Programowanie Genetyczne

dr Dariusz Pałka
dpalka@agh.edu.pl

Programowanie genetyczne

Q: „**What is Genetic Programming?**”
A:”Genetic programming (GP) is an automated method for creating a working computer program from a high-level problem statement of a problem. Genetic programming starts from a high-level statement of “what needs to be done” and automatically creates a computer program to solve the problem.”

zródło: www.genetic-programming.com

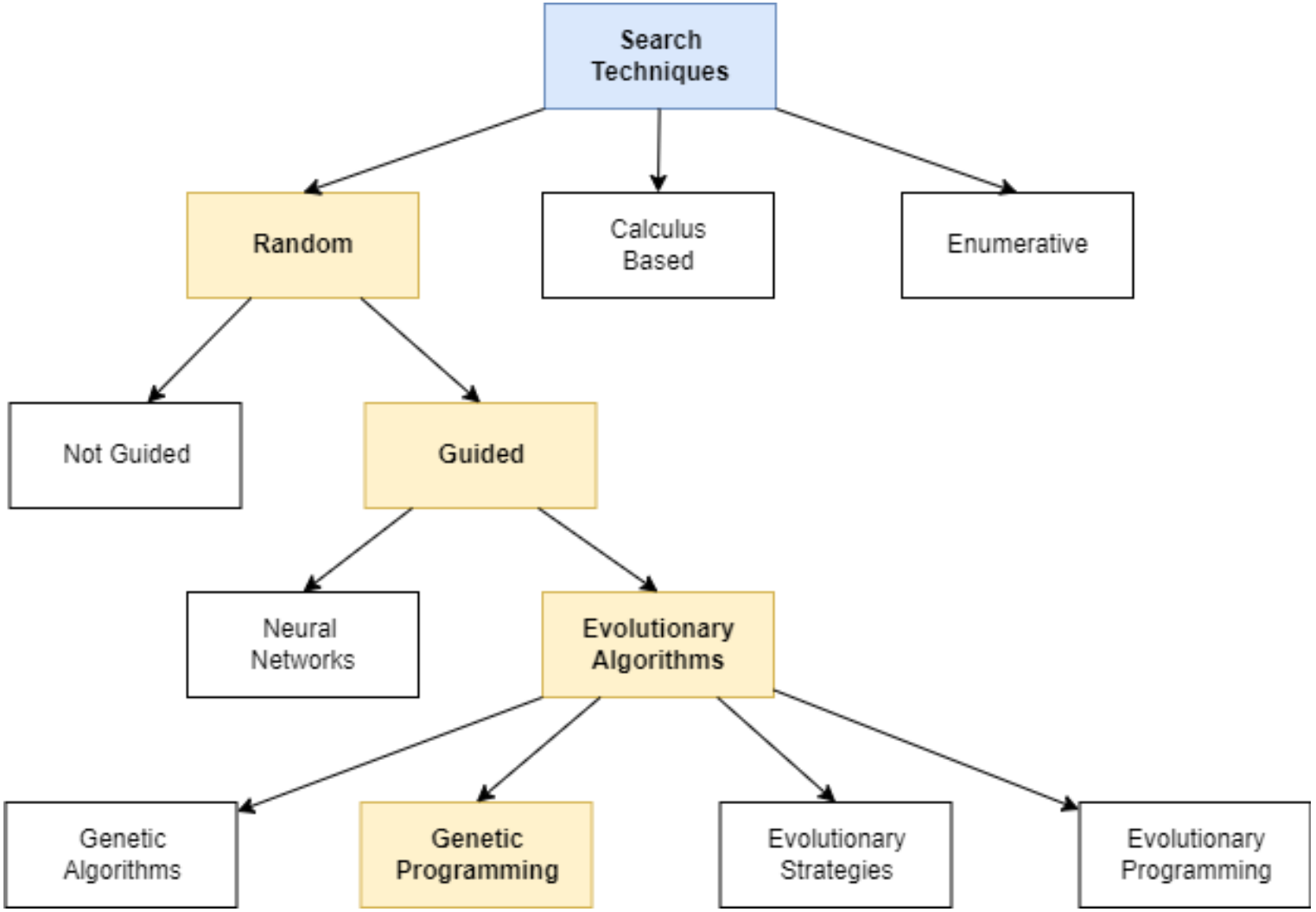
GP

- Kierowane wyszukiwanie losowe (guided random search)
- Wykorzystujące populację osobników (rozwiązań)
- Metoda inspirowana ewolucją naturalną
  - Osobniki lepiej przystosowane do środowiska mają większe szanse na reprodukcję i przekazanie swoich genów do następnego pokolenia
  - Funkcja dopasowania – ocena osobników
  - Operatory pozwalające na zmianę rozwiązania

Kiedy GP jest szczególnie użyteczne

- Olbrzymia przestrzeń poszukiwań (search space)
- Dokładna postać rozwiązania nie jest znana
- Można zaakceptować rozwiązanie przybliżone

Techniki wyszukiwania



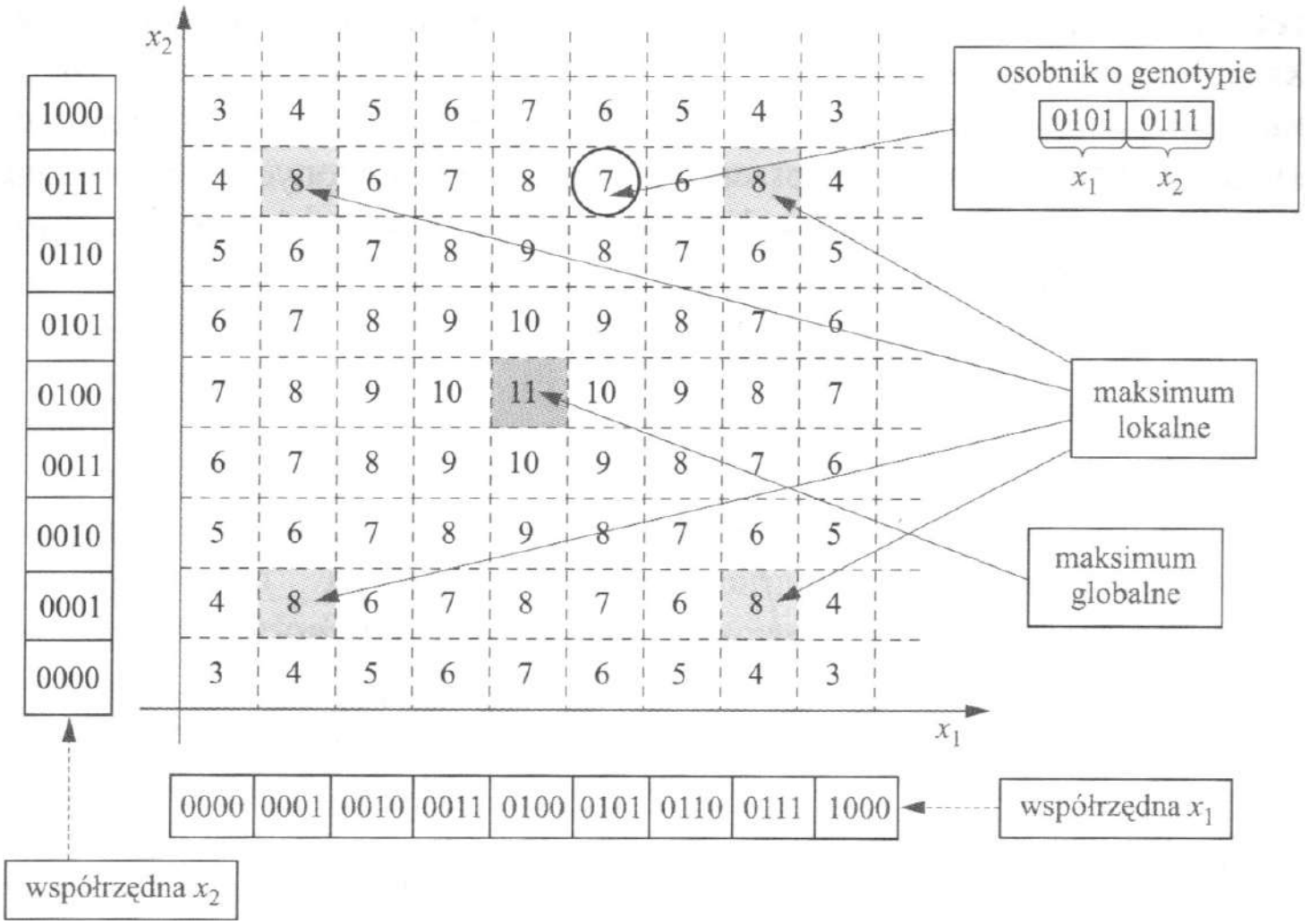
Geneza GP

- Pierwsze znane(?) koncepcje ewolucji programów pochodzą od Alana Turinga „Computing Machinery and Intelligence” (1950)
- Algorytmy genetyczne
  - prace Alexa Frasera (1957)
  - Upowszechnienie GA przez Johna Hollada 1975 „Adaptation in Natural and Artificial Systems: An Introductory Analysis with Applications to Biology, Control, and Artificial Intelligence”
- 1966 praca Fogel, Owens i Walsh – idea programowania ewolucyjnego w którym do rozwiązania postawionego problemu używane są CA podlegające ewolucji

