1 Syntax

$$
\begin{array}{rclll}
\mathit{Path} & \ni & q & ::= & \epsilon \mid q.n \\
\mathit{Place} & \ni & \pi & ::= & x.q \\
\mathit{Place} & \ni & p & ::= & \ell.q \\
\mathit{Const} & \ni & c & ::= & \textbf{false} \mid \textbf{true} \mid z \\
\mathit{RVal} & \ni & R & ::= & c \mid *\pi \mid *\pi_1 + *\pi_2 \\
\mathit{Instr} & \ni & I & ::= & \textbf{let } x = \textbf{new}(n) \mid \pi \coloneqq R \\
\mathit{FuncBody} & \ni & F & ::= & I; F \mid \textbf{letcont } k(\Gamma; \mathbf{T}; \overline{x{:}\,\ell}) = F_1 \textbf{ in } F_2 \mid \textbf{jump } k(\overline{x}) \\
 & & & \mid & \textbf{call } f(\overline{x}) \textbf{ ret } k \mid \textbf{if } *\pi \textbf{ then } F_1 \textbf{ else } F_2 \mid \textbf{abort} \\
\mathit{BType} & \ni & \beta & ::= & \texttt{bool} \mid \texttt{int} \\
\mathit{Type} & \ni & \tau & ::= & \textbf{fn}(\Gamma; \overline{\ell}) \to \Gamma'/\ell \mid \{x{:}\,\beta \mid r\} \mid \Pi(\overline{x{:}\,\tau}) \\
\mathit{Pred} & \ni & r & ::= & \ldots \\
\mathit{GEnv} & \ni & \Gamma & ::= & \emptyset \mid \Gamma, \ell{:}\,\tau \mid \Gamma, r \\
\mathit{TEnv} & \ni & \mathbf{T} & ::= & \emptyset \mid \mathbf{T}, x{:}\,\texttt{own}\,(\ell) \\
\mathit{KEnv} & \ni & \mathbf{K} & ::= & \emptyset \mid \mathbf{K}, k{:}\,\textbf{cont}(\Gamma; \mathbf{T}; \overline{\ell})
\end{array}
$$

2 Typing

Well-typed rvalues $\boxed{\Gamma; \mathbf{T} \vdash R : \tau \rightsquigarrow \Gamma'; \mathbf{T}'}$

R-ADD
$$\frac{(\Gamma; \mathbf{T})(\pi_1) = (p_1, \{\texttt{int} \mid r_1\}) \qquad (\Gamma; \mathbf{T})(\pi_2) = (p_2, \{\texttt{int} \mid r_2\})}{\Gamma; \mathbf{T} \vdash *\pi_1 + *\pi_2 : \{\texttt{int} \mid \nu = p_1 + p_2\} \rightsquigarrow \Gamma; \mathbf{T}}$$

R-COPY
$$\frac{(\Gamma; \mathbf{T})(\pi) = (p, \tau) \qquad \tau \text{ copy}}{\Gamma; \mathbf{T} \vdash *\pi : \texttt{self}(p, \tau) \rightsquigarrow \Gamma; \mathbf{T}}$$

R-MOVE
$$\frac{(\Gamma; \mathbf{T})(\pi) = (p, \tau) \qquad \tau \text{ noncopy} \qquad n = \text{size}(\tau)}{\Gamma; \mathbf{T} \vdash *\pi : \tau \rightsquigarrow (\Gamma; \mathbf{T})[\pi \mapsto \lightning_n]}$$

R-CONST
$$\Gamma; \mathbf{T} \vdash c : \{\delta(c) \mid \nu = c\} \rightsquigarrow \Gamma; \mathbf{T}$$

Well-typed instructions $\boxed{\Gamma; \mathbf{T} \vdash I \rightsquigarrow \Gamma; \mathbf{T}'}$

I-NEW
$$\frac{\Gamma' = \Gamma * \ell{:}\,\lightning_n \qquad \mathbf{T} = \mathbf{T} * x{:}\,\texttt{own}\,(\ell)}{\Gamma; \mathbf{T} \vdash \textbf{let } x = \textbf{new}(n) \rightsquigarrow \Gamma'; \mathbf{T}'}$$

