1 Semantics

1.1 Grammar and definitions

Todo: findable typing

FindPair queries

$$\begin{aligned} P \rightarrow\ & Rel(R) \\ & \mid RevRel(R) \\ & \mid Chain(P, P) \\ & \mid And(P, P) \\ & \mid AndRight(P, S) \\ & \mid AndLeft(P, S) \\ & \mid Or(P, P) \\ & \mid Distinct(P) \\ & \mid Id_A \\ & \mid Exactly(n, P) \\ & \mid Upto(n, P) \\ & \mid FixedPoint(P) \end{aligned} \tag{1}$$

and FindSingle queries

$$S \rightarrow Find(F) \mid From(S, P) \mid Narrow(S, F)$$

Object Types

$$\tau \rightarrow A \mid B \mid C \mid ..$$

Relations

$$R \rightarrow r_1 \mid r_2 \mid ..$$

Findables

$$F_A \rightarrow f_1 \mid f_2 \mid ...$$

are defined as partial functions

$$f \colon A \rightharpoonup \{True, False\}$$

For some given object type A

A schema Σ is made up of three partial functions:

$$\Sigma_{rel} \colon R \rightharpoonup \tau \times \tau$$

$$\Sigma_{findable} \colon R \rightharpoonup \tau$$

$$\Sigma_{table} \colon \tau \rightharpoonup \{True, False\}$$

Though, when it is obvious from the context, I shall use simply use $\Sigma(x)$ to signify application of either function.

A view $v \in V_\Sigma$, for a given schema represents the immutable state of a database.

It represents a pair of partial functions. Firstly the relation lookup function

$$v \in V_\Sigma \Rightarrow v_{rel}(r) \in \wp(s) \qquad if \qquad \Sigma(A) \downarrow s$$

if Σ is defined at r. That is, if a relation r is in the schema, then $v(r)$ is a set of objects of object type $\Sigma(r)$. Here, and from this point onwards I am using $\wp(s)$ to represent the powerset of a set, and $f(x) \downarrow y$ to mean f is defined at x and $f(x) = y$

The next function of a view is the table lookup function, it looks up a findable and returns

$$v \in V_\Sigma \Rightarrow v_{table}(A) \in \wp(A) \qquad if \qquad \Sigma(A) \downarrow True$$

That is, $v(A)$ is a set of object of type A stored in the view, and A is a member of the schema Σ. Again I shall overload these two functions where it is clear from the context which is to be used.

1.2 typing

Typing rules take two forms. Firstly typing of pair queries:

$$\Sigma \vdash P\colon (A, B)$$

Which means "under the schema Σ, pair query P returns a subset of $A \times B$". The second is for single queries:

$$\Sigma \vdash SA$$

Which means "under the schema Σ single query returns a subset of A

2 Inference-Based Program Analysis

This is a general technique in which an inference system specifies judgements of the form

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

where ϕ is a program property and Γ is a set of assumptions about free variables of e. One standard example (covered in more detail in the CST Part II 'Types' course) is the ML type system. Although the properties are here types and thus are not directly typical of program optimisation (the associated optimisation consists of removing types of values, evaluating in a typeless manner, and attaching the inferred type to the computed typeless result; non-typable programs are rejected) it is worth considering this as an archetype. For current purposes ML expressions e can here be seen as the λ-calculus:

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

and (assuming α to range over type variables) types t of the syntax

$$t ::= \alpha \mid int \mid t \to t'.$$

Now let Γ be a set of assumptions of the form $\{x_1 : t_1, \ldots, x_n : t_n\}$ which assume types t_i for free variables x_i; and write $\Gamma[x : t]$ for Γ with any assumption about x removed and with $x : t$ additionally assumed. We then have inference rules:

$$(\text{VAR})\frac{}{\Gamma[x : t] \vdash x : t}$$

$$(\text{LAM})\frac{\Gamma[x : t] \vdash e : t'}{\Gamma \vdash \lambda x.e : t \to t'}$$

$$(\text{APP})\frac{\Gamma \vdash e_1 : t \to t' \qquad \Gamma \vdash e_2 : t}{\Gamma \vdash e_1 e_2 : t'}.$$

Safety: the type-safety of the ML inference system is clearly not part of this course, but its formulation clearly relates to that for other analyses. It is usually specified by the *soundness* condition:

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

where $\llbracket e \rrbracket$ represents the result of evaluating e (its denotation) and $\llbracket t \rrbracket$ represents the set of values which have type t. Note that (because of $\{\}$) the safety statement only applies to closed programs (those with no free variables) but its inductive proof in general requires one to consider programs with free variables.

The following gives a more program-analysis–related example; here properties have the form

$$\phi ::= \mathit{odd} \mid \mathit{even} \mid \phi \to \phi'.$$

We would then have rules:

$$(\text{VAR})\frac{}{\Gamma[x : \phi] \vdash x : \phi}$$

$$(\text{LAM})\frac{\Gamma[x : \phi] \vdash e : \phi'}{\Gamma \vdash \lambda x.e : \phi \to \phi'}$$

$$(\text{APP})\frac{\Gamma \vdash e_1 : \phi \to \phi' \qquad \Gamma \vdash e_2 : \phi}{\Gamma \vdash e_1 e_2 : \phi'}.$$

Under the assumptions

$$\Gamma = \{2 : \mathit{even}, \quad + : \mathit{even} \to \mathit{even} \to \mathit{even}, \quad \times : \mathit{even} \to \mathit{odd} \to \mathit{even}\}$$

we could then show

$$\Gamma \vdash \lambda x.\lambda y.2 \times x + y : \mathit{odd} \to \mathit{even} \to \mathit{even}.$$

but note that showing

$$\Gamma' \vdash \lambda x.\lambda y.2 \times x + 3 \times y : \mathit{even} \to \mathit{even} \to \mathit{even}.$$

would require Γ' to have *two* assumptions for $\times$ or a single assumption of a more elaborate property, involving conjunction, such as:

$$\begin{array}{ll} \times : & \mathit{even} \to \mathit{even} \to \mathit{even} \ \wedge \\ & \mathit{even} \to \mathit{odd} \to \mathit{even} \ \wedge \\ & \mathit{odd} \to \mathit{even} \to \mathit{even} \ \wedge \\ & \mathit{odd} \to \mathit{odd} \to \mathit{odd}. \end{array}$$

Exercise: Construct a system for *odd* and *even* which can show that

$$\Gamma \vdash (\lambda f.f(1) + f(2))(\lambda x.x) : \mathit{odd}$$

for some Γ.