# This is the Project Title

*Your Name*

Master of Science
Computer Science
School of Informatics
University of Edinburgh
2020

# Abstract

Formal development of Frank.

# Chapter 1

# Formalisation of Frank

| | |
|---|---|
| (data types) | $D$ |
| (value type variables) | $X$ |
| (effect type variables) | $E$ |
| (value types) | $A, B ::= D\,\overline{R}$ |
| | $\mid\ \{C\} \mid X$ |
| (computation types) | $C ::= \overline{T \to} G$ |
| (argument types) | $T ::= \langle\Delta\rangle A$ |
| (return types) | $G ::= [\Sigma]A$ |
| (type binders) | $Z ::= X \mid [E]$ |
| (type arguments) | $R ::= A \mid [\Sigma]$ |
| (polytypes) | $P ::= \forall\overline{Z}.A$ |
| (interfaces) | $I$ |
| (term variables) | $x, y, z, f$ |
| (instance variables) | $s, a, b, c$ |
| (seeds) | $\sigma ::= \emptyset \mid E$ |
| (abilities) | $\Sigma ::= \sigma \mid \Xi$ |
| (extensions) | $\Xi ::= \iota \mid \Xi, I\,\overline{R}$ |
| (adaptors) | $\Theta ::= \iota \mid \Theta, I(S \to S')$ |
| (adjustments) | $\Delta ::= \Theta \mid \Xi$ |
| (instance patterns) | $S ::= s \mid S\,a$ |
| (kind environments) | $\Phi, \Psi ::= \cdot \mid \Phi, Z$ |
| (type environments) | $\Gamma ::= \cdot \mid \Gamma, x : A \mid \Gamma, f : P$ |
| (instance environments) | $\Omega ::= s : \Sigma \mid \Omega, a : I\,\overline{R}$ |

Figure 1.1: Types

$$
\begin{array}{ll}
\text{(constructors)} & k \\
\text{(commands)} & c \\
\text{(uses)} & m ::= x \mid f\,\overline{R} \mid m\,\overline{n} \mid \uparrow\!(n:A) \\
\text{(constructions)} & n ::= \downarrow\! m \mid k\,\overline{n} \mid c\,\overline{R}\,\overline{n} \mid \{e\} \\
 & \quad \mid \ \textbf{let}\ f:P=n\ \textbf{in}\ n' \mid \textbf{letrec}\ \overline{f:P=e}\ \textbf{in}\ n \\
 & \quad \mid \ \langle\Theta\rangle\ n \\
\text{(computations)} & e ::= \overline{\overline{r}\mapsto n} \\
\text{(computation patterns)} & r ::= p \mid \langle c\,\overline{p} \to z\rangle \mid \langle x\rangle \\
\text{(value patterns)} & p ::= k\,\overline{p} \mid x
\end{array}
$$

Figure 1.2: Terms

$$\boxed{\Phi;\Gamma\,[\Sigma]\!\vdash m \Rightarrow A}$$

$$\textsc{T-Var}\quad \frac{x : A \in \Gamma}{\Phi;\Gamma\,[\Sigma]\!\vdash x \Rightarrow A} \qquad\qquad \textsc{T-PolyVar}\quad \frac{\Phi \vdash \overline{R} \qquad f : \forall \overline{Z}.A \in \Gamma}{\Phi;\Gamma\,[\Sigma]\!\vdash f\,\overline{R} \Rightarrow A[\overline{R}/\overline{Z}]}$$

$$\textsc{T-App}\quad \frac{\begin{array}{c}\Sigma' = \Sigma \qquad (\Sigma \vdash \Delta_i \dashv \Sigma'_i)_i \\ \Phi;\Gamma\,[\Sigma]\!\vdash m \Rightarrow \{\overline{\langle\Delta\rangle A \to}\ [\Sigma']B\} \qquad (\Phi;\Gamma\,[\Sigma'_i]\!\vdash n_i : A_i)_i\end{array}}{\Phi;\Gamma\,[\Sigma]\!\vdash m\,\overline{n} \Rightarrow B} \qquad \textsc{T-Ascribe}\quad \frac{\Phi;\Gamma\,[\Sigma]\!\vdash n : A}{\Phi;\Gamma\,[\Sigma]\!\vdash \uparrow(n : A) \Rightarrow A}$$

