Początki Lispu
(((Archeologia Cyfrowa)))

Jakub Grobelny

19.11.2019

O czym będzie?

1. Do czego był potrzebny taki język?
2. „*Recursive Functions of Symbolic Expressions...*”
3. Pierwsze implementacje
4. Być może coś więcej (Lisp-maszyny, Scheme)

Narodziny sztucznej inteligencji

Narodziny sztucznej inteligencji

Lata 50. XX wieku były czasem, gdy sztuczna inteligencja pojawiła się jako dziedzina wiedzy.

Narodziny sztucznej inteligencji

Lata 50. XX wieku były czasem, gdy sztuczna inteligencja pojawiła się jako dziedzina wiedzy.

1950 Alan Turing publikuje „*Computing Machinery and Intelligence*"

Narodziny sztucznej inteligencji

Lata 50. XX wieku były czasem, gdy sztuczna inteligencja pojawiła się jako dziedzina wiedzy.

1950 Alan Turing publikuje „*Computing Machinery and Intelligence*"

- „*Czy maszyny myślą?*"

Narodziny sztucznej inteligencji

Lata 50. XX wieku były czasem, gdy sztuczna inteligencja pojawiła się jako dziedzina wiedzy.

1950 Alan Turing publikuje „*Computing Machinery and Intelligence*"

- „*Czy maszyny myślą?*"
- „*Czy maszyny mogą myśleć?*"

Narodziny sztucznej inteligencji

Lata 50. XX wieku były czasem, gdy sztuczna inteligencja pojawiła się jako dziedzina wiedzy.

1950 Alan Turing publikuje „*Computing Machinery and Intelligence*"

- ~~„*Czy maszyny myślą?*"~~
- ~~„*Czy maszyny mogą myśleć?*"~~
- „*Czy maszyny mogą działać nieodróżnialnie od ludzi?*"

Narodziny sztucznej inteligencji

Lata 50. XX wieku były czasem, gdy sztuczna inteligencja pojawiła się jako dziedzina wiedzy.

1950 Alan Turing publikuje „*Computing Machinery and Intelligence*"

- ~~„*Czy maszyny myślą?*"~~
- ~~„*Czy maszyny mogą myśleć?*"~~
- „*Czy maszyny mogą działać nieodróżnialnie od ludzi?*"

1951 Marvin Lee Minsky buduje sieć neuronową SNARC[1]

[1] Stochastic neural analog reinforcement calculator

Narodziny sztucznej inteligencji

Lata 50. XX wieku były czasem, gdy sztuczna inteligencja pojawiła się jako dziedzina wiedzy.

1950 Alan Turing publikuje „*Computing Machinery and Intelligence*"
- ~~„*Czy maszyny myślą?*"~~
- ~~„*Czy maszyny mogą myśleć?*"~~
- „*Czy maszyny mogą działać nieodróżnialnie od ludzi?*"

1951 Marvin Lee Minsky buduje sieć neuronową SNARC[1]

1951 Pierwsze programy grające w warcaby (Christopher Strachey) i szachy (Dietrich Prinz)

[1] Stochastic neural analog reinforcement calculator

Narodziny sztucznej inteligencji

1955 Allen Newell, Herbert A. Simon i Cliff Shaw tworzą program „*Logic Theorist*"

Narodziny sztucznej inteligencji

1955 Allen Newell, Herbert A. Simon i Cliff Shaw tworzą program „*Logic Theorist*"

- Program naśladujący ludzkie techniki rozwiązywania problemów

Narodziny sztucznej inteligencji

1955 Allen Newell, Herbert A. Simon i Cliff Shaw tworzą program „*Logic Theorist*"

- Program naśladujący ludzkie techniki rozwiązywania problemów
- Program manipulujący **symbolami**

Narodziny sztucznej inteligencji

1955 Allen Newell, Herbert A. Simon i Cliff Shaw tworzą program „*Logic Theorist*”

- Program naśladujący ludzkie techniki rozwiązywania problemów
- Program manipulujący **symbolami**
- Udowodnił 38 z pierwszych 52 twierdzeń z „*Principia Mathematica*”[2]

[2] Autorstwa Alfreda Northa Whiteheada i Bertranda Russella

Narodziny sztucznej inteligencji

1955 Allen Newell, Herbert A. Simon i Cliff Shaw tworzą program „*Logic Theorist*"

- Program naśladujący ludzkie techniki rozwiązywania problemów
- Program manipulujący **symbolami**
- Udowodnił 38 z pierwszych 52 twierdzeń z „*Principia Mathematica*"[2] (niektóre z dowodów były bardziej eleganckie niż wcześniej istniejące)...

[2] Autorstwa Alfreda Northa Whiteheada i Bertranda Russella

Narodziny sztucznej inteligencji

1955 Allen Newell, Herbert A. Simon i Cliff Shaw tworzą program „*Logic Theorist*"

- Program naśladujący ludzkie techniki rozwiązywania problemów
- Program manipulujący **symbolami**
- Udowodnił 38 z pierwszych 52 twierdzeń z „*Principia Mathematica*"[2] (niektóre z dowodów były bardziej eleganckie niż wcześniej istniejące)...
- ...i to wszystko zanim określenie „sztuczna inteligencja" w ogóle zostało stworzone.

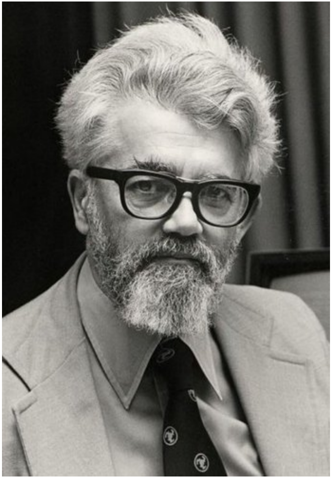
[2] Autorstwa Alfreda Northa Whiteheada i Bertranda Russella

Narodziny sztucznej inteligencji

1956 Konferencja w Dartmouth.

Narodziny sztucznej inteligencji

1956 Konferencja w Dartmouth. **John McCarthy** przekonuje zebranych do używania terminu „*sztuczna inteligencja*".



Rysunek: John McCarthy

Symbole

Symbole

Okazuje się, że operowanie na **symbolach** jest istotne przy tworzeniu sztucznej inteligencji.

Symbole

Okazuje się, że operowanie na **symbolach** jest istotne przy tworzeniu sztucznej inteligencji.

