Gramatyki (1-2)

Definiowanie języków programowania

Zagadnienia

- Jak zdefiniować język programowania?
- Gramatyki formalne
- Definiowanie składni
- Definiowanie semantyki

Pożądane cechy języka programowania

- Język programowania musi mieć
  - jednoznaczną składnię – trzeba móc bez pudła stwierdzić, co jest a co nie jest poprawnym programem
  - jednoznaczną semantykę – trzeba każdą konstrukcję języka programowania jednoznacznie zinterpretować. Przykład typowego problemu niejednoznaczności semantycznej:
    - jeśli jutro będzie dobra pogoda to jeśli będzie z nami Wojtek, to pojedziemy na ryby, a w przeciwnym razie pójdziemy do kina.

Problem z komputerem

- Człowiek często jest w stanie na podstawie kontekstu wychwycić intencje, nawet jeśli przekaz jest niejednoznaczny.
- Komputer jest na to zbyt głupi.
- Język naturalny siłą rzeczy jest niejednoznaczny.
- Język programowania taki być nie może.

Standardy

- Każdy język programowania ma dialekty związane z konkretnymi architekturami, kompilatorami, środowiskami, ale powinien mieć jeden standard.
- Program napisany w jednym dialekcie może być niezrozumiany w innym. Ważne jest pisanie w standardzie, bo wszystkie dialekty powinny być z nim zgodne.

Definiowanie języka programowania

- Język programowania w odróżnieniu od języka naturalnego jest tworem sztucznym.
- Odwieczny problem języków naturalnych: określić dla nich gramatykę. Dla wszystkich naturalnych języków jest to problem do dziś nierozwiązany.
- Języki sztuczne – w tym języki programowania – wychodzą od drugiej strony: określa się dla nich gramatykę i za poprawne uznaje tylko to, co z tej gramatyki można wyprowadzić. (Noam Chomsky)

Składnia i semantyka

- Definicję języka programowania rozbijemy na dwa etapy: określenie składni i nadanie znaczenia konstrukcjom języka, czyli semantyki.
- Do obydwu tych rzeczy użyjemy tego samego narzędzia: gramatyk

Gramatyki bezkontekstowe

- Gramatyka jest to czwórka <N,T,P,A>, gdzie
  - N – zbiór symboli pomocniczych (nieterminalnych)
  - T – zbior symboli końcowych (terminalnych)
  - P – zbior produkcji, czyli reguł postaci S→ w, gdzie S jest symbolem pomocniczym, a w jest słowem (ciągiem) zbudowanym zarówno z symboli pomocniczych, jak i końcowych (być może tylko jednych z nich).
  - A – symbol początkowy zwany aksjomatem gramatyki (A jest symbolem początkowym)

Oznaczenia

- Wprowadzamy następujące oznaczenia:
  - ε – puste słowo
  - Jeśli jeden symbol pomocniczy ma wiele produkcji go dotyczących, to warianty oddzielamy kreskami pionowymi |.
  - T* – wszystkie słowa nad alfabetem T, czyli złożone z symboli występujących w T, np. dla T={a,b} zbiór T*={ε,a,b,aa,ab,ba,bb,aaa,aab,aba,abb,baa,...}
  - Dla dowolnych słów u,v definiujemy ich konkatenację uv, jako dopisanie do u słowa v, np dla u=ab, v=ba mamy uv=abba. Oczywiście dla każdego słowa u zachodzi uε=εu=u, bo dopisanie pustego słowa z dowolnej strony niczego nie zmienia.

Przykład

- Gramatyka parzystych palindromów, czyli słów o parzystej długości, czytanych tak samo wprost i wspak: $G=\langle\{S\},\{a,b\},\{S \rightarrow aSa \mid bSb \mid \varepsilon\},S\rangle$

Przykładowe wyprowadzenie słowa w tej gramatyce:

$S \rightarrow aSa \rightarrow abSba \rightarrow abba$

Język generowany przez tę gramatykę:

$L(G) = \{\varepsilon, aa, bb, aaaa, abba, baab, bbbb, aaaaaa, \ldots\}$

Przykład 2

- Gramatyka wszystkich palindromów, czyli słów czytanych tak samo wprost i wspak: $G = \langle \{S\}, \{a,b\}, \{S \rightarrow aSa \mid bSb \mid a \mid b \mid \varepsilon\}, S \rangle$

Przykładowe wyprowadzenie słowa w tej gramatyce:

$S \rightarrow aSa \rightarrow abSba \rightarrow ababa$

Język generowany przez tę gramatykę:

$L(G) = \{\varepsilon, a, b, aa, bb, aaa, aba, bab, bbb, aaaa, abba, baab, bbbb, aaaaaa, \ldots\}$

Reguły wyprowadzania

- Zaczynamy od aksjomatu będącego zawsze symbolem pomocniczym.
- Jeżeli w wyprowadzanym słowie występuje jakiś symbol pomocniczy, to można go zastąpić przez prawą stronę produkcji dotyczącej tego symbolu.
- Wyprowadzanie kończymy dopiero wtedy, gdy pozbędziemy się wszystkich symboli pomocniczych.
- Język generowany przez gramatykę, to zbiór słów, które można wyprowadzić z aksjomatu.