Geneza GP

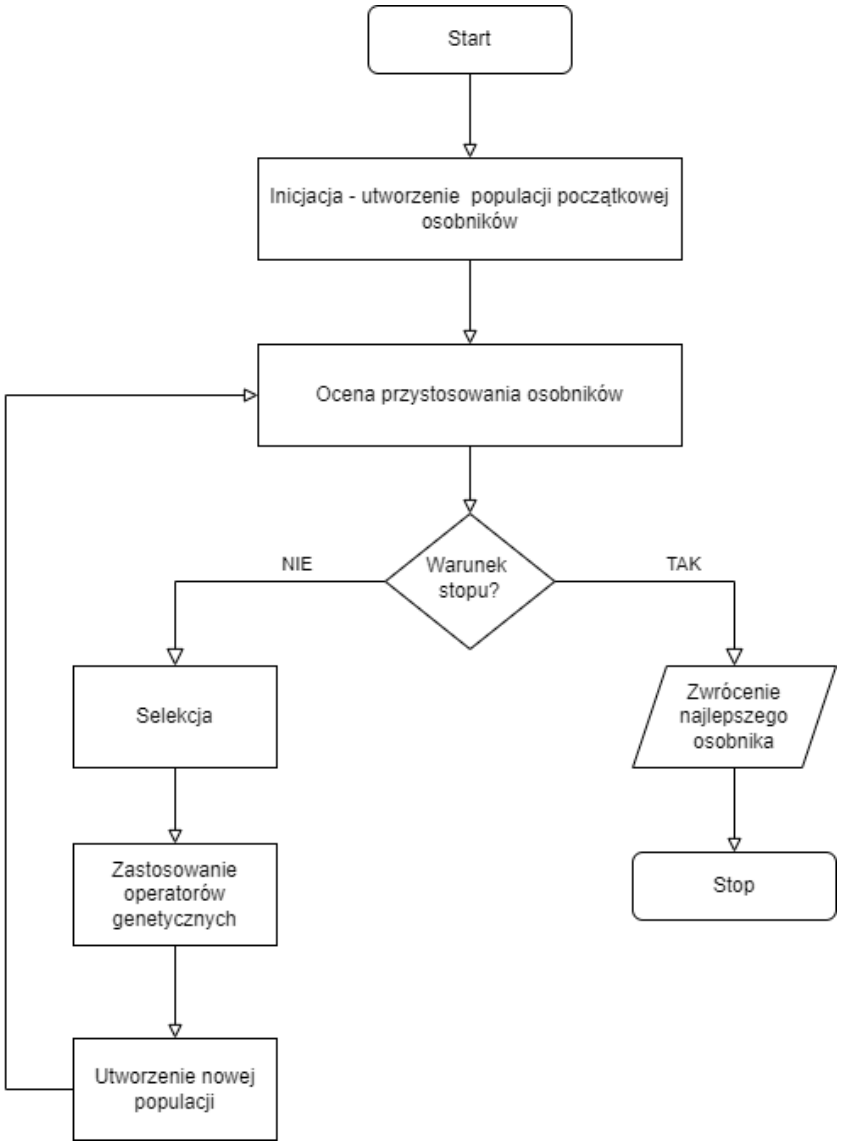
- 1985 Cramer jako pierwszy zastosował operację krzyżowania poddrzew dla potrzeb ewolucji w języku TB (tree based)
- Programowanie genetyczne - John Koza (doktorant Hollanda) prace od 1988
  - 1990 „ Genetic Programming: A Paradigm for Genetically Breeding Populations of Computer Programs to Solve Problems” http://www.genetic-programming.com/jkpdf/tr1314.pdf
  - 1992 „Genetic Programming: On the Programming of Computers by Means of Natural Selection”, MIT Press
  - 1994 Genetic Programming II: Automatic Discovery of Reusable Programs, MIT Press

GA



Źródło: Mariusz Flasiński „Wstęp do sztucznej inteligencji”

GA - schemat


Start
Inicjacja - utworzenie populacji początkowej osobników
Ocena przystosowania osobników
Warunek stopu?
NIE
TAK
Selekcja
Zastosowanie operatorów genetycznych
Utworzenie nowej populacji
Zwrócenie najlepszego osobnika
Stop

Algorytmy ewolucyjne terminologia

- **Populacja** – zbiór osobników o określonej liczebności
- **Osobniki** – zakodowane w postaci chromosomu zbiory parametrów rozwiązania (punkty w przestrzeni poszukiwań)
- **Gen** (cecha, znak) – pojedynczy element genptypu
- **Genotyp** – zespół (struktura) chromosomów danego osobnika. Osobnikami populacji mogą być genotypy lup pojedyncze chromosomy (jeśli genotyp składa się z jednego chromosomu)
- **Fenotyp** – zestaw wartości odpowiadający danemu genotypowi (zdekodowana struktura), zbiór parametrów rozwiązania
- **Allel** – wartość danego genu (wartość cechy)

GP

- Nie nakłada ograniczeń na formę rozwiązania – struktura rozwiązania jest dowolna
- Rozwiązania mogą reprezentować cokolwiek co da się „zakodować” - np.
  - Programy (w dowolnych językach)
  - Funkcje, formuły, równania
  - Układy elektroniczne
  - Projekty

GP

- „+” - Szeroki zakres zastosowań
- „-” - Rozwiązania mogą szybko się rozrastać i być niepotrzebnie złożone

GP Przykłady

- Projekt
  - 2006 NASA Space Technology 5 aircraft antena (https://www.jpl.nasa.gov/nmp/st5/TECHNOLOGY/antenna.html)

Wykorzystanie populacji

- Możliwość zastosowania obliczeń równoległych
- Możliwość przeszukiwania wielowymiarowych przestrzeni rozwiązań

Źródło: https://continuingstudies.uvic.ca/

GP - podstawy

- Rozszerzenie klasycznych algorytmów genetycznych
- Kodowanie binarne zastąpione (na ogół) kodowaniem drzewiastym (LISP)
- Zmiana sposobu działania operacji genetycznych (adaptacja do reprezentacji drzewiastej)

Lisp przykład

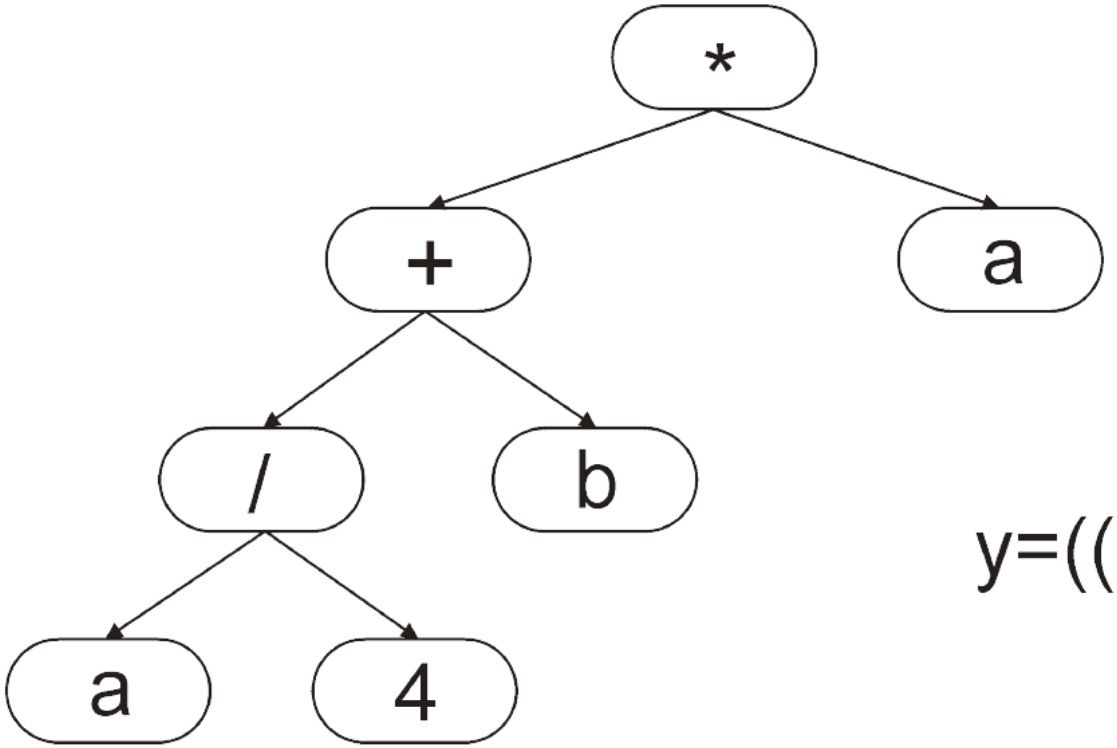
```
(defun factorial (n)
    (if (zerop n) 1
        (* n (factorial (1- n)))))

(defun factorial (n)
    (loop for i from 1 to n
        for fac = 1 then (* fac i)
        finally (return fac)))
```

GP - reprezentacja

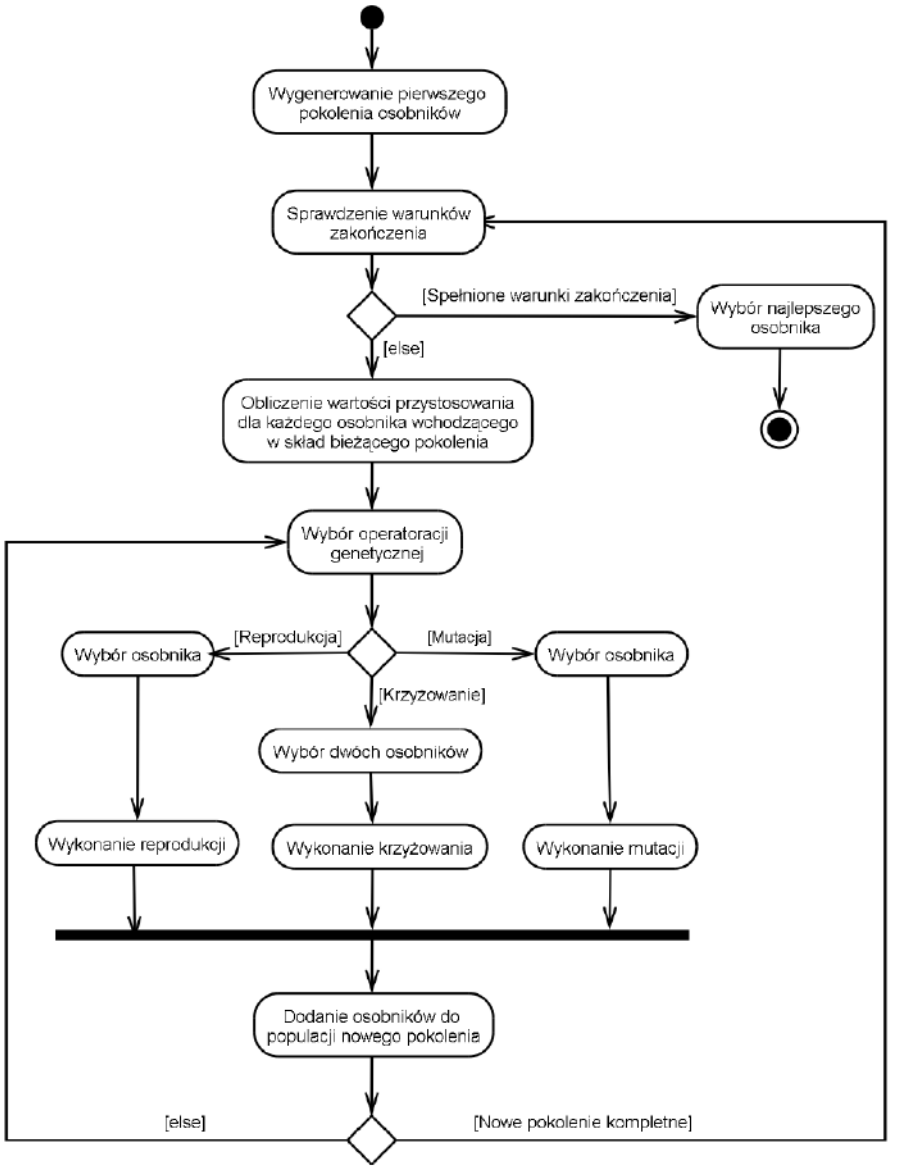
- Program reprezentowany jest za pomocą drzewa, które składa się z jednego lub większej ilości wierzchołków
- Wierzchołki pochodzą z 2-ch rozłącznych zbiorów:
  - Zbioru funkcji F
  - Zbioru terminali T
- Wyznaczenie zbiorów F i T to etap wstępny użycia GP

