Type and Effect Systems Lecture Notes

DANIEL SAINATI

1 INTRO

We all should have a decent intuition for what a side effect is: it's anything your program might do to the environment or world that is not captured in the type of the value it returns. Some examples of effects in real world languages include:

- I/O or file handling
- Reading or writing from memory
- Throwing an exception
- Nondeterminism
- Time passing
- Non-termination

2 SLTC WITH EFFECTS

To see how we can model side effects using types, we're going to add a new feature to the simply typed lambda calculus: `tick`, which is going to represent the program using an amount of time in the real world.

We extend the basic SLTC with some new syntax to support ticking, along with booleans and if statements to make things a bit more interesting:

$$e ::= x \mid \texttt{true} \mid \texttt{false} \mid \texttt{if } e \texttt{ then } e \texttt{ else } e \mid \lambda x : \tau . e \mid ee \mid \texttt{tick} \mid e; e$$

We also need some types:

$$\tau ::= \texttt{bool} \mid \texttt{unit} \mid \tau \to \tau$$

Here we are going to use `unit` as the type of `tick`, as it does not produce a value that we care about. However, this is not the only thing `tick` does, in addition to producing a dummy unit value, it also consumes time. What if we'd like to statically know how much time a program will take to run (or specifically, how many times it will `tick`)? As it turns out, we won't be able say precisely how many `tick`s a program will use statically, but we can place an upper bound on this number with an effect system.

Define an effect type ϕ, which in this case will be a natural number. We'll also want a way to combine effects ($+$) and compare effects ($\leq$), which in this case will be the corresponding operations on the naturals. However, in a system designed to track other effects you can imagine more interesting versions of these operators (e.g. for exceptions you might have your effect type be a set of possible error kinds, $+$ be defined as union and $\leq$ be set inclusion). What's important is that we can define an *algebra* over our effects, although this should not be confused with "algebraic effects", which means something different.

Now we want to work our effects into the type system. What will this look like? Our typing judgment for the original SLTC looked like $\Gamma \vdash e : \tau$, which only gave us information about the type of the value produced by evaluating the expression. We also want information now about the effects that evaluation will have, so we can achieve this by annotating the judgment with effects:

$\Gamma \vdash e :^{\phi} \tau$.

Let's start with the actual effect-producing expression: `tick`.

$$\textbf{T-Tick}\ \frac{}{\Gamma \vdash \texttt{tick} :^{1} \texttt{unit}}$$

This tells us that the `tick` expression has type `unit`, and produces 1 tick effect when you run it. Pretty simple.

What about sequencing?

$$\textbf{T-Seq}\ \frac{\Gamma \vdash e_1 :^{\phi_1} \texttt{unit} \qquad \Gamma \vdash e_2 :^{\phi_2} \tau}{\Gamma \vdash e_1; e_2 :^{\phi_1+\phi_2} \tau}$$

This is also straightforward: sequencing two expressions adds together their effects, so e.g. `tick; tick` will have have effect 2 and type `unit`.

Some more simple rules:

$$\textbf{T-True}\ \frac{}{\Gamma \vdash \texttt{true} :^{0} \texttt{bool}} \quad \textbf{T-False}\ \frac{}{\Gamma \vdash \texttt{false} :^{0} \texttt{bool}} \quad \textbf{T-Var}\ \frac{x : \tau \in \Gamma}{\Gamma \vdash x :^{0} \tau}$$

Primitive values and variable uses have no effect, that is, they are "pure".

How about if statements? Well, unlike sequencing, which always evaluates both of its constituent expressions, a conditional will only choose one of its two arms to evaluate. Adding together the effects of these branches would be an unnecessarily conservative overestimate of the effects. Instead, we can use our $\leq$ operator to choose the greater of the two branches:

$$\textbf{T-If}\ \frac{\Gamma \vdash e_1 :^{\phi_1} \texttt{bool} \qquad \Gamma \vdash e_2 :^{\phi_2} \tau \qquad \Gamma \vdash e_3 :^{\phi_3} \tau}{\Gamma \vdash \texttt{if}\ e_1\ \texttt{then}\ e_2\ \texttt{else}\ e_3 :^{\phi_1+\mathbf{max}(\phi_2,\phi_3)} \tau}$$

Now we come to functions. How might we type check a lambda expression in this system?