$$\boxed{\Phi;\Gamma\,[\Sigma]\!\vdash n : A}$$

$$\textsc{T-Switch}\quad \frac{\Phi;\Gamma\,[\Sigma]\!\vdash m \Rightarrow A \qquad A = B}{\Phi;\Gamma\,[\Sigma]\!\vdash \downarrow m : B} \qquad\qquad \textsc{T-Data}\quad \frac{k\,\overline{A} \in D\,\overline{R} \qquad (\Phi;\Gamma\,[\Sigma]\!\vdash n_j : A_j)_j}{\Phi;\Gamma\,[\Sigma]\!\vdash k\,\overline{n} : D\,\overline{R}}$$

$$\textsc{T-Command}\quad \frac{\Phi \vdash \overline{R} \qquad c : \forall \overline{Z}.\overline{A \to}\, B \in \Sigma \qquad (\Phi;\Gamma\,[\Sigma]\!\vdash n_j : A_j[\overline{R}/\overline{Z}])_j}{\Phi;\Gamma\,[\Sigma]\!\vdash c\,\overline{R}\,\overline{n} : B[\overline{R}/\overline{Z}]} \qquad \textsc{T-Thunk}\quad \frac{\Phi;\Gamma \vdash e : C}{\Phi;\Gamma\,[\Sigma]\!\vdash \{e\} : \{C\}}$$

$$\textsc{T-Let}\quad \frac{\begin{array}{c}P = \forall \overline{Z}.A \\ \Phi,\overline{Z};\Gamma\,[\emptyset]\!\vdash n : A \qquad \Phi;\Gamma, f : P\,[\Sigma]\!\vdash n' : B\end{array}}{\Phi;\Gamma\,[\Sigma]\!\vdash \mathbf{let}\ f : P = n\ \mathbf{in}\ n' : B}$$

$$\textsc{T-LetRec}\quad \frac{\begin{array}{c}(P_i = \forall \overline{Z}_i.\{C_i\})_i \\ (\Phi,\overline{Z}_i;\Gamma,\overline{f : P} \vdash e_i : C)_i \qquad \Phi;\Gamma,\overline{f : P}\,[\Sigma]\!\vdash n : B\end{array}}{\Phi;\Gamma\,[\Sigma]\!\vdash \mathbf{letrec}\ \overline{f : P = e}\ \mathbf{in}\ n : B} \qquad \textsc{T-Adapt}\quad \frac{\Sigma \vdash \Theta \dashv \Sigma' \qquad \Phi;\Gamma\,[\Sigma']\!\vdash n : A}{\Phi;\Gamma\,[\Sigma]\!\vdash \langle\Theta\rangle\ n : A}$$

$$\boxed{\Phi;\Gamma \vdash e : C}$$

$$\textsc{T-Comp}\quad \frac{\begin{array}{c}(\Phi \vdash r_{i,j} : T_j \dashv\![\Sigma]\ \exists \Psi_{i,j}.\Gamma'_{i,j})_{i,j} \\ (\Phi,(\Psi_{i,j})_j;\Gamma,(\Gamma'_{i,j})_j\,[\Sigma]\!\vdash n_i : B)_i \qquad ((r_{i,j})_i \text{ covers } T_j)_j\end{array}}{\Phi;\Gamma \vdash ((r_{i,j})_j \mapsto n_i)_i : (T_j \to)_j\ [\Sigma]B}$$

Figure 1.3: Term Typing Rules

$$
\begin{array}{lrl}
\text{(uses)} & m & ::= \cdots \mid \lceil \mathcal{E}[c\;\overline{R}\;\overline{w}] \rceil \\
\text{(constructions)} & n & ::= \cdots \mid \lceil \mathcal{E}[c\;\overline{R}\;\overline{w}] \rceil \\
\text{(use values)} & u & ::= x \mid f\;\overline{R} \mid \uparrow(v : A) \\
\text{(non-use values)} & v & ::= k\;\overline{w} \mid \{e\} \\
\text{(construction values)} & w & ::= \downarrow u \mid v \\
\text{(normal forms)} & t & ::= w \mid \lceil \mathcal{E}[c\;\overline{R}\;\overline{w}] \rceil \\
\text{(evaluation frames)} & \mathcal{F} & ::= [\,]\;\overline{n} \mid u\;(\bar{t},[\,],\overline{n}) \mid \uparrow([\,] : A) \\
 & & \mid\;\; \downarrow[\,] \mid k\;(\overline{w},[\,],\overline{n}) \mid c\;\overline{R}\;(\overline{w},[\,],\overline{n}) \\
 & & \mid\;\; \textbf{let}\; f : P = [\,]\;\textbf{in}\; n \mid \langle \Theta \rangle\;[\,] \\
\text{(evaluation contexts)} & \mathcal{E} & ::= [\,] \mid \mathcal{F}[\mathcal{E}]
\end{array}
$$

