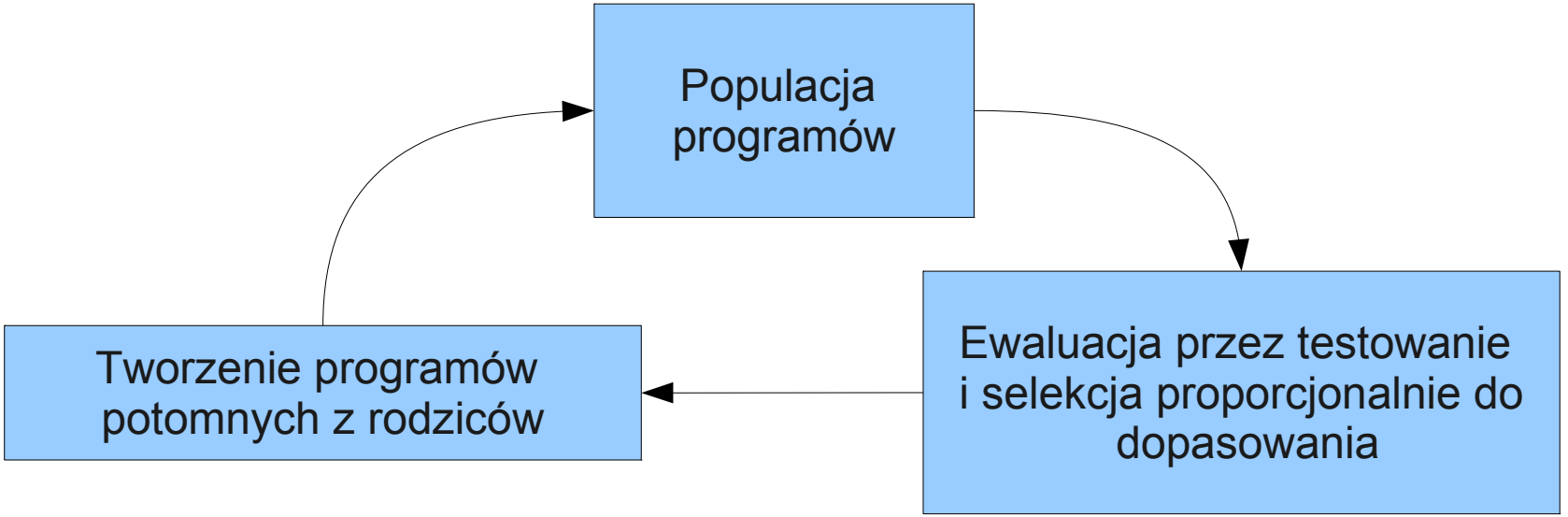
Programowanie genetyczne (ang. genetic programming)

Wstęp

- Spopularyzowane przez Johna Kozę na początku lat 90-tych.
- Polega na zastosowaniu paradygmatu obliczeń ewolucyjnych do generowania programów komputerowych rozwiązujących pewne zadanie.
- Programy są reprezentowane poprzez drzewa składniowe (ang. parse tree).
- Wymaga zaangażowania dużych zasobów obliczeniowych, (rozmiar populacji to tysiące a nawet miliony), ale wraz z upowszechnieniem się komputerów równoległych może być zastosowane do coraz trudniejszych problemów.

Pętla algorytmu



- Należy określić:
  - W jaki sposób reprezentować programy.
  - W jaki sposób wygenerować populację początkowych programów.
  - Jak wykonać krzyżowanie i mutację programów ?
  - W jaki sposób obliczyć dopasowanie programu ?

Reprezentacja oparta na drzewach

- Drzewa pozwalają na uniwersalną reprezentację np. rozważmy:

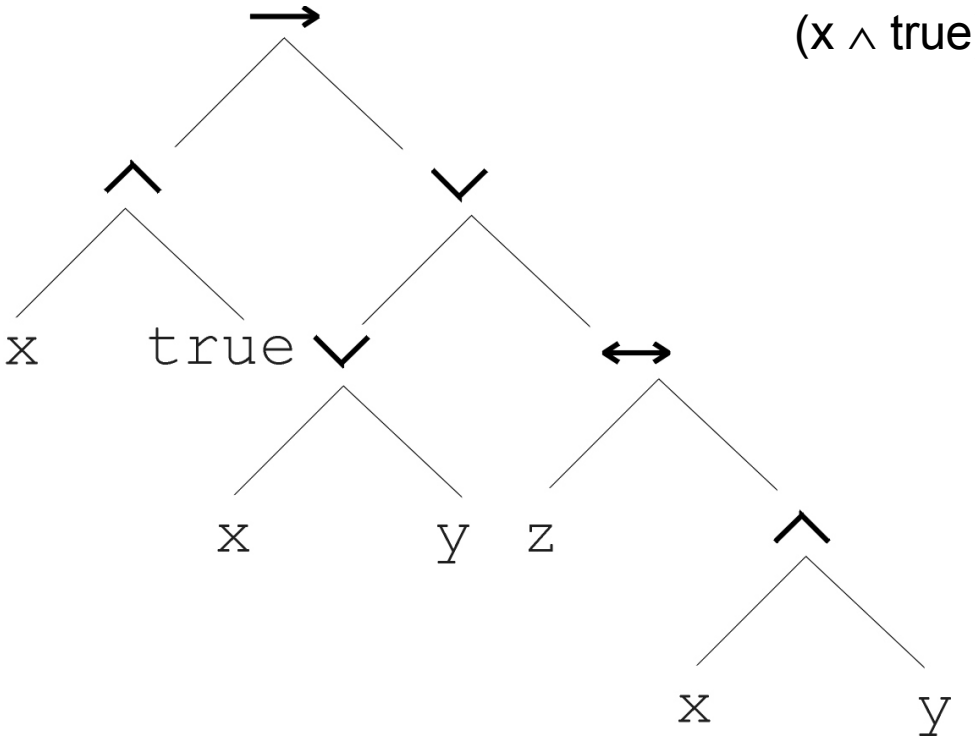
- Wyrażenie arytmetyczne: $2 \cdot \pi + \left((x+3) - \frac{y}{5+1} \right)$

