GOAL:

- Introduce a friend to the PL.
- Incrementally develop concepts
  - dependency order of a feature
    ⇕
    impl. order " the .
- Document whole feature-set [I keep forgetting]

MIND MAP

BY SYNTAX
- patterns
- expressions
- statements : const (7), class (8)
- modules
- source files

BY RUNTIME (AOT vs. RT)
- const / non-const (const)
- non-const immut / non-const mut (&)

BY TYPING
- Atomic Types : S_, U_, F_
- Aggregate Types : { ... }, enum
- Function Types : (T) → U
- Template Types : <T Any, u 532> → <T Any>
  could be <v Any> for value (14)
- Pointers : ^T
- Mutables : &T —— 6
- Classes : sets of types (8)

class C (T Any) <...> {
  type req T.x = F32 ;
  val chk (v T) = (v.x == 144)
} ;

chk imposes a condition on ALL values of a type.

replace with 'type' & 'val' ('test' 'check')

e.g.: ArrayLike class : T = <...> → {...}

MISC:

- '.' for static + dyn lookup
  - .(?) for indexing (1)
  - T.id ⇒ type of field (6) cf enum
- ':' for typing ops:
  - (2) '::' for eq,
  - (3) '<:' for ~~subtype~~ sup? fewer items in struct?

U8 <: U16 :: conv sub → sup

for compound, match eq size + fieldwise <:

{x U8} <: {x U16}

{y U8} <: {x U16}
name x

{x U8} <: {x U8, y U8}

NOTE: order independent
only top-level fields considered 'embed'

{x U8} <: ? No : tuple != struct

Types feature

req T >: {
  x Sca, y Sca
}

For templates, —— (11) (explicit call?)

① check spec indep:
  if specced constraints bad, dfn error.
② At tcall [explicit/implicit], B
  check class instanceship
  At vcall,
  check type instanceship
  Mismatch? Call error.
③ to emit, just emit per monomorphic permutation

(12) : enable <> (explicit) args
(7) : const stmt
(13) : enable [] (implicit) args
(14) : enable (v Cls) implicit template param

Templates ⇓ class OK + in class OK

④ — reintroduce → as *()., lambda " => "
  ↳ clearer typing ::
  T.x != T→x
— let &x = 42;
  let (&x, y) = (42, 52)
⑥

⑤

* Dynamic bdd allocation (WIP: allocators, library?)

① Fixed bound

'dim' stmt creates a REGION with opt (non-const) max-size in B.

e.g. dim positions Vec3 (n) @ r? / how do we know? / type of region arg?

Each region is ...

① disposed of on return (dropped) UNLESS RETURNED
② contained in heap or another region
   no "region"
③ used to allocate with 'new' stackwise

From CFGs, compiler can:

① trace each 'new' to a parent 'dim' / else error!
② check / compute bound for dim to compare

② Dynamic bound

Cost freq data & use to set resize steps < total upper bound

TICKETS

Things to implement.

① —— (14) in Black ink

may be a value!

vtcall

tcall ⇒ template call
vcall ⇒ runtime call / value call

(15) shorter structs

* IDEA:
  - use an allocator for numerous objects
  - tie allocators to CFG frames

① Change ':' to '.' for module lookup (no → possible)
  - in typer, T_MODULE known during postfix visit
  - early overloading: when overload determinable from scopes alone (no unknowns)
  - else, '.' assumes LHS struct

⑥ Typespec . ID ⇒ type of field (and →)
  - enum Opt <T Any> {
      some T ;
      none ; # ⇒ unit, ; vs. , with groupings
    } ;
    Opt.some value ⇝ cast expr syntactically — special enum case!
    enum Color <T Any> {
      rgba {r, g, b, a T} ;
      hsva {h, s, v, a T} ;
    } ;
    Color < T >.rgba.r :: T
    (enum field type; struct field type)

WHAT OF MONOMORPHIC
- structs — (-1)
- enums — (-2)

② '::' type eq binop
③ ':<' and '>:' type binops
④ '→' ptr. (like C/C++)
⑤ Lambda ' |pattern => body'
⑥ '&' in |pattern items
  + &T typing
  + &T typechecking
    - ^ : &T → ^&T
    - ^ : T → ^T (still take ptr to lit, not 'const' !!)
    - set : ^&T = T
    - call : &T → T but T ↛ &T
      ^&T → ^T but ^T ↛ ^&T
      &(^T) → (^T) but (^T) ↛ &(^T)
      (some case; not confusing &/x accids. switched.)
    - dot : (&T).x → &Y where T :< {x F32, y F32} (? switch?)
    - struct/enum: error if field is mut: mutability set by owner
      ↳ warning?

⑦ const stmt
⑧ class stmt
⑨ type
⑩ val
⑪ <> tcall defns
⑫ <> tcalls
⑬ [] tcall defns + tcalls (explicit but inferred)

IMP! scope chaining / multiple

vs.

