1 Semantics

1.1 Grammar and definitions

Todo: findable typing

FindPair queries

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

where n denotes a natural number. FindSingle queries

$$S \to Find(F) \mid From(S, P) \mid AndS(S, S') \mid OrS(S, S')$$

Object Types

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

Relations

$$R \to r_1 \mid r_2 \mid ..$$

Findables

$$F_A \to f_1 \mid f_2 \mid ...$$

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

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

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

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

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:

$$\Sigma \vdash S\colon A$$

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

The rules of the first kind are as follows

$$(\text{Rel})\frac{\Sigma(r) \downarrow (A, B)}{\Sigma \vdash Rel(r)\colon (A, B)}$$

$$(\text{Rev})\frac{\Sigma(r) \downarrow (B, A)}{\Sigma \vdash Rel(r)\colon (A, B)}$$

$$(\text{Id})\frac{\Sigma(A) \downarrow True}{\Sigma \vdash Id_A\colon (A, A)}$$

$$(\text{Chain})\frac{\Sigma \vdash P\colon (A, B) \qquad \Sigma \vdash Q\colon (B, C)}{\Sigma \vdash Chain(P, Q)\colon (A, C)}$$

$$(\text{And})\frac{\Sigma \vdash P\colon (A, B) \qquad \Sigma \vdash Q\colon (A, B)}{\Sigma \vdash And(P, Q)\colon (A, B)}$$

$$(\text{Or})\frac{\Sigma \vdash P\colon (A, B) \qquad \Sigma \vdash Q\colon (A, B)}{\Sigma \vdash Or(P, Q)\colon (A, B)}$$

$$(\text{Distinct})\frac{\Sigma \vdash P\colon (A, B)}{\Sigma \vdash Distinct(P)\colon (A, B)}$$

$$(\text{AndLeft})\frac{\Sigma \vdash P\colon (A, B) \qquad \Sigma \vdash S\colon (A)}{\Sigma \vdash AndLeft(P, S)\colon (A, B)}$$

$$(\text{AndRight})\frac{\Sigma \vdash P\colon (A, B) \qquad \Sigma \vdash S\colon B}{\Sigma \vdash \Sigma \vdash AndRight(P, S)\colon (A, B)}$$

$$(\text{Exactly})\frac{\Sigma \vdash P\colon (A, A)}{\Sigma \vdash Exactly(n, P)\colon (A, A)}$$

$$(\text{Upto})\frac{\Sigma \vdash P\colon (A, A)}{\Sigma \vdash Upto(n, P)\colon (A, A)}$$

$$(\text{FixedPoint})\frac{\Sigma \vdash P\colon (A, A)}{\Sigma \vdash FixedPoint(P)\colon (A, A)}$$

The rules for types of Single queries are similar:

$$(\text{Find})\frac{\Sigma(f) \downarrow (A)}{\Sigma \vdash Find(f)\colon A}$$

$$(\text{From})\frac{\Sigma \vdash P\colon (A, B) \qquad \Sigma \vdash S\colon A}{\Sigma \vdash From(S, P)\colon B}$$

$$(\text{AndS})\frac{\Sigma \vdash S\colon A \qquad \Sigma \vdash S'\colon A}{\Sigma \vdash AndS(S, S')\colon A}$$

$$(\text{OrS})\frac{\Sigma \vdash S\colon A \qquad \Sigma \vdash S'\colon A}{\Sigma \vdash OrS(S, S')\colon A}$$

1.3 Operational Semantics

Now we shall define a set of rules for determining if a pair of objects is a valid result of a query. We're interested in forming a relation $a \lhd p$ to mean "a is a valid result of query Q". This is dependent on the current view $v : View_{\Sigma}$, and the type of the expression. Hence we define $(a, b) \lhd_{(A,B),v} P$ for pair queries P and $a \lhd_{A,v} S$ for single queries S.