- Wyrażenie logiczne: $(x \wedge \text{true}) \rightarrow ((x \vee y) \vee (z \leftrightarrow (x \wedge y)))$

- Program:

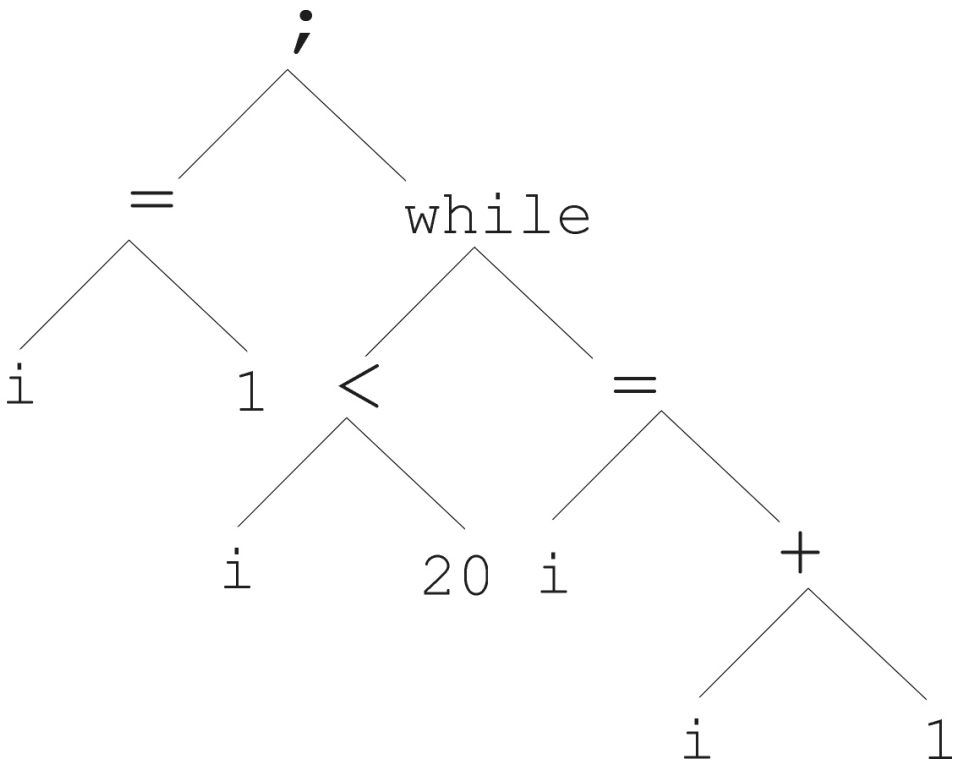
```
i =1;
while (i < 20)
{
        i = i +1
}
```

Reprezentacja dla wyrażenia logicznego



$(x \wedge \text{true}) \rightarrow ((\ x \vee y\ ) \vee (z \leftrightarrow (x \wedge y)))$

