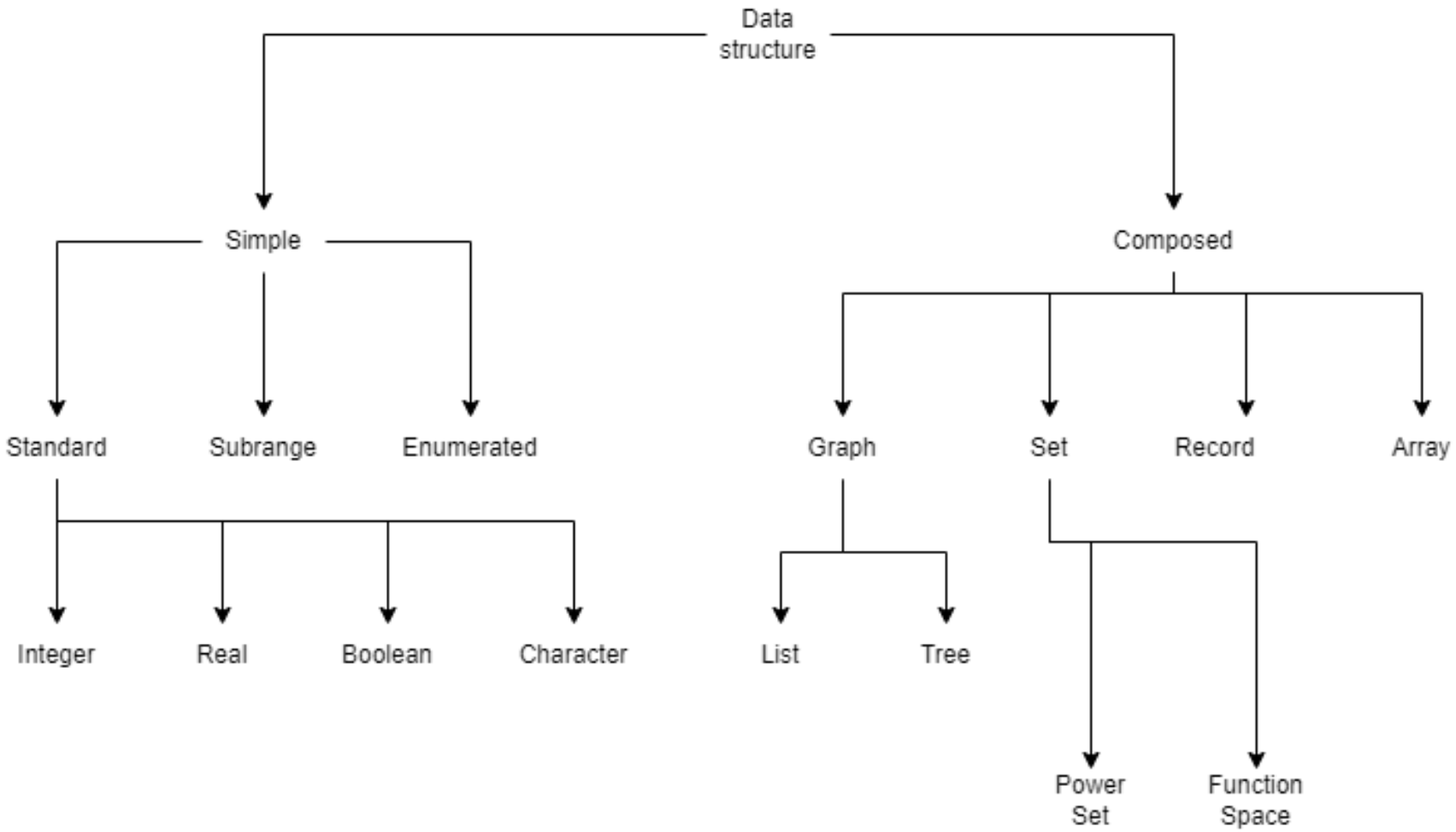
Reprezentacja osobników w Programowaniu Genetycznym

