Optimising Compilers 2004 - Paper 9 Question 3 (AM)

[Syllabus: "constraint based analysis (control flow analysis for λ-calculus)".]

(a) We label the nodes of the syntax tree of the program uniquely with their *occurrences* in the tree (formally sequences of integers representing the route from the root to the given node, but convenient integers are usually better). Now associate a *flow variable* α_i with each program point. In principle we wish to associate, with each flow variable α_i associated with expression e^i, an (overestimate of) the flow values which it yields during evaluation. Flow values are here most sensibly $\{o_1, \ldots, o_n, \iota\}$ where $o_1, \ldots, o_n$ are the n object definitions and ι represents an arbitrary integer value. [It is also OK to have one flow value for each integer constant, or even no values for integers (in which case the constraint for integer constants below will merely be $\alpha_i \supseteq \{\}$, i.e. always *true*.]

(b) We get constraints on the α_i determined by the program structure (the following constraints are in addition to the ones recursively generated by the subterms e, e_1, e_2 and e_3):

- for a term x^i we get the constraint $\alpha_i \supseteq \alpha_j$ where x^j is the associated binding (via $\texttt{let}\ x^j = \cdots$ or $f(x^j) = \cdots$);
- for a term c^i we get the constraint $\alpha_i \supseteq \{c^i\}$;
- for a term $(e_1^j.e_2^k)^i$ we get the constraint $\alpha_i \supseteq \{\iota\}$;
- for a term $(f(e_1^{i_1}, \ldots e_k^{i_k}))^{i_0}$ (and where the definition of f has arguments flow variables $\beta_1, \ldots, \beta_k$ and result flow variable β_0) get constraints $\alpha_{i_1} \supseteq \beta_1, \ldots, \alpha_{i_k} \supseteq \beta_k$, $\beta_0 \supseteq \alpha_{i_0}$
- for a term $(\texttt{let}\ x^l = e_1^j\ \texttt{in}\ e_2^k)^i$ we get the constraints $\alpha_i \supseteq \alpha_k$ and $\alpha_l \supseteq \alpha_j$;
- for a term $(\texttt{if}\ e_1^j\ \texttt{then}\ e_2^k\ \texttt{else}\ e_3^l)^i$ we get the constraints $\alpha_i \supseteq \alpha_k$ and $\alpha_i \supseteq \alpha_l$.

If e is the whole body of function $f(...) = e$ then the flow variable for the function result of the function is the same as that of e.

(c) Mumble that this is the same as using lambda expressions as in the notes (and note that we add fn values to flow values). We generate a compound constraint

$$((\alpha_{i_1}, \ldots, \alpha_{i_k}) \mapsto \alpha_{i_0}) \supseteq \alpha_j$$

(interpreted as above or as per notes) where e_0 is labelled with j.
[Remark to supervisors: this change effectively moves the constraint solving algorithm from ordinary transitive closure to dynamic transitive closure, but students are not expected to know this.]

(d) This is a cruel lie. It is quite acceptable as suggested above for flow variables to take values from the set $\{o_1, \ldots, o_n, \iota\}$ where ι represents *any* integer.

(e) We can fold adjacent operations on `x` into a single one (here a no-op) optimising

```
{ x.field++; print(y.field); x.field--; }
```

to

```
{ print(y.field); }
```

only if we know for certain that x and y cannot alias.