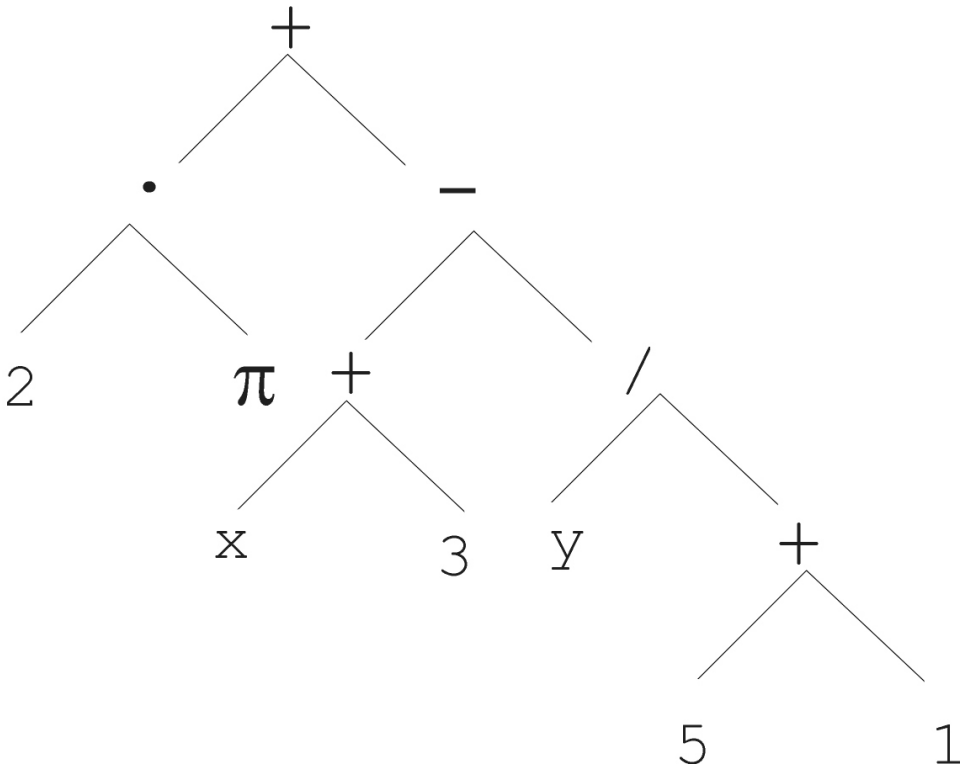
Reprezentacja dla programu




```
i =1;
while (i < 20)
{
        i = i +1
}
```

Bardziej formalnie

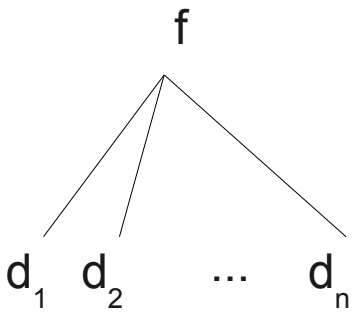
- Które drzewo jest poprawnym drzewem (reprezentuje poprawny program) ? Należy zdefiniować ich składnie.
- Zbiór termów (ang. terminal) - mogą być liśćmi drzewa.
- Zbiór funkcji - mogą być węzłami wewnętrznymi drzewa.
- Dla przykładu z wyrażeniami algebraicznymi zbiór funkcji to: {+,-,/,*} a zbiór termów to $\Re \cup \{x, y\}$



$$2 \cdot \pi + \left((x+3) - \frac{y}{5+1} \right)$$

Własność domknięcia (ang. closure)

1. Drzewo jednoelementowe (składające się z pojedynczego termu) reprezentuje poprawne wyrażenie.
2. Jeżeli f jest funkcją n-argumentową, a drzewa $d_1,d_2,..,d_n$ reprezentują poprawne wyrażenia, to drzewo:



   również reprezentuje poprawne wyrażenie.
3. Innych poprawnych wyrażeń nie ma.

- Mniej formalnie: każda funkcja musi być w stanie akceptować na wejściu dowolny typ danych i wartość, która może być zwrócona przez inną funkcję lub term.
    - Niezbędne w programowaniu genetycznym, ponieważ operatory krzyżowania oraz mutacji prowadzą do losowych zmian w drzewach.

Warunki wystarczalności (ang. sufficiency) oraz uniwersalności (ang. universality)

- Wystarczalność: zbiór termów i funkcji powinien pozwalać na rozwiązanie problemu.
  - np. zbiór funkcji {+,-} nie pozwala na reprezentację wyrażenia arytmetycznego, w którym występuje mnożenie.
  - dla naprawdę trudnych problemów nie jest łatwo ją zapewnić.

- Uniwersalność: dobrze jest dodać funkcje, które nie są niezbędne do rozwiązania zadania, mogą one ułatwić znalezienie rozwiązania.
  - Ale zbyt duża nadmiarowość przeszkadza w znalezieniu rozwiązania.

Inicjalizacja losowa drzewa

- Wybierz losowo funkcję na korzeń drzewa.
- Każda funkcja ma skończoną liczbę argumentów. Dla każdego argumentu funkcji w korzeniu wybierz losowo term lub funkcję.
  - Jeżeli wybrałeś term stanie się liściem drzewa.
  - Jeżeli wybrałeś funkcję wykonaj algorytm inicjalizacji rekurencyjnie.
- Ograniczenie na rekurencję: wysokość drzewa nie może być większa niż D_{max}. Zalecana wartość, to (D_{max}=2...6).
- Trzy możliwości inicjalizacji
  - grow initialization: opisana powyżej
  - full initialization: każda gałąź drzewa ma głębokość D_{max}. Wymaga losowania na poziomie $d<D_{max}$ spośród zbioru funkcji, a na poziomie $d=D_{max}$ spośród termów.
  - ramped half-and-half: połowa populacji jest inicjalizowana metodą grow, a druga połowa metodą full.

Obliczenie dopasowania

- Obliczenie dopasowania wykonywane jest na podstawie pewnej liczby przypadków testowych (ang. fitnes cases).

- Przypadek testowy to zestaw wejść dla programu + wyjście.

