
Эдсгер Вайб Дейкстра

"О вреде оператора Go To"

Технологический университет, Эйндховен,
Нидерланды

Edsger Wybe Dijkstra

"Go To Statement Considered Harmful"

Communications of the ACM, Vol. 11,
No. 3, March 1968, pp. 147-148.

За многие годы я утвердился во мнении о том, что квалификация программистов - функция, обратно зависящая от частоты появления операторов `go to` в их программах. Позже я открыл, почему оператор `go to` производит такой пагубный эффект, и я убежден в том, что оператор `go to` должен быть отменен в языках программирования "высокого уровня" (т.е. отовсюду, кроме, возможно, простого машинного кода). В то время я не счел это открытие слишком важным. Теперь же я отправляю свои соображения для публикации, потому что меня подтолкнула к этому развернувшаяся сейчас дискуссия на эту тему.

Мое первое замечание состоит в том, что хотя деятельность программиста заканчивается с конструированием корректной программы, процесс, который выполняется под управлением этой программы, является истинной сущностью этой деятельности; этот процесс должен завершиться с заданным эффектом; поведение этого процесса должно удовлетворять заданным спецификациям. Хотя, когда программа уже изготовлена, "изготовление" соответствующего процесса поручается машине.

Мое второе замечание - то, что наши интеллектуальные силы в большей степени связаны со статическими отношениями, и что наши способности представлять процессы, развивающиеся во времени, развиты относительно плохо. Исходя из этого, мы должны делать (как мудрые программисты, осознающие свои ограничения) все возможное, чтобы сократить концептуальную пропасть между статической программой и динамическим процессом, чтобы сделать соответствие между программой (разворачивающейся в пространстве текста) и процессом (разворачивающимся во времени) настолько очевидным, насколько это возможно.

Давайте теперь рассмотрим, как мы можем охарактеризовать состояние процесса. (Вы можете представить себе этот вопрос очень конкретно: предположите, что процесс, рассматриваемый как последовательность действий во времени, остановлен после произвольного действия, какие данные мы должны иметь, чтобы точно определить порядок, в котором следует повторить процесс, чтобы дойти до той же точки?) Если текст программы представляет собой просто сцепление, скажем, операторов присваивания (в рамках данной дискуссии будем рассматривать их как описания одиночных действий), то достаточно в тексте программы указать точку между двумя последовательными описаниями действий [two successive action descriptions]. (В отсутствии оператора `go to` я могу позволить себе синтаксическую неоднозначность в последних трех словах предыдущего предложения: если мы разбираем их как "последовательные (описания действий)", то мы имеем в виду последовательность в тексте; если же мы разбираем их как "описания (последовательных действий)", то мы имеем в виду последовательность во времени.) Давайте назовем указатель на соответствующее место в тексте "текстуальным индексом".

Когда мы вводим условное предложение (`if` *B* `then` *A*), предложение альтернатив (`if` *B* `then` *A1* `else` *A2*), предложения выбора, введенные C. A. R. Hoare (`case` [i] `of` (*A1, A2,..., An*)), или условные выражения J. McCarthy (*B1 -> E1, B2 -> E2, ..., Bn -> En*), сохраняется тот факт, что состояние процесса продолжает характеризоваться единственным текстуальным индексом.

Как только мы включаем в наш язык процедуры, мы должны признать, что единственного текстуального индекса уже недостаточно. В случае, если текстуальный индекс указывает внутрь тела процедуры, динамическое состояние может быть охарактеризовано только если там также дано, с каким вызовом процедуры он связан. С введением процедур мы можем характеризовать состояние процесса при помощи последовательности текстуальных индексов, длина этой последовательности должна быть равна динамической глубине процедурных вызовов.

Теперь давайте рассмотрим предложения повторения (как `while` *B* `repeat` *A* или `repeat` *A* `until` *B*). С позиций логики эти предложения избыточны, потому что мы можем выразить повторения при помощи рекурсивных процедур. Из соображений реалистичности я не хочу их исключать: с одной стороны, предложения повторений могут быть весьма удобно реализованы на современном ограниченном оборудовании, с другой стороны, модель мышления, известная как "индукция", делает нас хорошо подготовленными для интеллектуального восприятия процессов, порождаемых предложениями повторения. С введением предложений повторения текстуальных индексов уже недостаточно для описания динамического состояния процесса. С каждым входом в предложения повторения, однако, мы можем связать, так называемый, "динамический" индекс, исправно подсчитывающий порядковый номер текущего повторения. Если предложения повторения (также как вызовы процедур) могут применяться с вложением, мы находим, что теперь развитие процесса может быть однозначно охарактеризовано последовательностью (смешанной) текстуальных и/или динамических индексов.

