1. Конкретный синтаксис.

⟨*ident*⟩ ::= ⟨*nondigit*⟩
  | ⟨*nondigit*⟩ ⟨*ident_rest*⟩

⟨*ident_rest*⟩ ::= ⟨*alphanum*⟩ | ⟨*alphanum*⟩ ⟨*ident_rest*⟩

⟨*cmp-op*⟩ ::= '==' | '/= ' | '<=' | '<' | '>' | '>='

⟨*md-op*⟩ ::= '*' | '/'

⟨*pm-op*⟩ ::= '+' | '.-'

⟨*unop*⟩ ::= '-' | '!'

⟨*num-lit*⟩ ::= number

⟨*bool-lit*⟩ ::= 'T' | 'F'

⟨*lit*⟩ ::= ⟨*num-lit*⟩ | ⟨*bool-lit*⟩

⟨*constr-name*⟩ ::= ⟨*capital-alpha*⟩ | ⟨*capital-alpha*⟩ ⟨*ident*⟩

⟨*pat*⟩ ::= ⟨*var*⟩
  | ⟨*lit*⟩
  | ⟨*constr-name*⟩
  | ⟨*constr-name*⟩ ⟨*pats*⟩
  | '(' ⟨*pat*⟩ ')'

⟨*pats*⟩ ::= ⟨*pat*⟩ | ⟨*pat*⟩ ⟨*pats*⟩

⟨*expr*⟩ ::= ⟨*binop-expr*⟩

⟨*binop-expr*⟩ ::= ⟨*or-expr*⟩

⟨*or-expr*⟩ ::= ⟨*and-expr*⟩ '||' ⟨*or-expr*⟩ | ⟨*and-expr*⟩

⟨*and-expr*⟩ ::= ⟨*cmp-expr*⟩ '&&' ⟨*and-expr*⟩ | ⟨*cmp-expr*⟩

⟨*cmp-expr*⟩ ::= ⟨*pm-expr*⟩ ⟨*cmp-op*⟩ ⟨*pm-expr*⟩ | ⟨*pm-expr*⟩

⟨*pm-expr*⟩ ::= ⟨*pm-expr*⟩ ⟨*pm-op*⟩ ⟨*md-expr*⟩ | ⟨*md-expr*⟩

⟨*md-expr*⟩ ::= ⟨*md-expr*⟩ ⟨*md-op*⟩ ⟨*pow-expr*⟩ | ⟨*pow-expr*⟩

⟨*pow-expr*⟩ ::= ⟨*unop-expr*⟩ '^' ⟨*pow-expr*⟩ | ⟨*unop-expr*⟩

⟨*unop-expr*⟩ ::= ⟨*un-op*⟩ ⟨*lit*⟩
  | ⟨*un-op*⟩ ⟨*var*⟩
  | ⟨*un-op*⟩ '(' ⟨*atom-expr*⟩ ')'

⟨*atom-expr*⟩ ::= ⟨*ident*⟩
  | ⟨*lit*⟩
  | ⟨*app*⟩
  | 'if' ⟨*expr*⟩ 'then' ⟨*expr*⟩ 'else' ⟨*expr*⟩
  | 'let' var '=' ⟨*expr*⟩ 'in' ⟨*expr*⟩
  | 'case' '(' ⟨*expr*⟩ ')' 'of' '{' ⟨*case-body*⟩ '}'
  | ⟨*expr*⟩ ⟨*binop*⟩ ⟨*expr*⟩
  | ⟨*unop*⟩ ⟨*expr*⟩
  | '(' ⟨*expr*⟩ ')'

⟨*app*⟩ ::= ⟨*ident*⟩ ‘ ’ ⟨*app-args*⟩
| ‘(’ ⟨*expr*⟩ ‘)’ ‘ ’ ⟨*app-args*⟩

⟨*app-args*⟩ ::= ⟨*app-arg*⟩ | ⟨*app-arg*⟩ ‘ ’ ⟨*app-args*⟩

⟨*app-arg*⟩ ::= ⟨*lit*⟩ | ⟨*ident*⟩ | ‘(’ ⟨*expr*⟩ ‘)’