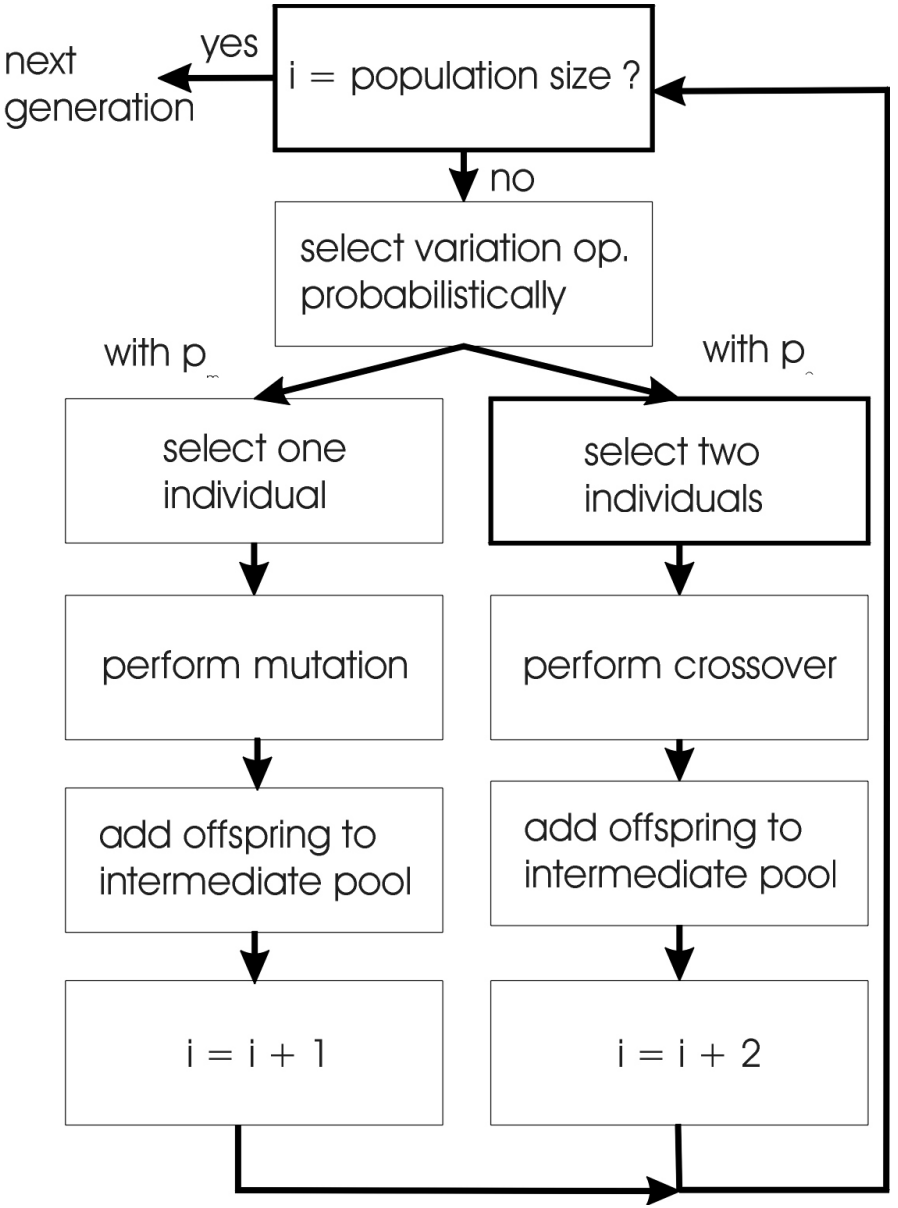
- Liczba wszystkich przypadków testowych jest na ogół nieskończona, dlatego należy wybrać „reprezentatywną” próbkę.

- Dla każdego przypadku testowego możemy rekurencyjne przejrzeć drzewo programu, zaczynając od korzenia, ale wykonując obliczenia w węźle będącym funkcją dopiero jak wykonamy obliczenia w poddrzewach będących argumentami funkcji.