$$(\text{Rel})\frac{(a,b) \in v(r)}{(a,b) \lhd_{(A,B),v} Rel(r)}$$

$$(\text{Rev})\frac{(b,a) \in v(r)}{(a,b) \lhd_{(A,B),v} RevRel(r)}$$

$$(\text{Id})\frac{a \in v(A)}{(a,a) \lhd_{(A,A),v} Id_A}$$

$$(\text{Distinct})\frac{(a,b) \lhd_{(A,B),v} P \qquad a \neq b}{(a,b) \lhd_{(A,B),v} Distinct(P)}$$

$$(\text{And})\frac{(a,b) \lhd_{(A,B),v} P \qquad ((a,b) \lhd_{(A,B),v} Q}{(a,b) \lhd_{(A,B),v} And(P,Q)}$$

$$(\text{Or1})\frac{(a,b) \lhd_{(A,B),v} P}{(a,b) \lhd_{(A,B),v} Or(P,Q)}$$

$$(\text{Or2})\frac{(a,b) \lhd_{(A,B),v} Q}{(a,b) \lhd_{(A,B),v} Or(P,Q)}$$

$$(\text{Chain})\frac{(a,b) \lhd_{(A,B),v} P \qquad (b,c) \lhd_{(B,C),v} Q}{(a,c) \lhd_{(A,C),v} Chain(P,Q)}$$

$$(\text{AndLeft})\frac{(a,b) \lhd_{(A,B),v} P \qquad a \lhd_{A,v} S}{(a,b) \lhd_{(A,B),v} AndLeft(P,S)}$$

$$(\text{AndRight})\frac{(a,b) \lhd_{(A,B),v} P \qquad b \lhd_{B,v} S}{(a,b) \lhd_{(A,B),v} AndRight(P,S)}$$

$$(\text{Exactly˙0})\frac{(a,b) \lhd_{(A,A),v} Id_A}{(a,b) \lhd_{(A,A),v} Exactly(0,P)}$$

$$(\text{Exactly˙n+1})\frac{(a,b) \lhd_{(A,A),v} P \qquad (b,c) \lhd_{(A,A),v} Exactly(n,P)}{(a,c) \lhd_{(A,A),v} Exactly(n+1,P)}$$

$$(\text{Upto˙0})\frac{(a,b) \lhd_{(A,A),v} Id_A}{(a,b) \lhd_{(A,A),v} Upto(0,P)}$$

$$(\text{Upto˙n})\frac{(a,b) \lhd_{(A,A),v} Upto(n,P)}{(a,b) \lhd_{(A,A),v} Upto(n+1,P)}$$

$$(\text{Upto˙n+1})\frac{(a,b) \lhd_{(A,A),v} P \qquad (b,c) \lhd_{(A,A),v} Upto(n,P)}{(a,c) \lhd_{(A,A),v} Upto(n+1,P)}$$

$$(\text{fix1})\frac{(a,b) \lhd_{(A,A),v} Id_A}{(a,b) \lhd_{(A,A),v} FixedPoint(P)}$$

$$(\text{fix2})\frac{(a,b) \lhd_{(A,A),v} P \qquad (b,c) \lhd_{(A,A),v} FixedPoint(P)}{(a,b) \lhd_{(A,A),v} FixedPoint(P)}$$

And the FindSingle rules

$$(\text{Find})\frac{a \in v(A) \qquad f(a) \downarrow True}{a \triangleleft_{A,v} Find(f)}$$

$$(\text{From})\frac{a \triangleleft_{A,v} S \qquad (a,b) \triangleleft_{)(A,B),v} P}{b \triangleleft_{A,v} From(S,P)}$$

$$(\text{AndS})\frac{a \triangleleft_{A,v} S \qquad a \triangleleft_{A,v} S'}{a \triangleleft_{A,v} And(S,S')}$$