⑭ (v Cls) implicit templates ⟶ ✗ : use `v Any<Cls>`

Typer Plan

- Types are just objects — not directly '==' comparable
- apply_rule ⟶ {Ok / Fail / Deferred} — diff 'rules' for diff syntactical cases
  - * rules ⟶ constraints : sup/sub, is_kind, convert_to (vcall), cast_to (cast<T>(v)), has_field
  - * diff kinds of constraints ⇒ diff kinds of types
  - * 'hypothesis' : free vars only
    - possible kinds bitset (so kinds Po2)
    - super & sub record vectors (sup + constraint/rule)
    - fields
  - * on apply_rule,
    - ① Rule (type) ⟶ Constraints ...
    - ② for each constraint c,
      if c is known (not dep on free var)
      Ok / Fail
      else
      update FreeVar with constraints
      - any conflicts? FAIL
      - any changes? DEFER after updating
      - else OK (confirm, no new info)

↳ constraint ⟶ rule
rule ⟶ loc / why

Constraint
- building block of rules
- tracks parent rule

KINDS:
① Unit
② Int
③ Float
④ Struct
⑤ Enum
⑥ VFunc
⑦ TFunc : maps FreeVars to sub parts
⑧ Any : emits an enum, cast from any type [x]. kind == type. kind. enum
⑨ Module
⑩ FreeVar : just a free variable, to be subbed later (cf metavar) (cf TFunc)
⑪ Type
- reqd for Module field, tfunc

t: Type ⇔ t is a type

* classes are not types.
  - ? — consider A<Cls> in specs?
  - ? — consider no TID / vid
    ∴ x a<Class>
    ✗
    x Class (not allowed)
    so
    x class ?
    so
    ✱ — using so 'a' in usable name-space?
    else type-a ✗

⑫ Class

RULES:
① ModRule (M, (string, T)[]) ↳ spans?
  kind (M, MODULE)
  for n, t in T
  M has field (n, t) ⟶ unnamed field ⇕ tuple

② Let Rule
③ Def Rule
⋮
of Typer ! ?

* IDEA
- change 'class' to 'template'
- add OO-style classes : class, {using, dim, def}
- add 'using' statement to import symbols.

value classes
Ints
Floats
Strings
Tuples
Arrays
Structs
Enums
Objects

* uses '->' access operator for methods
* use '.*' for ptr access
  e.g. a.*b.*c

(15) Change TIO/VIO semantics : don't need separate !

(16) Add 'All' class : set of all types

T All ⇒ T is any type

(17) Add 'objects' as a new obj kind

- support methods, overriding

* add '=>' operator for method access
* add ∅-arg methods

:: x => y

could be useful, is v. explicit

* note x.y still used for data members

(18) * using y T ⇒ import / embed

* using a class includes methods w/ overloading / shadowing

```
class B {
    hi (x T) → T {
        x
    }
};

class C {
    using b B;
    member y X;
    method hi (x T) → T {
        hello (x)
    };
    abstract hello (x T) → T;
};
```

→ MUST be subtype of shadowed base

(19) Multi-ID variation field w/ shared fs

* Subclassing , Subtyping , Conversion

① A, B types

A subtype B

⇒ Bs substited by As

e.g. : 132 :< 164

⇒ B from A in call OK

i.e. can pass 132 for 164 w/o losing info

e.g. : {x, y u, ... T[]} :< {x, y T}

⇔ T :< u

i.e. can pass {x, y u, ... T[]}

as {x, y T}

⇔ T :< u

e.g. I32 :< Float64

i.e can pass i32 as f64

* A sub B

⇔ if f(B) → x, then

f(A) → x' [s.t. x' sub x]

* Rules :

① Int & Float :

$I8 :< I16 :< I32 :< I64$

$U8 :< U16 :< U32 :< U64$

$F32 :< F64$

② Structs :

all names unique

$\{ a_1 \dots a_m \; T_1 \dots T_m \}$ ⋮ $\{ b_1 \dots b_n \; u_1 \dots u_n \}$

$\{ x \; I32 \}$ ⋮ $\{ x, y \; I64 \}$

⇑

① $m \geq n$ (sub bigger)

② $\forall \; 1 \leq i \leq m,$

$\exists \; 1 \leq j \leq n$ s.t.

$name(b_j) = name(a_i)$

$T_i :< U_i$

③ Enum

Each enum accepts a struct !

Follow struct typing rules.

④ VFunc

$A \dots \to B \;\; :< \;\; C \dots \to D$

⇑ (A

$len(A) = len(C)$ ∧ $\forall i \;\; A_i :< C_i$

⑤ TFunc

see VFunc, Class

* ⊕ Subclass

$C :< D$

$\Leftrightarrow \; \forall \; T \in C, \; T \in D$

$\Leftrightarrow \; C \subseteq D$

for traits, not really

* consider sep. subclass / subset op ?