Selekcja i sukcesja

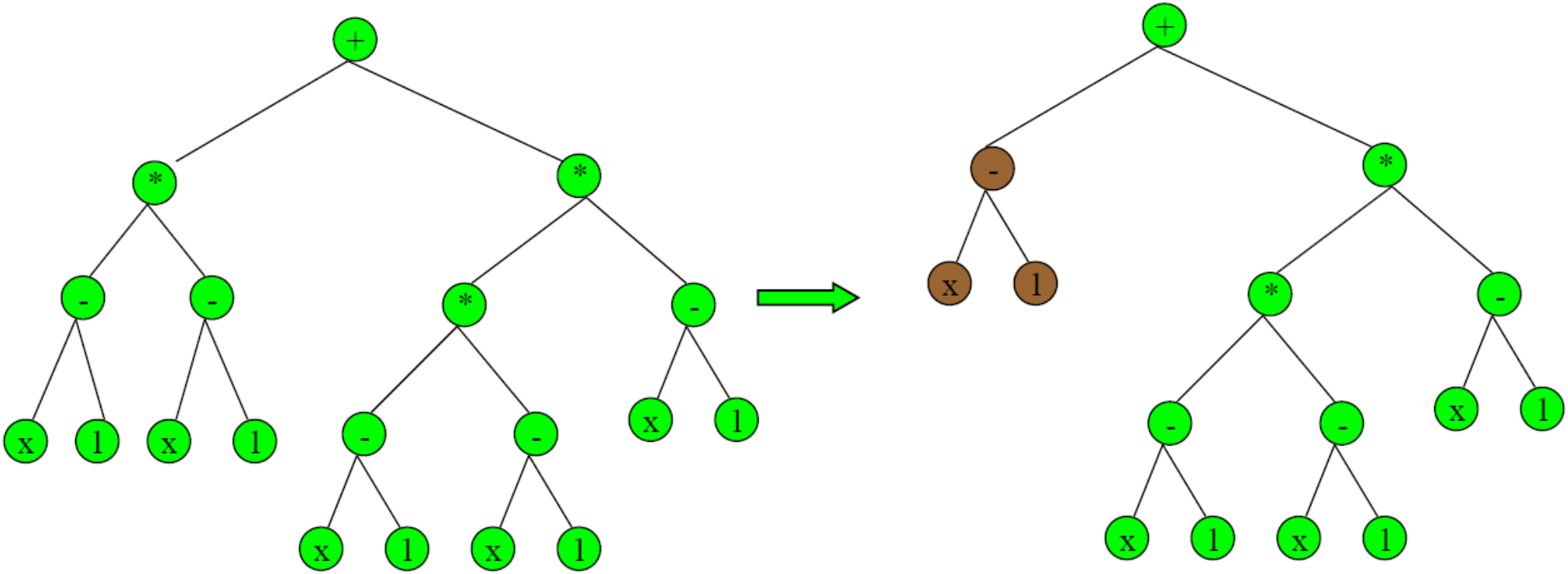
- W programowaniu genetycznym stosuje się generacyjny model sukcesji (podobnie jak w GA), chociaż ostatnio pojawiają się propozycje zbliżone do algorytmu genetycznego stanu stałego.
- Selekcja rodziców często wykorzystuje metodę nazwaną nad-selekcją (over-selection).
  - Posortuje osobnik względem dopasowania i podziel populację na dwie grupy. W pierwszej x% najlepiej dopasowanych i (100-x)% najgorzej dopasowanych.
  - Dla rozmiaru populacji S=1000 x=32%, dla S=8000 x=4%.
  - 80% rodziców wybierz z populacji najlepiej dopasowanych osobników, 20% z populacji gorzej dopasowanych.
- Stosuje się selekcję proporcjonalną (ostatnio również turniejową).

Wybór operatora zmieniającego osobnika



- Osobnik podlega
  - krzyżowaniu z prawdop. p_c.
  - mutacji z prawdop. p_m.
- Osobnik nie podlega nigdy krzyżowaniu i mutacji jednocześnie.
- Zalecane jest bardzo małe p_m w granicach 0.05 (Koza w swojej pierwszej książce zalecał $p_m \approx 0$).

Operacja mutacji



- Najczęściej stosowana operacja mutacji (subtree mutation) polega na zastąpieniu losowo wybranego poddrzewa w drzewie losowo wygenerowanym poddrzewem.

Alternatywne operacje mutacji

- point mutation – wybierz losowo węzeł drzewa i zastąp go innym tego samego typu (term innym termem, a funkcję funkcją o tej samej liczbie argumentów).

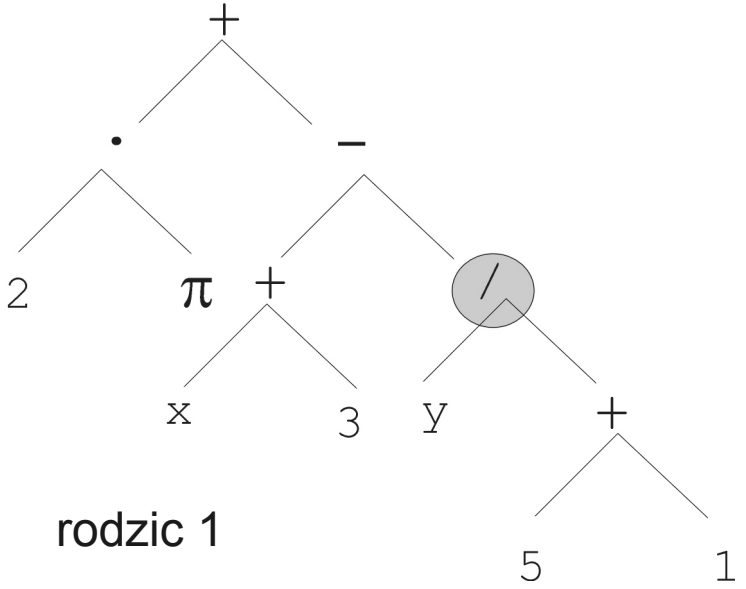
- permutation mutation – zmień kolejność argumentów losowo wybranego węzła wewnętrznego (funkcji).

- collapse subtree mutation – zastąp losowo wybrane poddrzewo losowo wygenerowanym termem.

