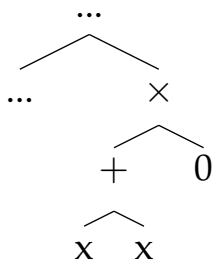

Рис. .35: Пример нежизнеспособного кода

4.6 Раздувание кода

Многие способы кодирования, описанные здесь, по своей природе имеют переменный размер. С этим связана одна из интересных проблем: с течением времени особи в популяции начинают увеличиваться в размерах. Такое явление принято называть **раздувание** (***bloat***) (или «раздувание кода», или «разрастание кода», выберите наиболее симпатичный для вас вариант[1]). Больше всего эффект раздувания изучен для генетического программирования, в котором он является серьезной проблемой. Раздувшиеся особи долго проходят процедуру оценивания. Занимают много памяти. И что хуже всего, раздувшиеся особи, как правило, находятся *очень далеко* от оптимального решения, которое обычно не настолько и велико. Налицо обманчивая задача.

Ранее исследователи ГП заметили, что большие, раздутые деревья содержат много поддеревьев, которые ничего не делают. Такие поддеревья называют **интронами** (***introns***), как и их аналоги в ДНК. На рис. .35 показан особый тип интрона — **нежизнеспособный код** (***inviable code***). Поддерево (+ x x) бесполезно, т.к. каким бы ни был результат его вычисления, он все равно будет умножен на 0. В ранних работах по ГП полагалось, что интроны являются основным источником трудностей. Это привело к появлению идеи: большинство результатов операции Tweak вредят приспособленности особей. Поэтому чтобы выжить не обязательно нужно себя совершенствовать: достаточно уметь держать голову над водой. Один из способов добиться этого — сделать так, чтобы операции Tweak было сложно повредить особь, даже если это затормозит ее улучшение. Если интронов много, в частности много нежизнеспособного кода, то, увеличив вероятность изменения операцией Tweak областей, занятых интронами, приспособленность не будет меняться, все равно изменения в таких областях не будут иметь силы. Вот так звучала эта идея. Однако она оказалась неверной: устранение возможности для операции Tweak работать в областях с нежизнеспособным кодом не привело к исчезновению раздувания[2].

Таким образом, на данный момент причина раздувания еще не известна. У меня есть своя теория на этот счет, согласно которой основной причиной раздувания являются особи, которым для улучшения нужно изменять все более глубокие уровни дерева, т.к. изменение операторов в таких участках оказывает все меньший эффект на приспособленность потомков. При этом наличие глубоко расположенных операторов хорошо коррелирует с размером дерева. Конечно же, есть и другие предположения.

Как бы то ни было, отсутствие твердого понимания, что же все-таки творится на самом деле, приводит к тому, что существующие методы борьбы с раздуванием эвристичны. Существует три способа сохранить небольшой размер особей:

- **Ограничить** размер особей при работе операции Tweak.
- **Редактировать** особи, удаляя интроны и другие подобные участки.
- **Штрафовать** особей, если их размер становится большим.

Традиционным для генетического программирования считается ограничение на глубину дерева (до 17 уровней). Но в настоящее время существует тенденция налагания штрафа, который уменьша-

[0]Перевод раздела из книги Luke S. Essentials of Metaheuristics. A Set of Undergraduate Lecture Notes. Zeroth Edition. Online Version 0.9. July, 2010 (`http://cs.gmu.edu/~sean/book/metaheuristics/`). Перевел – Юрий Цой, 2010 г. Любые замечания, касающиеся перевода, просьба присылать по адресу yurytsoy@gmail.com
Данный текст доступен по адресу: `http://qai.narod.ru/GA/meta-heuristics_4_6.pdf`

[1] Билл Лэнгдон называет это «выживание жирнейших».

[2] Я написал статью, в которой это показано: Sean Luke, 2000, Code growth is not caused by introns, in DarrellWhitley, editor, Late Breaking Papers at the 2000 Genetic and Evolutionary Computation Conference, pages 228–235.

