# Lecture 12
# Inference-based analysis

# Motivation

In this part of the course we're examining several methods of higher-level program analysis.

We have so far seen *abstract interpretation* and *constraint-based analysis*, two general frameworks for formally specifying (and performing) analyses of programs.

Another alternative framework is *inference-based analysis*.

# Inference-based analysis

Inference systems consist of *sets of rules* for determining *program properties*.

Typically such a property of an entire program depends recursively upon the properties of the program's subexpressions; inference systems can directly express this relationship, and show how to recursively compute the property.

# Inference-based analysis

An inference system specifies judgements:

$$\Gamma \vdash e : \phi$$

- e is an expression (e.g. a complete program)
- Γ is a set of assumptions about free variables of e
- ϕ is a program property

# Type systems

Consider the ML type system, for example.

This particular inference system specifies judgements about a *well-typedness* property:

$$\Gamma \vdash e : t$$

means "under the assumptions in Γ, the expression e has type *t*".

# Type systems

We will avoid the more complicated ML typing issues (see Types course for details) and just consider the expressions in the lambda calculus:

$$e ::= x \mid \lambda x.\, e \mid e_1\ e_2$$

Our program properties are types *t*:

$$t ::= \alpha \mid int \mid t_1 \rightarrow t_2$$

# Type systems

$\Gamma$ is a set of *type assumptions* of the form

$$\{ x_1 : t_1, \ldots, x_n : t_n \}$$

where each identifier $x_i$ is assumed to have type $t_i$.

We write

$$\Gamma[x : t]$$

to mean $\Gamma$ with the additional assumption that $x$ has type $t$ (overriding any other assumption about $x$).

# Type systems

In all inference systems, we use a set of *rules* to inductively define which judgements are valid.

In a type system, these are the *typing rules*.

# Type systems

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

$$\frac{\Gamma[x : t] \vdash e : t'}{\Gamma \vdash \lambda x.e : t \rightarrow t'} \quad (\textsc{Lam})$$

$$\frac{\Gamma \vdash e_1 : t \rightarrow t' \quad \Gamma \vdash e_2 : t}{\Gamma \vdash e_1 e_2 : t'} \quad (\textsc{App})$$

# Type systems

$\Gamma = \{\, 2 : int, add : int \to int \to int, multiply : int \to int \to int \,\}$

$e = \lambda x.\ \lambda y.\ add\ (multiply\ 2\ x)\ y$

$t = int \to int \to int$

$$\dfrac{\dfrac{\dfrac{}{\Gamma[x:int][y:int] \vdash add : int \to int \to int} \quad \dfrac{\vdots}{\Gamma[x:int][y:int] \vdash multiply\ 2\ x : int}}{\Gamma[x:int][y:int] \vdash add\ (multiply\ 2\ x) : int \to int} \quad \dfrac{}{\Gamma[x:int][y:int] \vdash y : int}}{\dfrac{\Gamma[x:int][y:int] \vdash add\ (multiply\ 2\ x)\ y : int}{\dfrac{\Gamma[x:int] \vdash \lambda y.\ add\ (multiply\ 2\ x)\ y : int \to int}{\Gamma \vdash \lambda x.\ \lambda y.\ add\ (multiply\ 2\ x)\ y : int \to int \to int}}}$$

# Optimisation

In the absence of a compile-time type checker, all values must be tagged with their types and run-time checks must be performed to ensure types match appropriately.

If a type system has shown that the program is well-typed, execution can proceed safely without these tags and checks; if necessary, the final result of evaluation can be tagged with its inferred type.

Hence the final result of evaluation is identical, but less run-time computation is required to produce it.

# Safety

The safety condition for this inference system is

$$(\{\} \vdash e : t) \Rightarrow (\llbracket e \rrbracket \in \llbracket t \rrbracket)$$

where $\llbracket e \rrbracket$ and $\llbracket t \rrbracket$ are the *denotations* of e and *t* respectively: $\llbracket e \rrbracket$ is the value obtained by evaluating e, and $\llbracket t \rrbracket$ is the set of all values of type *t*.