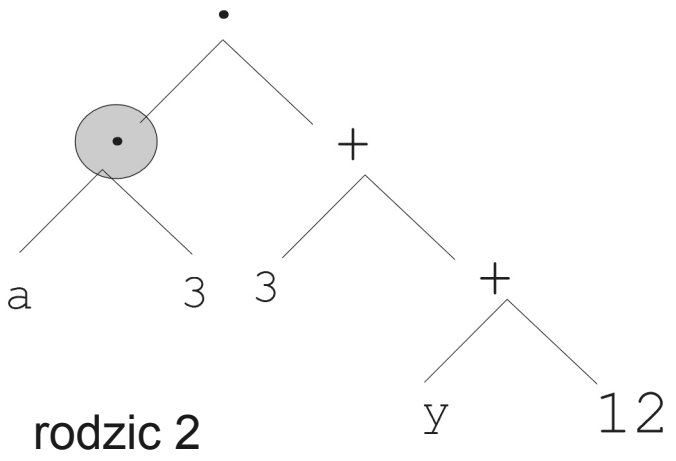
- hoist mutation – losowo wybrane poddrzewo staje się nowym osobnikiem.

- expansion mutation – losowo wybrany term zastąpiony zostaje losowo wybranym poddrzewem.

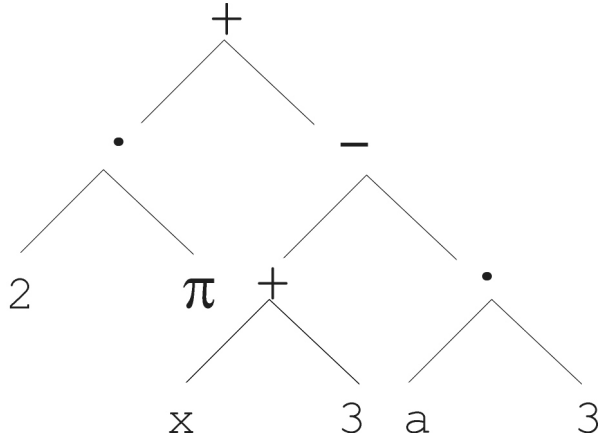
Operacja krzyżowania



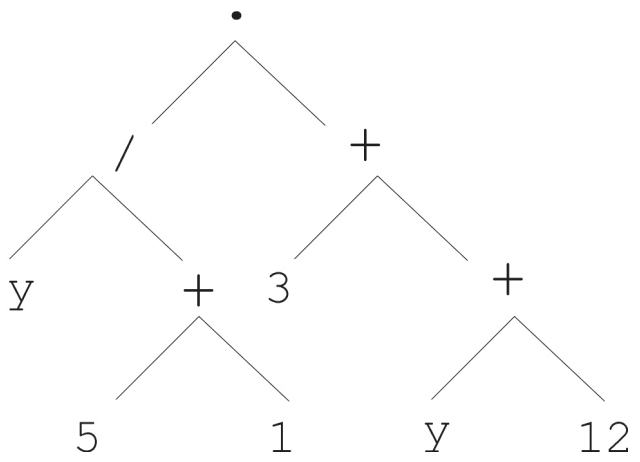
rodzic 1



rodzic 2



potomek 1



potomek 2

- Najczęściej stosowany operator krzyżowania wymienia losowo wybrane poddrzewa pomiędzy osobnikami

Warunek stopu i rozmiar populacji

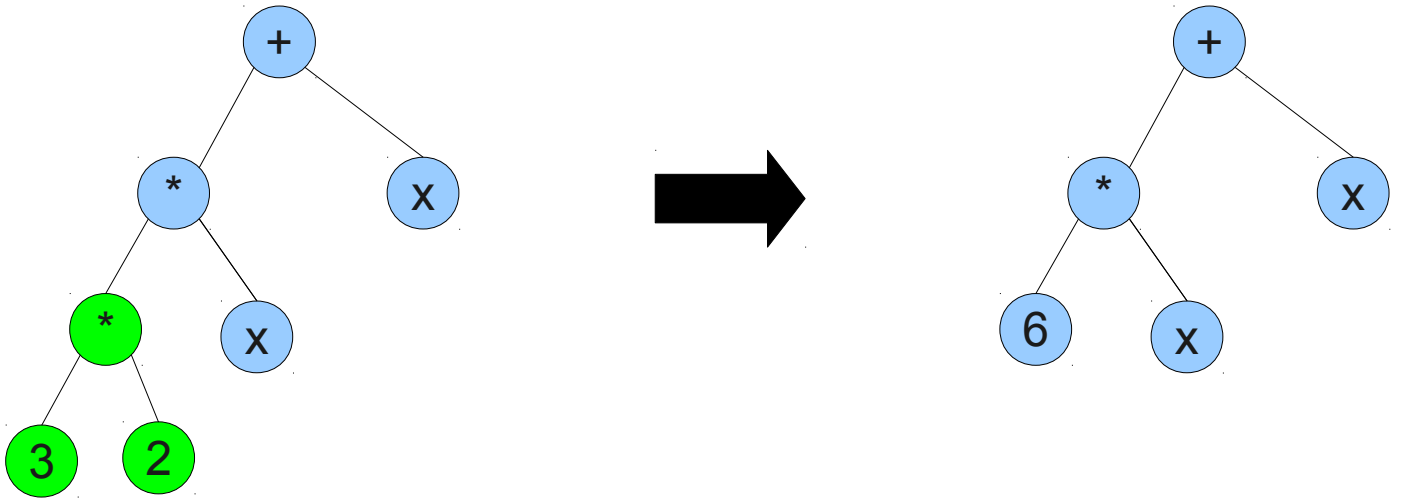
- W społeczności zajmującej się programowaniem genetycznym, przyjęła się praktyka preferowanie bardzo dużych rozmiarów populacji i zatrzymywania algorytmu po kilkudziesięciu (najwyżej kilkuset generacjach).