$$\textbf{T-Lam-Wrong}\ \frac{\Gamma, x : \tau_1 \vdash e :^{\phi} \tau_2}{\Gamma \vdash \lambda x : \tau_1 . e :^{0} \tau_1 \to \tau_2}$$

The body of the function will use some effect when it is evaluated, but the function itself is not evaluated until it is applied (in a CBV or CBN system at least). So, as written here, we lose information about the effects of the body, and won't have any place to recover them when applying the function elsewhere. To handle this, we modify the arrow type to "store" the effects of the body:

$$\textbf{T-Lam}\ \frac{\Gamma, x : \tau_1 \vdash e :^{\phi} \tau_2}{\Gamma \vdash \lambda x : \tau_1 . e :^{0} \tau_1 \xrightarrow{\phi} \tau_2}$$

And now, application:

$$\textbf{T-App}\ \frac{\Gamma \vdash e_1 :^{\phi_1} \tau_1 \xrightarrow{\phi_3} \tau_2 \qquad \Gamma \vdash e_2 :^{\phi_2} \tau_1}{\Gamma \vdash e_1 e_2 :^{\phi_1+\phi_2+\phi_3} \tau_2}$$

This way the application rule takes into account the effects not only of evaluating the argument and the function, but also of evaluating the body of the function with the supplied argument.

3 EFFECT SOUNDNESS

What properties might we care to prove about this system? The obvious type soundness and normalization properties, sure, but we may also care to show that the system actually has the effect property we described earlier. That is, we'd like it to be the case that the effects described by the type system are actually an upper bound on the side effects that occur during evaluation. To be able to talk about this, we first need to be able to formally describe "the side effects that occur during evaluation". We achieve this by *instrumenting* the semantics for the SLTC with tracking of effects. We define our terminal values and environments:

$$v ::= \texttt{true} \mid \texttt{false} \mid () \mid \langle\!\langle \rho, \lambda x.e \rangle\!\rangle$$
$$\rho ::= \cdot \mid x \mapsto v, \rho$$

The $\langle\!\langle \rho, \lambda x.e \rangle\!\rangle$ denotes a closure capturing an environment ρ; this will be the result of evaluating a lambda expression. Now, the big step semantics:

$$\textbf{E-True}\ \frac{}{\rho \vdash \texttt{true} \Downarrow^{0} \texttt{true}} \qquad \textbf{E-False}\ \frac{}{\rho \vdash \texttt{false} \Downarrow^{0} \texttt{false}} \qquad \textbf{E-Var}\ \frac{x \mapsto v \in \rho}{\rho \vdash x \Downarrow^{0} v}$$

$$\textbf{E-Tick}\ \frac{}{\rho \vdash \texttt{tick} \Downarrow^{1} ()} \qquad \textbf{E-Seq}\ \frac{\rho \vdash e_1 \Downarrow^{\phi_1} () \qquad \rho \vdash e_2 \Downarrow^{\phi_2} v}{\Gamma \vdash e_1; e_2 \Downarrow^{\phi_1+\phi_2} v}$$

$$\textbf{E-If-True}\ \frac{\rho \vdash e_1 \Downarrow^{\phi_1} \texttt{true} \qquad \rho \vdash e_2 \Downarrow^{\phi_2} v}{\rho \vdash \texttt{if}\ e_1\ \texttt{then}\ e_2\ \texttt{else}\ e_3 \Downarrow^{\phi_1+\phi_2} v}$$

$$\textbf{E-If-False}\ \frac{\rho \vdash e_1 \Downarrow^{\phi_1} \texttt{false} \qquad \rho \vdash e_3 \Downarrow^{\phi_2} v}{\rho \vdash \texttt{if}\ e_1\ \texttt{then}\ e_2\ \texttt{else}\ e_3 \Downarrow^{\phi_1+\phi_2} v} \qquad \textbf{E-Lam}\ \frac{}{\rho \vdash \lambda x : \tau . e \Downarrow^{0} \langle\!\langle \rho, \lambda x.e \rangle\!\rangle}$$

$$\textbf{E-App}\ \frac{\rho \vdash e_1 :^{\phi_1} \langle\!\langle \rho', \lambda x.e_3 \rangle\!\rangle \qquad \rho \vdash e_2 :^{\phi_2} v' \qquad \rho', x \mapsto v' \vdash e_3 :^{\phi_3} v}{\rho \vdash e_1 e_2 :^{\phi_1+\phi_2+\phi_3} v}$$

Notice how the effect annotations on the semantics mirror those on the typing judgment.