Ludzkie rozumowanie opiera się na manipulacji symbolami (*physical symbol system hypothesis* – Allan Newell i Herbert A. Simon)

Symbole

Okazuje się, że operowanie na **symbolach** jest istotne przy tworzeniu sztucznej inteligencji.

Ludzkie rozumowanie opiera się na manipulacji symbolami (*physical symbol system hypothesis* – Allan Newell i Herbert A. Simon)

System symboli składa się z symboli, składania ich w struktury (wyrażenia) i manipulowania nimi (przetwarzania) w celu tworzenia nowych wyrażeń.

Symbole

W 1959 roku John McCarthy pisze pracę „*Programs with common sense*".

Symbole

W 1959 roku John McCarthy pisze pracę „*Programs with common sense*".

Proponuje w niej stworzenie programu „*Advice Taker*", który rozwiązywałby problemy poprzez manipulację zdaniami (**symbole!**).

Symbole

W 1959 roku John McCarthy pisze pracę „*Programs with common sense*".

Proponuje w niej stworzenie programu „*Advice Taker*", który rozwiązywałby problemy poprzez manipulację zdaniami (**symbole!**).

„*Our ultimate objective is to make programs that learn from their experience as effectively as humans do.*"

„Programs with common sense"

„*Programs with common sense*"

„*A class of entities called terms is defined and a term is an expression. A sequence of expressions is an expression. These expressions are represented in the machine by list structures*" – reprezentacja przesłanek/faktów w postaci **list symboli**

Przykłady:

Przykłady:

$$at(I, desk)$$

$$at(desk, home)$$

$$at(car, home)$$

$$at(home, county)$$

$$at(airport, county)$$

Przykłady:

$$at(I, desk)$$

$$at(desk, home)$$

$$at(car, home)$$

$$at(home, county)$$

$$at(airport, county)$$

$$at(x, y), at(y, z) \rightarrow at(x, z)$$

Przykłady:

$$at(I, desk)$$

$$at(desk, home)$$

$$at(car, home)$$

$$at(home, county)$$

$$at(airport, county)$$

$$at(x, y), at(y, z) \rightarrow at(x, z)$$

$$transitive(at)$$

$$transitive(u) \rightarrow (u(x, y), u(y, z) \rightarrow u(x, z))$$

Przykłady:

$$at(I, desk)$$

$$at(desk, home)$$

$$at(car, home)$$

$$at(home, county)$$

$$at(airport, county)$$

$$at(x, y), at(y, z) \rightarrow at(x, z)$$

$$transitive(at)$$

$$transitive(u) \rightarrow (u(x, y), u(y, z) \rightarrow u(x, z))$$

Prawie jak Prolog?

Zapotrzebowanie na nowe języki

Information Processing Language – niskopozimowy język programowania do manipulowania listami stworzony przez Newella, Shawa i Simona, który posłużył do napisania programu. „*Logic Theorist*".

Zapotrzebowanie na nowe języki

Information Processing Language – niskopozimowy język programowania do manipulowania listami stworzony przez Newella, Shawa i Simona, który posłużył do napisania programu. „*Logic Theorist*".

Dwa rodzaje wyrażeń:

- dane reprezentowane jako listy
- procedury operujące na danych

```
 IPL-V  List
  Structure
   Example
Name SYMB LINK
L1   9-1  100
100  S4   101
101  S5   0
9-1  0    200
200  A1   201
201  V1   202
202  A2   203
203  V2   0
```

Rysunek: Lista zapisana w IPL-V

Information Processing Language

Nowe feature'y:

Information Processing Language

Nowe feature'y:

- Manipulacja listami (tylko listy atomów)

Information Processing Language

Nowe feature'y:

- Manipulacja listami (tylko listy atomów)
- Funkcje wyższego rzędu

Information Processing Language

Nowe feature'y:

- Manipulacja listami (tylko listy atomów)
- Funkcje wyższego rzędu
- Obliczenia na symbolach (tylko litera+liczba)

Information Processing Language

Nowe feature'y:

- Manipulacja listami (tylko listy atomów)
- Funkcje wyższego rzędu
- Obliczenia na symbolach (tylko litera+liczba)

Alternatywy?

Information Processing Language

Nowe feature'y:

- Manipulacja listami (tylko listy atomów)
- Funkcje wyższego rzędu
- Obliczenia na symbolach (tylko litera+liczba)

Alternatywy? Brak.

Wymyślenie Lispu

Wymyślenie Lispu

W 1960 ukazuje się praca Johna McCarthy'ego pt.„*Recursive Functions of Symbolic Expressions Their Computation by Machine, Part I*".

Wymyślenie Lispu

W 1960 ukazuje się praca Johna McCarthy'ego pt.„*Recursive Functions of Symbolic Expressions Their Computation by Machine, Part I*".

1. Introduction

A programming system called LISP (for LISt Processor) has been developed for the IBM 704 computer by the Artificial Intelligence group at M.I.T. The system was designed to facilitate experiments with a proposed system called the Advice Taker, whereby a machine could be instructed to handle declarative as well as imperative sentences and could exhibit "common sense" in carrying out its instructions.

Cele

Cele

- Lisp został wymyślony ze względu na „*Advice Taker'a*".

Cele

- Lisp został wymyślony ze względu na „*Advice Taker'a*".
- Języka programowania do manipulacji wyrażeniami reprezentującymi zdania, przy użyciu których „*Advice Taker*" mógłby przeprowadzać wnioskowania.

Wyrażenia warunkowe

Wyrażenia warunkowe

Po raz pierwszy pojawiła się idea *wyrażeń warunkowych*.

Wyrażenia warunkowe

Po raz pierwszy pojawiła się idea *wyrażeń warunkowych*.

> We shall need a number of mathematical ideas and notations concerning functions in general. Most of the ideas are well known, but the notion of *conditional expression* is believed to be new, and the use of conditional expressions permits functions to be defined recursively in a new and convenient way.

Wyrażenia warunkowe

Po raz pierwszy pojawiła się idea *wyrażeń warunkowych*.

> We shall need a number of mathematical ideas and notations concerning functions in general. Most of the ideas are well known, but the notion of *conditional expression* is believed to be new, and the use of conditional expressions permits functions to be defined recursively in a new and convenient way.