- Przyjmuje się, że jeżeli ewolucja nie jest w stanie odkryć dobrego programu przez kilkadziesiąt iteracji, to nie będzie w stanie tego zrobić później.

- Słowo bardzo duży rozmiar populacji – oznacza tysiące w przypadku komputerów sekwencyjnych miliony w przypadku komputerów równoległych.
  - Przy czym dla niewielkiej liczby procesorów i systemu ze wspólną pamięcią można łatwo zrównoleglić obliczanie wartości funkcji dopasowania (jest to najbardziej czasochłonny krok ze względu na konieczność przejścia drzewa/uruchomienia programu dla każdego przypadku testowego).
  - Dla systemów składających się z tysięcy procesorów można zastosować model wyspowy (ang. island) zrównoleglenia.

Operator edycji (ang. editing)

- Operator który umożliwia uproszczenie drzew.
- Rekurencyjnie poczynając od korzenia drzewa stosuje zestaw reguł upraszczających, nie zmieniających interpretacji drzewa. Przykład (zastąpienie funkcji operującej na stałych):



Zjawisko rozdęcia kodu (ang. code bloat)

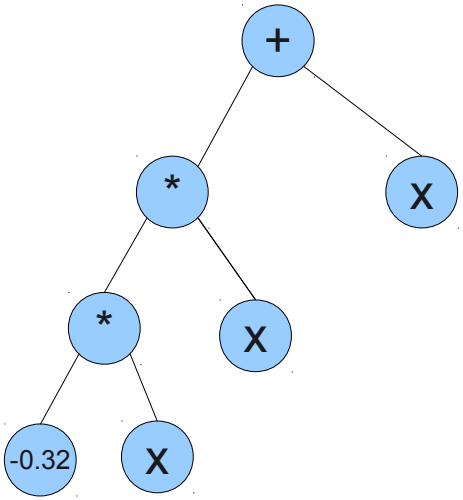
- Bloat = survival of the fattest
- Operatory krzyżowania i mutacji mogą prowadzić do sytuacji w której wysokość potomków jest większa od wysokości rodziców.
- Okazuje się, że przeciętny rozmiar drzewa w populacji zaczyna rosnąć wraz z postępem algorytmu.
- Aby kontrolować rozdęcie należy przedsięwziąć środki zaradcze.
  - Ograniczenie maksymalnej wysokości drzewa, i zakazanie (lub przerwanie) operatorów rekombinacji gdy wysokość potomka przekroczyłaby maksimum. (Prosta technika ale wymaga dodatkowego parametru).
  - Nacisk na oszczędność (ang. parsimony pressure) Powiązanie dopasowania z rozmiarem drzewa, tak że im większy rozmiar drzewa tym mniejsze dopasowanie. (a zatem wprowadzamy karę za duży rozmiar).