Figure 1.4: Runtime Syntax

$$\boxed{\Phi;\Gamma\,[\Sigma]\!\!-\; m \Rightarrow A} \qquad \boxed{\Phi;\Gamma\,[\Sigma]\!\!-\; n : A}$$

$$
\textsc{T-Freeze-Use}\quad
\frac{\neg(\mathcal{E}\ \text{handles}\ c) \qquad \Phi;\Gamma\,[\Sigma]\!\!-\; \mathcal{E}[c\;\overline{R}\;\overline{w}] \Rightarrow A}{\Phi;\Gamma\,[\Sigma]\!\!-\; \lceil \mathcal{E}[c\;\overline{R}\;\overline{w}] \rceil \Rightarrow A}
$$

$$
\textsc{T-Freeze-Cons}\quad
\frac{\neg(\mathcal{E}\ \text{handles}\ c) \qquad \Phi;\Gamma\,[\Sigma]\!\!-\; \mathcal{E}[c\;\overline{R}\;\overline{w}] : A}{\Phi;\Gamma\,[\Sigma]\!\!-\; \lceil \mathcal{E}[c\;\overline{R}\;\overline{w}] \rceil : A}
$$

Figure 1.5: Frozen Commands

$$\boxed{m \rightsquigarrow_{\mathrm{u}} m'} \quad \boxed{n \rightsquigarrow_{\mathrm{c}} n'} \quad \boxed{m \longrightarrow_{\mathrm{u}} m'} \quad \boxed{n \longrightarrow_{\mathrm{c}} n'}$$

R-HANDLE

$$\frac{k = \min_i \{i \mid \exists\overline{\theta}.(r_{i,j} : \langle\Delta_j\rangle A_j \leftarrow t_j \dashv[\Sigma]\ \theta_j)_j\} \qquad (r_{k,j} : \langle\Delta_j\rangle A_j \leftarrow t_j \dashv[\Sigma]\ \theta_j)_j}{\uparrow(\{((r_{i,j})_j \to n_i)_i\} : \{\overline{\langle\Delta\rangle A \to}\ [\Sigma]B\})\ \bar{t} \rightsquigarrow_{\mathrm{u}} \uparrow((\overline{\theta}(n_k) : B)}$$

R-ASCRIBE-USE

$$\frac{}{\uparrow(\downarrow u : A) \rightsquigarrow_{\mathrm{u}} u}$$

R-ASCRIBE-CONS

$$\frac{}{\downarrow\uparrow(w : A) \rightsquigarrow_{\mathrm{c}} w}$$

R-LET

$$\frac{}{\mathbf{let}\ f : P = w\ \mathbf{in}\ n \rightsquigarrow_{\mathrm{c}} n[\uparrow(w : P)/f]}$$

R-LETREC

$$\frac{\overline{e = \overline{\overline{r} \to n}}}{\mathbf{letrec}\ \overline{f : P = e}\ \mathbf{in}\ n' \rightsquigarrow_{\mathrm{c}} n'\overline{[\uparrow(\overline{\{\overline{r} \to \mathbf{letrec}\ \overline{f : P = e}\ \mathbf{in}\ n\}} : P)/f]}}$$

R-ADAPT

$$\frac{}{\langle\Theta\rangle\ w \rightsquigarrow_{\mathrm{c}} w}$$

R-FREEZE-COMM

$$\frac{}{c\ \overline{R}\ \overline{w} \rightsquigarrow_{\mathrm{c}} \lceil c\ \overline{R}\ \overline{w}\rceil}$$

R-FREEZE-FRAME-USE

$$\frac{\neg(\mathcal{F}[\mathcal{E}]\ \text{handles}\ c)}{\mathcal{F}[\lceil\mathcal{E}[c\ \overline{R}\ \overline{w}]\rceil] \rightsquigarrow_{\mathrm{u}} \lceil\mathcal{F}[\mathcal{E}[c\ \overline{R}\ \overline{w}]]\rceil}$$