GP - reprezentacja



y=((a/4)+b)*a

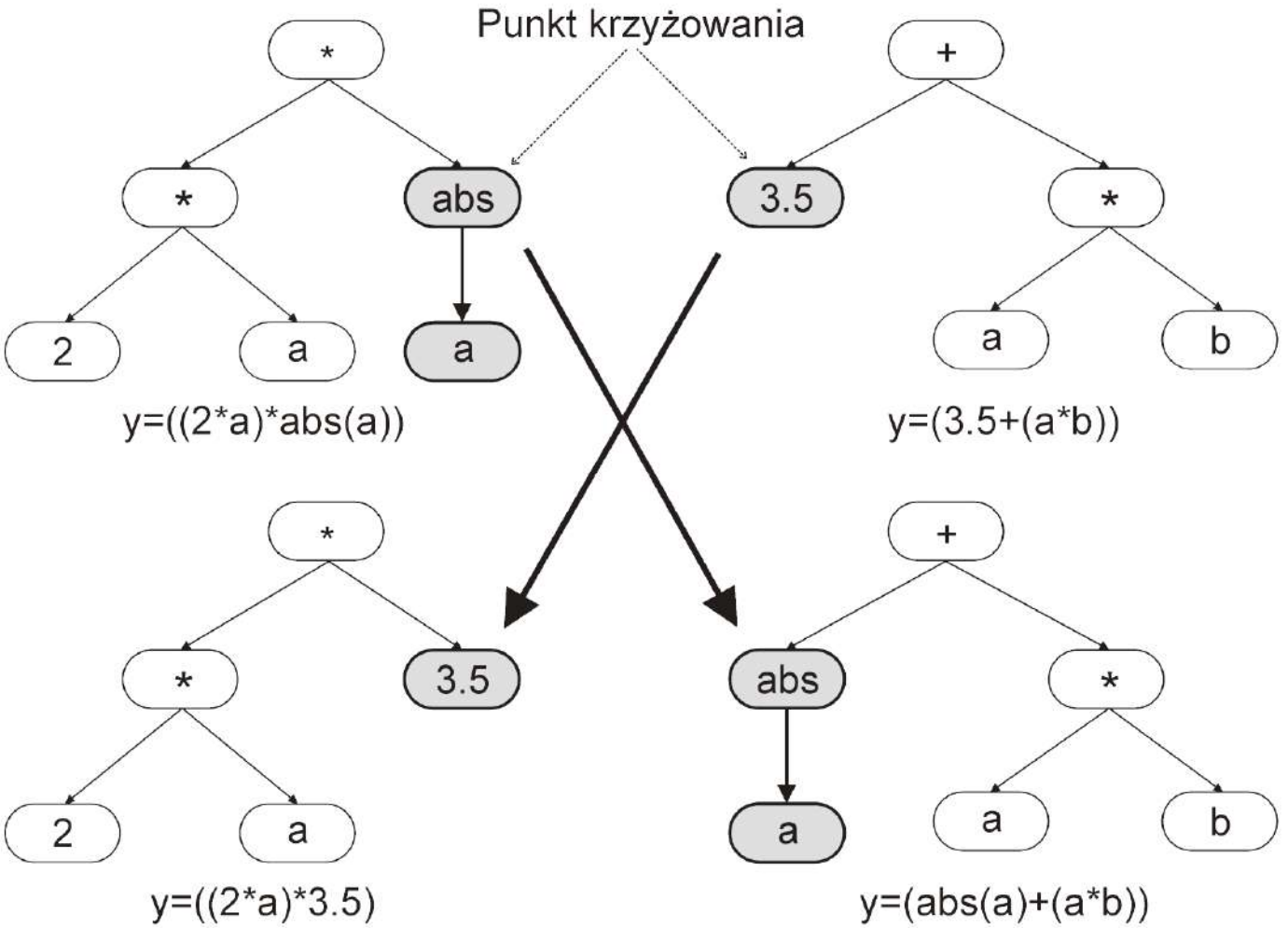
GP - schemat


Wygenerowanie pierwszego pokolenia osobników
Sprawdzenie warunków zakończenia
[Spełnione warunki zakończenia]
Wybór najlepszego osobnika
[else]
Obliczenie wartości przystosowania dla każdego osobnika wchodzącego w skład bieżącego pokolenia
Wybór operatoracji genetycznej
[Reprodukcja]
[Mutacja]
Wybór osobnika
Wybór osobnika
[Krzyzowanie]
Wybór dwóch osobników
Wykonanie reprodukcji
Wykonanie krzyżowania
Wykonanie mutacji
Dodanie osobników do populacji nowego pokolenia
[else]
[Nowe pokolenie kompletne]

Etapy GP

- Generowanie pierwszego pokolenia
  - Metoda „full”
  - Metoda „grow”
  - Metoda „ramped half and half”
- Ocena osobników
- Selekcja osobników
- Operacje genetyczne...

GP crossover



GP przykłady

- Regresja symboliczna
- ...
- GP in Deep Learning
- Annual "Humies" Awards For Human-Competitive Results (https://www.human-competitive.org/)

GP Regresja symboliczna

- Funkcja **f(x) = x1*x1+2**
- (21 pokolenie) Best Individual : ((((X1 * X1) + -3.023394996582597) - -4.801968845897364) - (-0.8545071961179138 / ((2.044217854474746 7 + 1.815372932099357 7) + (0.03050658950392204 4 / (((((-2.938533036758191 7 - (2.004923266640129 3 - ((0.01134529576916421 - 1.486820610944873 2) + (-0.8417323372865 04 * 2.044217854474746 7)))) / ((((-0.7646475814000189 * ((-0.6671452250416889 - -2.000774052090731 6) * (0.6446568470410696 + (3.291738029142850 4 + 1.321302928915177 9)))) - 0.01134529576916421) / (((-0.4997098444746708 - -2.000774052090731 6) * ((((3.383413944946442 + (-0.4997098444746708 * (2.003699960875358 - -2.730819519126850 3))) - (-3.4126448179984092 + -0.2129944646391814 8)) / 2.044217854474746 7) * (((-2.859536313550714 6 * 4.199601223568122) / ((-2.938533036758191 7 - (2.004923266640129 3 - 2.004923266640129 3)) + (-0.8417323372865 04 * 1.486820610944873 2))) - 1.486820610944873 2))) + (0.941998416153610 1 - (3.835093090071073 / 0.01134529576916421)))) - -0.2129944646391814 8)) - ((2.042149625659270 4 + -3.063836515978517 5) / ((1.321302928915177 9 / 0.01134529576916421) + -3.546239445571934))) * (1.219114280923 18 / 2.004923266640129 3)) * -2.917639541161473 3)))))

Czy GP ma sens?

- Analogie do metodyk zwinnych
  - W TDD kryterium poprawności rozwiązania są testy unitarne (automatyczne)
  - W metodykach zwinnych np. w XP to czy dana funkcjonalność została poprawnie zaimplementowana jest oceniane przez testy akceptacyjne (automatyczne)

Czy GP ma sens?

- Niezrozumiały (wygenerowany) kod?
  - Co z funkcjami systemowymi i wstawkami np. w asemblerze?
  - A co np. z parserami LR (tablice Action i GOTO)?
  - Wprowadzanie zmian w nieznanym języku programowania
  - Co z programowaniem Maszyny Turinga albo językami typu brainf**k (https://en.wikipedia.org/wiki/Brainfuck)?