Суть в том, что значения этих индексов не управляются программистом; они генерируются (то ли при написании программы, то ли при динамическом выполнении процесса) независимо от того, желает он этого или нет. Они обеспечивают независимые координаты, в которых описывается состояние процесса.

Почему нам нужны эти независимые координаты? Причина в том - и это представляется неотъемлемым для последовательных процессов - что мы можем интерпретировать значение переменной только с учетом состояния процесса. Если мы хотим подсчитать число *n*, скажем, людей в комнате, изначально пустой, мы можем достичь этого путем увеличения *n* на единицу всякий раз, когда мы видим, что кто-то вошел в комнату. В промежуточный момент, когда мы увидели кого-то входящего в комнату, но еще не выполнили последующего увеличения *n*. Его значение равно числу людей в комнате минус один!

Разнузданное применение оператора `go to` имеет прямым следствием то, что становится ужасно трудно найти осмысленный набор координат, в которых описывается состояние процесса. Обычно люди принимают во внимание также и значения некоторых избранных переменных, но это не вопрос, потому что состояния процесса зависит, как понимать эти значения! С оператором `go to` можно, конечно, все еще описывать состояние процесса однозначно при помощи счетчика числа действий, выполненных от старта программы (т.е., некоторой разновидности нормализованных часов). Трудность в том, что такие координаты, хотя они и уникальные, просто бесполезные. С такими координатами система становится чрезвычайно сложной, сталкиваясь с необходимостью определения всех тех точек состояния, в которых, скажем, *n* равно числу людей в комнате минус один!

Оператор `go to` сам по себе просто слишком примитивен; он создает слишком сильное побуждение внести путаницу в программу. Рассмотренные предложения могут оцениваться и приветствоваться как сдерживающие его применение. Я не заявляю, что упомянутые предложения являются исчерпывающими в том смысле, что они удовлетворяют любым требованиям, но какие бы предложения не приводились (например, предложение прекращения), они должны удовлетворять тому требованию, чтобы при них могла бы поддерживаться независимая от программиста система координат для описания процесса полезным и осуществимым способом.

Трудно закончить эти заметки без благодарностей. Могу ли я судить о том, кто оказал влияние на мои рассуждения? Совершенно очевидно, что не обошлось без влияния Peter Landin и Christopher Strachey. Наконец, я хочу отметить (я помню это совершенно точно), как Heinz Zemanek на встрече "pre-ALGOL" в начале 1959 г. в Копенгагене предельно ясно выразил свои сомнения в том, должен ли оператор `go to` трактоваться как такой же синтаксический базис, как оператор присваивания. До некоторой степени я виню себя за то, что не имел тогда представления о значимости его замечания.

Замечания о нежелательности оператора `go to` далеко не новы. Я помню, что читал четкие рекомендации ограничивать использование оператора `go to` аварийным выходом, но не помню точно, где, вероятно, это было сделано C. A. R. Hoare. В [1, Sec. 3.2.1.] Wirth и Hoare вместе делают замечание в том же направлении, мотивируя конструкцию выбора: "Как и условия, она отражает динамическую структуру программы более ясно, чем оператор `go to` и переключатели и она устраняет необходимость введения в программу большого числа меток."

В [2] Guiseppe Jacopini, кажется, доказывает (логически) ненужность оператора `go to`. Упражнения по более или менее механическому переводу произвольной схемы алгоритма в схему без переходов, однако, не рекомендуются. При этом не ожидается, что результирующая схема будет более понятной, чем исходная.

Ссылки:

1. Wirth, Niklaus, and Hoare C. A. R. A contribution to the development of ALGOL. Comm. ACM 9 (June 1966), 413-432.
2. Bohm, Corrado, and Jacopini Guiseppe. Flow diagrams, Turing machines and languages with only two formation rules. Comm. ACM 9 (May 1966), 366-371.