dr Dariusz Pałka
dpalka@agh.edu.pl

Reprezentacje

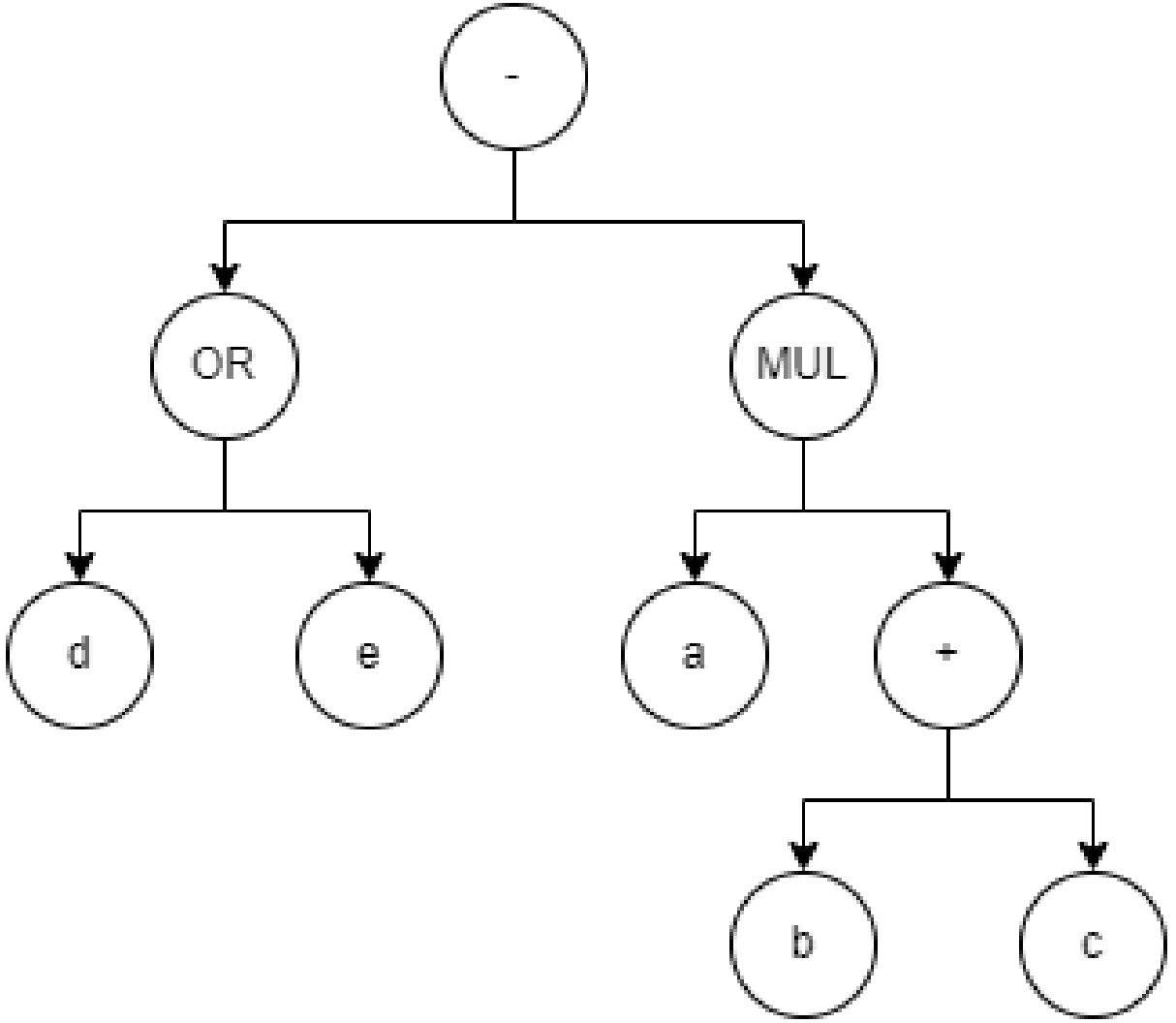
Struktury danych



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

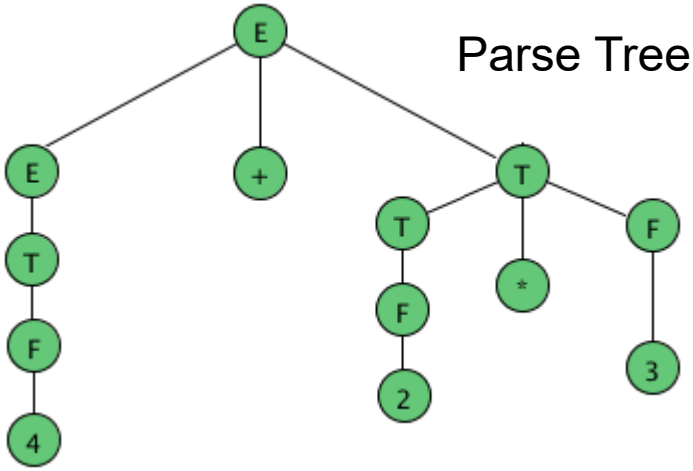
  $E \rightarrow E + T \mid E - T \mid T$