Wyrażenia warunkowe są następującej postaci:

$$(p_1 \to e_1, \dots p_n \to e_n)$$

Wyrażenia warunkowe

Wyrażenia warunkowe są następującej postaci:

$$(p_1 \rightarrow e_1, \ldots p_n \rightarrow e_n)$$

Interpretacja:

Wyrażenia warunkowe

Wyrażenia warunkowe są następującej postaci:

$$(p_1 \rightarrow e_1, \ldots p_n \rightarrow e_n)$$

Interpretacja:
„Jeżeli p_1 to e_1, w przeciwnym razie jeżeli p_2 to e_2, … , w przciwnym razie jeżeli p_n to e_n''

Wyrażenia warunkowe

Wyrażenia warunkowe są następującej postaci:

$$(p_1 \rightarrow e_1, \dots p_n \rightarrow e_n)$$

Interpretacja:
„Jeżeli p_1 to e_1, w przeciwnym razie jeżeli p_2 to e_2, …, w przciwnym razie jeżeli p_n to e_n''

lub

Wyrażenia warunkowe

Wyrażenia warunkowe są następującej postaci:

$$(p_1 \rightarrow e_1, \; ... \; p_n \rightarrow e_n)$$

Interpretacja:
„Jeżeli p_1 to e_1, w przeciwnym razie jeżeli p_2 to e_2, ..., w przciwnym razie jeżeli p_n to e_n"

lub
„p_1 daje e_1, ..., p_n daje e_n"

Wyrażenia warunkowe

Zasady obliczania wartości wyrażeń warunkowych:

Wyrażenia warunkowe

Zasady obliczania wartości wyrażeń warunkowych:

- Rozpatruj *p* od lewej do prawej.

Wyrażenia warunkowe

Zasady obliczania wartości wyrażeń warunkowych:

- Rozpatruj p od lewej do prawej.
- Jeżeli p, którego wartość to T, występuje przed jakimkolwiek innym p, którego wartość jest niezdefiniowana, to wartością wyrażenia warunkowego jest wartość odpowiadającego e (jeżeli jest zdefiniowane).

Wyrażenia warunkowe

Zasady obliczania wartości wyrażeń warunkowych:

- Rozpatruj p od lewej do prawej.
- Jeżeli p, którego wartość to T, występuje przed jakimkolwiek innym p, którego wartość jest niezdefiniowana, to wartością wyrażenia warunkowego jest wartość odpowiadającego e (jeżeli jest zdefiniowane).
- Jeżeli jakiekolwiek niezdefiniowane p jest napotkane przed prawdziwym p, lub gdy wszystkie p są fałszywe, bądź gdy e odpowiadającego pierwszemu prawdziwemu p jest niezdefiniowane, to wartość wyrażenia jest niezdefiniowana.

Wyrażenia warunkowe

Przykłady:

Wyrażenia warunkowe

Przykłady:

$$(2 < 1 \rightarrow 4, T \rightarrow 3) =$$

Wyrażenia warunkowe

Przykłady:

$$(2 < 1 \rightarrow 4, T \rightarrow 3) = 3$$

Wyrażenia warunkowe

Przykłady:

$$(2 < 1 \rightarrow 4, T \rightarrow 3) = 3$$

$$(2 < 1 \rightarrow \frac{0}{0}, T \rightarrow 3) =$$

Wyrażenia warunkowe

Przykłady:

$$(2 < 1 \to 4, T \to 3) = 3$$

$$(2 < 1 \to \frac{0}{0}, T \to 3) = 3$$

Wyrażenia warunkowe

Przykłady:

$$(2 < 1 \to 4, T \to 3) = 3$$

$$(2 < 1 \to \frac{0}{0}, T \to 3) = 3$$

$$(2 < 1 \to 3, T \to \frac{0}{0}) =$$

Wyrażenia warunkowe

Przykłady:

$$(2 < 1 \rightarrow 4, T \rightarrow 3) = 3$$

$$(2 < 1 \rightarrow \frac{0}{0}, T \rightarrow 3) = 3$$

$$(2 < 1 \rightarrow 3, T \rightarrow \frac{0}{0}) = \textit{undefined}$$

Wyrażenia warunkowe

Przykłady:

$$(2 < 1 \rightarrow 4, T \rightarrow 3) = 3$$

$$(2 < 1 \rightarrow \frac{0}{0}, T \rightarrow 3) = 3$$

$$(2 < 1 \rightarrow 3, T \rightarrow \frac{0}{0}) = \mathit{undefined}$$

$$(2 < 1 \rightarrow 3, 4 < 1 \rightarrow 1) =$$

Wyrażenia warunkowe

Przykłady:

$$(2 < 1 \to 4,\ T \to 3) = 3$$

$$(2 < 1 \to \frac{0}{0},\ T \to 3) = 3$$

$$(2 < 1 \to 3,\ T \to \frac{0}{0}) = \textit{undefined}$$

$$(2 < 1 \to 3, 4 < 1 \to 1) = \textit{undefined}$$

Wyrażenia warunkowe

Wyrażenia warunkowe pozwalają na eleganckie zapisywanie funkcji:

Wyrażenia warunkowe

Wyrażenia warunkowe pozwalają na eleganckie zapisywanie funkcji:

$$|x| = (x < 0 \to -x, T \to x)$$

Wyrażenia warunkowe

Wyrażenia warunkowe pozwalają na eleganckie zapisywanie funkcji:

$$|x| = (x < 0 \to -x, T \to x)$$

$$sgn(x) = (x < 0 \to -1, x = 0 \to 0, T \to 1)$$

Funkcje rekurencyjne

Bardzo zwięzły staje się też zapis funkcji rekurencyjnych:

Funkcje rekurencyjne

Bardzo zwięzły staje się też zapis funkcji rekurencyjnych:

$$n! = (n = 0 \to 1, T \to n \cdot (n-1)!)$$

Funkcje rekurencyjne

Bardzo zwięzły staje się też zapis funkcji rekurencyjnych:

$$n! = (n = 0 \rightarrow 1, T \rightarrow n \cdot (n-1)!)$$

$$\begin{gathered} gcd(m,n) = (m > n \rightarrow gcd(n,m), \\ rem(n,m) = 0 \rightarrow m, \\ T \rightarrow gcd(rem(n,m), m)) \end{gathered}$$

Funkcje rekurencyjne

There is no guarantee that the computation determined by a recursive definition will ever terminate and, for example, an attempt to compute n! from our definition will only succeed if n is a non-negative integer. If the computation does not terminate, the function must be regarded as undefined for the given arguments.