R-FREEZE-FRAME-CONS

$$\frac{\neg(\mathcal{F}[\mathcal{E}]\ \text{handles}\ c)}{\mathcal{F}[\lceil\mathcal{E}[c\ \overline{R}\ \overline{w}]\rceil] \rightsquigarrow_{\mathrm{c}} \lceil\mathcal{F}[\mathcal{E}[c\ \overline{R}\ \overline{w}]]\rceil}$$

R-LIFT-UU

$$\frac{m \rightsquigarrow_{\mathrm{u}} m'}{\mathcal{E}[m] \longrightarrow_{\mathrm{u}} \mathcal{E}[m']}$$

R-LIFT-UC

$$\frac{m \rightsquigarrow_{\mathrm{u}} m'}{\mathcal{E}[m] \longrightarrow_{\mathrm{c}} \mathcal{E}[m']}$$

R-LIFT-CU

$$\frac{n \rightsquigarrow_{\mathrm{c}} n'}{\mathcal{E}[n] \longrightarrow_{\mathrm{u}} \mathcal{E}[n']}$$

R-LIFT-CC

$$\frac{n \rightsquigarrow_{\mathrm{c}} n'}{\mathcal{E}[n] \longrightarrow_{\mathrm{c}} \mathcal{E}[n']}$$

Figure 1.6: Operational Semantics

$\boxed{r : T \leftarrow t \dashv[\Sigma]\ \theta}$

B-Value

$$\frac{\begin{array}{c}\Sigma \vdash \Delta \dashv \Sigma' \\ p : A \leftarrow w \dashv \theta\end{array}}{p : \langle\Delta\rangle A \leftarrow w \dashv[\Sigma]\ \theta}$$

B-Request

$$\frac{\begin{array}{c}\Sigma \vdash \Delta \dashv \Sigma' \qquad \mathcal{E}\ \textsf{poisedfor}\ c \\ \Delta = \Theta \mid \Xi \qquad c : \forall \overline{Z}.\overline{B \to}\ B' \in \Xi \qquad (p_i : B_i \leftarrow w_i \dashv \theta_i)_i\end{array}}{\langle c\ \overline{p} \to z\rangle : \langle\Delta\rangle A \leftarrow \lceil \mathcal{E}[c\ \overline{R}\ \overline{w}]\rceil \dashv[\Sigma]\ \overline{\theta}[\uparrow(\{x \mapsto \mathcal{E}[x]\} : \{B' \to [\Sigma']A\})/z]}$$

B-CatchAll-Value

$$\frac{\Sigma \vdash \Delta \dashv \Sigma'}{\langle x\rangle : \langle\Delta\rangle A \leftarrow w \dashv[\Sigma]\ [\uparrow(\{w\}{:}\{[\Sigma']A\})/x]}$$

B-CatchAll-Request

$$\frac{\begin{array}{c}\Sigma \vdash \Delta \dashv \Sigma' \qquad \mathcal{E}\ \textsf{poisedfor}\ c \\ \Delta = \Theta \mid \Xi \qquad c : \forall \overline{Z}.\overline{B \to}\ B' \in \Xi\end{array}}{\langle x\rangle : \langle\Delta\rangle A \leftarrow \lceil \mathcal{E}[c\ \overline{R}\ \overline{w}]\rceil \dashv[\Sigma]\ [\uparrow(\{\lceil \mathcal{E}[c\ \overline{R}\ \overline{w}]\rceil\}{:}\{[\Sigma']A\})/x]}$$

$\boxed{p : A \leftarrow w \dashv \theta}$

B-Var

$$\frac{}{x : A \leftarrow w \dashv [\uparrow(w : A)/x]}$$

B-Data

$$\frac{k\ \overline{A} \in D\ \overline{R} \qquad (p_i : A_i \leftarrow w_i \dashv \theta_i)_i}{k\ \overline{p} : D\ \overline{R} \leftarrow k\ \overline{w} \dashv \overline{\theta}}$$

Figure 1.7: Pattern Binding

# Chapter 2

# Arbitrary Thread Interruption

## 2.1 Motivation

One important part of our asynchronous effect handling system is the ability to interrupt arbitrary computations. This is essential for pre-emptive concurrency, which relies on being able to suspend computations for resumption later.

