СОДЕРЖАНИЕ

ВВЕДЕНИЕ

1 Анализ средств статического вывода типов стековых языков программирования

Использование стека данных в качестве единого хранилища порождает несколько возможных проблем типизации. Первая – проблема несовпадение типов, когда на вход некоторой команде с вершины стека поступают данные неправильного типа. Вторая – проблема исчерпания стека, когда команда при выполнении пытается взять данные с пустого стека. Данные ошибки приводят к аварийным завершениям программ и тем самым снижают их надежность. В то же время от подобных ошибок можно защититься, если в момент компиляции программы статически определять состояния стека во все будущие моменты времени исполнения и в случае нахождения ошибки блокировать компиляцию программы как небезопасной. Просчет возможных состояний стека для программы на стековом языке называется выводом ее типа.

Для стековых языков тип для некоторой команды определяется через эффект, который оказывается эта команда на конфигурацию стека данных при выполнении. В свою очередь конфигурация стека состоит из размера стека (сколько данных на нем лежит) и «формы» стека (данных каких типов на нем лежат). Задача вывода типов для стековых языков сводится к нахождению типа для всей программы – как программа поменяет конфигурацию стека при выполнении.

В общем случае выводом типов в языках программирования называется алгоритм, который позволяет установить неуказанный явно тип некоторого выражения в тексте программы по ограничениям окружения, где это выражение встречается. Разные языки обладают разными возможностями вывода в зависимости от размера области программы, которая используется для вывода типа. Некоторые языки допускают простые выводы, основанные на анализе только самого выражения, для которого выводится тип. Например, вывод типов в язык программирования Java ограничивается автоматическим определением типа переменной при ее объявлении по выражению, которое инициализирует данную переменную. Другие языки идут дальше и подзволяют выводить типы по совокупности выражений и совокупности переменных, задействованных в этих выражениях в рамках функций. К примеру, Rust, который позволяет выводить тип переменной из контекста выражений, где эта переменная используется – так, если функция возвращает целое 32-битное число, то для гипотетической переменной, из которой делается возврат значения и которая может быть объявлена где угодно внутри такой функции, выводится соответственно тип целого 32-битного числа (то есть тип вывелся исходя из как минимум двух выражений, где переменная была задействована). Наконец существуют языки, вывод типов в которых осуществляется в рамках всей единицы трансляции – зачастую это языки с функциональной

парадигмой, в таком случае типы для всей программы можно не указывать вовсе, так как все они будут выведены неявно автоматически.

Для стековых языков алгоритмы автоматического вывода типов для всей программы (то есть, последний случай, рассмотренный в предыдущем абзаце) актуальны, так как зачастую программисту затруднительно самому вычислить тип программы, что и порождает ошибки работы со стеком данных.

Для вывода типов в функциональных языках используется алгоритм Хиндли-Милнера. Вкратце смысл алгоритма при его запуске над несколькими выражениями, для которых надо вывести типы, сводится к двум шагам: сначала вычисляются ограничения, которые задаются формой выражений (например, одна переменная равна другой или переменная участвует в сложении, а потому должна быть числом), ограничения составляют систему уравнений, для которой вторым шагом ищется решение – такой набор типов, при подстановке которых выражений останутся непротиворечивыми. Если набор найден, то типы выведены успешно, иначе – ошибка типизации.

2 Анализ устройства байткода и модели исполнения WebAssembly

ЗАКЛЮЧЕНИЕ

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