Spójniki logiczne

Spójniki logiczne

$$p \wedge q = (p \rightarrow q, T \rightarrow F)$$

$$p \vee q = (p \rightarrow T, T \rightarrow q)$$

$$\sim p = (p \rightarrow F, T \rightarrow T)$$

$$p \Rightarrow q = (p \rightarrow q, T \rightarrow T)$$

Spójniki logiczne

It is readily seen that the right-hand sides of the equations have the correct truth tables. If we consider situations in which p or q may be undefined, the connectives $\wedge$ and $\vee$ are seen to be noncommutative. For example if p is false and q is undefined, we see that according to the definitions given above p $\wedge$ q is false, but q $\wedge$ p is undefined. For our applications this noncommutativity is desirable, since p $\wedge$ q is computed by first computing p, and if p is false q is not computed. If the computation for p does not terminate, we never get around to computing q.

Rysunek: Ewaluacja leniwa?

Spójniki logiczne

$$p \wedge q = (p \rightarrow q, T \rightarrow F)$$

Argumenty funkcji „$\wedge$” nie są poddawane ewaluacji przed jej aplikacją.

Formy a funkcje

Formy a funkcje

Pojęcie formy zostało zapożyczone z rachunku lambda Churcha.

Formy a funkcje

Pojęcie formy zostało zapożyczone z rachunku lambda Churcha.

$y^2 + x$ – forma

Formy a funkcje

Pojęcie formy zostało zapożyczone z rachunku lambda Churcha.

$y^2 + x$ – forma
$(y^2 + x)(3, 4)$ – źle. Nie wiadomo czy wartością powinno być 13 czy 19.

Formy a funkcje

Pojęcie formy zostało zapożyczone z rachunku lambda Churcha.

$y^2 + x$ – forma
$(y^2 + x)(3, 4)$ – źle. Nie wiadomo czy wartością powinno być 13 czy 19.

Formę $\mathcal{E}$ możemy skonwertować na funkcję jeżeli ustalimy powiązanie między zmiennymi w formie a uporządkowaną listą argumentów funkcji:

Formy a funkcje

Pojęcie formy zostało zapożyczone z rachunku lambda Churcha.

$y^2 + x$ – forma
$(y^2 + x)(3, 4)$ – źle. Nie wiadomo czy wartością powinno być 13 czy 19.

Formę $\mathcal{E}$ możemy skonwertować na funkcję jeżeli ustalimy powiązanie między zmiennymi w formie a uporządkowaną listą argumentów funkcji:

$\lambda((x_1, \ldots, x_n), \mathcal{E}))$ – funkcja

Formy a funkcje

Pojęcie formy zostało zapożyczone z rachunku lambda Churcha.

$y^2 + x$ – forma
$(y^2 + x)(3, 4)$ – źle. Nie wiadomo czy wartością powinno być 13 czy 19.

Formę $\mathcal{E}$ możemy skonwertować na funkcję jeżeli ustalimy powiązanie między zmiennymi w formie a uporządkowaną listą argumentów funkcji:

$\lambda((x_1, \dots, x_n), \mathcal{E}))$ – funkcja
$\lambda((x, y), y^2 + x)$ – też funkcja

Rekurencyjne λ-wyrażenia

Rekurencyjne λ-wyrażenia

$$sqrt = \lambda((a, x, \epsilon),$$
$$(|x^2 - a| < \epsilon \to x, T \to sqrt(a, \frac{1}{2}(x + \frac{a}{x}), \epsilon)))$$

Rekurencyjne λ-wyrażenia

$$sqrt = \lambda((a, x, \epsilon),$$
$$(|x^2 - a| < \epsilon \to x, T \to sqrt(a, \frac{1}{2}(x + \frac{a}{x}), \epsilon)))$$

Prawa strona wyrażenia nie może być wyrażeniem dla tej funkcji, bo nic nie wskazuje na to, że *sqrt* do odnosi się do całego wyrażenia (*sqrt* nie jest związane).

Rekurencyjne λ-wyrażenia

$$sqrt = \lambda((a, x, \epsilon),$$
$$(|x^2 - a| < \epsilon \rightarrow x, T \rightarrow sqrt(a, \frac{1}{2}(x + \frac{a}{x}), \epsilon)))$$

Prawa strona wyrażenia nie może być wyrażeniem dla tej funkcji, bo nic nie wskazuje na to, że *sqrt* do odnosi się do całego wyrażenia (*sqrt* nie jest związane).

Operator punktu stałego?

Rekurencyjne λ-wyrażenia

$$sqrt = \lambda((a, x, \epsilon),$$
$$(|x^2 - a| < \epsilon \rightarrow x, T \rightarrow sqrt(a, \frac{1}{2}(x + \frac{a}{x}), \epsilon)))$$

Prawa strona wyrażenia nie może być wyrażeniem dla tej funkcji, bo nic nie wskazuje na to, że *sqrt* do odnosi się do całego wyrażenia (*sqrt* nie jest związane).

Operator punktu stałego? – zbyt długie i nieczytelne wyrażenia.

Rekurencyjne λ-wyrażenia

Nowa notacja:

Rekurencyjne λ-wyrażenia

Nowa notacja:

$$label(a, \mathcal{E})$$

Rekurencyjne λ-wyrażenia

Nowa notacja:

$$\mathit{label}(a, \mathcal{E})$$

Wyrażenie $\mathcal{E}$, w którym wszystkie wystąpienia a interpretowane są jako odniesienia do całego wyrażenia $\mathcal{E}$.

Rekurencyjne λ-wyrażenia

Nowa notacja:

$$label(a, \mathcal{E})$$

Wyrażenie $\mathcal{E}$, w którym wszystkie wystąpienia a interpretowane są jako odniesienia do całego wyrażenia $\mathcal{E}$.

S-wyrażenia

S-wyrażenia

„*A Class of Symbolic Expressions*".

S-wyrażenia

„*A Class of Symbolic Expressions*”.

Dopuszczalne znaki:

- .

S-wyrażenia

„*A Class of Symbolic Expressions*”.

Dopuszczalne znaki:

- .
-)

S-wyrażenia

„*A Class of Symbolic Expressions*”.

Dopuszczalne znaki:

- .
-)
- (

S-wyrażenia

„*A Class of Symbolic Expressions*".

Dopuszczalne znaki:

- .
-)
- (
- Nieskończony zbiór rozróżnialnych symboli atomowych – napisów złożonych z wielkich liter alfabetu łacińskiego, cyfr i pojedynczych spacji.

S-wyrażenia

„*A Class of Symbolic Expressions*”.

Dopuszczalne znaki:

- .
-)
- (
- Nieskończony zbiór rozróżnialnych symboli atomowych – napisów złożonych z wielkich liter alfabetu łacińskiego, cyfr i pojedynczych spacji.
  Na przykład:
  - A
  - ABA
  - APPLE PIE NUMBER 3

S-wyrażenia

Czemu symbole składające się z wielu znaków?