  $T \rightarrow T * F \mid F$

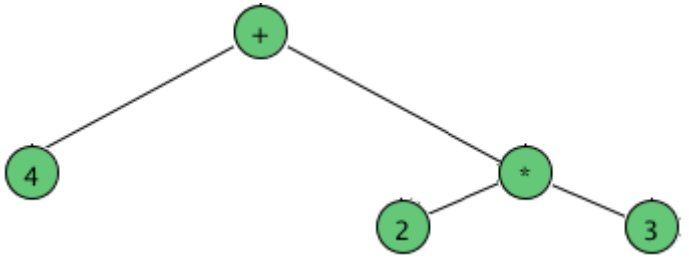
  $F \rightarrow a \mid (E)$

- Słowo: **4 + 2 * 3**

Parse Tree

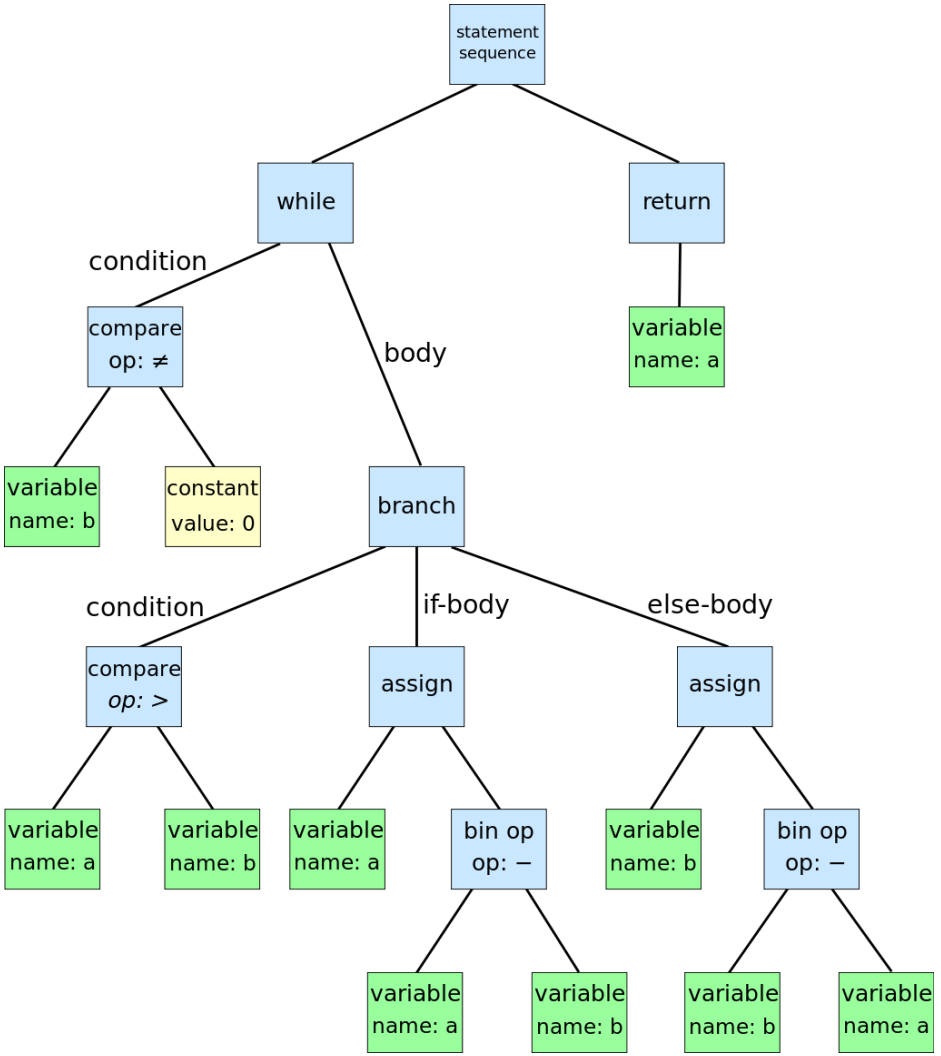


Abstract Syntax Tree



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

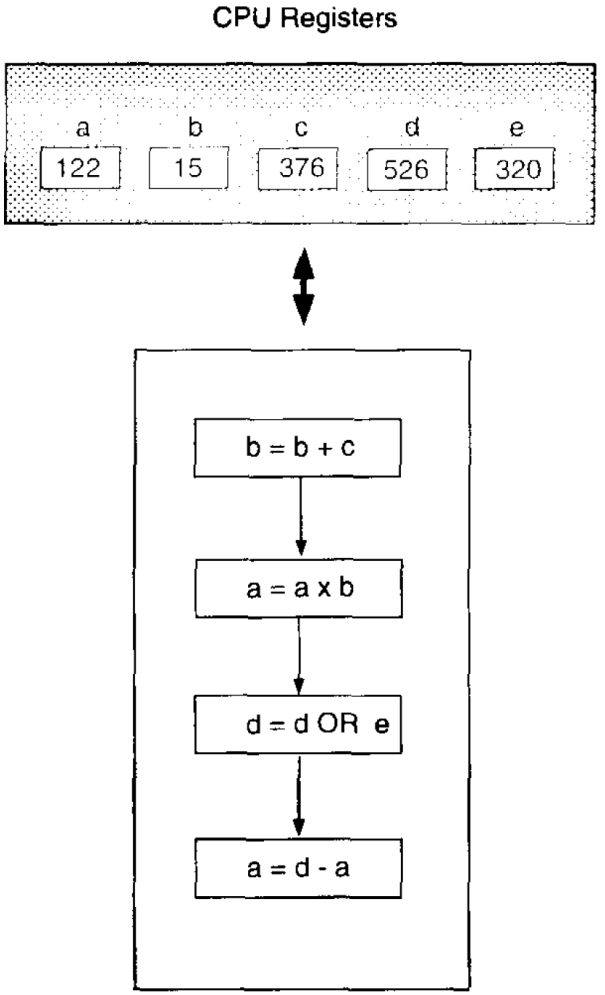
AST



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***.”