Zasoby

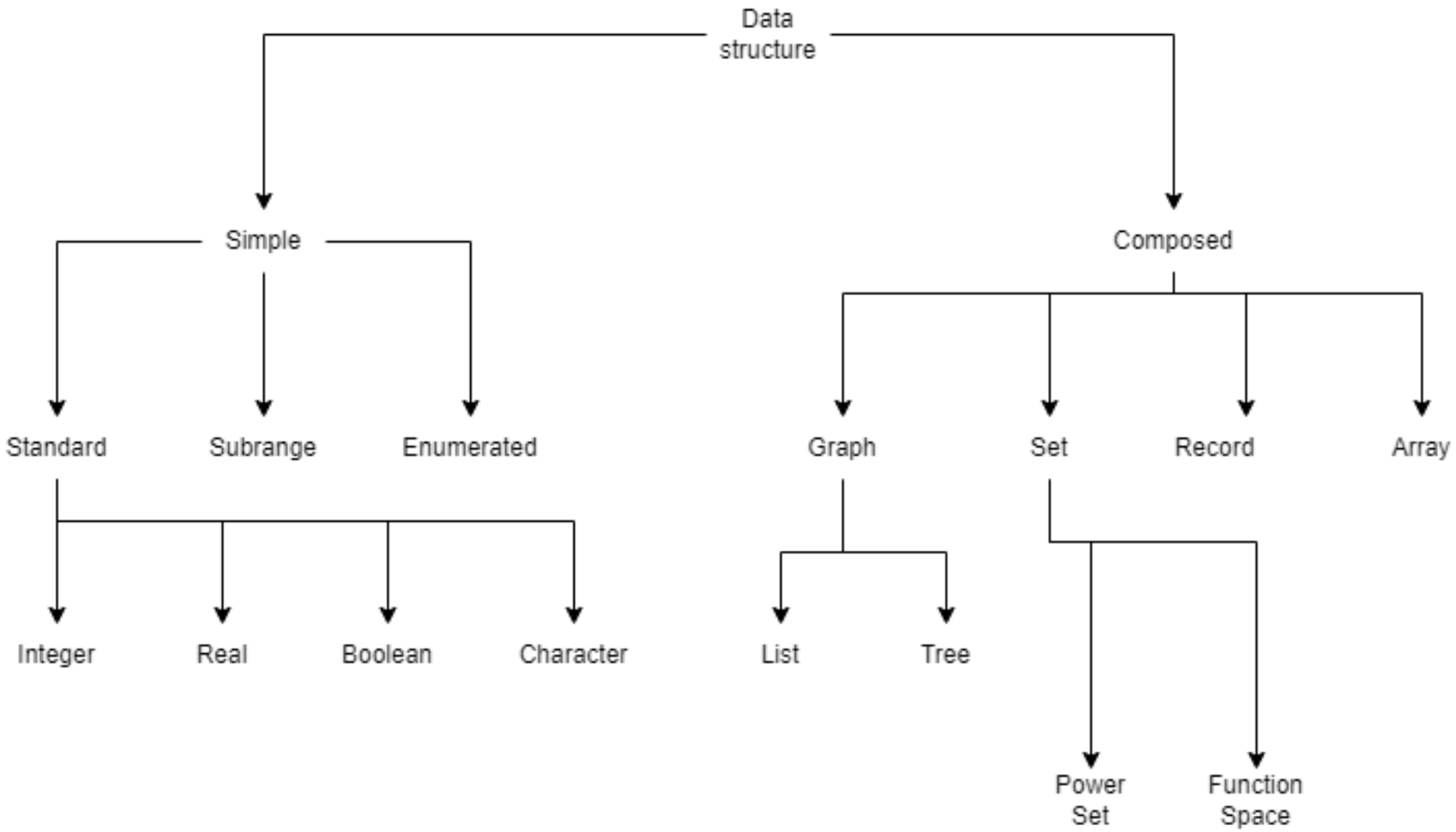
- John Koza Genetic Programming - The Movie - Part 1 https://www.youtube.com/watch?v=tTMpKrKkYXo
- Regresja symboliczna 2:47, 9:47
- John Koza Genetic Programming - The Movie - Part 2 https://www.youtube.com/watch?v=pRk6cth7Bpg

Reprezentacja osobników w Programowaniu Genetycznym

dr Dariusz Pałka
dpalka@agh.edu.pl

Reprezentacje

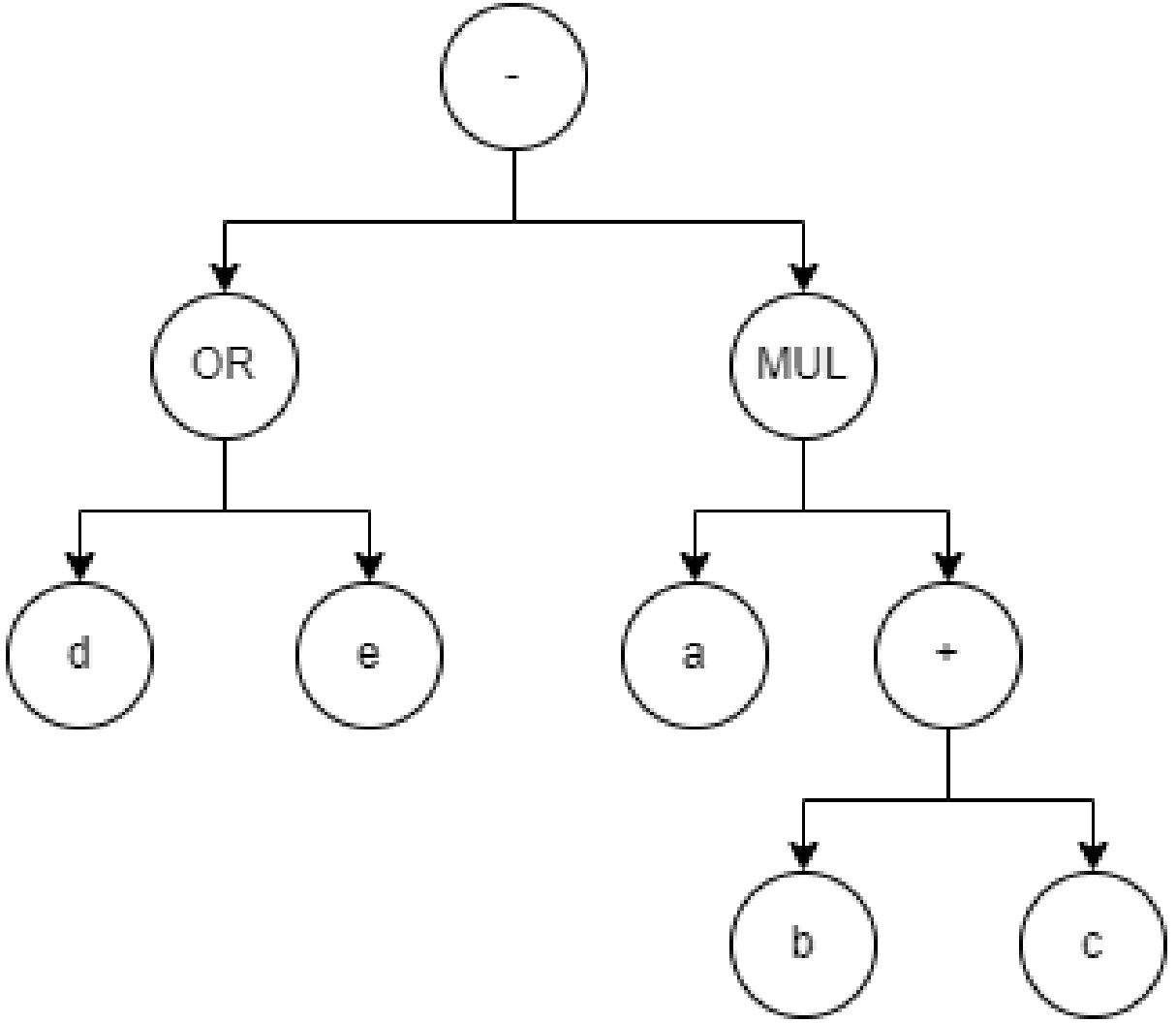
Struktury danych



Wolfgang Banzhaf, Peter Nordin, Robert E. Keller, Frank D. Francone;
Genetic Programming: An Introduction on the Automatic Evolution of computer programs and its Applications; 1998 Morgan Kaufmann Publishers

Reprezentacja za pomocą drzewa

Przykład
(d OR e) – (a * (b + c))



Parse Tree vs Abstract Syntax Tree

PT vs AST

- Gramatyka

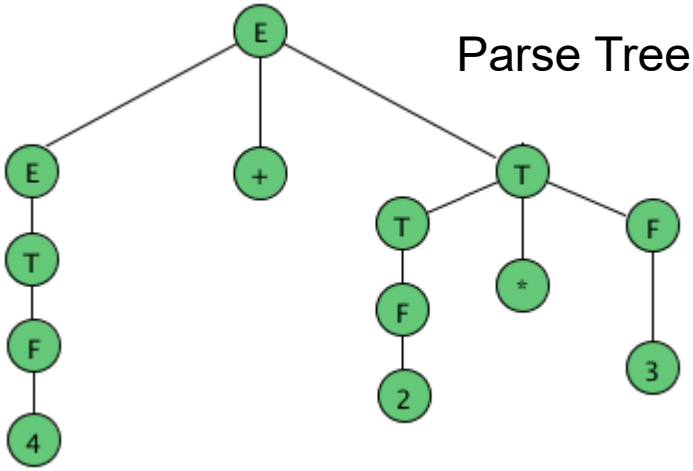
  $E \rightarrow E + T \mid E - T \mid T$