S-wyrażenia

Czemu symbole składające się z wielu znaków?

Odpowiedź:

- IBM 704 miał tylko 47 drukowalnych znaków

S-wyrażenia

Czemu symbole składające się z wielu znaków?

Odpowiedź:

- IBM 704 miał tylko 47 drukowalnych znaków
- Nazywanie atomowych bytów angielskimi słowami i zdaniami jest wygodne

S-wyrażenia

Definicja indukcyjna S-wyrażeń:

S-wyrażenia

Definicja indukcyjna S-wyrażeń:

1. Symbole atomowe są S-wyrażeniami

S-wyrażenia

Definicja indukcyjna S-wyrażeń:

1. Symbole atomowe są S-wyrażeniami
2. Jeżeli e_1 i e_2 są S-wyrażeniami, to $(e_1 \cdot e_2)$ również jest S-wyrażeniem.

S-wyrażenia

Definicja indukcyjna S-wyrażeń:

1. Symbole atomowe są S-wyrażeniami
2. Jeżeli e_1 i e_2 są S-wyrażeniami, to $(e_1 \cdot e_2)$ również jest S-wyrażeniem.

Przykładowe S-wyrażenia:

$$AB$$

$$(A \cdot B)$$

$$((AB \cdot C) \cdot D)$$

Listy (S-wyrażenia)

Możemy reprezentować listy dowolnej długości przy użyciu S-wyrażeń w następujący sposób:

Listy (S-wyrażenia)

Możemy reprezentować listy dowolnej długości przy użyciu S-wyrażeń w następujący sposób:

$$(m_1, m_2, \ldots m_n) = (m_1 \cdot (m_2 \cdot (\ldots(m_n \cdot NIL)\ldots)))$$

Listy (S-wyrażenia)

Możemy reprezentować listy dowolnej długości przy użyciu S-wyrażeń w następujący sposób:

$(m_1, m_2, \dots m_n) = (m_1 \cdot (m_2 \cdot (\dots(m_n \cdot NIL)\dots)))$
Gdzie NIL jest specjalnym symbolem kończącym listy.

Listy (S-wyrażenia)

Możemy reprezentować listy dowolnej długości przy użyciu S-wyrażeń w następujący sposób:

$(m_1, m_2, \dots m_n) = (m_1 \cdot (m_2 \cdot (\dots(m_n \cdot NIL)\dots)))$
Gdzie NIL jest specjalnym symbolem kończącym listy.

Wprowadzamy zatem specjalną notację dla list:

Listy (S-wyrażenia)

Możemy reprezentować listy dowolnej długości przy użyciu S-wyrażeń w następujący sposób:

$(m_1, m_2, \dots m_n) = (m_1 \cdot (m_2 \cdot (\dots(m_n \cdot NIL)\dots)))$
Gdzie NIL jest specjalnym symbolem kończącym listy.

Wprowadzamy zatem specjalną notację dla list:

1. $(m) = (m \cdot NIL)$
2. $(m_1, m_2, \dots m_n) = (m_1 \cdot (\dots(m_n \cdot NIL)\dots))$
3. $(m_1, \dots, m_n \cdot x) = (m_1 \cdot (\dots(m_n \cdot x)\dots))$

M-wyrażenia

M-wyrażenia

Aby odróżnić wyrażenia reprezentujące aplikację funkcji od S-wyrażeń, będziemy małych liter do nazywania funkcji i zmiennych.

M-wyrażenia

Aby odróżnić wyrażenia reprezentujące aplikację funkcji od S-wyrażeń, będziemy małych liter do nazywania funkcji i zmiennych.

Będziemy również używać nawiasów kwadratowych i średników zamiast nawiasów i przecinków.

M-wyrażenia

Aby odróżnić wyrażenia reprezentujące aplikację funkcji od S-wyrażeń, będziemy małych liter do nazywania funkcji i zmiennych.

Będziemy również używać nawiasów kwadratowych i średników zamiast nawiasów i przecinków.
Na przykład:

$$car[x]$$

$$car[cons[(A \cdot B); x]]$$

M-wyrażenia

Aby odróżnić wyrażenia reprezentujące aplikację funkcji od S-wyrażeń, będziemy małych liter do nazywania funkcji i zmiennych.

Będziemy również używać nawiasów kwadratowych i średników zamiast nawiasów i przecinków.
Na przykład:

$$car[x]$$

$$car[cons[(A \cdot B); x]]$$

M-wyrażenia – meta-wyrażenia

Funkcje i predykaty dla S-wyrażeń

Funkcje i predykaty dla S-wyrażeń

$atom[x]$ – prawda tylko jeżeli x jest symbolem

$eq[x; y]$ – prawda tylko jeżeli x i y są tym samym symbolem

$car[x]$ – zdefiniowany tylko jeżeli x nie jest symbolem.

Funkcje i predykaty dla S-wyrażeń

$atom[x]$ – prawda tylko jeżeli x jest symbolem

$eq[x; y]$ – prawda tylko jeżeli x i y są tym samym symbolem

$car[x]$ – zdefiniowany tylko jeżeli x nie jest symbolem.

$$car[(X \cdot A)] = X$$
$$car[((X \cdot A) \cdot Y)] = (X \cdot A)$$

Funkcje i predykaty dla S-wyrażeń

$atom[x]$ – prawda tylko jeżeli x jest symbolem

$eq[x; y]$ – prawda tylko jeżeli x i y są tym samym symbolem

$car[x]$ – zdefiniowany tylko jeżeli x nie jest symbolem.

$$car[(X \cdot A)] = X$$
$$car[((X \cdot A) \cdot Y)] = (X \cdot A)$$

$cdr[x]$ – zdefiniowany tylko jeżeli x nie jest symbolem

Funkcje i predykaty dla S-wyrażeń

$atom[x]$ – prawda tylko jeżeli x jest symbolem

$eq[x; y]$ – prawda tylko jeżeli x i y są tym samym symbolem

$car[x]$ – zdefiniowany tylko jeżeli x nie jest symbolem.

$$car[(X \cdot A)] = X$$
$$car[((X \cdot A) \cdot Y)] = (X \cdot A)$$

$cdr[x]$ – zdefiniowany tylko jeżeli x nie jest symbolem

$$cdr[(X \cdot A)] = A$$
$$cdr[((X \cdot A) \cdot Y)] = Y$$

Funkcje i predykaty dla S-wyrażeń

$atom[x]$ – prawda tylko jeżeli x jest symbolem

$eq[x; y]$ – prawda tylko jeżeli x i y są tym samym symbolem

$car[x]$ – zdefiniowany tylko jeżeli x nie jest symbolem.