R-ASSIGN
$$\frac{\Gamma; \mathbf{T} \vdash R : \tau' \rightsquigarrow \Gamma'; \mathbf{T}' \qquad (\Gamma; \mathbf{T})(\pi) = (p, \tau) \qquad \text{size}(\tau) = \text{size}(\tau')}{\Gamma; \mathbf{T} \vdash \pi \coloneqq R \rightsquigarrow (\Gamma'; \mathbf{T}')[\pi \mapsto \tau']}$$

Well-typed functions $\boxed{\mathbf{T}|\mathbf{K} \vdash F}$

$$\textsc{F-instr}\quad \frac{\Gamma; \mathbf{T} \vdash I \rightsquigarrow \Gamma'; \mathbf{T}' \qquad \Gamma; \mathbf{T}|\mathbf{K} \vdash F}{\Gamma; \mathbf{T}|\mathbf{K} \vdash I; F}$$

$$\textsc{F-letcont}\quad \frac{\Gamma'; \mathbf{T}', \overline{x\colon \mathtt{own}\,(\ell)}|\mathbf{K}, k\colon \mathbf{cont}(\Gamma'; \mathbf{T}'; \overline{\ell}) \vdash F_1 \qquad \Gamma; \mathbf{T}|\mathbf{K}, k\colon \mathbf{cont}(\Gamma'; \mathbf{T}'; \overline{\ell}) \vdash F_2}{\Gamma; \mathbf{T}|\mathbf{K} \vdash \mathbf{letcont}\ k(\Gamma'; \mathbf{T}'; \overline{\ell}) = F_1\ \mathbf{in}\ F_2}$$

$$\textsc{F-jump}\quad \frac{\mathbf{K}(k) = \mathbf{cont}(\Gamma'; \mathbf{T}'; \overline{\ell}) \qquad \Gamma; \mathbf{T} \Rightarrow^{\theta} \Gamma'; \mathbf{T}' * \overline{x\colon \mathtt{own}\,(\ell)} \qquad \mathbf{T} \Subset \overline{x\colon \mathtt{own}\,(\ell)} * \mathbf{T}' \qquad \Gamma \preceq \theta\Gamma'}{\Gamma; \mathbf{T}|\mathbf{K} \vdash \mathbf{jump}\ k(\overline{x})}$$

$$\textsc{F-call}\quad \frac{\begin{array}{c}\mathbf{T}(f) = \mathbf{fn}(\Gamma_f; \overline{\ell_f}) \to \Gamma_o/\ell_o \qquad \mathbf{K}(k) = \mathbf{cont}(\Gamma_k; \mathbf{T}_k; \ell_k) \\ \Gamma; \mathbf{T} \Rightarrow^{\theta_1} \Gamma_f; \overline{x\colon \mathtt{own}\,(\ell_f)} \qquad \Gamma; \mathbf{T} * y\colon \mathtt{own}\,(\ell_o) \Rightarrow^{\theta_2} \Gamma_k; \mathbf{T}_k * y\colon \mathtt{own}\,(\ell_k) \\ \Gamma \preceq \theta_1\Gamma_f \qquad \Gamma * \theta_1\Gamma_o \preceq \theta_2\Gamma_k \qquad \mathbf{T} \Subset \overline{x\colon \mathtt{own}\,(\ell_f)} * \mathbf{T}_k\end{array}}{\Gamma; \mathbf{T}|\mathbf{K} \vdash \mathbf{call}\ f(\overline{x})\ \mathbf{ret}\ k}$$

$$\textsc{F-if}\quad \frac{(\Gamma; \mathbf{T})(\pi) = (p, \{\mathtt{bool} \mid r\}) \qquad \Gamma, p; \mathbf{T}|\mathbf{K} \vdash F_1 \qquad \Gamma, \neg p; \mathbf{T}|\mathbf{K} \vdash F_2}{\Gamma; \mathbf{T}|\mathbf{K} \vdash \mathbf{if}\ {*\pi}\ \mathbf{then}\ F_1\ \mathbf{else}\ F_2}$$

$$\textsc{F-abort}\quad \Gamma; \mathbf{T}|\mathbf{K} \vdash \mathbf{abort}$$

3 Environment inclusion

Environment inclusion $\boxed{\mathbf{T}_1 \Subset \mathbf{T}_2}$

$$\frac{\mathrm{dom}(\mathbf{T}_2) \subseteq \mathrm{dom}(\mathbf{T}_1)}{\mathbf{T}_1 \Subset \mathbf{T}_2}$$