  $T \rightarrow T * F \mid F$

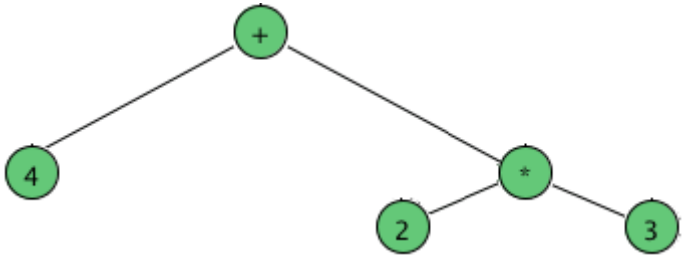
  $F \rightarrow a \mid (E)$

- Słowo: **4 + 2 * 3**

Parse Tree

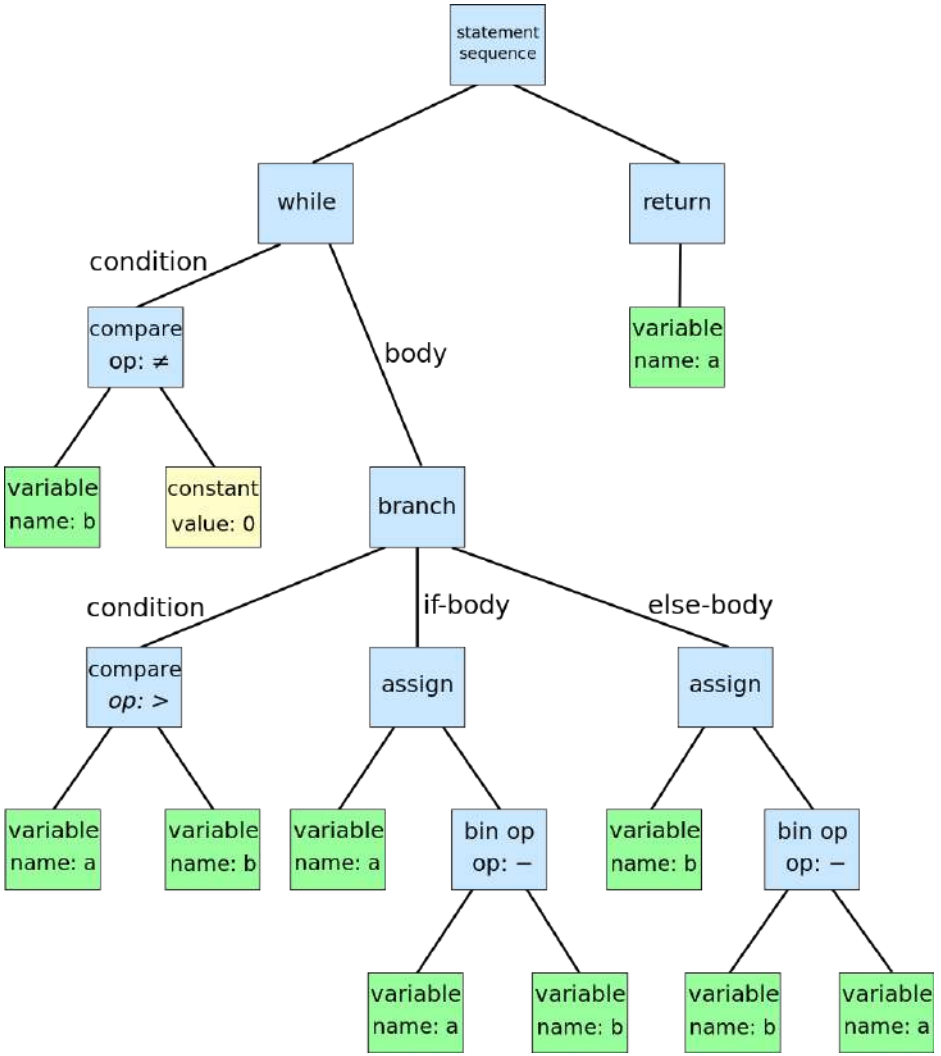


Abstract Syntax Tree



Źródło: https://www.cs.wcupa.edu/rkline/fcs/parse-trees.html

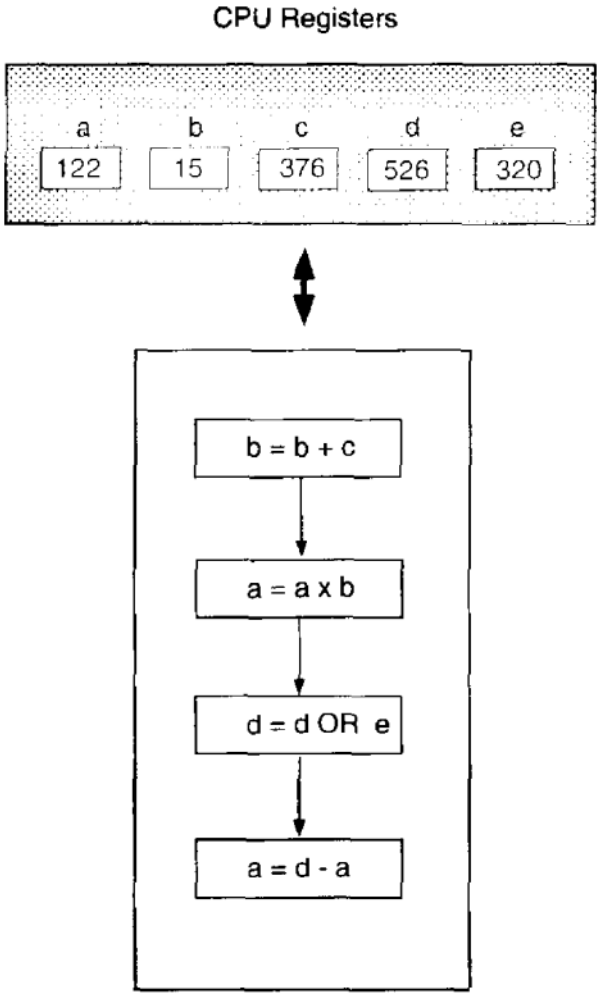
AST



Źródło: https://commons.wikimedia.org/w/index.php?curid=14676451

Reprezentacja liniowa

Przykład
(d OR e) – (a * (b + c))



Wolfgang Banzhaf, Peter Nordin, Robert E. Keller, Frank D. Francone;
Genetic Programming: An Introduction on the Automatic Evolution of computer programs and its Applications; 1998 Morgan Kaufmann Publishers

Linear Genetic Programming

Markus Brameier, Wolfgang Banzhaf

Linear GP

- Linear GP wykorzystuje programowanie imperatywne
- Programowanie imperatywne (inaczej niż np. funkcyjne) jest mocno związane z językiem maszynowym
- Większość obecnych CPU oparta jest na architekturze von Neumanna i bazuje na rejestrach.

Linear GP

- Większość współczesnych języków maszynowych wykorzystuje instrukcje działające na 2 lub 3 rejestrach
- W prezentowanym podejściu rejestry przechowują wartości typu float.

Linear GP - przykładowy program (w notacji C)

```
void gp(r)
double r[8];
{ ...
  r[0] = r[5] + 71;
  // r[7] = r[0] - 59;
  if (r[1] > 0)
  if (r[5] > 2)
  r[4] = r[2] * r[1];
  // r[2] = r[5] + r[4];
  r[6] = r[4] * 13;
  r[1] = r[3] / 2;
  // if (r[0] > r[1])
  // r[3] = r[5] * r[5];
  r[7] = r[6] - 2;
  // r[5] = r[7] + 15;
  if (r[1] <= r[6])
  r[0] = sin(r[7]);
}
```

Markus Brameier, Wolfgang Banzhaf; Linear Genetic Programming; 2007 Springer Science

LGP instrukcje

Instruction type	General notation	Input range
Arithmetic operations	$r_i := r_j + r_k$ $r_i := r_j - r_k$ $r_i := r_j \times r_k$ $r_i := r_j \ / \ r_k$	$r_i, r_j, r_k \in \mathbb{R}$
Exponential functions	$r_i := {r_j}^{(r_k)}$ $r_i := e^{r_j}$ $r_i := ln(r_j)$ $r_i := {r_j}^2$ $r_i := \sqrt{r_j}$	$r_i, r_j, r_k \in \mathbb{R}$
Trigonomic functions	$r_i := sin(r_j)$ $r_i := cos(r_j)$	$r_i, r_j, r_k \in \mathbb{R}$
Boolean operations	$r_i := r_j \wedge r_k$ $r_i := r_j \vee r_k$ $r_i := \neg \ r_j$	$r_i, r_j, r_k \in \mathbb{B}$
Conditional branches	$if \ (r_j > r_k)$ $if \ (r_j \leq r_k)$ $if \ (r_j)$	$r_j, r_k \in \mathbb{R}$ $r_j \in \mathbb{B}$

Markus Brameier, Wolfgang Banzhaf; Linear Genetic Programming; 2007 Springer Science

Slash/A

- https://github.com/arturadib/slash-a
- Przykładowy program

```
input/0/save/input/add/output/.


input/   # gets an input from user and saves it to
register F

0/       # sets register I = 0

save/    # saves content of F into data vector D[I]
(i.e. D[0] := F)

input/   # gets another input, saves to F

add/     # adds to F current data pointed to by I (i.e.
D[0] := F)

output/. # outputs result from F
```

Slash/A

- Język kompletny w sensie Turinga
- „*The instructions are atomic in that they don't need any arguments (unlike some x86 assembly instructions, for example), so* ***any random sequence of Slash/A instructions is a semantically correct program***.”

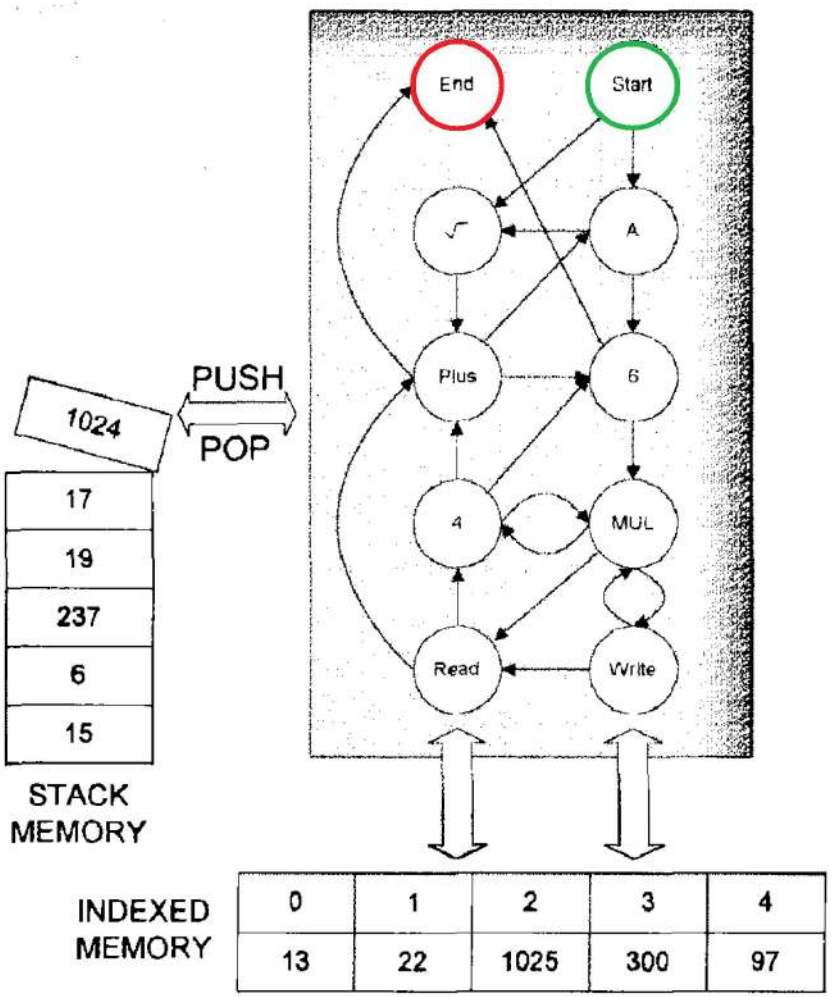
Slash/A - Operacje Genetyczne

- „*When expressed as Bytecodes, a Slash/A program is represented by a simple C++ vector of unsigned numbers, each of which corresponds to an instruction. A mutation operation is thus a simple replacement of a number in such a vector by another random integer, while a crossing-over operation can be accomplished by simply cutting-and-pasting the appropriate vector segments into another vector.*”

