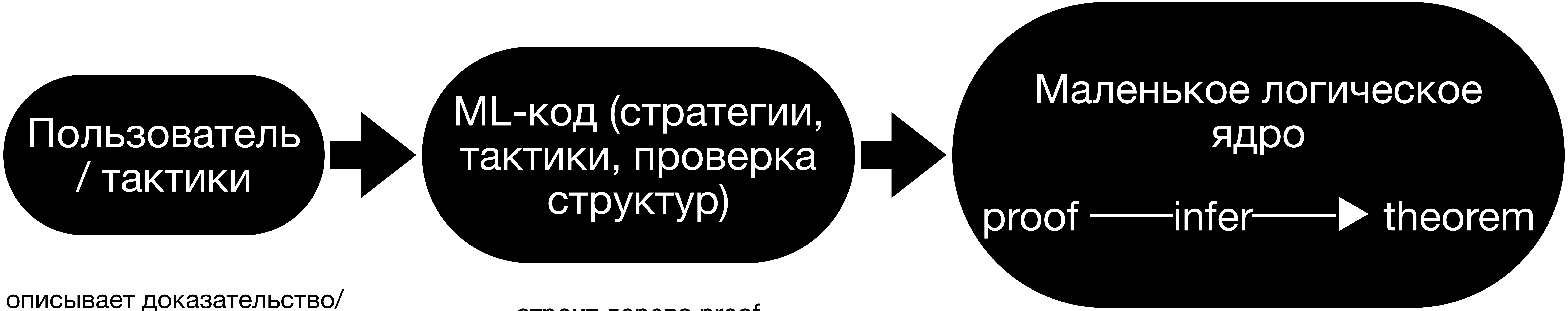
ML как метаязык для доказательств

Мини-ядро натурального вывода на Standard ML

Фатахов Тимур, Буслама Анис, гр. 5030102/20201

Идея LCF: маленькое ядро и ML как метаязык



описывает доказательство/
тактику на ML

строит дерево proof

- реализует правила вывода
- создаёт только корректные theorem

Объектный язык: формулы логики высказываний

```
datatype prop =
        Var of string
      | And of prop * prop
      | Imp of prop * prop
```

Var "A" → пропозициональная переменная A

And (p, q) → формула p∧q

Imp (p, q) → формула p→q

ML выступает как **метаязык**, в котором мы описываем синтаксис формул

Представление теорем: контекст ⊢ заключение

```
datatype theorem = Theorem of prop list * prop
```

Theorem (Γ, ϕ) читается как: Γ ⊢ ϕ

Γ — список допущений: prop list

ϕ — заключение: prop

Theorem ([Var "A", Var "B"], And (Var "A", Var “B"))

отображается как: A, B ⊢ A ∧ B

В реальных LCF-подобных системах конструктор Theorem обычно скрыт в модуле (абстрактный тип thm)

Доказательства как дерево: тип proof

Определяем синтаксис выводов как алгебраический тип данных:

```
datatype proof =
        Assume of prop
      | AndIntro of proof * proof
      | AndElimLeft of proof
      | AndElimRight of proof
      | ImpIntro of prop * proof
      | ImpElim of proof * proof
```

Тип proof описывает структуру доказательства, но не проверяет его корректность — это делает функция infer.

Функция infer: логическое ядро системы

Все правила вывода реализуются одной функцией:

fun infer : proof -> theorem

Ключевые идеи:

- Вход: синтаксическое дерево proof.

- Выход: объект theorem, то есть пара (контекст, заключение).

- При некорректном применении правила (не та формула, не тот тип вывода) функция выбрасывает исключение Fail.

Именно infer реализует правила вывода и является доверенной (trusted) частью системы.

Правила для конъюнкции и допущения

- \- Assume p ⇒ теорема [p] ⊢ p (аксиома-допущение).
- \- AndIntro: рекурсивно считаем теоремы для левой и правой части, объединяем контексты и строим Γ ⊢ (p ∧ q).
- \- AndElimLeft/AndElimRight: проверяем, что заключение действительно конъюнкция;
- если да — берём нужную часть, контекст не меняется;
- если нет — выбрасываем Fail и отвергаем некорректное применение правила.

```
fun infer (Assume p) = Theorem ([p], p)
  | infer (AndIntro (left_pf, right_pf)) =
      let
          val Theorem (ctx_l, left_prop) = infer left_pf
          val Theorem (ctx_r, right_prop) = infer right_pf
          val new_ctx = union_ctx ctx_l ctx_r
      in
          Theorem (new_ctx, And (left_prop, right_prop))
      end
  | infer (AndElimLeft pf) =
      let
          val Theorem (ctx, concl) = infer pf
      in
          case concl of
              And (p, _) => Theorem (ctx, p)
            | _ => raise Fail "ожидалась конъюнкция для левого устранения"
      end
  | infer (AndElimRight pf) =
      let
          val Theorem (ctx, concl) = infer pf
      in
          case concl of
              And (_, q) => Theorem (ctx, q)
            | _ => raise Fail "ожидалась конъюнкция для правого устранения"
      end
```

Ядро само сверяет форму заключения с ожидаемой: нельзя «притвориться», что у нас есть конъюнкция, если её нет.