$$car[(X \cdot A)] = X$$
$$car[((X \cdot A) \cdot Y)] = (X \cdot A)$$

$cdr[x]$ – zdefiniowany tylko jeżeli x nie jest symbolem

$$cdr[(X \cdot A)] = A$$
$$cdr[((X \cdot A) \cdot Y)] = Y$$

$cons[x; y] = (e_1 \cdot e_2)$

Funkcje i predykaty dla S-wyrażeń

Widać, że *car*, *cdr* i *cons* spełniają poniższe zależności:

Funkcje i predykaty dla S-wyrażeń

Widać, że car, cdr i $cons$ spełniają poniższe zależności:

$car[cons[x; y]] = x$

$cdr[cons[x; y]] = y$

$cons[car[x]; cdr[x]] = x$ (pod warunkiem, że x nie jest atomowe)

Funkcje i predykaty dla S-wyrażeń

Dalej McCarthy opisuje różne funkcje, które można zapisać przy użyciu wcześniej opisanej notacji.

- $append[x; y]$
- $among[x; y]$ – czy y zawiera x?
- $pair[x; y]$ – `zip`
- $assoc[x; y]$ – szukanie klucza x na liście par
- $sublis[x; y]$ – mamy listę symboli y pod które chcemy podstawić przypisane im wartości w liście x.

Tłumaczenie M-wyrażeń na S-wyrażenia

Chcemy przetłumaczyć M-wyrażenie $\mathcal{E}$ na S-wyrażenie $\mathcal{E}^*$

Tłumaczenie M-wyrażeń na S-wyrażenia

Chcemy przetłumaczyć M-wyrażenie $\mathcal{E}$ na S-wyrażenie $\mathcal{E}^*$

1. Jeżeli $\mathcal{E}$ jest S-wyrażeniem, to $\mathcal{E}^* = (QUOTE, \mathcal{E})$

Tłumaczenie M-wyrażeń na S-wyrażenia

Chcemy przetłumaczyć M-wyrażenie $\mathcal{E}$ na S-wyrażenie $\mathcal{E}^*$

1. Jeżeli $\mathcal{E}$ jest S-wyrażeniem, to $\mathcal{E}^* = (QUOTE, \mathcal{E})$
2. Zmienne i nazwy funkcji napisane małymi literami zamieniamy na odpowiadające napisy złożone z wielkich liter (np. $car^* = CAR$).

Tłumaczenie M-wyrażeń na S-wyrażenia

Chcemy przetłumaczyć M-wyrażenie $\mathcal{E}$ na S-wyrażenie $\mathcal{E}^*$

1. Jeżeli $\mathcal{E}$ jest S-wyrażeniem, to $\mathcal{E}^* = (QUOTE, \mathcal{E})$
2. Zmienne i nazwy funkcji napisane małymi literami zamieniamy na odpowiadające napisy złożone z wielkich liter (np. $car^* = CAR$).
3. Forma $f[e_1; \dots; e_n] = (f^*, e_1^*, \dots, e_n^*)$

Tłumaczenie M-wyrażeń na S-wyrażenia

Chcemy przetłumaczyć M-wyrażenie $\mathcal{E}$ na S-wyrażenie $\mathcal{E}^*$

1. Jeżeli $\mathcal{E}$ jest S-wyrażeniem, to $\mathcal{E}^* = (QUOTE, \mathcal{E})$
2. Zmienne i nazwy funkcji napisane małymi literami zamieniamy na odpowiadające napisy złożone z wielkich liter (np. $car^* = CAR$).
3. Forma $f[e_1; \ldots; e_n] = (f^*, e_1^*, \ldots, e_n^*)$
4. $\{[p_1 \to e_1; \ldots; p_n \to e_n]\}^* = (COND, (p_1^*, e_1^*), \ldots (p_n^*, e_n^*))$

Tłumaczenie M-wyrażeń na S-wyrażenia

Chcemy przetłumaczyć M-wyrażenie $\mathcal{E}$ na S-wyrażenie $\mathcal{E}^*$

1. Jeżeli $\mathcal{E}$ jest S-wyrażeniem, to $\mathcal{E}^* = (QUOTE, \mathcal{E})$
2. Zmienne i nazwy funkcji napisane małymi literami zamieniamy na odpowiadające napisy złożone z wielkich liter (np. $car^* = CAR$).
3. Forma $f[e_1; \dots; e_n] = (f^*, e_1^*, \dots, e_n^*)$
4. $\{[p_1 \to e_1; \dots; p_n \to e_n]\}^* = (COND, (p_1^*, e_1^*), \dots (p_n^*, e_n^*))$
5. $\{[\lambda[[x_1; \dots; x_n]; \mathcal{E}]]\}^* = (LAMBDA, (x_1^*, \dots, x_n^*), \mathcal{E}^*)$

Tłumaczenie M-wyrażeń na S-wyrażenia

Chcemy przetłumaczyć M-wyrażenie $\mathcal{E}$ na S-wyrażenie $\mathcal{E}^*$

1. Jeżeli $\mathcal{E}$ jest S-wyrażeniem, to $\mathcal{E}^* = (QUOTE, \mathcal{E})$
2. Zmienne i nazwy funkcji napisane małymi literami zamieniamy na odpowiadające napisy złożone z wielkich liter (np. $car^* = CAR$).
3. Forma $f[e_1; \ldots; e_n] = (f^*, e_1^*, \ldots, e_n^*)$
4. $\{[p_1 \to e_1; \ldots; p_n \to e_n]\}^* = (COND, (p_1^*, e_1^*), \ldots (p_n^*, e_n^*))$
5. $\{[\lambda[[x_1; \ldots; x_n]; \mathcal{E}]]\}^* = (LAMBDA, (x_1^*, \ldots, x_n^*), \mathcal{E}^*)$
6. $label[a; \mathcal{E}]^* = (LABEL, a^*, \mathcal{E}^*)$

S-wyrażenia

(LABEL, SUBST, (LAMBDA, (X, Y, Z), (COND ((ATOM, Z), (COND, (EQ, Y, Z), X), ((QUOTE, T), Z))), ((QUOTE, T), (CONS, (SUBST, X, Y, (CAR Z)), (SUBST, X, Y, (CDR, Z)))))))