For instance, consider the two programs below;

```
controller : {[Stop, Go, Console] Unit}
controller! = go!; stop!; print ``stop 1'';
              go!; stop!; print ``stop 2'';
              go!; stop!; print ``stop 3''

runner : {[Console] Unit}
runner! = print ``1 ``; print ``2 ``; print ``3 '';
```

We ideally want a multihandler that can run these two programs in parallel, such that the result will be `1 stop 1 2 stop 2 3 stop 3`; that is to say, the `stop` and `go` operations from

## 2.2 Interruption with Yields

One way we can get this behaviour is using the `Yield` interface. This offers a single operation, `yield : Unit`. With this, we can write a multihandler `suspend`;

```
runner : {[Console, Yield] Unit}
runner! = print "1 "; yield!; print "2 "; yield!; print "3 ";
   yield!
```

```
suspend : {<Yield> Unit -> <Stop, Go> Unit -> Maybe {[Console,
    Yield] Unit} -> [Console] Unit}
suspend <yield -> k> <stop -> l> _ = suspend unit (l unit) (
   just {k unit})
suspend <k>          <go -> l> (just res) = suspend (res!) (l
   unit) nothing
suspend <yield -> k> <m> maybe = suspend (k unit) m! maybe
suspend unit _ _ = unit
suspend _ unit _ = unit
```

**TODO:** Maybe make more concise?

**TODO:** change to not be letter l cos it looks like a 1

Running `suspend runner! controller! nothing` then prints out `1 stop 1 2 stop 2 3 stop 3` as desired. So far so good; this works as planned. However, observe that we had to change the code of the original runner program to `yield` every time it prints. We would rather not have this requirement; the threads should be suspendable without knowing in advance they will be suspended, and thus without needing to explicitly `yield`. Furthermore, see that the `yield` operation adds no more information; it is just used as a placeholder operation; any operation would work. As such, we keep searching for a better solution.

## 2.3 Relaxing Catches

The key to this lies in the catchall pattern, $\langle x \rangle$, and the pattern binding rules of Figure 1.7.

Recall that the catchall pattern $\langle x \rangle$ matches either a value — e.g. `unit` — or a command that is handled in the surrounding ability. For instance, the pattern `<k>` in the code above matches either `unit` or `<yield -> k>`. The variable `k` is then bound to whatever this match is, leaving the (potentially) invoked effect unhandled. This is expressed in the B-CatchAll-Request rule in Figure 1.7.

Important to note is that only effects that are handled in that position are able to be caught. `runner` also makes use of the `print` effect, but these are not able to be caught by the catchall command. Formally, this is due to the fourth requirement of B-CatchAll-Request; that the command $c$ that is invoked is a member of $\Xi$.

As such, we propose to remove this constraint from B-CatchAll-Request. The

$$\textsc{B-CatchAll-Request-Loose}\quad \frac{\Sigma \vdash \Delta \dashv \Sigma'}{\langle x \rangle : \langle \Delta \rangle A \leftarrow \lceil \mathcal{E}[c\ \overline{R}\ \overline{w}] \rceil \dashv_{[\Sigma]} [\uparrow(\{\lceil \mathcal{E}[c\ \overline{R}\ \overline{w}] \rceil\}{:}\{[\Sigma']A\})/x]}$$

Figure 2.1: Updated B-CatchAll-Request

resulting rule can then be seen in Figure 2.1. This lets us update the previous `suspend` code to the following, which yields the same results as last time;

```
runner : {[Console] Unit}
runner! = print "1 "; print "2 "; print "3 "

suspend : {Unit -> <Stop, Go> Unit -> Maybe {[Console] Unit}
   -> [Console] Unit}
suspend <k> <stop -> l> _          = suspend unit   (l unit) (
   just {k unit})
suspend <k> <go -> l>   (just res) = suspend (res!) (l unit)
   nothing
suspend unit _ _ = unit
suspend _ unit _ = unit
```

**TODO:** Verify that this particular example really works.

Why does this work?

**TODO:** explain

## 2.4 Interrupting Arbitrary Terms

The approach of Section 2.3 can only interrupt command invocations. If `runner` were instead a sequence of pure computations[1], we would be unable to interrupt it; it does not invoke commands.

As such, we need to further change the pattern binding rules of Figure 1.7. This is to let us interrupt arbitrary computation terms. In Figure 2.2, we see an updated version of the runtime syntax; this allows for the suspension of arbitrary *uses*, being function applications and constructions.