$$(\text{OrS1})\frac{a \triangleleft_{A,v} S}{a \triangleleft_{A,v} Or(S,S')}$$

$$(\text{OrS1})\frac{a \triangleleft_{A,v} S'}{a \triangleleft_{A,v} Or(S,S')}$$

1.4 Denotational Semantics

The operational semantics clearly demonstrate membership of a query, but don't give a means to efficiently generate the results of query. To this end, we introduce denotations $[\![P]\!]$ and $[\![S]\!]$ such that

$$\Sigma \vdash P\colon (A,B) \Rightarrow [\![P]\!]\colon View_{\Sigma} \to \wp(A \times B)$$

and

$$\Sigma \vdash S\colon A \Rightarrow [\![S]\!]\colon View_{\Sigma} \to \wp(A)$$

Such denotations should be compositional and syntax directed, whilst still corresponding to the operational semantics.

$$[\![Rel(r)]\!](v) = v(r)$$

$$[\![RevRel(r)]\!](v) = swap(v(r))$$

$$[\![Id_A]\!](v) = dup(v(a))$$

$$[\![Chain(P,Q)]\!](v) = join([\![P]\!](v), [\![Q]\!](v))$$

$$[\![And(P,Q)]\!](v) = [\![P]\!](v) \cap [\![Q]\!](v)$$

$$[\![Or(P,Q)]\!](v) = [\![P]\!](v) \cup [\![Q]\!](v)$$

$$[\![AndLeft(P,S)]\!](v) = filterLeft([\![P]\!](v), [\![S]\!](v))$$

$$[\![AndRight(P,S)]\!](v) = filterRight([\![P]\!](v), [\![S]\!](v))$$

$$[\![Distinct(P)]\!](v) = distinct([\![P]\!](v))$$

$$[\![Exactly(n,P)]\!](v) = (\lambda pairs.join([\![P]\!](v), pairs))^n [\![Id_A]\!](v)$$

$$[\![Upto(n,P)]\!](v) = (\lambda pairs.join([\![P]\!](v), pairs) \cup pairs)^n [\![Id_A]\!](v)$$

$$[\![FixedPoint(P)]\!](v) = fix(\lambda pairs.join([\![P]\!](v), pairs) \cup pairs) \text{ in the domain } closure(A,v)$$

And similarly with single queries

$$[\![Find(f)]\!](v) = \{a \in v(A) \mid f(a) \downarrow True\} \text{ for } \Sigma(f) = A$$

$$[\![From(S,P)]\!](v) = \{b \mid (a,b) \in [\![P]\!](v) \wedge a \in [\![S]\!](v)\}$$

$$[\![AndS(S,S')]\!](v) = [\![S]\!](v) \cap [\![S']\!](v)$$

$$[\![OrS(S,S')]\!](v) = [\![S]\!](v) \cup [\![S']\!](v)$$

with the following definitions:

$$swap(s) = \{(b,a) \mid (a,b) \in s\}$$

$$dup(s) = \{(a,a) \mid a \in s\}$$

$$join(p,q) = \{(a,c) \mid \exists b.(a,b) \in p \wedge (b,c) \in q\}$$

$$distinct(s) = \{(a,b) \in s \mid a \neq b\}$$

$$filterLeft(p,s) = \{(a,b) \in p \mid a \in s\}$$

$$filterRight(p,s) = \{(a,b) \in p \mid b \in s\}$$

1.5 The domain closure(A, v)

Todo: Serious tidying

For the subsequent proofs it is necessary to define the scott domain $closure(A, v)$ for some object type A and $View_\Sigma$ v. This domain is the set of subsets x $[\![Id_A]\!](v) \subseteq x \subseteq A \times A$ with bottom element $\perp = [\![Id_A]\!](v)$ and partial order $x \sqsubseteq y \Leftrightarrow x \subseteq y$

Theorem: $closure(A, v)$ is a domain

Firstly, by definition, $\forall x.x \in closure(A, v) \Rightarrow x \supseteq [\![Id_A]\!](v)$, so $[\![Id_A]\!](v)$ is the bottom element.