How now to prove effect soundness? Since we are using a big-step semantics, we will prove soundness along with normalization together (big-step semantics cannot express non-terminating reduction sequences by nature, so proving that a reduction sequence exists for every well-typed term also entails proving that every well-typed term terminates). Thus, we will use a logical relation (which will look quite similar to the hereditary termination relation [2] we used for the regular SLTC).

$$\begin{aligned}
[\![\texttt{bool}]\!] &= \{\texttt{true}, \texttt{false}\} \\
[\![\texttt{unit}]\!] &= \{()\} \\
[\![\tau_1 \xrightarrow{\phi} \tau_2]\!] &= \{\langle\!\langle \rho, \lambda x.e \rangle\!\rangle \mid \forall v \in [\![\tau_1]\!], (\rho, x \mapsto v) \vdash e \Downarrow^{\phi'} v' \wedge \phi' \leq \phi \wedge v' \in [\![\tau_2]\!]\}
\end{aligned}$$

The only interesting case in this definition is the arrow type case, which is going to require that evaluating the body of the function uses at most the effects annotated on the arrow type. We define our notion of semantic well-typedness:

$$\begin{aligned}
\Gamma \vDash \rho &\triangleq x : \tau \in \Gamma \implies x \mapsto v \in \rho \wedge v \in [\![\tau]\!] \\
\Gamma \vDash e :^{\phi} \tau &\triangleq \forall \rho, \Gamma \vDash \rho \implies \exists v, \phi' \text{ such that } \rho \vdash e \Downarrow^{\phi'} v \wedge \phi' \leq \phi \wedge v \in [\![\tau]\!]
\end{aligned}$$

Notice that we don't attempt to promise that evaluation will have exactly the effects specified in the semantic typing judgment, as branching execution in the if statement prevents us from doing so. Instead we require that evaluation have at most the effects specified in the type.

Note also that the ρ here is performing a dual duty as both the execution environment for the big step semantics and also the closing substitution for the logical relation. This is typical for logical relations for big step semantics.

Now, let's prove the fundamental lemma:

Lemma 3.1 (Fundamental Lemma: Effect Soundness)**.** If $\Gamma \vdash e :^{\phi} \tau$, then $\Gamma \vDash e :^{\phi} \tau$.

Proof. As one might expect, we prove this by induction on the typing derivation of $\Gamma \vdash e :^{\phi} \tau$.