**TODO:** verify that uses are “just” apps and constructions

[1] I.e. `runner! = 1 + 1; 1 + 1; 1 + 1; ...`

$$
\begin{array}{lrl}
\text{(uses)} & m & ::= \cdots \mid \lceil \mathcal{E}[c\,\overline{R}\,\overline{w}] \rceil \\
\text{(constructions)} & n & ::= \cdots \mid \lceil \mathcal{E}[c\,\overline{R}\,\overline{w}] \rceil \\
\text{(use values)} & u & ::= x \mid f\,\overline{R} \mid {\uparrow}(v : A) \\
\text{(non-use values)} & v & ::= k\,\overline{w} \mid \{e\} \\
\text{(construction values)} & w & ::= {\downarrow}u \mid v \\
\text{(normal forms)} & t & ::= w \mid \lceil \mathcal{E}[c\,\overline{R}\,\overline{w}] \rceil \mid !(m) \\
\text{(evaluation frames)} & \mathcal{F} & ::= [\,]\,\overline{n} \mid u\ (\bar{t}, [\,], \overline{n}) \mid {\uparrow}([\,] : A) \\
 & & \mid\ \ {\downarrow}[\,] \mid k\ (\overline{w}, [\,], \overline{n}) \mid c\,\overline{R}\ (\overline{w}, [\,], \overline{n}) \\
 & & \mid\ \ \mathbf{let}\ f : P = [\,]\ \mathbf{in}\ n \mid \langle \Theta \rangle\ [\,] \\
\text{(evaluation contexts)} & \mathcal{E} & ::= [\,] \mid \mathcal{F}[\mathcal{E}]
\end{array}
$$

Figure 2.2: Runtime Syntax, Updated with Suspension of Uses

B-CatchAll-Interrupt

$$
\frac{\Sigma \vdash \Delta \dashv \Sigma'}{\langle x \rangle : \langle \Delta \rangle A \leftarrow !(m) \dashv [\Sigma]\ [{\uparrow}(\{m\} : \{[\Sigma']A\})/x]}
$$

Figure 2.3: Catching Interrupts rule.

**TODO:** Do we need to add interrupts to evaluation frames?

With this in mind, we now give the updated rule for the catchall pattern matching on interrupted terms (denoted $!(m)$). This can be seen in Figure 2.3. It expresses that an arbitrary suspended *use* can be matched against the computation pattern $\langle x \rangle$. The suspended computation $\{m\}$ is then bound to $x$, in a similar way to other B-CatchAll rules.

Figure 2.4 shows how uses $m$ become interrupted. This rule supplements the operational semantics of Figure 1.6.

**TODO:** Check that just uses is enough

**TODO:** Talk about how this achieves our goal.

The addition of this rule introduces non-determinism into the language; at any point, a use can either step as normal (e.g. through the R-Handle rule), or it can be interrupted. An interrupted term $!(m)$ can no longer reduce; it is blocked until it is resumed by the R-Handle rule.

**TODO:** Talk about non-determinism as a result of this

**TODO:** Maybe move non-determinism to the next section?

R-Interrupt

$$\frac{}{m \rightsquigarrow_{\mathrm{u}} !(m)}$$

Figure 2.4: Use interruption rule

## 2.5 Interrupting In Practice

Due to the non-determinism introduced by R-Interrupt, this system is difficult to implement; how do we choose whether to apply a handler to its arguments or to just interrupt it?

In our implementation of this system, we instead maintain a counter which is incremented every time a handler is evaluated. When the counter reaches a certain threshold value $t$, we interrupt the current term $m$, applying the R-Interrupt rule. This converts the non-deterministic system to a deterministic one; we never have any question of *what* to do.

Observe that the process of interrupting a computation is a familiar one; we stop computing and offer up the continuation to the programmer. Where is a similar control flow already around? That's right — invocation of an effect.

**TODO:** Rewrite above paragraph to be less camp

As such, we can just use a normal effect to perform the interruption. Sticking with previous themes, we choose to invoke a `yield` effect when interrupting. This lets the programmer choose to handle the effect as they wish. Note that now interrupts are not restricted to the catch-all pattern $\langle m \rangle$ but are normal computation patterns.

These `yield` effects are only inserted in a computation when the `Yield` interface is present in the ability of this computation. This is important, as it lets the programmer get fine-grained control over which computations can be interrupted and which cannot. Consider the example in Section 2.1; we want the `runner` computation to be controller by the `controller`. As such, we want the `runner` to be interruptible, whilst the `controller` is not. We can reflect this by adding the `Yield` interface to the ability of the former computation.