S-wyrażenia

(LABEL, SUBST, (LAMBDA, (X, Y, Z), (COND ((ATOM, Z), (COND, (EQ, Y, Z), X), ((QUOTE, T), Z))), ((QUOTE, T), (CONS, (SUBST, X, Y, (CAR Z)), (SUBST, X, Y, (CDR, Z)))))))

This notation is writable and somewhat readable. It can be made easier to read and write at the cost of making its structure less regular. If more characters were available on the computer, it could be improved considerably.

apply

Funkcja *apply* aplikuje S-wyrażenie f reprezentujące S-funkcję f' do listy argumentów *args* postaci $(arg_1 \ldots arg_n)$.

apply

Funkcja *apply* aplikuje S-wyrażenie f reprezentujące S-funkcję f' do listy argumentów *args* postaci $(arg_1 \dots arg_n)$.

$$apply[f; args] = eval[cons[f; appq[args]]; NIL]$$

gdzie

$$\begin{gathered} appq[m] = \\ [null[m] \to NIL; \\ T \to cons[list[QUOTE; car[m]]; appq[cdr[m]]]] \end{gathered}$$

eval

$$eval[e; a] = [$$

eval

$$eval[e; a] = [$$
$$atom[e] \rightarrow assoc[e; a]$$

eval

$$eval[e; a] = [$$
$$\quad atom[e] \rightarrow assoc[e; a]$$
$$\quad atom[car[e]] \rightarrow [$$

eval

$$
\begin{aligned}
&eval[e; a] = [\\
&\quad atom[e] \rightarrow assoc[e; a]\\
&\quad atom[car[e]] \rightarrow [\\
&\quad\quad eq[car[e]; QUOTE] \rightarrow cadr[e];
\end{aligned}
$$

eval

$$
\begin{aligned}
&eval[e; a] = [\\
&\quad atom[e] \to assoc[e; a]\\
&\quad atom[car[e]] \to [\\
&\qquad eq[car[e]; QUOTE] \to cadr[e];\\
&\qquad eq[car[e]; ATOM] \to atom[eval[cadr[e]; a]];
\end{aligned}
$$

eval

$$
\begin{aligned}
&eval[e; a] = [\\
&\quad atom[e] \rightarrow assoc[e; a]\\
&\quad atom[car[e]] \rightarrow [\\
&\quad\quad eq[car[e]; QUOTE] \rightarrow cadr[e];\\
&\quad\quad eq[car[e]; ATOM] \rightarrow atom[eval[cadr[e]; a]];\\
&\quad\quad eq[car[e]; EQ] \rightarrow eval[cadr[e]; a] = eval[caddr[e]; a];
\end{aligned}
$$

eval

$$
\begin{aligned}
& eval[e; a] = [\\
& \quad atom[e] \to assoc[e; a] \\
& \quad atom[car[e]] \to [\\
& \quad\quad eq[car[e]; QUOTE] \to cadr[e]; \\
& \quad\quad eq[car[e]; ATOM] \to atom[eval[cadr[e]; a]]; \\
& \quad\quad eq[car[e]; EQ] \to eval[cadr[e]; a] = eval[caddr[e]; a]; \\
& \quad\quad eq[car[e]; COND] \to evcon[cdr[e]; a];
\end{aligned}
$$

eval

$$
\begin{aligned}
&eval[e; a] = [\\
&\quad atom[e] \to assoc[e; a]\\
&\quad atom[car[e]] \to [\\
&\qquad eq[car[e]; QUOTE] \to cadr[e];\\
&\qquad eq[car[e]; ATOM] \to atom[eval[cadr[e]; a]];\\
&\qquad eq[car[e]; EQ] \to eval[cadr[e]; a] = eval[caddr[e]; a];\\
&\qquad eq[car[e]; COND] \to evcon[cdr[e]; a];\\
&\qquad eq[car[e]; CAR] \to car[eval[cadr[e]; a]];
\end{aligned}
$$

eval

$$
\begin{aligned}
&eval[e; a] = [\\
&\quad atom[e] \rightarrow assoc[e; a]\\
&\quad atom[car[e]] \rightarrow [\\
&\qquad eq[car[e]; QUOTE] \rightarrow cadr[e];\\
&\qquad eq[car[e]; ATOM] \rightarrow atom[eval[cadr[e]; a]];\\
&\qquad eq[car[e]; EQ] \rightarrow eval[cadr[e]; a] = eval[caddr[e]; a];\\
&\qquad eq[car[e]; COND] \rightarrow evcon[cdr[e]; a];\\
&\qquad eq[car[e]; CAR] \rightarrow car[eval[cadr[e]; a]];\\
&\qquad eq[car[e]; CDR] \rightarrow cdr[eval[cadr[e]; a]];
\end{aligned}
$$

eval

$$
\begin{aligned}
&eval[e; a] = [\\
&\quad atom[e] \to assoc[e; a]\\
&\quad atom[car[e]] \to [\\
&\quad\quad eq[car[e]; QUOTE] \to cadr[e];\\
&\quad\quad eq[car[e]; ATOM] \to atom[eval[cadr[e]; a]];\\
&\quad\quad eq[car[e]; EQ] \to eval[cadr[e]; a] = eval[caddr[e]; a];\\
&\quad\quad eq[car[e]; COND] \to evcon[cdr[e]; a];\\
&\quad\quad eq[car[e]; CAR] \to car[eval[cadr[e]; a]];\\
&\quad\quad eq[car[e]; CDR] \to cdr[eval[cadr[e]; a]];\\
&\quad\quad eq[car[e]; CONS] \to\\
&\quad\quad\quad cons[eval[cadr[e]; a]; eval[caddr[e]; a]];
\end{aligned}
$$

eval

$$
\begin{aligned}
&eval[e; a] = [\\
&\quad atom[e] \rightarrow assoc[e; a]\\
&\quad atom[car[e]] \rightarrow [\\
&\quad\quad eq[car[e]; QUOTE] \rightarrow cadr[e];\\
&\quad\quad eq[car[e]; ATOM] \rightarrow atom[eval[cadr[e]; a]];\\
&\quad\quad eq[car[e]; EQ] \rightarrow eval[cadr[e]; a] = eval[caddr[e]; a];\\
&\quad\quad eq[car[e]; COND] \rightarrow evcon[cdr[e]; a];\\
&\quad\quad eq[car[e]; CAR] \rightarrow car[eval[cadr[e]; a]];\\
&\quad\quad eq[car[e]; CDR] \rightarrow cdr[eval[cadr[e]; a]];\\
&\quad\quad eq[car[e]; CONS] \rightarrow\\
&\quad\quad\quad cons[eval[cadr[e]; a]; eval[caddr[e]; a]];\\
&\quad\quad T \rightarrow eval[cons[assoc[car[e]; a]; evlis[cdr[e]; a]]; a]];
\end{aligned}
$$

eval cd.