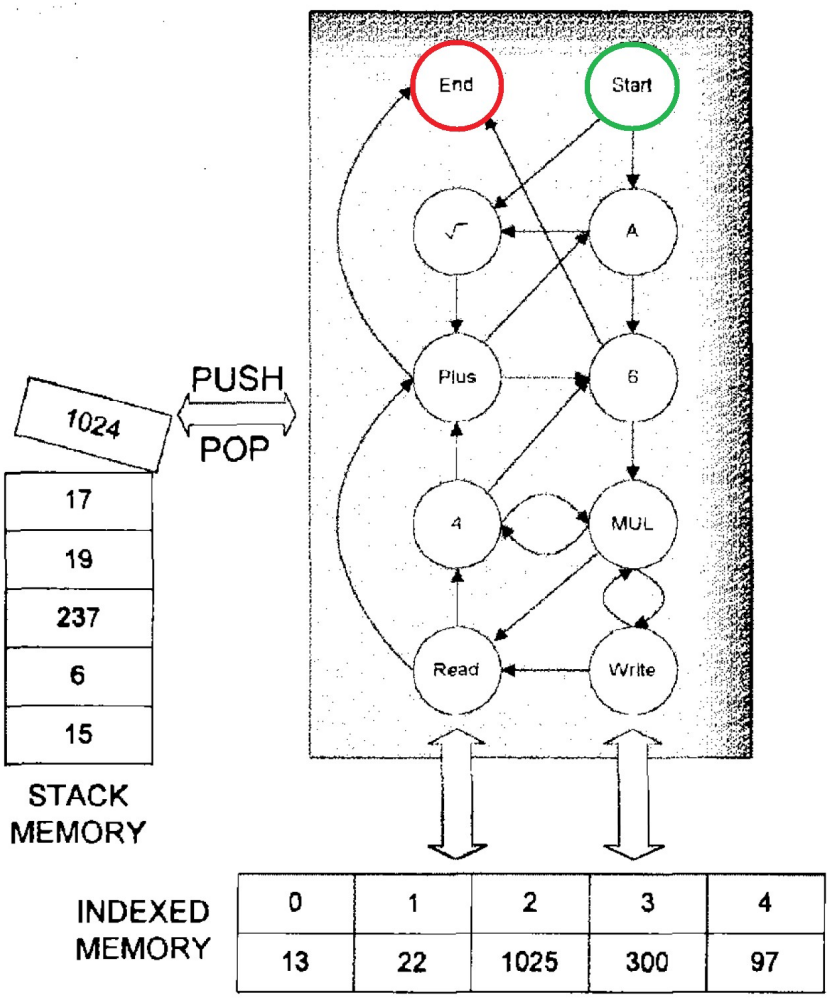
Slash/A - Operacje Genetyczne

- „*