Przykład zastosowania: regresja symboliczna

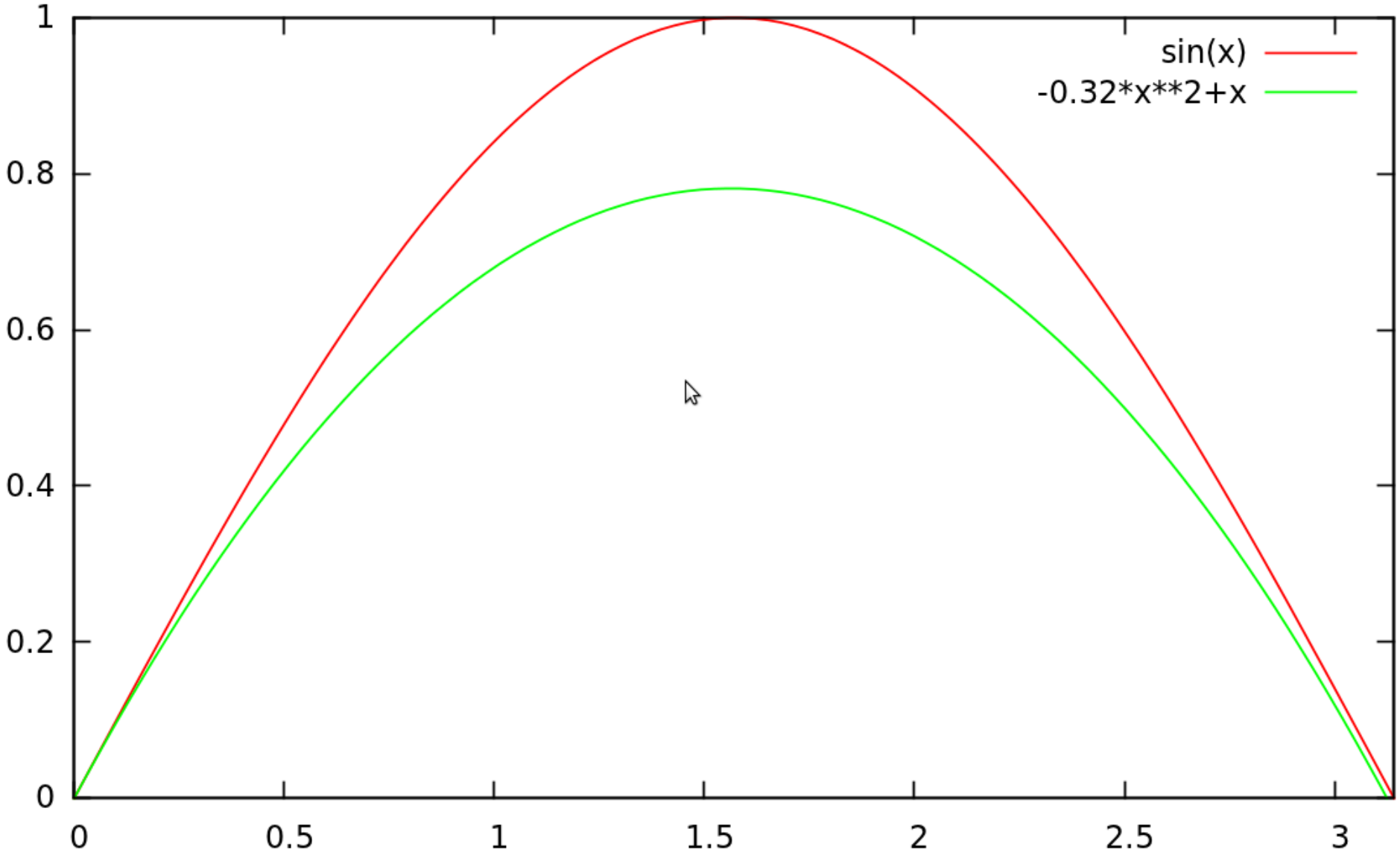
- Dany jest zbiór punktów z R^2: $\{(x_1,y_1),(x_2,y_2), \ldots , (x_n,y_n)\}$. Zgodnie z pierwszą książką Kozy punkty wygenerowano jako $y_i=\sin(x_i)$, a x_i losowo z przedziału $(0,\pi)$
- Znajdź funkcję f(x) taką że $f(x_i)\approx y_i$.
- Zbiór termów: $\Re \cup \{x\}$
- Zbiór funkcji: {+,-,*,/} (wszystkie to funkcje dwuargumentowe).
- Funkcja dopasowania $\sum_{i=1}^{n} (f(x_i)-y_i)^2$
- Algorytm po 29 generacjach znalazł drzewo:

reprezentujące wyrażenie

$$-0.32 * x^2 + x$$



Regresja symboliczna - porównanie



Literatura dodatkowa

- J. R. Koza, Genetic Programming: On the Programming of Computers by Means of Natural Selection, 1993, MIT Press, książka która spopularyzowała temat, ponad 2400 cytowań.
- R. Poli, W. B. Langdon, N. F. McPhee, A Field Guide to Genetic Programming, 2008, książka wydana na licencji creative commons, przewodnik dla osób chcących poeksperymentować z GP.
- J. R. Koza, M. A. Keane, M. J. Streeter, W. Mydlowec, J. Yu, G. Lanza, Genetic Programming IV: Routine Human-Competitive Machine Intelligence, 2003, Springer. Najnowsza książka wynalazcy GP. Większość problemów została rozwiązana na klastrze składającym się z 500 dwu-procesorowych komputerów z Pentium II 350 MHz z 64MB RAM. Typowy przebieg wykonywał się 50 godzin, a każdy z 1000 procesorów zajmował się populacją 1000 osobników.
- Wydawnictwo Springer wydaje czasopismo naukowe Genetic Programming and Evolvable Machines, (http://www.springer.com/computer/ai/journal/10710) poświęcone programowaniu genetycznemu.

Oprogramowanie

- ECJ (Evolutionary Computation in Java) - biblioteka do obliczeń ewolucyjnych w Javie (http://cs.gmu.edu/~eclab/projects/ecj/). Zawiera implementacje programowania genetycznego.
- EO (Evolving Objects) biblioteka w języku C++ (http://eodev.sourceforge.net/).
- Open BEAGLE (http://beagle.sourceforge.net/) jeszcze jedna biblioteka w C++, wydaje się że rozwój się zakończył w 2007 roku.
- JGAP (Java Genetic Algorithms Package - http://jgap.sourceforge.net/) inny framework napisany w Javie, aktywnie rozwijany.
- GPLAB (http://gplab.sourceforge.net/) - darmowy toolbox do MATLAB-a implementujący GP.
- HeuristicLab (http://dev.heuristiclab.com/trac/hl/core) pakiet z bogatym interfejsem użytkownika, oparty o .NET 4.0.