Reprezentacja grafowa

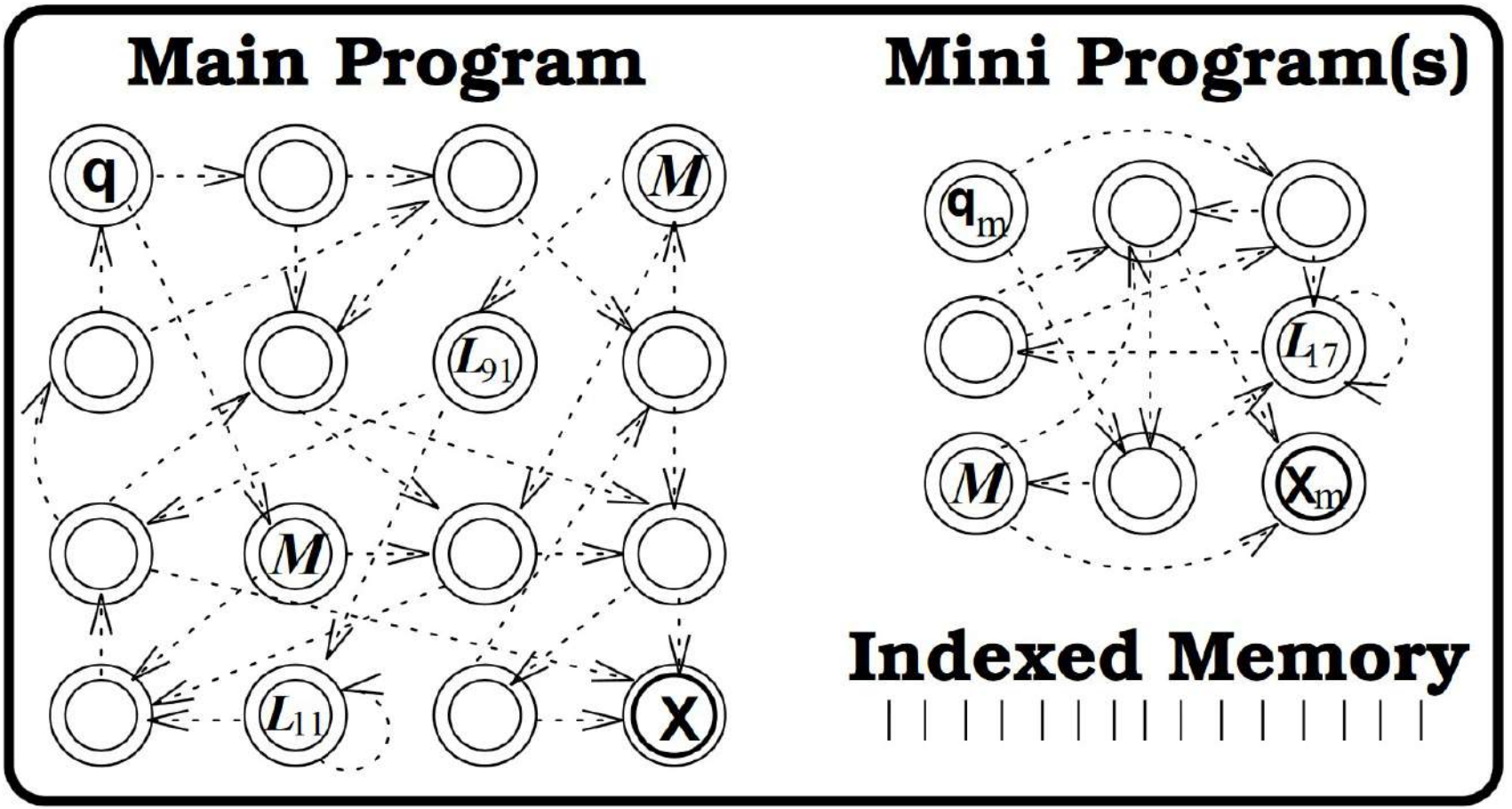
PADO (Teller and Veloso, 1995)

PADO



Wolfgang Banzhaf, Peter Nordin, Robert E. Keller, Frank D. Francone;
Genetic Programming: An Introduction on the Automatic Evolution of computer programs and its Applications; 1998 Morgan Kaufmann Publishers

PADO



Astro Teller, Manuela Veloso; PADO: A New Learning Architecture for Object Recognition; Carnegie Meollon Uniwersity

Cartesian Genetic Programming

Cartesian

- „Cartesian” - ponieważ „program” reprezentowany jest jako 2-wymiarowa siatka węzłów
- Koncepcja pojawiła się po raz pierwszy w 1999 roku – praca **Miller**, J.F.: **An Empirical Study of the Efficiency of Learning Boolean Functions using a Cartesian Genetic Programming Approach**. In: Proc. Genetic and Evolutionary Computation Conference, pp. 1135–1142. Morgan Kaufmann (1999)

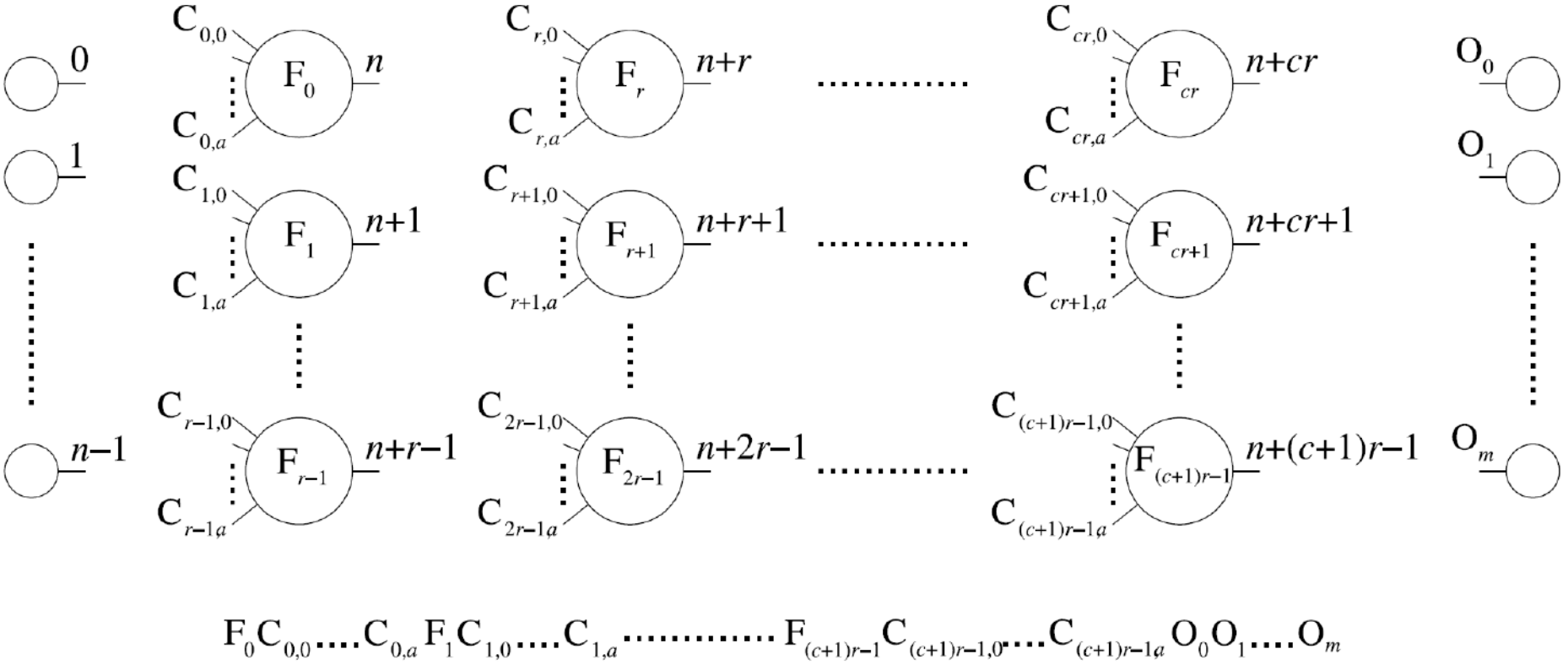
Cartesian GP

- Reprezentacja programu w formie skierowanego grafu acyklicznego
- Graf reprezentowany jest jako 2-wymiarowa siatka węzłów obliczeniowych
- Geny tworzące genotyp w CGP są liczbami całkowitymi, które określają skąd węzły pobierają dane, jakie operacje wykonują i jak uzyskać dane wyjściowe
- Genotyp w CGP ma stałą dlugość
- Przy dekodowaniu genotypu niektóre węzły mogą być ignorowane (jeśli jego dane wyjściowe nie są wykorzystywane do stworzenia danych wyjściowych dla użytkownika) – geny **‘non-coding’**