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

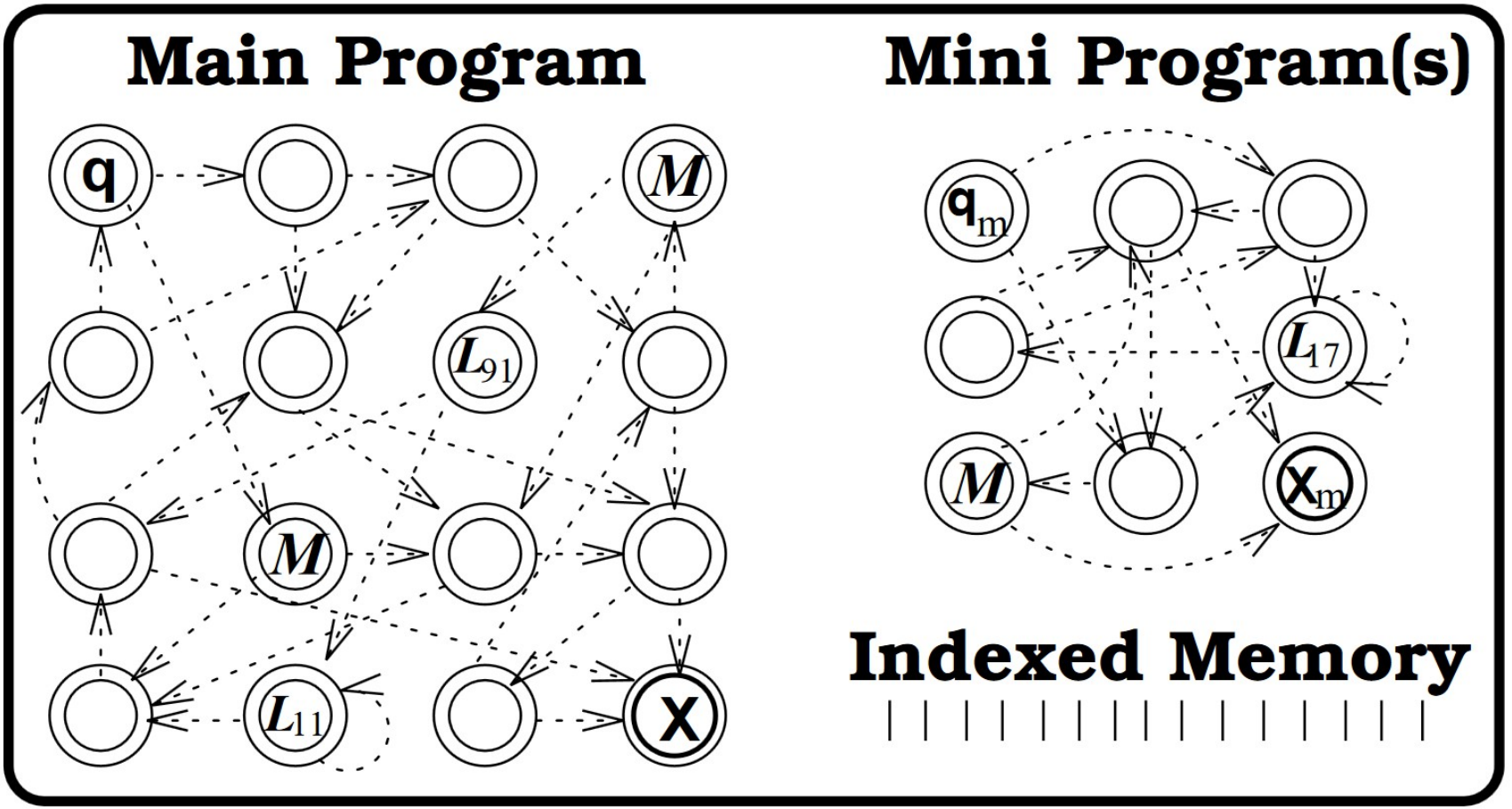
PADO (Teller and Veloso, 1995)