4 Subtyping

Environment subtyping $\boxed{\Gamma_1 \preceq \Gamma_2}$

$$\preceq\textsc{-env-empty}\quad \Gamma_1 \preceq \emptyset$$

$$\preceq\textsc{-env-var}\quad \frac{\Gamma_1 \preceq \Gamma_2 \qquad \Gamma_1 \vdash \Gamma_1(x) \preceq \tau}{\Gamma_1 \preceq x\colon \tau, \Gamma_2}$$

Subtyping $\boxed{\Gamma \vdash \tau_1 \preceq \tau_2}$

$$\preceq\textsc{-refine}\quad \frac{\mathtt{Valid}(\llbracket\Gamma\rrbracket \wedge r_1 \Rightarrow r_2[x/y])}{\Gamma \vdash \{x\colon \beta \mid r_1\} \preceq \{y\colon \beta \mid r_2\}}$$

$$\preceq\textsc{-fun}\quad \frac{\theta = [\overline{\ell_2}/\overline{\ell_1}] \qquad \Gamma * \Gamma_2 \preceq \theta\Gamma_1 \qquad \Gamma * \Gamma_2 * \theta\Gamma_1' \preceq \Gamma_2'[\ell_1'/\ell_2']}{\Gamma \vdash \mathbf{fn}(\Gamma_1; \overline{\ell_1}) \to \Gamma_1'/\ell_1' \preceq \mathbf{fn}(\Gamma_2; \overline{\ell_2}) \to \Gamma_2'/\ell_2'}$$

$$\preceq\textsc{-prod}\quad \frac{\Gamma * \overline{\ell\colon \tau} \preceq (\overline{\ell\colon \tau'})[\overline{\ell}/\overline{y}]}{\Gamma \vdash \Pi(\overline{x\colon \tau}) \preceq \Pi(\overline{y\colon \tau'})}$$

5 Metafunctions

$\boxed{\Gamma_1 * \Gamma_2 = \Gamma}$

$$\Gamma_1 * \Gamma_2, \ell\colon \tau \quad = \quad \Gamma_1, \ell\colon \tau * \Gamma_2$$
$$\quad \mathbf{where}\ \ \ell \notin \mathtt{dom}(\Gamma_1)$$

$\boxed{\mathbf{T}_1 * \mathbf{T}_2 = \mathbf{T}}$

$$\begin{array}{lll} \mathbf{T}_1 * \mathbf{T}_2, x{:}\,\tau & = & \mathbf{T}_1, x{:}\,\tau * \mathbf{T}_2 \\ \quad \textbf{where}\ x \notin \mathtt{dom}(\mathbf{T}_1) & & \end{array}$$

$\boxed{(\Gamma; \mathbf{T})[\pi \mapsto \tau] = \Gamma; \mathbf{T}}$

$$\begin{array}{l} (\Gamma; \mathbf{T})[x.p \mapsto \tau] \quad = \quad (\Gamma, \ell'{:}\,\tau'; \mathbf{T}[x \mapsto \mathbf{own}\,(\ell')]) \\ \quad \textbf{where}\ \begin{array}{lcl} \mathbf{own}\,(\ell) & = & \mathbf{T}(x) \\ \tau' & = & \Gamma(\ell)[p \mapsto \tau] \\ \text{fresh } \ell' & & \end{array} \end{array}$$