This condition asserts that the run-time behaviour of the program will agree with the type system's prediction.

# Odds and evens

Type-checking is just one application of inference-based program analysis.

The properties do not have to be types; in particular, they can carry more (or completely different!) information than traditional types do.

We'll consider a more program-analysis–related example: detecting odd and even numbers.

# Odds and evens

This time, the program property $\phi$ has the form

$$\phi ::= \mathit{odd} \mid \mathit{even} \mid \phi_1 \rightarrow \phi_2$$

# Odds and evens

$$\frac{}{\Gamma[x : \phi] \vdash x : \phi} \quad (\textsc{Var})$$

$$\frac{\Gamma[x : \phi] \vdash e : \phi'}{\Gamma \vdash \lambda x.e : \phi \to \phi'} \quad (\textsc{Lam})$$

$$\frac{\Gamma \vdash e_1 : \phi \to \phi' \quad \Gamma \vdash e_2 : \phi}{\Gamma \vdash e_1 e_2 : \phi'} \quad (\textsc{App})$$

# Odds and evens

$\Gamma = \{\ 2 : even,\ add : even \to even \to even,$
$\qquad\qquad multiply : even \to odd \to even\ \}$

$e = \lambda x.\ \lambda y.\ add\ (multiply\ 2\ x)\ y$

$\phi = odd \to even \to even$

$$
\dfrac{\dfrac{\dfrac{\dfrac{}{\Gamma[x : odd][y : even] \vdash add : even \to even \to even} \quad \dfrac{\vdots}{\Gamma[x : odd][y : even] \vdash multiply\ 2\ x : even}}{\Gamma[x : odd][y : even] \vdash add\ (multiply\ 2\ x) : even \to even} \quad \dfrac{}{\Gamma[x : odd][y : even] \vdash y : even}}{\dfrac{\Gamma[x : odd][y : even] \vdash add\ (multiply\ 2\ x)\ y : even}{\Gamma[x : odd] \vdash \lambda y.\ add\ (multiply\ 2\ x)\ y : even \to even}}}{\Gamma \vdash \lambda x.\ \lambda y.\ add\ (multiply\ 2\ x)\ y : odd \to even \to even}
$$

# Safety

The safety condition for this inference system is

$$(\{\} \vdash e : \phi) \Rightarrow ([\![e]\!] \in [\![\phi]\!])$$

where $[\![\phi]\!]$ is the denotation of $\phi$:

$$[\![\mathit{odd}]\!] = \{ z \in \mathbb{Z} \mid z \text{ is odd} \},$$

$$[\![\mathit{even}]\!] = \{ z \in \mathbb{Z} \mid z \text{ is even} \},$$

$$[\![\phi_1 \rightarrow \phi_2]\!] = [\![\phi_1]\!] \rightarrow [\![\phi_2]\!]$$

# Richer properties

Note that if we want to show a judgement like

$$\Gamma \vdash \lambda x.\, \lambda y.\, add\ (multiply\ 2\ x)\ (multiply\ 3\ y) : even \rightarrow even \rightarrow even$$

we need more than one assumption about *multiply*:

$\Gamma$ = { ..., *multiply* : *even* → *even* → *even*,
*multiply* : *odd* → *even* → *even*, ... }

# Richer properties

This might be undesirable, and one alternative is to enrich our properties instead; in this case we could allow *conjunction* inside properties, so that our single assumption about *multiply* looks like:

$$
\begin{aligned}
\mathit{multiply} : \; & \mathit{even} \to \mathit{even} \to \mathit{even} \;\wedge \\
& \mathit{even} \to \mathit{odd} \to \mathit{even} \;\wedge \\
& \mathit{odd} \to \mathit{even} \to \mathit{even} \;\wedge \\
& \mathit{odd} \to \mathit{odd} \to \mathit{odd}
\end{aligned}
$$

We would need to modify the inference system to handle these richer properties.

# Summary

- Inference-based analysis is another useful framework
- Inference rules are used to produce judgements about programs and their properties
- Type systems are the best-known example
- Richer properties give more detailed information
- An inference system used for analysis has an associated safety condition