Автоматное программирование

Дмитрий Халанский

11 марта 2021 г.

1. Программы и автоматы

Теория формальных языков полезная

В Стенфорде провели опрос среди программистов, которые выпустились за пять лет до этого, на тему самых полезных в реальной жизни курсов.
Базовые предметы вроде “Программирования” лидировали, но “Теория Автоматов” занимала странно высокую позицию. (В три раза больше баллов, чем “Искусственный Интеллект”).
`https://www.ics.uci.edu/~goodrich/teach/cs162/notes/intro.pdf`

Чем полезна теория формальных языков?

Согласно курсу Стенфорда:

- Крепкое понимание регулярных выражений;
- Модельное описание протоколов: техника *model checking* позволяет проверить корректность несложных программ перебором всех возможных состояний и проверки, всегда ли выполняются какие-то предположения;
- Контекстно-свободные грамматики (дальше по курсу) описывают, как парсятся $\pm$ все современные языки программирования;
- Машины Тьюринга позволяют легче увидеть, что возможно и что невозможно вычислить на компьютере.

Программы как автоматы

Почти теорема: любая процедура такая, что

- стек вызовов из неё ограничен по глубине константой и
- переменные, доступные для чтения в ней (и во всех транзитивно вызываемых ей функциях), хранят конечный объём данных,

представима как конечный автомат.

Почти доказательство почти теоремы: запустим программу исполняться. Закроем глаза. В случайный момент остановим её. Откроем глаза и посмотрим на состояние стека, на то, какая строчка исполняется, и состояние переменных. Этого достаточно, чтобы понять происходящее, но все эти сведения занимают конечный объём информации.

Автоматы как программы: пример

```
def A(lst):
    state = initStateA
    for e in lst:
        state = stepA(state, e)
    return state.isTerminal

def B(lst):
    state = initStateB
    for e in lst:
        state = stepB(state, e)
    return state.isTerminal

def AAndB(lst):
    state = (initStateA, initStateB)
    for e in lst:
        state = (stepA(state[0], e), stepB(state[1], e))
    return state[0].isTerminal and state[1].isTerminal
```

Автоматное программирование (тема этой практики)

Предложение: давайте писать программы так, будто мы их оттранслировали из конечных автоматов. Согласно “почти теореме”, это не менее выразительно, чем широкий класс полезных программ.
Достоинства: легко анализировать, прозрачный процесс написания, *понятно, с чего начинать*.
Суть:

- Вводим конечное множество состояний.
- Обрабатываем поток событий (принимаем запросы, идём по списку и т. п.) и попутно обновляем состояния. Иногда делаем что-то полезное по пути.

Другие применения конечных автоматов в программировании

С их помощью компиляторы могут избавляться от использования стека в случае хвостовой рекурсии, в том числе для поддержки асинхронного программирования (см. trampoline + continuation-passing style).