⟨*case-entry*⟩ ::= ‘|’ ⟨*pat*⟩ ‘->’ ⟨*expr*⟩

⟨*case-body*⟩ ::= ⟨*case-entry*⟩ | ⟨*case-entry*⟩ ⟨*case-body*⟩

⟨*bind*⟩ ::= ⟨*ident*⟩ (⟨*arg*⟩ | ε) ‘=’ ⟨*expr*⟩

⟨*arg*⟩ ::= ⟨*ident*⟩ | ⟨*ident*⟩ ‘ ’ ⟨*arg*⟩

⟨*data*⟩ ::= ‘data’ ⟨*constr-name*⟩ ‘=’ ⟨*data-body*⟩

⟨*data-body*⟩ ::= ⟨*data-entry*⟩ | ⟨*data-entry*⟩ ‘|’ ⟨*data-body*⟩

⟨*data-entry*⟩ ::= ⟨*constr-name*⟩ | ⟨*constr-name*⟩ ⟨*pats*⟩

⟨*type-def*⟩ ::= ‘:’ ident ‘:’ ⟨*type*⟩

⟨*type*⟩ ::= ‘Int’
| ’Bool’
| ⟨*type*⟩ ’->’ ⟨*type*⟩

⟨*decl*⟩ ::= ⟨*bind*⟩ | ⟨*data*⟩ | ⟨*type-def*⟩

⟨*decls*⟩ ::= ⟨*decl*⟩ ’;’ | ⟨*decl*⟩ ’;’ ⟨*decls*⟩

Введены две новые синтаксические категории: объявление типа функций (`type-def`) и объявление типов данных (`data`).

Каждой функции должно предшествовать объявление её типа.

Примеры:

1. Объявление типы данных.

```
data EitherBI = Left Bool | Right Int;
```

2. Объявление типа функции.

```
:f: Int -> Int;
```

Добавлена синтаксическая конструкция `case`.

```
:f: EitherBI -> Bool
f x = case (x) of {
    | Right b -> b
    | Left  i -> T
};
```

И паттерн-матчинг.

```
:f: EitherBI -> Bool
f (Right b) = b;
f (Left  i) = T;
```

Правила проверки типов тогда сводятся к проверке непротиворечивости объявления шаблона и его использовании.

Правила типизации шаблонов.

$$\frac{x \in \text{Int}}{\text{PatLit x : Int}}$$

$$\frac{x \in \text{Bool}}{\text{PatBool x : Bool}}$$

$$\frac{}{\Delta \vdash \text{PatVar x} : A}$$

$$\frac{\Delta D : \{A_i\}_{i \in 1 \ldots n} \to A \quad \Delta \vdash \{a_i\}_{i \in 1 \ldots n} : \{A_i\}_{i \in 1 \ldots n}}{\Delta \text{PatConstr D } \{a_i\}_{i \in 1 \ldots n} : A}$$

Тогда case будет типизироваться как

$$\frac{\Gamma, \Delta \vdash x : A \quad \Delta \vdash \{p_i\}_{i \in 1 \ldots n} : A \quad \Gamma, \Delta \vdash \{e_i\}_{i \in 1 \ldots n} : B}{\Gamma, \Delta \vdash \text{case } x \text{ of } \{p_i \to e_i\}_{i \in 1 \ldots n} : B}$$

2. Абстрактный синтаксис

Представлен в виде АСД.

```
type Ast = [Decl]

data Decl = BindDecl Bind | DataDecl Data | TypeDecl Name Type

data Data = Data Name [Constr]

data Constr = Constr Name [Name]

data Type = Arrow Type Type | TInt | TBool | TData Name

data Bind = Bind Name [Pat] Expr

data Pat = PatVar Name | PatCtr Name [Pat] | PatLit Lit

data Expr = Var Name
          | Lit Lit
          | App Expr Expr
          | If Expr Expr Expr
          | Case Expr [(Pat, Expr)]
          | Let Name Expr Expr
          | UnOp UnOperator Expr
          | BinOp BinOperator Expr Expr

data Lit = ILit Integer | BLit Bool
```