Secondly for any chain $x_1 \subseteq x_2 \subseteq x_3 \subseteq ...$, $x_i \in closure(A, v)$, there exists a value $\bigsqcup_n x_n \in closure(A, v)$ such that $\forall i. \bigsqcup_n x_n \supseteq x_i$ and $\forall y.(\forall i.x_i \subseteq y) \Rightarrow y \supseteq \bigsqcup_n x_n$

proof:

Take $\bigsqcup_n x_n = \bigcup_n x_n$. This is in $closure(A, v)$, since both $\bigcup_n x_n \supseteq [\![Id_A]\!](v)$ due to $\forall i.x_i \supseteq [\![Id_A]\!](v)$ by definition and $\bigcup_n x_n \subseteq A \times A$, by

$$\forall a.(a \in \bigcup_n x_n \wedge \neg(a \in A \times A)) \Rightarrow (\exists i.a \in x_i \wedge \neg a \in A \times A) \Rightarrow (\exists i.\neg x_i \subseteq A \times A)$$

yielding a contradiction if $\bigcup_n x_n \subseteq A \times A$ does not hold.

We know $\forall i. \bigcup_n x_n \supseteq x_i$ by definition.

For any y such that $(\forall i.)y \supseteq x_i$ then

$$\forall a.(\exists i.a \in x_i) \Rightarrow a \in y$$

$$\forall a. \bigvee_n (a \in x_n) \Rightarrow a \in y$$

$$\forall a.(a \in \bigcup_n x_n) \Rightarrow a \in y$$

$$\bigcup_n x_n \subseteq yu$$

$\square$

1.6 Correspondence of operational and denotational semantics

In order to use the denotational semantics to construct an interpreter or compiler we need to prove they are equivalent to the operational semantics. Namely:

For any pair query P, schema Σ and $View_\Sigma$ v: $\Sigma \vdash P\colon (A, B) \Rightarrow (a, b \triangleleft_{(A,B),v} P \Leftrightarrow (a, b) \in [\![P]\!](v))$

And for any single query S, schema Σ and $View_\Sigma$ v: $\Sigma \vdash S\colon A \Rightarrow (a \triangleleft_{A,v} S \Leftrightarrow a \in [\![S]\!](v))$

In order to prove these two propositions, we define two induction hypotheses

$$\Phi(\Sigma, P, A, B) \Leftrightarrow (\Sigma \vdash P{:}\,(A,B) \Rightarrow \forall v \in View_\Sigma, (a,b) \in A \times B. \quad ((a,b) \lhd_{(A,B),v} P) \Leftrightarrow ((a,b) \in [\![P]\!](v)))$$

$$\Psi(\Sigma, S, A) \Leftrightarrow (\Sigma \vdash S{:}\,A \Rightarrow \forall v \in View_\Sigma, a \in A. \quad (a \lhd_{A,v} S) \Leftrightarrow (a \in [\![S]\!](v)))$$

Now we shall induct over the structures of P and S starting with the SingleQuery cases
case $S = AndS(S', S'')$

$$\begin{aligned} a \in [\![S]\!](v) &\Leftrightarrow a \in ([\![S']\!](v) \cap [\![S'']\!](v)) \\ &\Leftrightarrow (a \in [\![S']\!](v) \wedge a \in [\![S'']\!](v)) \\ &\Leftrightarrow (a \lhd_{A,v} S' \wedge a \lhd_{A,v} S'') \text{ by } \Psi(\Sigma, S', A),\ \Psi(\Sigma, S'', A) \\ &\Leftrightarrow (a \lhd_{A,v} AndS(S', S'') \text{ by inversion of (AndS)} \end{aligned} \tag{2}$$