$$
\begin{array}{l}
eq[caar[e]; LABEL] \rightarrow \\
\quad [eval[cons[caddar[e]; cdr[e]]; \\
\quad cons[list[cadar[e]; car[e]; a]]];
\end{array}
$$

eval cd.

$$
\begin{aligned}
&eq[caar[e]; LABEL] \rightarrow \\
&\quad [eval[cons[caddar[e]; cdr[e]]; \\
&\quad cons[list[cadar[e]; car[e]; a]]; \\
&eq[caar[e]; LAMBDA] \rightarrow \\
&\quad eval[caddar[e]; append[pair[cadar[e]; evlis[cdr[e]; a]; a]]
\end{aligned}
$$

]

eval cd.

$$
\begin{aligned}
&eq[caar[e]; LABEL] \rightarrow \\
&\quad [eval[cons[caddar[e]; cdr[e]]; \\
&\quad cons[list[cadar[e]; car[e]; a]]; \\
&eq[caar[e]; LAMBDA] \rightarrow \\
&\quad eval[caddar[e]; append[pair[cadar[e]; evlis[cdr[e]; a]; a]] \\
]&
\end{aligned}
$$

gdzie

$evcon[e; a] = [eval[caar[e]; a] \rightarrow eval[cadar[e]; a]; T \rightarrow evcon[cdr[e]; a]]$

$evlis[m; a] = [null[m] \rightarrow NIL; T \rightarrow cons[eval[car[m]; a]; evlis[cdr[m]; a]]]$

Funkcje wyższego rzędu

Funkcje wyższego rzędu

g. *Functions with Functions as Arguments.* There are a number of useful functions some of whose arguments are functions. They are especially useful in defining other functions.

Funkcje wyższego rzędu

g. *Functions with Functions as Arguments.* There are a number of useful functions some of whose arguments are functions. They are especially useful in defining other functions.

$$
\begin{aligned}
&maplist[x; f] = [\\
&\quad null[x] \rightarrow NIL;\\
&\quad T \rightarrow cons[f[x]; maplist[cdr[x]; f]]]
\end{aligned}
$$

Funkcje wyższego rzędu

g. *Functions with Functions as Arguments.* There are a number of useful functions some of whose arguments are functions. They are especially useful in defining other functions.

$$
\begin{aligned}
&maplist[x; f] = [\\
&\quad null[x] \to NIL;\\
&\quad T \to cons[f[x]; maplist[cdr[x]; f]]]
\end{aligned}
$$

$$
\begin{aligned}
&search[x; p; f; u] = [\\
&\quad null[x] \to u;\\
&\quad p[x] \to f[x];\\
&\quad T \to search[cdr[x]; p; f; u]]
\end{aligned}
$$

diff [y; x] = [atom [y] → [eq [y; x] → ONE; T → ZERO]; eq [car [y]; PLUS] → cons [PLUS; maplist [cdr [y]; λ[[z]; diff[car [z]; x]]]]; eq[car [y]; TIMES] → cons[PLUS; maplist[cdr[y]; λ[[z]; cons [TIMES; maplist[cdr [y]; λ[[w]; ∼eq [z; w] → car [w]; T → diff [car [[w]; x]]]]]]]]

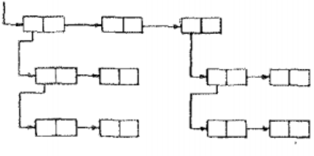
The derivative of the allowed expression, as computed by this formula, is

(PLUS, (TIMES, ONE, (PLUS, X, A), Y),
(TIMES, X, (PLUS, ONE, ZERO), Y),
(TIMES, X, (PLUS, X, A), ZERO))

Rysunek: Funkcja obliczająca pochodne funkcji zaaplikowana do wyrażenia $(TIMES, X, (PLUS, X, A), Y)$

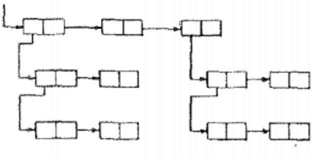
Reprezentacja S-wyrażeń w pamięci

Reprezentacja S-wyrażeń w pamięci



Lista jest zbiorem słów maszynowych zaaranżowanych w sposób podobny jak na powyższym rysunku. Każde słowo jest reprezentowane jako jeden z podzielonych na pół prostokątów.

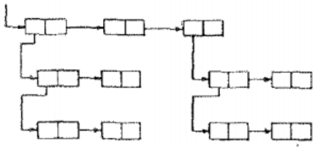
Reprezentacja S-wyrażeń w pamięci



Lista jest zbiorem słów maszynowych zaaranżowanych w sposób podobny jak na powyższym rysunku. Każde słowo jest reprezentowane jako jeden z podzielonych na pół prostokątów.

Każde słowo dzieli się na dwa pola: lewe *address* oraz prawe *decrement*.

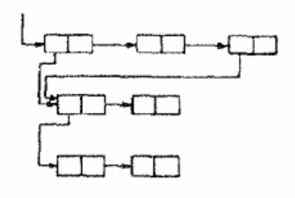
Reprezentacja S-wyrażeń w pamięci



Lista jest zbiorem słów maszynowych zaaranżowanych w sposób podobny jak na powyższym rysunku. Każde słowo jest reprezentowane jako jeden z podzielonych na pół prostokątów.

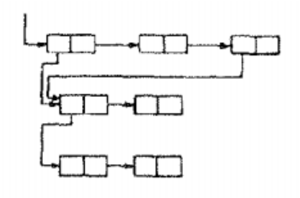
Każde słowo dzieli się na dwa pola: lewe *address* oraz prawe *decrement*.

Reprezentacja S-wyrażeń w pamięci

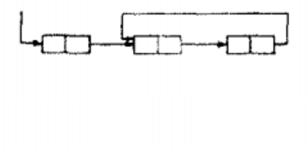


Jedna podstruktura może pojawiać się więcej niż w jednym miejscu w strukturze...

Reprezentacja S-wyrażeń w pamięci



Jedna podstruktura może pojawiać się więcej niż w jednym miejscu w strukturze...



...ale struktury nie mogą zawierać cyklów.

Reprezentacja S-wyrażeń cd.