CGP węzły

- Rodzaje funkcji obliczeniowych używanych w CGP są określane przez użytkownika i znajdują się w look-up table.
- Każdy węzeł w DAG jest kodowany za pomocą kilku genów
  - Gen adresu funkcji w LUT (tzw. function gene)
  - Geny określające skąd wezeł pobiera dane (tzw. connection genes)
    Węzły pobierają dane albo z wejścia albo z poprzedniej kolumny siatki

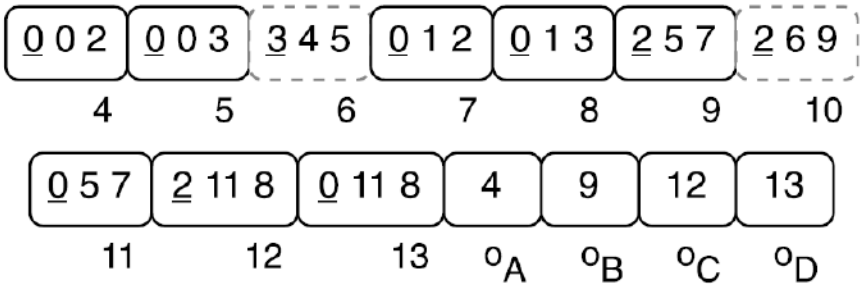
CGP ogólna postać

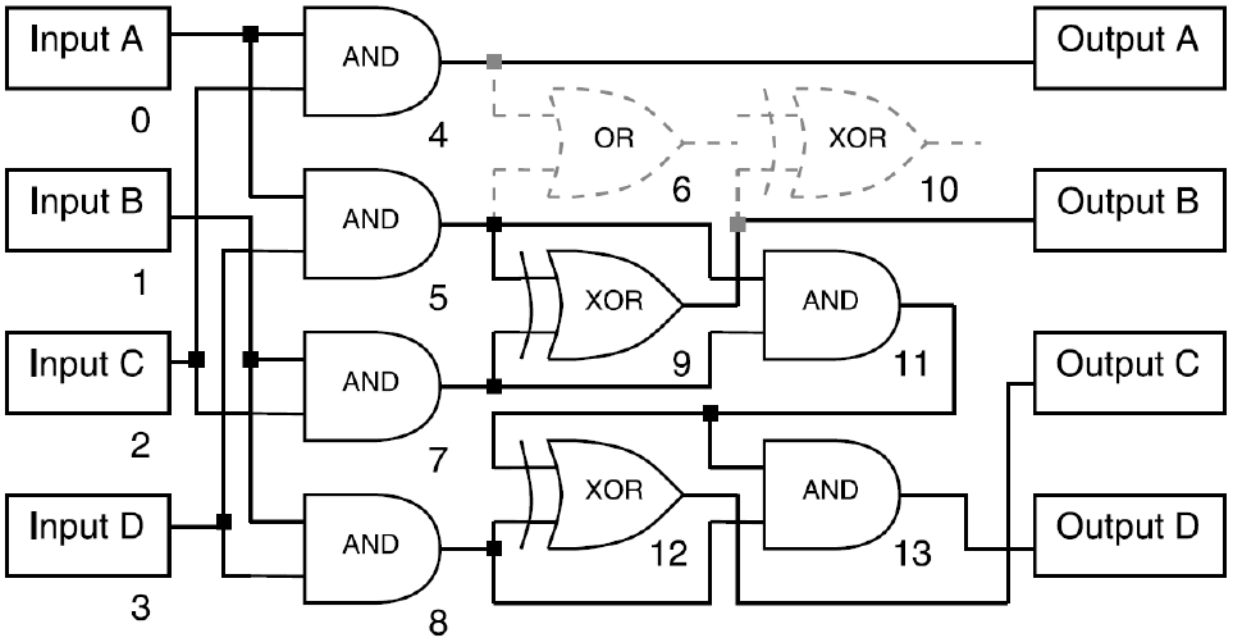


$$F_0 C_{0,0} \ldots C_{0,a} F_1 C_{1,0} \ldots C_{1,a} \ldots\ldots\ldots\ldots F_{(c+1)r-1} C_{(c+1)r-1,0} \ldots C_{(c+1)r-1a} O_0 O_1 \ldots O_m$$

J.F. Miller (ed.), Cartesian Genetic Programming, Natural Computing Series,DOI 10.1007/978-3-642-17310-3 2, ' Springer-Verlag Berlin Heidelberg 2011

CGP przykład





J.F. Miller (ed.), Cartesian Genetic Programming, Natural Computing Series,DOI 10.1007/978-3-642-17310-3 2, ' Springer-Verlag Berlin Heidelberg 2011

Grammatical evolution

GE

- Gramatyka

$E \rightarrow E + T \quad (1)$
$\quad | \; E - T \quad (2)$
$\quad | \; T \quad (3)$

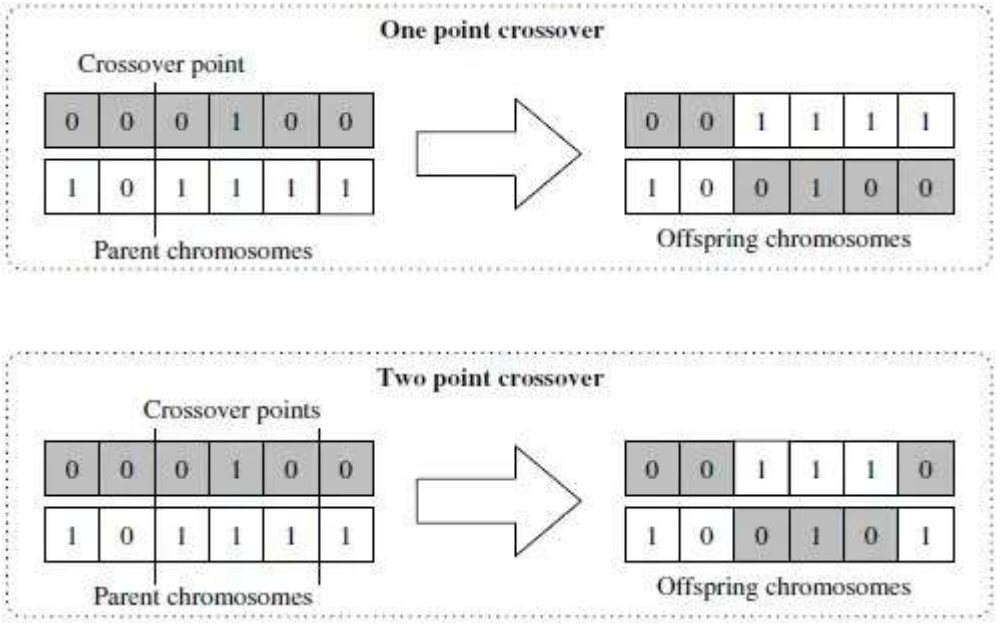
$T \rightarrow T * F \quad (1)$
$\quad | \; F \quad (2)$

$F \rightarrow a \quad (1)$
$\quad | \; (\; E \;) \quad (2)$

Operatory w GP

Krzyżowanie jedno- i wielo-punktowe

W algorytmach genetycznych (GA)



Size Fair Crossover

- Wybór 2 rodziców (w sposób tradycyjny)
- Wybór punktu krzyżowania w 1-szym rodzicu (tradycyjnie)
- Obliczany jest rozmiar poddrzewa, które ma zostać wymienione (poddrzewo poniżej 1-szego punku krzyżowania) – rozmiar ten stanowi wskazówkę do wyboru poddrzewa w 2-gim rodzicu
- Następnie obliczany jest rozmiar każdego poddrzewa w drugim rodzicu
- Poddrzewa o rozmiarze większym niż1 + 2*[rozmiar wybranego poddrzewa w 1-szym rodzicu] są pomijane

Size Fair Crossover

- Obliczane są liczby:
  - n- liczba poddrzew w 2-gim rodzicu mniejszych niż wybrane poddrzewo w 1-szym rodzicu
  - n0 liczba poddrzew (w 2-gim rodzicu) o takim samym rozmiarze jak wybrane poddrzewo
  - n+ liczba poddrzew o większym rozmiarze
  - mean- średni rozmiar mniejszych drzew (w 2-gim poddrzewie) niż wybrane w 1-szym drzewie
  - mean+ średni rozmiar większych drzew (w 2-gim poddrzewie) niż wybrane w 1-szym drzewie

Size Fair Crossover

- Jeżeli nie ma mniejszych lub większych drzew, to jedyny sposób zachowania balansu między zwiększeniem rozmiaru drzewa i jego zmniejszeniem jest niezmienianie tego rozmiaru (czyli wybór poddrzewa o takim samym rozmiarze).
  Jeżeli w 2-gim drzewie nie ma poddrzew o rozmiarze takim jak poddrzewo wybrane w 1-szym drzewie to punkt krzyżowania jest wybierany ponownie

Size Fair Crossover

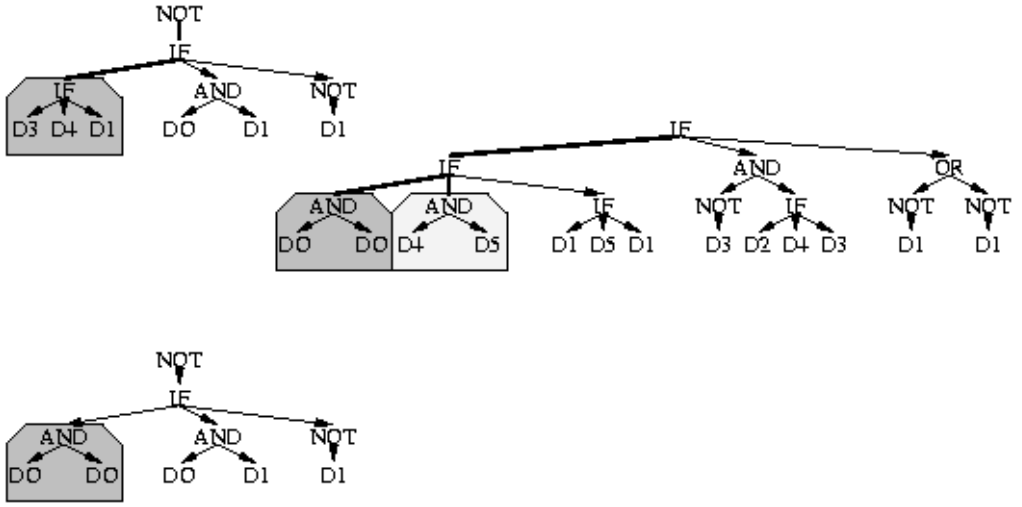
- Jeżeli są zarówno mniejsze jak i większe poddrzewa wtedy wybieramy pomiędzy nimi losowo, używając selekcji ruletkowej do wyboru długości poddrzewa.
- Jeśli istnieje więcej niż jedno poddrzewo o wylosowanej długości to wybór pomiędzy alternatywami jest losowy (z jednakowym prawdopodobieństwem)