case $S = OrS(S', S'')$

$$\begin{aligned} a \in [\![S]\!](v) &\Leftrightarrow a \in ([\![S']\!](v) \cup [\![S'']\!](v)) \\ &\Leftrightarrow (a \in [\![S']\!](v) \vee a \in [\![S'']\!](v)) \\ &\Leftrightarrow (a \lhd_{A,v} S' \vee a \lhd_{A,v} S'') \text{ by } \Psi(\Sigma, S', A),\ \Psi(\Sigma, S'', A) \\ &\Leftrightarrow (a \lhd_{A,v} OrS(S', S'') \text{ by inversion of (OrS)} \end{aligned} \tag{3}$$

case $S = From(S', P)$

$$\begin{aligned} b \in [\![S]\!](v) &\Leftrightarrow \exists a \in A. \quad (a \in [\![S']\!](v) \wedge (a,b) \in [\![[\![P]\!](v)]\!](v)) \\ &\Leftrightarrow \exists a \in A. \quad (a \lhd_{A,v} S' \wedge (a,b) \lhd_{(A,B),v} P) \text{ by } \Psi(\Sigma, S', A),\ \Phi(\Sigma, P, A, B) \\ &\Leftrightarrow b \lhd_{B,v} From(S', P) \qquad \text{by inversion of (OrS)} \end{aligned} \tag{4}$$

case $S = Find(f)$

$$\begin{aligned} a \in [\![S]\!](v) &\Leftrightarrow a \in v(A) \wedge f(a) \downarrow True \qquad \text{by inversion of the type rule (Find)} \\ &\Leftrightarrow a \lhd_{A,v} Find(f) \text{ by definition} \end{aligned} \tag{5}$$

Now, looking at the FindPair queries
case $P = Rel(r)$

$$\begin{aligned} (a,b) \in [\![P]\!](v) &\Leftrightarrow (a,b) \in v(r) \\ &\Leftrightarrow (a,b) \lhd_{(A,B),v} Rel(r) \text{ by definition} \end{aligned} \tag{6}$$

case $P = RevRel(r)$

$$\begin{aligned} (a,b) \in [\![P]\!](v) &\Leftrightarrow (b,a) \in v(r) \\ &\Leftrightarrow (a,b) \lhd_{(A,B),v} RevRel(r) \text{ by definition} \end{aligned} \tag{7}$$

case $P = Id_A$

$$\begin{aligned} (a,b) \in [\![P]\!](v) &\Leftrightarrow a \in v(A) \wedge a = b \\ &\Leftrightarrow (a,b) \lhd_{(A,B),v} Id_A \text{ by definition} \end{aligned} \tag{8}$$

case $P = Chain(P', Q)$

We have by inversion of the (Chain) type rule $\Sigma \vdash P\colon (A, C) \Leftrightarrow \exists B. \quad \Sigma \vdash P'\colon (A, B) \wedge \Sigma \vdash Q\colon (B, C)$

$$\begin{aligned}
(a, c) \in [\![P]\!](v) &\Leftrightarrow (a, c) \in join([\![P']\!](v), denoQ) \\
&\Leftrightarrow \exists b \in B. \quad (a, b) \in [\![P']\!](v) \wedge (b, c) \in [\![Q]\!](v) \\
&\Leftrightarrow \exists b \in B. \quad (a, b) \lhd_{(A,B),v} P' \wedge (b, c) \lhd_{(B,C),v} Q \quad \text{by } \Phi(\Sigma, P', A, B),\ \Phi(\Sigma, Q, B, C) \\
&\Leftrightarrow (a, c) \lhd_{(A,C),v} Chain(P', Q) \text{ by definition}
\end{aligned} \tag{9}$$

case $P = And(P', Q)$

$$\begin{aligned}
(a, b) \in [\![P]\!](v) &\Leftrightarrow (a, b) \in ([\![P']\!](v) \cap denoQ) \\
&\Leftrightarrow (a, b) \in [\![P']\!](v) \wedge (a, b) \in [\![Q]\!](v) \\
&\Leftrightarrow (a, b) \lhd_{(A,B),v} P' \wedge (a, b) \lhd_{(A,B),v} Q \quad \text{by } \Phi(\Sigma, P', A, B),\ \Phi(\Sigma, Q, A, B) \\
&\Leftrightarrow (a, b) \lhd_{(A,B),v} And(P', Q) \text{ by inversion of (And)}
\end{aligned} \tag{10}$$