$\boxed{\Gamma_1; \mathbf{T}_1 \Rightarrow^{\theta} \Gamma_2; \mathbf{T}_2}$

$$\Gamma_1; \mathbf{T}_1 \Rightarrow^{\emptyset} \Gamma_2; \emptyset \qquad \frac{\mathbf{own}\,(\ell_1) = \mathbf{T}_1(x) \qquad \Gamma_1; \mathbf{T}_1 \Rightarrow^{\theta_1} \Gamma_2; \mathbf{T}_2 \qquad \mathtt{subst}(\Gamma_1(\ell_1); \Gamma_2(\ell_2)) = \theta_2}{\Gamma_1; \mathbf{T}_1 \Rightarrow^{\theta_1 \cdot \theta_2 \cdot [\ell_1/\ell_2]} \Gamma_2; \mathbf{T}_2, x{:}\,\mathbf{own}\,(\ell_2)}$$

$\boxed{\mathtt{subst}(\tau_1; \tau_2) = \theta}$

$$\begin{array}{lcl} \mathtt{subst}(\mathbf{own}\,(\ell_1)\,; \mathbf{own}\,(\ell_2)) & = & [\ell_1/\ell_2] \\ \mathtt{subst}(\tau_1; \tau_2) & = & \emptyset \end{array}$$

$\boxed{\tau.q = \tau'}$

$$\begin{array}{lcl} \tau.\epsilon & = & \tau \\ \Pi(x_0{:}\,\tau_0, \ldots, x_n{:}\,\tau_n, x_m{:}\,\tau_m).n.q & = & \tau_n.q \end{array}$$

$\boxed{(\Gamma; \mathbf{T})(\pi) = (p, \tau)}$

$$\begin{array}{l} (\Gamma; \mathbf{T})(x.q) \quad = \quad (\ell.q, \tau.q) \\ \quad \textbf{where}\ \begin{array}{lcl} \mathbf{own}\,(\ell) & = & \mathbf{T}(x) \\ \tau & = & \Gamma(y) \end{array} \end{array}$$

$\boxed{\mathtt{self}(p, \tau) = \tau'}$

$$\begin{array}{lcl} \mathtt{self}(p, \{x{:}\,\beta \mid r\}) & = & \{x{:}\,\beta \mid x = p\} \\ \mathtt{self}(p, \tau) & = & \tau \end{array}$$

$\boxed{[\![\Gamma]\!] = r}$

$$\begin{array}{lcl} [\![\emptyset]\!] & = & \mathbf{true} \\ [\![\Gamma, r]\!] & = & [\![\Gamma]\!] \wedge r \\ [\![\Gamma, \ell{:}\,\tau]\!] & = & [\![\Gamma]\!] \wedge [\![\tau]\!]_{\ell} \end{array}$$

$\boxed{[\![\tau]\!]_p = r}$

$$\begin{array}{lcl} [\![\{y{:}\,\beta \mid r\}]\!]_p & = & r[p/y] \\ [\![\Pi(\overline{x_i{:}\,\tau_i})]\!]_p & = & \bigwedge_i [\![\tau]\!]_{p.i} \\ [\![\tau]\!]_p & = & \mathbf{true} \end{array}$$

6 Examples

6.1 Tracking variable versions

```
fn ris(;;) ret k($r_0$:(); $r_0$) =
  let p = new(2) in    // $p_0$: $\lightning_2$; $p$: own($p_0$)
  p.0 := 1;            // ..., $p_1$: $\{i32 \mid \nu = 1\} \times \lightning_1$; $p$: own($p_1$)
  p.1 := 2;            // ..., $p_2$: $\{i32 \mid \nu = 1\} \times \{i32 \mid \nu = 2\}$; $p$: own($p_2$)
  let x = new(1) in    // ..., $x_0$: $\lightning_1$; $p$: own($p_2$), $x$: own($x_0$)
  x := *p.0;           // ..., $x_1$: $\{i32 \mid \nu = p_2.0\}$; $p$: own($p_2$), $x$: own($x_1$)
  p.0 := 3;            // ..., $p_3$: $\{i32 \mid \nu = 3\} \times \{i32 \mid \nu = 2\}$; $p$: own($p_3$), $x$: own($x_1$)
  let y = new(1) in    // ..., $y_0$: $\lightning_1$; $p$: own($p_3$), $x$: own($x_1$), $y$: own($y_0$)
  y := *p.0;           // ..., $y_0$: $\{i32 \mid \nu = p_3.0\}$; $p$: own($p_3$), $x$: own($x_1$), $y$: own($y_1$)
  let z = new(1) in    // ..., $z_0$: $\lightning_1$; $p$: own($p_3$), $x$: own($x_1$), $y$: own($y_1$), $z$: own($z_0$)
  z := *p.1;           // ..., $z_1$: $\{i32 \mid \nu = p_3.1\}$; $p$: own($p_3$), $x$: own($x_1$), $y$: own($y_1$), $z$: own($z_1$)
  jump(())
```