- Case **T-Var**:
  In this case $\Gamma \vdash x :^0 \tau$, and we would like to show $\Gamma \vDash x :^0 \tau$. Unfolding the definition, we can assume we have some ρ such that $\Gamma \vDash \rho$, and want to show that $\rho \vdash x \Downarrow^{\phi'} v \wedge \phi' \leq 0 \wedge v \in [\![\tau]\!]$ for some v and ϕ'. We can just choose ϕ' to be 0 and v to be $\rho(x)$, since our assumption that $\Gamma \vDash \rho$ tells us that $x \mapsto v \in \rho \wedge v \in [\![\tau]\!]$.
- Case **T-True**:
  In this case $\Gamma \vdash \texttt{true} :^0 \texttt{bool}$, and we would like to show $\Gamma \vDash \texttt{true} :^0 \texttt{bool}$. Unfolding the definition, this requires us to prove $\rho \vdash \texttt{true} \Downarrow^{\phi'} v \wedge \phi' \leq 0 \wedge v \in [\![\texttt{bool}]\!]$. The case is immediate with a choice of 0 for ϕ' and `true` for v.
- Case **T-False**:
  This case is identical to the `true` case, just with `false` chosen for v.
- Case **T-Tick**:
  In this case, $\Gamma \vdash \texttt{tick} :^1 \texttt{unit}$, and we would like to show $\Gamma \vDash \texttt{tick} :^1 \texttt{unit}$. This requires us to show $\rho \vdash \texttt{tick} \Downarrow^{\phi'} v \wedge \phi' \leq 1 \wedge v \in [\![\texttt{unit}]\!]$. This is immediate with a choice of 1 for ϕ' and () for v.
- Case **T-Seq**:
  In first of our inductive cases, we have $\Gamma \vdash e_1; e_2 :^{\phi_1+\phi_2} \tau$. Our IHs tell us that $\Gamma \vDash e_1 :^{\phi_1} \texttt{unit}$ and $\Gamma \vDash e_2 :^{\phi_2} \tau$, and we would like to show $\Gamma \vDash e_1; e_2 :^{\phi_1+\phi_2} \tau$. Unrolling definitions, we can assume we have a ρ such that $\Gamma \vDash \rho$, which allows us to use our IHs to derive that there is some ϕ_1' and v_1' such that $\rho \vdash e_1 \Downarrow^{\phi_1'} v_1' \wedge \phi_1' \leq \phi_1 \wedge v_1' \in [\![\texttt{unit}]\!]$, and some ϕ_2' and v_2' such that $\rho \vdash e_2 \Downarrow^{\phi_2'} v_2' \wedge \phi_2' \leq \phi_2 \wedge v_2' \in [\![\tau]\!]$.
  From the definition of our logical relation, we can see that $v_1' \in [\![\texttt{unit}]\!]$ tells us that v_1' must be ( ). Our goal in this case is to show that $\rho \vdash e_1; e_2 :^{\phi'} v \wedge \phi' \leq \phi_1 + \phi_2 \wedge v \in [\![\tau]\!]$. We choose $\phi' = \phi_1' + \phi_2'$ and $v = v_2'$, which gives us the derivation we want using the **E-Seq** rule, along with inclusion in the logical relation. We then just need to show that $\phi_1' + \phi_2' \leq \phi_1 + \phi_2$, which is trivial to show in the algebra of the naturals that we chose for these effects, but in general should also hold for arbitrary other effect algebras we might care about.
- Case **T-If**:
  In this case we have $\Gamma \vdash \texttt{if } e_1 \texttt{ then } e_2 \texttt{ else } e_3 :^{\phi_1+\mathbf{max}(\phi_2,\phi_3)} \tau$, and our IHs give us that $\Gamma \vDash e_1 :^{\phi_1}$ bool, $\Gamma \vDash e_2 :^{\phi_2} \tau$, and $\Gamma \vDash e_3 :^{\phi_3} \tau$. We would like to show that $\Gamma \vDash \texttt{if } e_1 \texttt{ then } e_2 \texttt{ else } e_3 :^{\phi_1+\mathbf{max}(\phi_2,\phi_3)} \tau$.
  Unrolling definitions, we have some ρ such that $\Gamma \vDash \rho$, which allows us to use our first IH to derive that (after some additional simplification), that $\rho \vdash e_1 \Downarrow^{\phi_1'} v_1' \wedge \phi_1' \leq \phi_1 \wedge v_1' \in [\![\texttt{bool}]\!]$ (and thus that v_1' is either `true` or `false`). We must consider both cases. Let's start with the case where v_1' is `true`.
  In this case, we can use another of our IHs to derive that $\rho \vdash e_2 \Downarrow^{\phi_2'} v_2' \wedge \phi_2' \leq \phi_2 \wedge v_2' \in [\![\tau]\!]$. Our goal is to show that $\rho \vdash \texttt{if } e_1 \texttt{ then } e_2 \texttt{ else } e_3 \Downarrow^{\phi'} v \wedge \phi' \leq \phi_1 + \mathbf{max}(\phi_2, \phi_3) \wedge v \in [\![\tau]\!]$. Choose ϕ' to be $\phi_1' + \phi_2'$, and v to be v_2'. The rest of the case follows directly from the usage of the **E-If-True** rule and some simple algebra. The case where v_1' is `false` is symmetric, using the **E-If-False** rule.
- Case **T-Lam**:
  In this case, we have $\Gamma \vdash \lambda x : \tau_1.e :^0 \tau_1 \xrightarrow{\phi} \tau_2$, and our IH gives us that $\Gamma, x : \tau_1 \vDash e :^{\phi} \tau_2$. We would like to show $\Gamma \vDash \lambda x : \tau_1.e :^0 \tau_1 \xrightarrow{\phi} \tau_2$.
  Unrolling our definitions, we assume we have a ρ such that $\Gamma \vDash \rho$, and we want to show that there is some ϕ' and v such that $\rho \vdash \lambda x : \tau_1.e \Downarrow^{\phi'} v \wedge \phi' \leq 0 \wedge v \in [\![\tau_1 \xrightarrow{\phi} \tau_2]\!]$. If we choose v to