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)

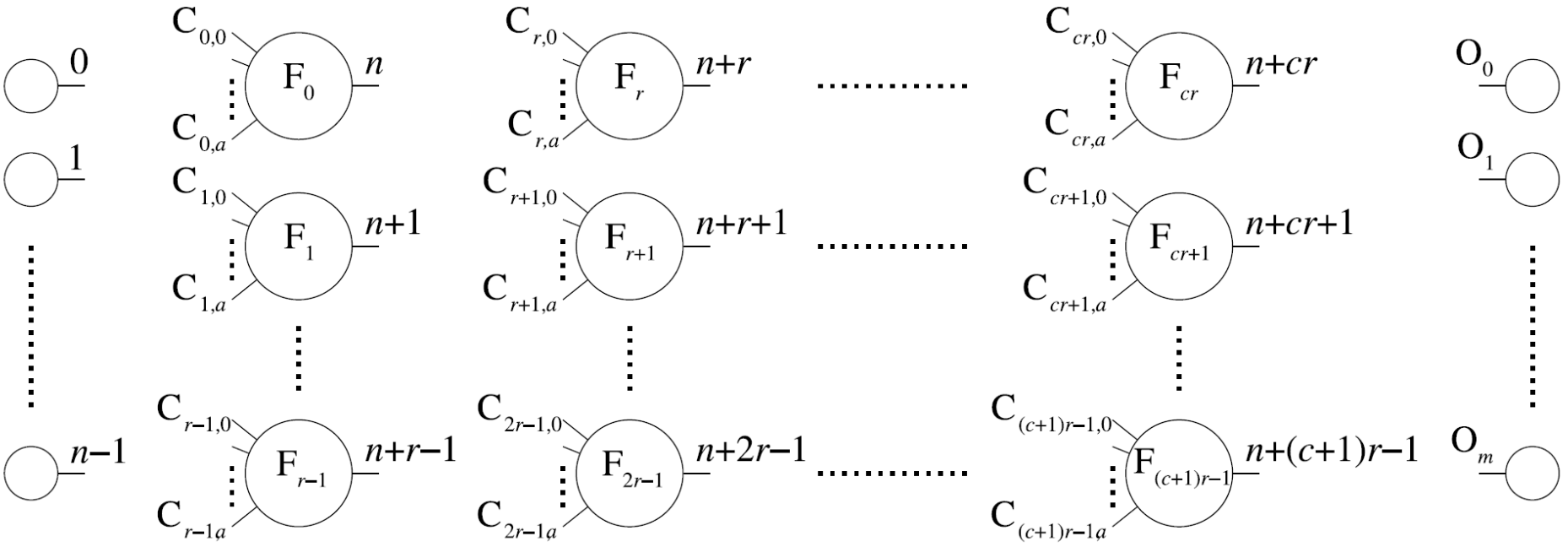
Cartesian GP

- 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

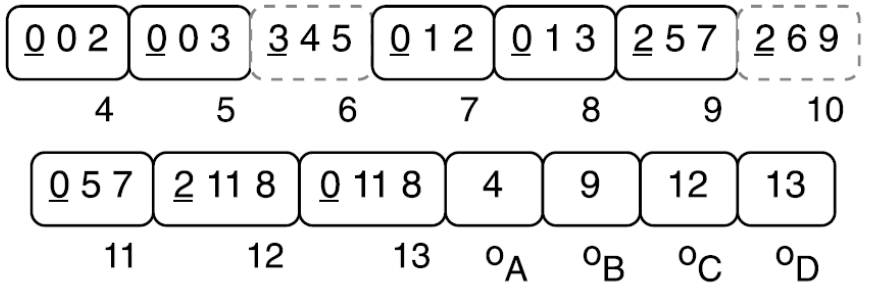
CGP ogólna postać

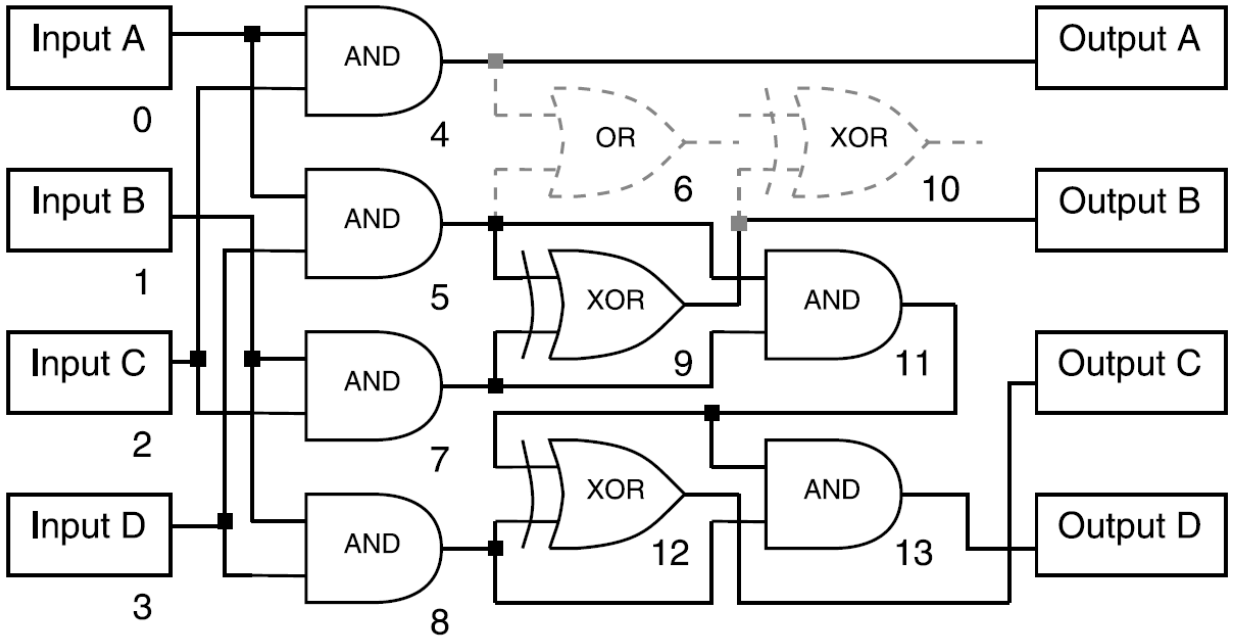


$$F_0 C_{0,0} \ldots C_{0,a} F_1 C_{1,0} \ldots C_{1,a} \ldots\ldots\ldots\ldots F_{(c+1)r-1} C_{(c+1)r-1,0} \ldots C_{(c+1)r-1a} O_0 O_1 \ldots O_m$$

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

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

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

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