Thus our example from before becomes;

**TODO:** updated example

# Appendix A

# Remaining Formalisms

$$\boxed{\Sigma \vdash \Delta \dashv \Sigma'}$$

$$\begin{array}{l}\textsc{A-Adj}\\ \dfrac{\Sigma \vdash \Theta \dashv \Sigma' \qquad \Sigma' \vdash \Xi \dashv \Sigma''}{\Sigma \vdash \Theta \mid \Xi \dashv \Sigma''}\end{array}$$

$$\boxed{\Sigma \vdash \Xi \dashv \Sigma'}$$

$$\begin{array}{l}\textsc{A-Ext-Id}\\ \dfrac{}{\Sigma \vdash \iota \dashv \Sigma}\end{array} \qquad \begin{array}{l}\textsc{A-Ext-Snoc}\\ \dfrac{\Sigma \vdash \Xi \dashv \Sigma'}{\Sigma \vdash \Xi, I\,\overline{R} \dashv \Sigma', I\,\overline{R}}\end{array}$$

$$\boxed{\Sigma \vdash \Theta \dashv \Sigma'}$$

$$\begin{array}{l}\textsc{A-Adapt-Id}\\ \dfrac{}{\Sigma \vdash \iota \dashv \Sigma}\end{array} \qquad \begin{array}{l}\textsc{A-Adapt-Snoc}\\ \dfrac{\Sigma \vdash \Theta \dashv \Sigma' \qquad \Sigma' \vdash I(S \to S') \dashv \Sigma''}{\Sigma \vdash \Theta, I(S \to S') \dashv \Sigma''}\end{array}$$

$$\boxed{\Sigma \vdash I(S \to S') \dashv \Sigma'}$$

$$\begin{array}{l}\textsc{A-Adapt-Com}\\ \dfrac{\Sigma \vdash S : I \dashv \Sigma';\Omega \qquad \Omega \vdash S' : I \dashv \Xi \qquad \Sigma' \vdash \Xi \dashv \Sigma''}{\Sigma \vdash I(S \to S') \dashv \Sigma''}\end{array}$$

$$\boxed{\Sigma \vdash S : I \dashv \Sigma';\Omega}$$

$$\begin{array}{l}\textsc{I-Pat-Id}\\ \dfrac{}{\Sigma \vdash s : I \dashv \Sigma; s : \Sigma}\end{array} \qquad \begin{array}{l}\textsc{I-Pat-Bind}\\ \dfrac{\Sigma \vdash S : I \dashv \Sigma';\Omega}{\Sigma, I\,\overline{R} \vdash S\,a : I \dashv \Sigma';\Omega, a : I\,\overline{R}}\end{array} \qquad \begin{array}{l}\textsc{I-Pat-Skip}\\ \dfrac{\Sigma \vdash S\,a : I \dashv \Sigma';\Omega \qquad I \neq I'}{\Sigma, I'\,\overline{R} \vdash S\,a : I \dashv \Sigma', I'\,\overline{R};\Omega}\end{array}$$

$$\boxed{\Omega \vdash S : I \dashv \Xi}$$

$$\begin{array}{l}\textsc{I-Inst-Id}\\ \dfrac{s \in \mathsf{dom}(\Omega)}{\Omega \vdash s : I \dashv \iota}\end{array} \qquad \begin{array}{l}\textsc{I-Inst-Lkp}\\ \dfrac{a \in \mathsf{dom}(\Omega) \qquad \Omega \vdash S : I \dashv \Xi \qquad \Omega(a) = I\,\overline{R}}{\Omega \vdash S\,a : I \dashv \Xi, I\,\overline{R}}\end{array}$$

Figure A.1: Action of an Adjustment on an Ability and Auxiliary Judgements

$$\mathcal{X} ::= A \mid C \mid T \mid G \mid Z \mid R \mid P \mid \sigma \mid \Sigma \mid \Xi \mid \Theta \mid \Delta \mid \Gamma \mid \exists \Psi.\Gamma \mid \Omega$$