be $\langle\!\langle \rho, \lambda x.e_3 \rangle\!\rangle$ and ϕ' to be 0, the first two components of our goal conjunction are simple, and we need only now prove that $\langle\!\langle \rho, \lambda x.e \rangle\!\rangle \in [\![\tau_1 \xrightarrow{\phi} \tau_2]\!]$. Further unrolling our definitions, we see that this is equivalent to proving that $\forall v \in [\![\tau_1]\!], (\rho, x \mapsto v) \vdash e \Downarrow^{\phi'} v' \wedge \phi' \leq \phi \wedge v' \in [\![\tau_2]\!]$. We can prove an easy technical lemma (omitted here) showing that given $\Gamma \vDash \rho$ and $v \in [\![\tau_1]\!]$, then $\Gamma, x : \tau \vDash \rho, x \mapsto v$. With this result, we can use our IH to derive that there is some v' and ϕ' such that $(\rho, x \mapsto v) \vdash e \Downarrow^{\phi'} v' \wedge \phi' \leq \phi \wedge v' \in [\![\tau_2]\!]$. This is, however, exactly what we need to show, and so we are done here.

- Case **T-App**:

  In this case we have $\Gamma \vdash e_1 e_2 :^{\phi_1+\phi_2+\phi_3} \tau_2$, and our IHs give us $\Gamma \vDash e_1 :^{\phi_1} \tau_1 \xrightarrow{\phi_3} \tau_2$ and $\Gamma \vDash e_2 :^{\phi_2} \tau_1$. We want to show $\Gamma \vDash e_1 e_2 :^{\phi_1+\phi_2+\phi_3} \tau_2$. We start by assuming $\Gamma \vDash \rho$, and want to show that there is some v and ϕ' such that $\rho \vdash e_1 e_2 \Downarrow^{\phi'} v \wedge \phi' \leq \phi_1 + \phi_2 + \phi_3 \wedge v \in [\![\tau_2]\!]$.
  We use our assumption that $\Gamma \vDash \rho$ and our IHs to derive that there is some v_1 and ϕ_1' such that $\rho \vdash e_1 \Downarrow^{\phi_1'} v_1 \wedge \phi_1' \leq \phi_1 \wedge v_1 \in [\![\tau_1 \xrightarrow{\phi_3} \tau_2]\!]$, and some v_2 and ϕ_2' such that $\rho \vdash e_2 \Downarrow^{\phi_2'} v_2 \wedge \phi_2' \leq \phi_2 \wedge v_2 \in [\![\tau_1]\!]$.
  Unfolding our logical relation definition in the first premise here tells us that v_1 is some $\langle\!\langle \rho', \lambda x.e' \rangle\!\rangle$ and that $\forall v \in [\![\tau_1]\!], (\rho, x \mapsto v) \vdash e \Downarrow^{\phi_3'} v' \wedge \phi_3' \leq \phi_3 \wedge v' \in [\![\tau_2]\!]$. We can use this, providing v_2 as our v here to conclude that $(\rho, x \mapsto v_2) \vdash e \Downarrow^{\phi_3'} v' \wedge \phi_3' \leq \phi_3 \wedge v' \in [\![\tau_2]\!]$ for some v' and ϕ_3'.
  Now, we choose our original v and ϕ' from our goal to be v' and $\phi_1' + \phi_2' + \phi_3'$, respectively, which along with our premises, some algebra and the **E-App** rule, gives us what we need to complete the case.

□

4 POLYMORPHIC EFFECTS

We've proven the correctness of effects for a simply typed language, but there's something very unsatisfying about this system. Namely, the types for functions are extremely restrictive. Consider a function we might wish to write in a system without effects that takes a function and applies it twice to an input:

$$\texttt{twice} \triangleq \lambda f : \texttt{bool} \to \texttt{bool}.\lambda x : \texttt{bool}.f(fx))$$

This is straightforward in a language without effects, but suppose we wanted to be able to write it in our effectful language. We would have to choose a specific effect to place on the type of f, and then our definition of `twice` would only accept input functions that use those effects. For example:

$$\texttt{twice2} \triangleq \lambda f : \texttt{bool} \xrightarrow{2} \texttt{bool}.\lambda x : \texttt{bool}.\texttt{tick}; f(fx))$$

defines a function `twice2` with type $\texttt{bool} \xrightarrow{2} \texttt{bool} \xrightarrow{0} \texttt{bool} \xrightarrow{5} \texttt{bool}$. If we wanted to apply a function that ticks 3 times twice, we'd have to define a different function (e.g., `twice3`) that accepts an f with effect 3. That's bad language design.

