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 позволяют 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 при компиляции.

Для стековых языков алгоритмы вывода типов всей программы (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 алгоритм при запуске над несколькими выражениями 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 складываемой), ограничения составляют систему уравнений, для которой вторым шагом ищется решение – такой набор типов, при подстановке которых, выражения останутся непротиворечивыми. Если набор найден, то типы выведены успешно, иначе – ошибка типизации.

Согласно идее, предложенной Кристофером Диггинсом, для стековых языков использование алгоритма Хиндли-Милнера также возможно, но с поправкой на row-полиморфизм. Row-полиморфизм – полиморфизм по размеру строки («row»), состоянию стека данных, над которым выполняются команды. Все команды row-полиморфны и оперируют всем стеком сразу, который состоит из хвоста (типовой вектор, обозначается большими латинскими буквами) и единичных значений, лежащих на вершине стека (обозначаются маленькими латинскими буквами либо названиями типов, например «a» и «Bool»). Для цитат используется запись со скобками, показывающая отложенность описанных вычислений (пример, конфигурация стека, на котором лежит цитата «A (A -> B)»).

Диггинс демонстрирует возможность статического вывода типов на примере своего экспериментального стекового языка Cat. Для команд встроенных команд приводятся типы, показаны на листинге 1.

Листинг 1 —

```
// Применение цитаты, которая приводит конфигурацию S к
конфигурации R, к стеку S
apply   : (S (S -> R) -> R)
// Формирование цитаты, путем захвата значения с вершины стека
quote   : (S a -> S (R -> R a))
// Композиция двух цитат
compose : (S (B -> C) (A -> B) -> S (A -> C))
// Дублирование значения на вершине стека
dup     : (S a -> S a a)
// Удаление значения с вершины стека
```

Продолжение листинга 1

```
pop     : (S a -> S)
// Перестановка двух значений с вершины стека
swap    : (S a b -> S b a)
// Примитив ветвления, выбирающий то или иное значения со
стека в зависимости от значения Bool
cond    : (S Bool a a -> S a)
// Примитив циклических операций, выполняющий тело (цитата на
вершине) пока верно условие (цитата под вершиной)
while   : ((S -> R Bool) (R -> S) S -> S)
```

Алгоритм вывода типа заключается в последовательном «сцеплении» типов команд, образующих программу. В своем техническом отчете Диггинс приводит текстовое описание алгоритма и пример, в котором происходит вывод типа для программы «[dup] apply». В результате применения алгоритма получается следующая цепочка типов: «A a -> A a (A a -> A a a) -> A a a» (примечание, в отчете приводятся только начальная и конечная конфигурации стека, но аналогичным образом возможно построить и промежуточные конфигурации). Как видно, цитата и команда «apply» взаимно уничтожаются, из-за чего результирующий тип эквивалентен применению одиночной команды «dup» с точностью до промежуточных конфигураций.

Предложенный алгоритм имеет ограничение, заключающееся в невозможности вычислять рекурсивные типы. Из-за этого ограничения нельзя вычислить тип для программ, которые имеют рекурсивный поток управления. Также нельзя вычислить тип программ, предполагающих рекурсию за счет использования конструкции «dup apply» (объяснить это можно так: «apply» предполагает, что на стеке лежит некоторая цитата, которая переводит стек под собой из начального состояния в некоторое другое, но при этом «dup» говорит, что сама цитата также является частью этого начального состояния под собой). Такие ограничения можно снять, если расширить систему типов дополнительными конструкциями.

Таким образом для стековых языков возможно статически вывести форму и размер стека, и в случае наличия ошибок заранее их заметить. Признаком исчерпания стека служит наличие значений на вершине стека в начальной конфигурации программы – то есть программа ожидает, что на стеке до ее запуска уже что-то лежит, хотя это не так. Признак несовпадения типов выражается невозможностью найти решение системы уравнений в процессе очередной сцепки типов команд в программе.

2 Анализ устройства байткода и модели исполнения WebAssembly

ЗАКЛЮЧЕНИЕ

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