Size Fair Crossover

- Ruletka jest „zmanipulowana” - prawdopodobieństwa wyboru poddrzew:
  - O takim samym rozmiarze
    p0 = 1/[rozmiar poddrzewa z 1-go rodzica]
  - Wszystkie poddrzewa o mniejszym rozmiarze mają takie samo prawdopodobieństwo wyboru podobnie jak o większym
    Prawdopodobieństwo wyboru poddrzewa o większym rozmiarze
    $p+ = (1-p0) / n_+(1+mean_+/mean_-)$

    Używamy średniego rozmiaru aby zrównoważyć prawdopodobieństwo wzrostu rozmiaru jak i redukcji rozmiaru

Size Fair Crossover



Źródło: http://www0.cs.ucl.ac.uk/staff/ucacbbl/wsc6/sizefair.html

- W pierwszym rodzicu zostało wybrane (do usunięcia) poddrzewo o rozmiarze 4
- W drugim poddrzewie zostało wybrane poddrzewo o rozmiarze 3 (selekcja ruletkowa)
- Z 2-ch poddrzew o rozmiarze 3 zostało wybrane pierwsze z lewej (wybór z jednakowym prawdopodobieństwem)

Homologous Crossover

- Standardowo w GP fragmenty kodu (poddrzewa) są przenoszone pomiędzy osobnikami.
- Zakłada się, że fragment (poddrzewo), które „przeżyło”proces selekcji ma pewną wartość i może przyczynić się do powstania lepszych programów
- Ważny jest jednak kontekst, w którym występuje poddrzewo.
- Przeniesienie poddrzewa w losowe miejsce może zniszczyć ten kontekst
- Przykładem krzyżowania, które ma za zadanie lepiej zachowywać kontekst jest krzyżowanie homologiczne - William B. Langdon. Size fair and homologous tree genetic programming crossovers. Genetic Programming And Evolvable Machines, 1(1/2):95-119, April 2000.

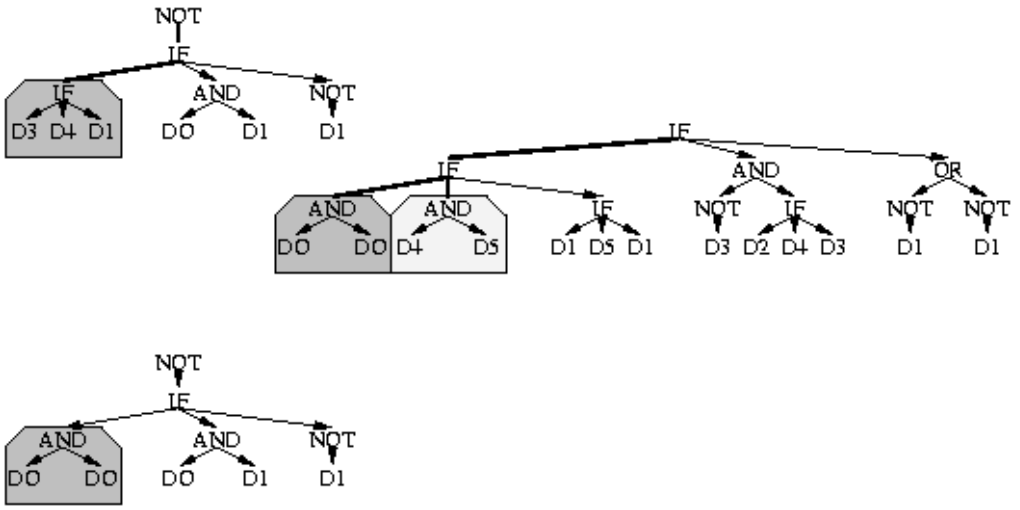
Homologous Crossover

- Schemat działania Homologous Crossover jest taki sam jak Size Fair Crossover poza ostatnim krokiem
- Zamiast losowo (z jednakowym prawdopodobieństwem) wybierać jedno z poddrzew o określonym rozmiarze (w 2-gim rodzicu) wybieramy deterministycznie poddrzewo najbardziej zbliżone do poddrzewa w pierwszym rodzicu

Homologous Crossover

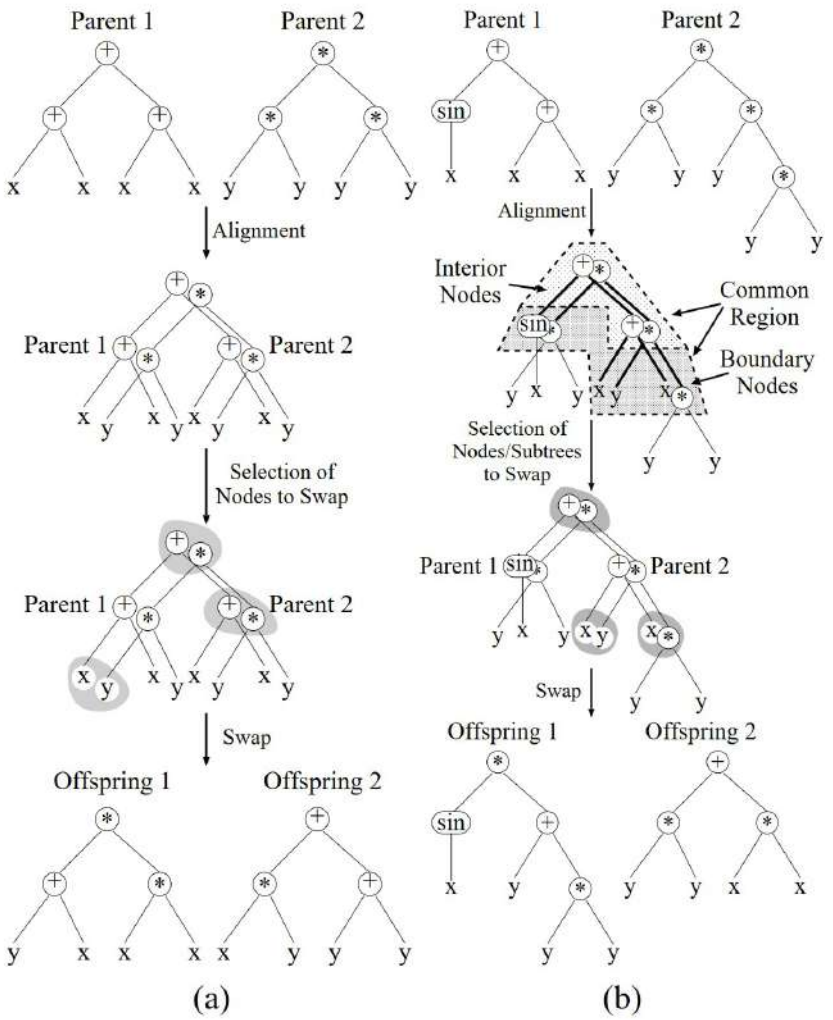
- Definiujemy odległość pomiędzy dwoma punktami krzyżowania przy użyciu jedynie ich pozycji i kształtu 2-ch drzew (ignorowane są funkcje)
- Jest to wykonywane w ten sposób, że śledzimy pozycję poddrzewa w kierunku korzenia.
- Odległość (bliskość) dwóch wierzchołków w drzewie jest określona jako głębokość na której ich drogi w kierunku korzenia się różnią

Homologous Crossover



Źródło: http://www0.cs.ucl.ac.uk/staff/ucacbbl/wsc6/sizefair.html

Uniform Crossover



a) taki sam kształt drzew b) ogólna postać

źródło: Riccardo Poli and W. B. Langdon On the Search Properties of Different Crossover Operators in Genetic Programming. Presented at GP-98

Uniform Crossover

- Wierzchołki w obszarach wspólnych mają taką samą liczbę dzieci (arity)
- Wierzchołki w obszarach wspólnych są wybierane z jednakowym prawdopodobieństwem
- Jeżeli wierzchołek należ do brzegu obszaru wspólnego (jest liściem obszaru wspólnego) i jest funkcją to również poddrzewo poniżej niego jest wymieniane w przeciwnym razie wymieniana jest tylko etykieta

Krzyżowanie w Grammatical Evolution

- Normalne krzyżowanie jak w algorytmach genetycznych (GA)

Mutacja

- Zamiana poddrzewa dla wybranego wierzchołka drzewa na nowe poddrzewo
  - Generowanie poddrzewa tak, aby głębokość całego drzewa nie wzrosła więcej niż o 15% (K. E. Kinnear, Jr., “Evolving a sort: Lessons in genetic programming)
  - Generowanie poddrzewa o rozmiarze l +/- l/2 (50% - 150% mutation) (W. B. Langdon, “The evolution of size in variable length representations)
  - Generowanie poddrzewa o takim samy rozmiarze jak wybrane poddrzewo (W. B. Langdon)

Mutacja

- Shrink mutation – zamiana wybranego poddrzewa na terminal (P. J. Angeline, “An investigation into the sensitivity of genetic programming to the frequency of leaf selection during subtree crossover”)
- Point mutation (node replacement mutation) – zamiana wybranego wezła na inny o takiej samej ilości dzieci (arity)
- Hoist mutation – wybranie losowego poddrzewa i zamiana drzewa (z którego pochodzi to poddrzewo) na to poddrzewo. Potencjalnie mocno destrukcyjna

Mutacja w Grammatical Evolution

- Normalna mutacja jak w algorytmach genetycznych (GA)