6.2 Basic control flow

```
fn abs($x_0$: $i32$; x: own($x_0$)) ret k($r_0$: $\{i32 \mid \nu > 0\}$; r: own($r_0$)) =
  // $x_0$: $i32$; $x$: own($x_0$)
  let b: bool = *x < 0 in   // ..., $b_0$: $\{bool \mid \nu \Leftrightarrow x_0 < 0\}$; $b$: own($b_0$)
  if b then                 // ..., $b$
    x := 0 - *x             // ..., $x_1$: $\{i32 \mid \nu = 0 - x_0\}$; $x$: own($x_1$)
    jump k(x)
      // $x_0$: $i32$, $b_0$: $\{i32 \mid \nu \Leftrightarrow x_0 < 0\}$, $b$, $x_1$: $\{i32 \mid \nu = 0 - x_0\}$; $x$: own($x_1$), $b$: own($b_0$)
      // $\preceq$
      // $m_0$: $\{i32 \mid \nu > 0\}$; $m$: own($m_0$)

  else  // ..., $\neg b$
    jump k(x)
      // $x_0$: $i32$, $b_0$: $\{i32 \mid \nu \Leftrightarrow x_0 < 0\}$, $\neg b$; $x$: own($x_1$), $b$: own($b_0$)
      // $\preceq$
      // $r_0$: $\{i32 \mid \nu > 0\}$; $x$: own($r_0$)
```

6.3 Dependent function

```
fn ira($a_0$: $i32$, $b_0$: $\{i32 \mid \nu > a_0\}$; a:own($a_0$), b:own($b_0$)) ret k($r_0$: $\{i32 \mid \nu > 0\}$; own($r_0$)) =
  // $a_0$: $i32$, $b_0$: $\{i32 \mid \nu > a_0\}$;  $a$: own($a_0$),  $b$: own($b_0$)
  let t = new(1);   // $\ldots, t_0$: $\lightning_1$;  $\ldots, t$ : own($b_0$)
  t := *a - *b;     // $\ldots, t_1$: $\{i32 \mid \nu = a_0 - b_0\}$;  $\ldots, t$: own($t_1$)
  jump k(t) // $a_0$: $i32$, $b_0$: $\{i32 \mid \nu > a_0\}$, $t_0$: $\lightning_1$, $t_1$: $\{i32 \mid \nu = a_0 - b_0\}$; $a$: own($a_0$), $b$: own($b_0$), $t$: own($t_1$)
            // $\preceq$
            // $r_0$: $\{i32 \mid \nu > 0\}$; $t$: own($r_0$)
```

