x.\ \xi!3.\ x + 1$ else $\lambda x.\ x + 2$

$int \xrightarrow{\{W_\xi\}} int \qquad int \xrightarrow{\{\}} int$

Effect subtyping

We can solve this problem by adding a new rule to handle *subtyping*.

Effect subtyping

$$\frac{\Gamma \vdash e : t \xrightarrow{F'} t', F \quad F' \subseteq F''}{\Gamma \vdash e : t \xrightarrow{F''} t', F} \quad (\textsc{Sub})$$

Effect subtyping

$$\{W_\xi\}$$
$$int \longrightarrow int$$

$$(S_{UB})$$

$$\{\}$$
$$int \longrightarrow int$$

$$\{W_\xi\}$$
$$int \longrightarrow int$$

if x then $\lambda x.\,\xi!3.\,x + 1$ else $\lambda x.\,x + 2$

Effect subtyping

$$int \xrightarrow{\{W_\xi\}} int \quad \checkmark$$

$$int \xrightarrow{\{W_\xi\}} int \qquad int \xrightarrow{\{\}} int$$

if x then $\lambda x.\, \xi!3.\, x + 1$ else $\lambda x.\, x + 2$

Optimisation

The information discovered by the effect system is useful when deciding whether particular transformations are safe.

An expression with no immediate side-effects is *referentially transparent*: it can safely be replaced with another expression (with the same value and type) with no change to the semantics of the program.

For example, referentially transparent expressions may safely be removed if LVA says they are dead.

Safety

$$(\{\} \vdash e : t, F) \Rightarrow$$

$$(v \in [\![t]\!] \wedge f \subseteq F \text{ where } (v, f) = [\![e]\!])$$

Extra structure

In this analysis we are using *sets* of effects.

As a result, we aren't collecting any information about how many times each effect may occur, or the order in which they may happen.

$$\xi ?x.\ \zeta !x.\ x \qquad\qquad F = \{\ R_\xi, W_\zeta\ \}$$

$$\zeta !y.\ \xi ?x.\ x \qquad\qquad F = \{\ R_\xi, W_\zeta\ \}$$

$$\zeta !y.\ \xi ?x.\ \zeta !x.\ x \qquad F = \{\ R_\xi, W_\zeta\ \}$$

Extra structure

If we use a different representation of effects, and use different operations on them, we can keep track of more information.

One option is to use *sequences* of effects and use an append operation when combining them.

Extra structure

$$\frac{\Gamma[x : int] \vdash e : t, F}{\Gamma \vdash \xi?x.e : t, \langle R_\xi \rangle \, @ \, F} \quad (\text{Read})$$

$$\frac{\Gamma \vdash e_1 : int, F \quad \Gamma \vdash e_2 : t, F'}{\Gamma \vdash \xi!e_1.e_2 : t, F \, @ \, \langle W_\xi \rangle \, @ \, F'} \quad (\text{Write})$$

Extra structure

In the new system, these expressions
all have different effects:

$$\xi?x.\ \zeta!x.\ x \qquad F = \langle\ R_\xi;\ W_\zeta\ \rangle$$

$$\zeta!y.\ \xi?x.\ x \qquad F = \langle\ W_\zeta;\ R_\xi\ \rangle$$

$$\zeta!y.\ \xi?x.\ \zeta!x.\ x \qquad F = \langle\ W_\zeta;\ R_\xi;\ W_\zeta\ \rangle$$

Extra structure

Whether we use sequences instead of sets depends upon whether we care about the order and number of effects. In the channel example, we probably don't.

But if we were tracking file accesses, it would be important to ensure that no further read or write effects occurred after a file had been closed.

And if we were tracking memory allocation, we would want to ensure that no block of memory got deallocated twice.

Summary

- Effect systems are a form of inference-based analysis

- Side-effects occur when expressions are evaluated

- Function types must be annotated to account for latent effects

- A type system can be modified to produce judgements about both types and effects

- Subtyping may be required to handle annotated types

- Different effect structures may give more information