ет шансы особей на отбор. Это называется **давлением скупости** (***parsimony pressure***)[3]. Наиболее простым способом реализовать давление скупости является включение размера особи в вычисление приспособленности. Например, можно определить припособленность особи как $f = \alpha r - (1-\alpha)s$, где r — *грубая* (или «фактическая») приспособленность особи, а s — размер особи. Данный подход широко известен как **линейное давление скупости** (***linear parsimony pressure***). Основная его проблема заключается в том, что необходимо решить насколько размер особи важнее ее приспособленности. А, как уже обсуждалось ранее, функция приспособленности часто определяется достаточно произвольно. Несмотря на это, линейное давление скупости весьма неплохо работает.

Альтернативный подход: использовать **метод непараметрического давления скупости** (***non-parametric parsimony pressure***), который не рассматривает конкретные значения приспособленности или жирности, а выбирает на основании того, какая особь приспособленнее, а какая – жирнее. Есть много реализаций. Например, в **лексикографическом давлении скупости** (***lexicographic parsimony pressure***) предлагается модификация турнирной селекции, в которой побеждают более приспособленные особи, но в случае ничьей, победителем считается более компактная особь[4].

Алгоритм 63 Лексикографическая турнирная селекция

$P \leftarrow$ популяция
$t \leftarrow$ размер турнира, $t \geq 1$

$Best \leftarrow$ случайная особь, выбранная из P с замещением
for i от 2 до t **do**
  $Next \leftarrow$ случайная особь, выбранная из P с замещением
  if Приспособленность($Next$) $>$ Приспособленность($Best$), или Приспособленность($Next$) $=$ Приспособленность($Best$) и $Next$ имеет меньший размер **then**
    $Best \leftarrow Next$
  end if
end for
return $Best$

Данные алгоритм хорошо работает в окружениях с большим количеством ничьих. Однако это малораспространенный случай. Другим подходом является одновременное движение в сторону улучшения приспособленности и компактности выбранных особей. В **двойном турнире** (***double tournament***) осуществляется турнирная селекция на основе приспособленности. Однако особи в этот турнир попадают не из основной популяции, а по результатам *другой* турнирной селекции, работающей с *компактностью*.

Здесь используются два турнира размером t_1 и t_2. Предполагая, что мы следуем традициям ГП и выбрали $t_1 = 7$, хорошим вариантом для t_2 является 2. Вообще, наилучшее значение приблизительно равно 1.4: вспомним, что для значений t меньших, чем 2, с вероятностью $t - 1.0$ осуществляется турнирный отбор размером $t = 2$, а в противном случае выбирается случайная особь. Разумеется, можно поступить и наоборот: сначала делать отбор по приспособленности, а потом по компактности. Есть множество вариантов для давления скупости, однако приведенные два призваны показать основную идею (и к тому же оба основаны на турнирном отборе!).

[3] Ливью Панаит (*Liviu Panait*) и я сделали работу по сравнительному анализу всех описываемых здесь методов давления скупости и других популярных подходов в статье Sean Luke and Liviu Panait, 2006, A comparison of bloat control methods for genetic programming, *Evolutionary Computation*, 14(3), 309–344.

[4] Лексикографическое давление скупости известно с приблизительно 1994 г., появившись в двух статьях: Conor Ryan, 1994, Pygmies and civil servants, in Kenneth E. Kinnear, Jr., editor, *Advances in Genetic Programming*, chapter 11, pages 243–263, MIT Press (и) Simon Lucas, 1994, Structuring chromosomes for context-free grammar evolution, in *Proceedings of the First IEEE Conference on Evolutionary Computation*, pages 130–135, IEEE.

Алгоритм 64 Двойная турнирная селекция

```
P ← популяция
t_1 ← размер турнира для приспособленности, t_1 ≥ 1
t_2 ← размер турнира для компактности, t_2 ≥ 1

Best ← ТурнирПоКомпактности (P, t_2)
for i от 2 до t_1 do
   Next ← ТурнирПоКомпактности (P, t_2)
   if Приспособленность(Next) > Приспособленность(Best) then
      Best ← Next
   end if
end for
return Best

procedure ТурнирПоКомпактности (P, t_2)
Best ← случайная особь, выбранная из P с замещением
for i от 2 до t_2 do
   Next ← случайная особь, выбранная из P с замещением
   if Next компактнее, чем Best then
      Best ← Next
   end if
end for
return Best
```