Синтаксис и семантика

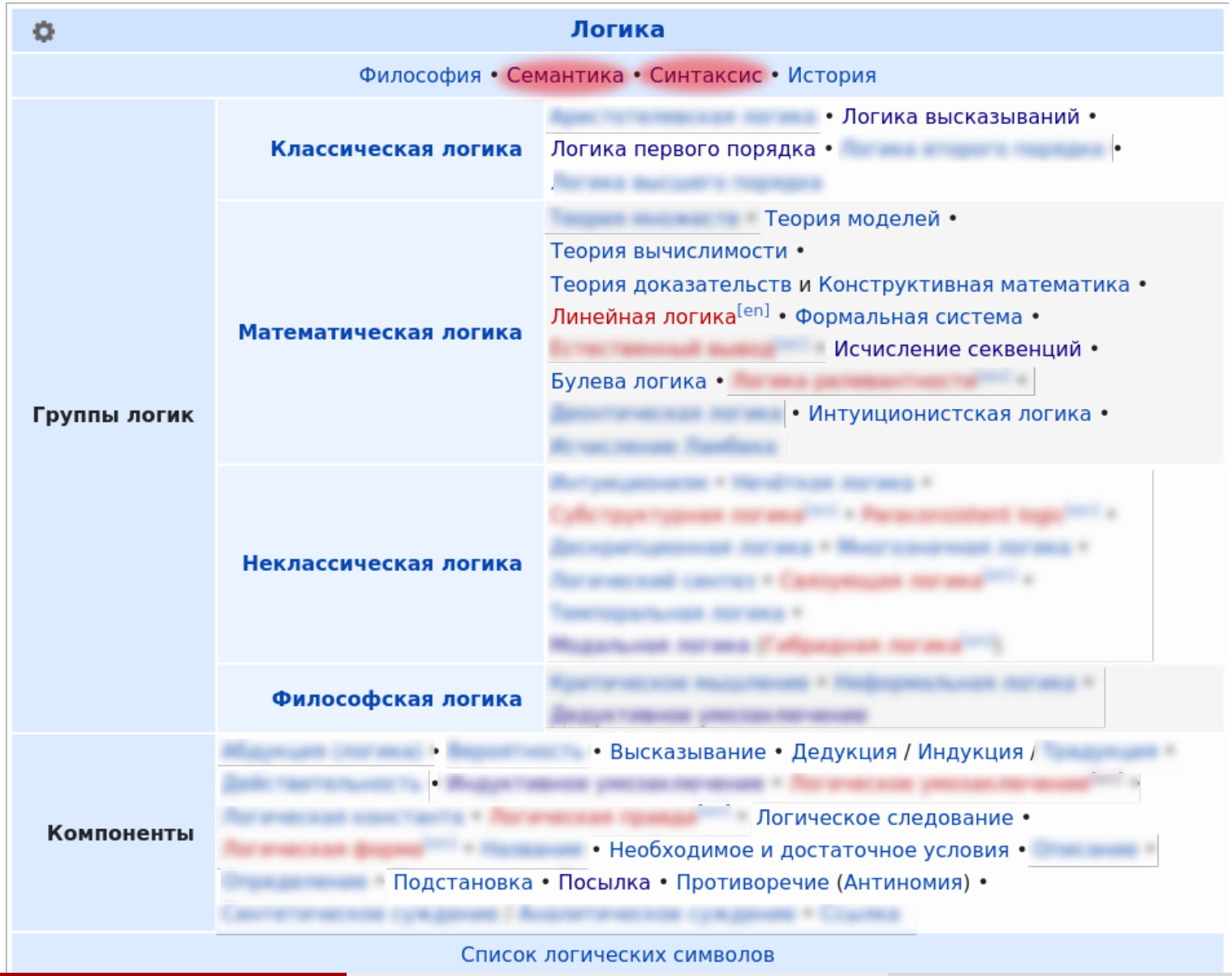
Дмитрий Халанский

18 февраля 2021 г.

Синтаксис и семантика

Зачем нужны логические формулы?
Чтобы выражать текстом функции $\mathbb{B}^n \to \mathbb{B}$.
Функции — это то, что мы *имеем в виду*, когда работаем с формулами.
Формулы же — это просто их *представления*.
Иными словами, формула — это синтаксическая сущность, выражающая функцию над логическими величинами, семантическую сущность.

Википедия о логике



Логика

Философия • Семантика • Синтаксис • История

Группы логик

Классическая логика: [illegible] • Логика высказываний • Логика первого порядка • [illegible] • [illegible]

Математическая логика: [illegible] • Теория моделей • Теория вычислимости • Теория доказательств и Конструктивная математика • Линейная логика[en] • Формальная система • [illegible] • Исчисление секвенций • Булева логика • [illegible] • [illegible] • Интуиционистская логика • [illegible]

Неклассическая логика: [illegible]

Философская логика: [illegible]

Компоненты: [illegible] • Высказывание • Дедукция / Индукция / [illegible] • Логическое следование • Необходимое и достаточное условия • [illegible] • Подстановка • Посылка • Противоречие (Антиномия) • [illegible]

Список логических символов

Связь между синтаксисом и семантикой: полнота

Полнота (Completeness) — Свойство синтаксической сущности, которое показывает, что данным синтаксисом можно покрыть всю семантическую область.
Пример: полная система связок позволяет через заданный ею синтаксис выразить любую логическую формулу.