case $P = Or(P', Q)$

$$\begin{aligned}
(a, b) \in [\![P]\!](v) &\Leftrightarrow (a, b) \in ([\![P']\!](v) \cup denoQ) \\
&\Leftrightarrow (a, b) \in [\![P']\!](v) \vee (a, b) \in [\![Q]\!](v) \\
&\Leftrightarrow (a, b) \lhd_{(A,B),v} P' \vee (a, b) \lhd_{(A,B),v} Q \quad \text{by } \Phi(\Sigma, P', A, B),\ \Phi(\Sigma, Q, A, B) \\
&\Leftrightarrow (a, b) \lhd_{(A,B),v} Or(P', Q) \text{ by inversion of (Or1), (Or2)}
\end{aligned} \tag{11}$$

case $P = AndLeft(P', S)$

$$\begin{aligned}
(a, b) \in [\![P]\!](v) &\Leftrightarrow (a, b) \in [\![P']\!](v) \wedge a \in [\![S]\!](v) \\
&\Leftrightarrow (a, b) \lhd_{(A,B),v} P' \wedge a \lhd_{A,v} S \quad \text{by } \Phi(\Sigma, P', A, B),\ \Psi(\Sigma, S, A) \\
&\Leftrightarrow (a, b) \lhd_{(A,B),v} AndLeft(P', S) \text{ by inversion of (AndLeft)}
\end{aligned} \tag{12}$$

case $P = AndRight(P', S)$

$$\begin{aligned}
(a, b) \in [\![P]\!](v) &\Leftrightarrow (a, b) \in [\![P']\!](v) \wedge b \in [\![S]\!](v) \\
&\Leftrightarrow (a, b) \lhd_{(A,B),v} P' \wedge b \lhd_{B,v} S \quad \text{by } \Phi(\Sigma, P', A, B),\ \Psi(\Sigma, S, B) \\
&\Leftrightarrow (a, b) \lhd_{(A,B),v} AndRight(P', S) \text{ by inversion of (AndRight)}
\end{aligned} \tag{13}$$

case $P = Distinct(P')$

$$\begin{aligned}
(a, b) \in [\![P]\!](v) &\Leftrightarrow (a, b) \in [\![P']\!](v) \wedge a \neq b \\
&\Leftrightarrow (a, b) \lhd_{(A,B),v} P' \wedge a \neq b \quad \text{by } \Phi(\Sigma, P', A, B) \\
&\Leftrightarrow (a, b) \lhd_{(A,B),v} Distinct(P') \text{ by inversion of (AndRight)}
\end{aligned} \tag{14}$$

case $P = Exactly(n, P')$

We have by inversion of the (Exactly) type rule $\Sigma \vdash P\colon (A, A) \wedge \Sigma \vdash P'\colon (A, A)$

$$\text{let } f = (\lambda pairs.join([\![P']\!](v), pairs)) \tag{15}$$

then

$$(a,b) \in [\![P]\!](v) \Leftrightarrow (a,b) \in f^n[\![Id_A]\!](v) \tag{16}$$

hence it suffices to prove

$$(a,b) \in f^n[\![Id_A]\!](v) \Leftrightarrow (a,b) \lhd_{(A,A),v} Exactly(n, P') \tag{17}$$

Case $Exactly(0, P')$:

$$f^0[\![Id_A]\!](v) = [\![Id_A]\!](v)$$

so by $\Phi(\Sigma, (a,b), A, A), Id_A)$

$$\begin{aligned}(a,b) \in f^0[\![Id_A]\!](v) &\Leftrightarrow (a,b) \in [\![Id_A]\!](v) \\ &\Leftrightarrow (a,b) \lhd_{(A,A),v} Id_A \\ &\Leftrightarrow (a,b) \lhd_{(A,A),v} Exactly(0, P')\end{aligned} \tag{18}$$