We can get around this by introducing effect polymorphism [3, 4]. There are many ways to do this, but one simple one is to add new syntax (like in System F). We'll borrow the System F syntax to highlight the relationship between parametric polymorphism and effect polymorphism.

$$e ::= \ldots \mid \Lambda\Phi.e \mid e[\phi]$$
$$\tau ::= \ldots \mid \forall\Phi.^{\phi}\tau$$

We use capital Φ to represent an effect variable, while lowercase ϕ is a specific effect. Note that just like the regular arrow type, the new forall type also has a latent effect on it, which makes sense when we consider the polymorphic abstraction to also suspend computation the same way as a normal abstraction.

We add new typing rules (again reminiscent of System F). We also extend our typing judgment with a Δ that tracks in-scope effect variables, so the judgment now looks like $\Gamma \mid \Delta \vdash e :^{\phi} \tau$:

$$\textbf{T-Eff-Lam}\ \frac{\Gamma \mid \Delta, \Phi \vdash e :^{\phi} \tau}{\Gamma \mid \Delta \vdash \Lambda\Phi.e :^{0} \forall\Phi.^{\phi}\tau}$$

$$\textbf{T-Eff-App}\ \frac{\Gamma \mid \Delta \vdash e :^{\phi_1} \forall\Phi.^{\phi_2}\tau \qquad \Phi \notin FV(\phi_1)}{\Gamma \mid \Delta \vdash e[\phi] :^{\phi_1 + \phi_2[\Phi \mapsto \phi]} \tau[\Phi \mapsto \phi]}$$

The proof of correctness for this system is more complicated than the simply typed one, but it draws on many of the same principles, along with some of the techniques used for the logical relation for System F.

5 EFFECT HANDLERS

It's also possible that we might wish to be able to interact with effects during evaluation of a program, rather than just observing them after the fact. One way to accomplish this is with *effect handlers* [1], which provide a way to "react" to effects as they happen during execution. These handlers often have certain algebraic properties, which leads to effects with handlers often being referred to as "algebraic effects". One can think of this as a generalization of `try-catch` syntax from common languages like JavaScript or Python. With that in mind, we can define a language extension for our calculus:

$$e ::= \ldots \mid \texttt{handle}\ e\ \texttt{with}\ h$$
$$h ::= \phi \Rightarrow e, h \mid \_ \Rightarrow e$$

So, for example, we might define a program:

$$\texttt{handle}\ fx\ \texttt{with}\ (1 \Rightarrow \texttt{true}, 4 \Rightarrow \texttt{false}, \_ \Rightarrow \texttt{tick}; \texttt{false})$$

which will return `true` if f ticks once during evaluation, `false` if it ticks 4 times, and otherwise will return `false` and tick once itself.

More practical systems will also reify effects as values that can be interacted with at runtime. One could also imagine a program that does different things depending on whether a function ticks an odd or even number of times:

$$\texttt{handle } fx \texttt{ with } \phi \Rightarrow \phi\%2 = 0$$

Additionally, in some effect systems (e.g. in one where the effects are exceptions), it might not make sense to wait for the completion of the handled expression to execute, as these examples implicitly do. In such cases handlers typically are able to suspend the execution of their handled expression, and also often come with *continuations*, expressions to evaluate upon completion of the effect handler.

REFERENCES

[1] Andrej Bauer and Matija Pretnar. 2015. Programming with algebraic effects and handlers. 84, 1 (2015), 108–123. doi:10.1016/j.jlamp.2014.02.001

[2] Robert Harper. 2025. How to (Re)Invent Tait's Method. https://www.cs.cmu.edu/~rwh/courses/atpl/pdfs/tait.pdf

[3] J. M. Lucassen and D. K. Gifford. 1988. Polymorphic effect systems. In *Proceedings of the 15th ACM SIGPLAN-SIGACT symposium on Principles of programming languages - POPL '88* (San Diego, California, United States). ACM Press, 47–57. doi:10.1145/73560.73564

[4] Lukas Rytz, Martin Odersky, and Philipp Haller. 2012. Lightweight Polymorphic Effects. In *ECOOP 2012 – Object-Oriented Programming* (Berlin, Heidelberg, 2012), James Noble (Ed.). Springer, 258–282. doi:10.1007/978-3-642-31057-7_13