Связь между синтаксисом и семантикой: корректность

Корректность (Soundness) — Свойство синтаксической сущности, которое показывает, что данный синтаксис не покрывает ничего помимо заданной семантической сущности.
Пример: просто типизированное лямбда-исчисление позволяет представить только завершающиеся программы (но не все, так что оно не является полным).
Контрпример[1]:

```
Giraffe[] giraffes = new[] { new Giraffe() };
Animal[] animals = giraffes; // This is legal!
animals[0] = new Tiger(); // crashes at runtime
```

Система типов должна гарантировать, что хорошо протипизированная программа не вылетит с ошибкой типизации.

[1] https://stackoverflow.com/a/23968075, "Is C# type system sound and decidable?"

Где встречаются полнота и корректность

- В логике.
- В статическом анализе. Например, если кто-то заявляет, что написал анализатор, который находит утечки памяти по исходному коду, то этот анализ *полный*, если

Где встречаются полнота и корректность

- В логике.
- В статическом анализе. Например, если кто-то заявляет, что написал анализатор, который находит утечки памяти по исходному коду, то этот анализ *полный*, если находит все утечки, и *корректный*, если

Где встречаются полнота и корректность

- В логике.
- В статическом анализе. Например, если кто-то заявляет, что написал анализатор, который находит утечки памяти по исходному коду, то этот анализ *полный*, если находит все утечки, и *корректный*, если он не жалуется там, где нет утечек. Любители статического анализа считают, что термин “корректность” путает (интуитивно, анализ корректный, если не пропускает ошибок), и поэтому произносят по-английски, “*soundness*”.
- В системах типов, потому что типы — разновидность статического анализа. Любители типов не видят проблем с термином “корректность”.

Суть логики

Формальная логика изучает связь между синтаксисом и семантикой: как (автоматизируемые) манипуляции синтаксическими конструкциями позволяют что-то узнать о предметной области.
Примеры таких связей:

- Нахождение ДНФ формулы не меняет логическую интерпретацию её как функции над логическими величинами.
- Если есть логические функции f и g, которые можно записать в виде формул P и Q соответственно, то можно получить функцию, которая возвращает `true`, когда f и g разом возвращают `true`, как интерпретацию формулы $P \wedge Q$.
- Система типов может запретить компилироваться программам, которые подают строку туда, где требуется число.

Что такое интерпретация?

См. лекцию 1, слайд “Оценки переменных формул” раздела “Истинностное значение формул”.
Интерпретация — это способ сопоставить синтаксис семантике.
Интерпретация (смысл) формулы p обозначается как $[\![p]\!]$.
Рассмотрим пропозициональную формулу $p \to q \to (p \wedge q) \vee a$. Она может обозначать не только функцию $\mathbb{B}^n \to \mathbb{B}$, но и что вздумается.
Например, ничто не запрещает задать такие значения истинностных связок:

$[\![x \to y]\!]$	$[\![y]\!]^{[\![x]\!]}$
$[\![x \wedge y]\!]$	$[\![x]\!] \cdot [\![y]\!]$
$[\![x \vee y]\!]$	$[\![x]\!] + [\![y]\!]$
$[\![\neg y]\!]$	$\max(1 - [\![y]\!], 0)$

Есть ли у этого польза? Без разницы: это просто *можно* сделать.

Соответствие Карри-Говарда

Введём определение `data Empty` (да, именно так, без конструкторов). Зададим интерпретацию формул как типов:

$[\![x]\!]$	x как типовая переменная
$[\![x \to y]\!]$	$[\![y]\!]$ `->` $[\![x]\!]$
$[\![x \wedge y]\!]$	`(`$[\![x]\!]$`,` $[\![y]\!]$`)`
$[\![x \vee y]\!]$	`Either` $[\![x]\!]$ $[\![y]\!]$
$[\![\neg y]\!]$	$[\![y]\!]$ `-> Empty`

$p \to q \to (p \wedge q) \vee a$ оценивается в `p -> q -> Either (p, q) a`.

Теорема (пока без доказательства): если тип населён тотальной функцией, то формула, интерпретирующаяся как этот тип, истинностно интерпретируется как тавтология.

Отступление: где синтаксис, а где семантика?

Предыдущий слайд по-разному подчёркивает “тип”, а то и двумя линиями сразу. Почему?

В одном контексте тип — синтаксическая конструкция, ограничивающая множество допустимых программ; в другом — тип является множеством значений, результатом интерпретации формул.

Можно эту цепочку провести дальше. Можно сказать, что компилятор осуществляет интерпретацию программ на C++ как машинного кода, а машинный код интерпретируется компьютером в конкретное исполнение.

Обычно, правда, говорят иначе: компилятор транслирует программу в машинный код *с сохранением семантики*, то есть

$$[\![\text{“}\texttt{int main() \{\}}\text{”}]\!] = [\![\mathrm{gcc}(\text{“}\texttt{int main() \{\}}\text{”})]\!],$$

где слева интерпретируется программа, а справа — машинный код.