Case $Exactly(n+1, P')$, assuming $\Phi(\Sigma, Exactly(n, P'), A, A)$:

$$f^{n+1}[\![Id_A]\!](v) = f(f^n([\![Id_A]\!](v)))$$

so by $\Phi(\Sigma, Exactly(n, P'), A, A)$

$$\begin{aligned}(a,b) \in f^{n+1}[\![Id_A]\!](v) &\Leftrightarrow \exists a'.(a,a') \in [\![P']\!](v) \land (a',b) \in f^n([\![Id_A]\!](v)) \quad \text{by definition of } join \text{ and } f \\ &\Leftrightarrow (a,a') \lhd_{(A,A),v} P \land (a',b) \lhd_{(A,A),v} Exactly(n,p) \\ &\Leftrightarrow (a,b) \lhd_{(A,A),v} Exactly(n+1, P') \text{ by (Exactly n+1)}\end{aligned} \tag{19}$$

case $P = Upto(n, P')$
We have by inversion of the (Upto) type rule $\Sigma \vdash P{:}\,(A,A) \land \Sigma \vdash P'{:}\,(A,A)$

$$\text{let } f = (\lambda pairs.join([\![P']\!](v), pairs) \cup pairs) \tag{20}$$

then

$$[\![P]\!](v) = f^n[\![Id_A]\!](v) \text{We now case split on n} \tag{21}$$

Case $Upto(0, P')$

$$\begin{aligned}(a,b) \in [\![Upto(0,P')]\!](v) &\Leftrightarrow (a,b) \in f^0[\![Id_A]\!](v) \\ &\Leftrightarrow (a,b) \in [\![Id_A]\!](v) \\ &\Leftrightarrow (a,b) \lhd_{(A,A),v} Id_A \text{ by } \Phi(\Sigma, Id_A, A, A) \\ &\Leftrightarrow (a,b) \lhd_{(A,A),v} Upto(0, P') \text{ by (Upto0)}\end{aligned} \tag{22}$$

Case $Upto(n+1, P')$

$$
\begin{aligned}
(a,b) \in [\![Upto(m+1,P')]\!](v) &\Leftrightarrow (a,b) \in f^{m+1}[\![Id_A]\!](v) \\
&\Leftrightarrow (a,b) \in (join([\![P']\!](v), f^m[\![Id_A]\!](v)) \cup f^m[\![Id_A]\!](v)) \\
&\Leftrightarrow (a,b) \in join([\![P']\!](v), [\![Upto(m,P)]\!](v)) \vee (a,b) \in [\![Upto(m,P')]\!](v) \\
&\Leftrightarrow (\exists a'.(a,a') \in [\![P']\!](v) \wedge (a',b) \in [\![Upto(m,P')]\!](v)) \vee (a,b) \lhd_{(A,A),v} Upto(m,P') \\
&\Leftrightarrow (\exists a'.(a,a') \lhd_{(A,A),v} P' \wedge (a',b) \lhd_{(A,A),v} Upto(m,P')) \vee (a,b) \lhd_{(A,A),v} Upto(m,P') \\
&\Leftrightarrow (a,b) \lhd_{(A,A),v} Upto(m+1,P') \text{ by (Upto n+1), (Upto n)}
\end{aligned} \tag{23}
$$

case $P = FixedPoint(P')$

We have by inversion of the (FixedPoint) type rule $\Sigma \vdash P\colon (A,A) \wedge \Sigma \vdash P'\colon (A,A)$

$$\text{let } f = (\lambda pairs.join([\![P']\!](v), pairs) \cup pairs) \tag{24}$$

then

$$[\![P]\!](v) = fix(f^n) \text{ In the domain } closure(A,v)$$

Lemma: f is continuous in the domain $closure(A,v)$

Firstly, f is monotonous

Let $x \subseteq y$

$$
\begin{aligned}
(a,b) \in f(x) &\Rightarrow (a,b) \in x \vee (\exists a'.(a,a') \in [\![P']\!](v) \wedge (a',b) \in x) \\
&\Rightarrow (a,b) \in y \vee (\exists a'.(a,a') \in [\![P']\!](v) \wedge (a',b) \in y) \\
&\Rightarrow (a,b) \in f(y)
\end{aligned} \tag{25}
$$