Введение импликации и Modus Ponens

- ImpIntro:

  - Строим теорему для поддоказательства pf.

  - Проверяем, что допущение assumption действительно присутствует в контексте.

  - Снимаем его один раз (remove_once) и получаем теорему Γ \ {assumption} ⊢ (assumption → conclusion).

  - Если допущения нет — Fail: «attempted to discharge missing assumption».

- ImpElim (Modus Ponens):

  - Первая теорема должна иметь вид premise → result, иначе Fail.

  - Вторая теорема должна доказывать ровно premise, иначе Fail "modus ponens mismatch".

  - При успехе объединяем контексты и получаем Γ ⊢ result.

```
| infer (ImpIntro (assumption, pf)) =
      let
          val Theorem (ctx, conclusion) = infer pf
          val remaining_ctx =
              if List.exists (fn p => p = assumption) ctx
              then remove_once assumption ctx
              else raise Fail "attempted to discharge missing assumption"
      in
          Theorem (remaining_ctx, Imp (assumption, conclusion))
      end
 | infer (ImpElim (imp_pf, arg_pf)) =
      let
          val Theorem (ctx_imp, imp_concl) = infer imp_pf
          val (premise, result) =
              case imp_concl of
                  Imp pair => pair
                | _ => raise Fail "ожидалась импликация для Modus Ponens"
          val Theorem (ctx_arg, arg_prop) = infer arg_pf
          val _ =
              if arg_prop = premise
              then ()
              else raise Fail "modus ponens mismatch"
          val combined_ctx = union_ctx ctx_imp ctx_arg
      in
          Theorem (combined_ctx, result)
      end
```

Так мы гарантируем, что импликации и Modus Ponens используются строго по их логическим правилам.

Целевая теорема: (A ∧ B) → (B ∧ A)

Пошаговый смысл proof_swap:

1. Цель: доказать (A ∧ B) → (B ∧ A).

2. Допускаем A ∧ B как гипотезу (Assume conjunction).

3. Из A ∧ B получаем A (AndElimLeft hyp).

4. Из той же гипотезы A ∧ B получаем B (AndElimRight hyp).

5. Собираем B ∧ A с помощью AndIntro (proj_b, proj_a).

6. Вводим импликацию ImpIntro (conjunction, …), снимая допущение A ∧ B и получая теорему без контекста.

```
val a = Var "A"
val b = Var "B"
val conjunction = And (a, b)
val swapped = And (b, a)
val target_theorem = Imp (conjunction, swapped)

val proof_swap =
    ImpIntro (conjunction,
      let
          val hyp = Assume conjunction
          val proj_a = AndElimLeft hyp
          val proj_b = AndElimRight hyp
          val reordered = AndIntro (proj_b, proj_a)
      in
          reordered
      end)
```

Это дерево proof_swap кодирует стандартное доказательство коммутативности конъюнкции.

Проверка доказательства и текстовый вывод

Ключевые моменты:

- derived имеет вид Theorem ([], Imp (And (Var "A", Var "B"), And (Var "B", Var "A"))).

- Пустой контекст [] означает, что теорема доказана без неснятых допущений.

- Функция prop_to_string печатает формулу в удобном виде с символами ∧ и →.

```
val derived = infer proof_swap

val _ =
    case derived of
        Theorem ([], conclusion) =>
            print ("Теорема доказана: " ^ prop_to_string conclusion ^ "\n")
      | th =>
            print ("Доказательство содержит незакрытые допущения: "
                   ^ theorem_to_string th ^ "\n")
```

Связь с реальными теорем-доказателями

Наше мини-ядро на ML — это учебная модель подхода, который используется в реальных proof assistant’ах семейства LCF.

- LCF (Logic for Computable Functions)
  - Одна из первых систем, где появилась идея:
    - абстрактный тип теорем;
    - небольшой набор примитивных правил вывода;
    - тактики и стратегии на ML поверх этого ядра.
  - ML (Meta Language) изначально разрабатывался именно как язык описания тактик.

Итоги

- Мы использовали Standard ML как метаязык для описания:
  - формул логики высказываний (datatype prop),
  - структуры доказательства (datatype proof),
  - результата проверки (datatype theorem).
- Функция infer : proof -> theorem играет роль логического ядра:
  - реализует правила для $\wedge$ и $\rightarrow$;
  - проверяет корректность применения правил;
  - формирует только те теоремы, которые получены из допустимых шагов вывода.
- Конкретный пример:
  - мы задали дерево proof_swap для теоремы $(A \wedge B) \rightarrow (B \wedge A)$;
  - запустили проверку в SML/NJ;
  - получили подтверждение: Теорема доказана: $((A \wedge B) \rightarrow (B \wedge A))$.