6.4 Function call

```
fn count_zeros($n_0$: $\{i32 \mid \nu \geq 0\}$; n: own($n_0$)) ret k($r_0$: $\{i32 \mid \nu \geq 0\}$; own($r_0$)) =
  letcont b0($i_1$: $\{i32 \mid \nu \geq 0\}$, $c_1$: $\{i32 \mid c \geq 0\}$; n: own($n_0$), i: own($i_1$); c: own($c_1$); ) =
    let t1: bool = *i < *n;
    if *t1 then
      letcont b1($x_0$: $i32$; $n$: own($n_0$), $i$: own($i_1$), $c$: own($c_1$); x:own($x_0$)) =
        letcont b2(; $n$: own($n_0$), $i$: own($i_1$), $c$: own($c_1$); ) =
          i := *i + 1;
          jump b0()
        in
        let t2: bool = *x == 0;
        if *t2 then
          c := *c + 1;
          jump b2()
        else
          jump b2()
      in
      call f(i) ret b1
        // $n_0$: $\{i32 \mid \nu \geq 0\}$, $i_1$: $\{i32 \mid \nu \geq 0\}$, $c_1$: $\{i32 \mid c \geq 0\}$, $t1_0$: $\{bool \mid i_1 < n_0\}$, $t1_0$
        // ;$n$: own($n_0$), $i$: own($i_1$), $c$: own($c_1$)
        // $\preceq$
        // $y_0$: $\{i32 \mid \nu \geq 0\}$;  $i$: own($y_0$)
        //
        // $n_0$: $\{i32 \mid \nu \geq 0\}$, $i_1$: $\{i32 \mid \nu \geq 0\}$, $c_1$: $\{i32 \mid c \geq 0\}$, $t1_0$: $\{bool \mid i_1 < n_0\}$, $t1_0$, $r_0$: $\{i32 \mid \nu \geq y_0[y_0/i_1]\}$;
        // $n$: own($n_0$), $i$: own($i_1$), $c$: own($c_1$), $r$: own($r_0$)
        // $\preceq$
        // $x_0$: $i32$;  $n$: own($n_0$), $i$: own($i_1$), $c$: own($c_1$), $r$: own($x_0$)
    else
      jump k(c1)
  in
  let i: int = 0;
  let c: int = 0;
  jump b0()
```

6.5 The sum example with references

```
fn sum($n_0$: $i32$; $n$: own($n_0$)) ret k($m_0$: $\{i32 \mid \nu \geq n_0\}$; m: own($m_0$)) =
  // $n_0$: $i32$;  $n$: own($n_0$)
  let i = new(1);
  i := 0;            // $\ldots, i_0$: $\{i32 \mid \nu = 0\}$;  $\ldots, i$: own($i_0$)
  let r = new(1);
  r := 0;            // $\ldots, r_0$: $\{i32 \mid \nu = 0\}$;  $\ldots, r$: own($r_0$)
  letcont loop($i_1$: $i32$, $r_1$: $\{i32 \mid \nu \geq i_1\}$; n: own($n_0$), i: own($i_1$), r: own($r_1$)) =
    // $\ldots, i_1$: $i32$, $r_1$: $\{i32 \mid \nu \geq i_1\}$;  $n$: own($n_0$), $i$: own($i_1$), $r$: own($r_1$)
    let b = new(1);
```

```
  b := *i < *n;    // ..., b₀: {i32 | ν ⇔ i₁ > n₀};  b: own (b₀)
  if b then
    i := *i + 1;   // ..., i₂: {i32 | ν = i₁ + 1};  i: own (i₂)
    r := *i + *r;  // ..., r₂: {i32 | ν = i₂ + r₁};  r: own (r₂)
    jump loop()
      // n₀: i32, i₀: {i32 | ν = 0}, r₀: {i32 | ν = 0}, i₁: i32, r₁: {i32 | ν ≥ i₁}, b₀: {i32 | ν ⇔ i₁ > n₀},
      // b, i₂: {i32 | ν = i₁ + 1}, r₂: {i32 | ν = i₂ + r₁};
      // n: own (n₀), i: own (i₂), r: own (r₂)
      // ⪯
      // i₁: i32, r₁: {i32 | ν ≥ i₁};
      // n: own (n₀), i: own (i₁), r: own (r₁)
  else
    jump k(r)
jump loop()
```

6.6 Non-copy type

```
fn hipa(r_0:Point × Point, r: own(r_0)) ret k(m_0:i32; m: own(m_0))
  // r_0:Point × Point; r:own(r_0)
  let a: Point = *r.0 in  // ..., r_1:↯_2 × Point; r:own(r_1), a:own(r_0.0)
  let b: i32 = *a.0 in    // ..., b_0:{i32 | ν = r_0.0}; ..., b:own(b_0)
  let c: i32 = *r.1.1 in  // ..., c_0:{i32 | ν = r_1.1.1}; ..., c:own(c_0)
  let d: i32 = *b + *d in // ..., d_0:{i32 | ν = b_0 + d_0}; ..., d:own(d_0)
  jump k(d)
```