Hence $f(x) \subseteq f(y)$

Consider a chain $x1 \subseteq x2 \subseteq ...$ in $closure(A,v)$ Since $closure(A,v)$ is a domain, the lub, $\bigcup_n x_n$ is also in $closure(A,v)$

$$x_n \subseteq \bigcup_n x_n$$

so

$$
\begin{aligned}
&\forall m.f(x_n) \subseteq f(\bigcup_n (x_n)) \\
&\therefore \bigcup_n f(x_n) \subseteq f(\bigcup_n (x_n))
\end{aligned} \tag{26}
$$

Going the other way,

$$(a,b) \in f(\bigcup_n (x_n)) \Rightarrow ((\exists n.(a,b) \in x_n) \vee (\exists m,a'.(a,a') \in [\![P']\!](v) \wedge (a',b) \in x_m) \tag{27}$$

let $n' = max(n,m) so x_n \subseteq x_{n'} \wedge x_m \subseteq x_{n'}$

$$\exists n'.((a,b) \in x_{n'}) \vee (\exists a'.(a,a') \in [\![P']\!](v) \wedge (a',b) \in x_{n'}) \tag{28}$$

$$\therefore \exists n'.(a,b) \in f(x_{n'})$$

$$\therefore \exists n'.f(\bigcup_n x_n) \subseteq f(x_{n'}$$

$$\therefore f(\bigcup_n x_n) \subseteq \bigcup_n f(x_{n'})$$

So f is Scott-continuous.

Now, by Tarski's fixed point theorem

$$[\![FixedPoint(P')]\!](v) = fix(f) = \bigsqcup_n f^n(\bot)$$

Lemma: $(a,b) \lhd_{(A,A),v} FixedPoint(P') \Leftrightarrow \exists n.(a,b) \in f^n(\bot)$

By inversion of the operational rules (FixedPoint0), (FixedPoint n) and $(a,b) \lhd_{(A,A),v} FixedPoint(P')$, we get two cases.

Case $(a,b) \lhd_{A,A,v} Id_A$:

by $\Phi(\Sigma, Id_A, A, A), (a,b) \in [\![Id_A]\!](v) = \bot$

so $n = 0$

Case $(a,b) \lhd_{(A,A),v} P' \wedge (b,c) \lhd_{(A,A),v} FixedPoint(P')$

(hence $(a,c) \lhd_{(A,A),v} FixedPoint(P')$)

by $\Phi(\Sigma, P', A, A), and (b,c) \lhd_{(A,A),v} FixedPoint(P')$

$$(a,b) \in [\![P']\!](v) \wedge \exists n.(b,c) \in f^n(\bot)$$

Instantiating with $m = n$ gives

$$(a,b) \in [\![P']\!](v) \wedge (b,c) \in f^m(\bot)$$

so $(a,c) \in f(f^m(\bot)) = f^{m+1}(\bot)$

Hence $(a,b) \lhd_{(A,A),v} FixedPoint(P') \Rightarrow \exists n.(a,b) \in f^n(\bot)$

To go the other way, we need to prove $(a,b) \in \bigsqcup_n f^n(\bot) \Rightarrow (a,b) \lhd_{(A,A),v} FixedPoint(P')$

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:

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

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

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

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:

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

where $[\![e]\!]$ represents the result of evaluating e (its denotation) and $[\![t]\!]$ 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

$$\phi ::= odd \mid even \mid \phi \to \phi'.$$

We would then have rules:

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

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

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

Under the assumptions

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

we could then show

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

but note that showing

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

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

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

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.