$\boxed{\Phi \vdash \mathcal{X}}$

$$\overset{\textsc{WF-Val}}{\frac{}{\Phi, X \vdash X}} \qquad \overset{\textsc{WF-Eff}}{\frac{}{\Phi, [E] \vdash E}} \qquad \overset{\textsc{WF-Poly}}{\frac{\Phi, \overline{Z} \vdash A}{\Phi \vdash \forall \overline{Z}.A}}$$

$$\overset{\textsc{WF-Data}}{\frac{(\Phi \vdash R)_i}{\Phi \vdash D\,\overline{R}}} \qquad \overset{\textsc{WF-Thunk}}{\frac{\Phi \vdash C}{\Phi \vdash \{C\}}} \qquad \overset{\textsc{WF-Comp}}{\frac{(\Phi \vdash T)_i \qquad \Phi \vdash G}{\Phi \vdash \overline{T \to} \, G}} \qquad \overset{\textsc{WF-Arg}}{\frac{\Phi \vdash \Delta \qquad \Phi \vdash A}{\Phi \vdash \langle \Delta \rangle A}}$$

$$\overset{\textsc{WF-Ret}}{\frac{\Phi \vdash \Sigma \qquad \Phi \vdash A}{\Phi \vdash [\Sigma]A}} \qquad \overset{\textsc{WF-Ability}}{\frac{\Phi \vdash \Sigma}{\Phi \vdash [\Sigma]}} \qquad \overset{\textsc{WF-Pure}}{\frac{}{\Phi \vdash \emptyset}} \qquad \overset{\textsc{WF-Id}}{\frac{}{\Phi \vdash \iota}} \qquad \overset{\textsc{WF-Ext}}{\frac{\Phi \vdash \Xi \qquad (\Phi \vdash R)_i}{\Phi \vdash \Xi, I\,\overline{R}}}$$

$$\overset{\textsc{WF-Adapt}}{\frac{\Phi \vdash \Theta}{\Phi \vdash \Theta, I\,(S \to S')}}$$

$$\overset{\textsc{WF-Empty}}{\frac{}{\Phi \vdash \cdot}} \qquad \overset{\textsc{WF-Mono}}{\frac{\Phi \vdash \Gamma \qquad \Phi \vdash A}{\Phi \vdash \Gamma, x : A}} \qquad \overset{\textsc{WF-Poly}}{\frac{\Phi \vdash \Gamma \qquad \Phi \vdash P}{\Phi \vdash \Gamma, f : P}}$$

$$\overset{\textsc{WF-Existential}}{\frac{\Phi, \Psi \vdash \Gamma}{\Phi \vdash \exists \Psi.\Gamma}} \qquad \overset{\textsc{WF-Interface}}{\frac{\Phi \vdash \Omega \qquad (\Phi \vdash R)_i}{\Phi \vdash \Omega, x : I\,\overline{R}}}$$

Figure A.2: Well-Formedness Rules

$$\boxed{\Phi \vdash p : A \dashv \Gamma}$$

$$\frac{}{\Phi \vdash x : A \dashv x : A}\ \textsc{P-Var} \qquad \frac{k\,\overline{A} \in D\,\overline{R} \qquad (\Phi \vdash p_i : A_i \dashv \Gamma)_i}{\Phi \vdash k\,\overline{p} : D\,\overline{R} \dashv \overline{\Gamma}}\ \textsc{P-Data}$$

$$\boxed{\Phi \vdash r : T \dashv[\Sigma]\ \exists\Psi.\Gamma}$$

$$\frac{\Sigma \vdash \Delta \dashv \Sigma' \qquad \Phi \vdash p : A \dashv \Gamma}{\Phi \vdash p : \langle\Delta\rangle A \dashv[\Sigma]\ \Gamma}\ \textsc{P-Value} \qquad \frac{\Sigma \vdash \Delta \dashv \Sigma'}{\Phi \vdash \langle x\rangle : \langle\Delta\rangle A \dashv[\Sigma]\ x : \{[\Sigma']A\}}\ \textsc{P-CatchAll}$$

$$\frac{\Sigma \vdash \Delta \dashv \Sigma' \qquad \Delta = \Theta \mid \Xi \qquad c : \forall\overline{\mathrm{Z}}.\overline{A \to}B \in \Xi \qquad (\Phi, \overline{Z} \vdash p_i : A_i \dashv \Gamma_i)_i}{\Phi \vdash \langle c\ \overline{p} \to z\rangle : \langle\Delta\rangle B' \dashv[\Sigma]\ \exists\overline{\mathrm{Z}}.\overline{\Gamma}, z : \{\langle\iota \mid \iota\rangle B \to [\Sigma']B'\}}\ \textsc{P-Command}$$

Figure A.3: Pattern Matching Typing Rules