Przykład 3

- Gramatyka generująca wszystkie liczby podzielne przez 3:
  - G=<{Z,J,D},{0,1,2,3,4,5,6,7,8,9},P,Z>, gdzie

    P={Z→0 | 3 | 6 | 9 | Z0 |Z3 | Z6 | Z9 | J2 | J5 | J8 | D1 | D4 | D7,

    J → 1 | 4 | 7 | Z1 | Z4 | Z7 | J0 | J3 | J6 | J9 | D2 | D5 | D8,

    D → 2 | 5 | 8 | Z2 | Z5 | Z8 | J1 | J4 | J7 | D0 | D3 | D6 | D9}

Przykład wyprowadzenia

Z → J8 → Z18 → D718 → 5718, co dzieli się przez 3

Uwaga: ta sama gramatyka o aksjomacie J, zamiast Z,wygeneruje wszystkie liczby dające przy dzieleniu przez 3 resztę 1.

Zauważmy, że też **każda** liczba podzielna przez 3 da się wyprowadzić z tej gramatyki!

Pytania sprawdzające

Jak wyglądają gramatyki generujące:

2. Liczby dające przy dzieleniu przez 3 resztę 2
3. Liczby podzielne przez 4
4. Liczby podzielne przez 5
5. Liczby podzielne przez 7
6. Słowa mające tę samą liczbę liter a i b
7. Słowa mające różną liczbę liter a i b?

Notacja Backusa-Naura

- Zamiast dużych liter na oznaczenie symboli pomocniczych używamy ich opisów wziętych w nawiasy kątowe, np. <cyfra>, <instrukcja wyboru> itp.
- Zamiast strzałki piszemy ::=
- Ciągi powtarzających się słów bierzemy w nawiasy klamrowe, np produkcja

  <ciąg instrukcji> ::= ε | <instrukcja>{;<instrukcja>}

  oznacza dowolny (być może pusty) ciąg instrukcji oddzielonych średnikami: do pojedynczej instrukcji parę średnik-instrukcja możemy dopisać dowolnie wiele razy.

Przykład reguł gramatycznych z języka naturalnego

- <proste zdanie> ::= <podmiot><orzeczenie><okolicznik>
- <podmiot>::=<rzeczownik>|<zaimek>
- <orzeczenie>::=<czasownik>
- <okolicznik>::=<okolicznik miejsca>
- <okolicznik miejsca>::=<przyimek><rzeczownik w bierniku>
- <rzeczownik>::=koń | ptak
- <zaimek>::=on
- <czasownik>::=przeskoczył | przeskoczy | przeleci
- <przyimek>::= przez | za
- <rzeczownik w bierniku>::=płot | księżyc | furtkę

Proste zdania

- Przykłady:
  - Koń przeskoczył przez płot
  - Ptak przeleci za księżyc
  - On przeskoczył przez furtkę
  - On przeleci przez księżyc
  - Koń przeskoczy za furtkę

- Tu oczywiście zbiorem symboli pomocniczych jest

N={<proste zdanie>, <podmiot>, <orzeczenie>, <rzeczownik>, <zaimek>, <okolicznik>, <okolicznik miejsca>, <zaimek>,<rzeczownik w bierniku>,<przyimek>, <czasownik>},

zbiorem symboli końcowych jest

T={koń,ptak,on,przeskoczył,przeskoczy,przeleci,za,przez},

zaś symbolem początkowym A jest <proste zdanie>

Zdania trochę bradziej złożone

- <zdanie>::=<zdanie proste> | <zdanie proste><spójnik><zdanie>
- <spójnik>::=lub | i | ,

- Przykłady zdań:
  - Koń przeskoczy za płot lub ptak przeleci przez furtkę
  - On przeskoczył za furtkę, ptak przeskoczył za furtkę

Kłopot z językiem naturalnym

- Chomsky próbował tego typu reguły stworzyć dla całego języka naturalnego i na tym poległ. Okazuje się bowiem, że bogactwo języka naturalnego nie pozwala na wciśnięcie go w ramy skończonej liczby reguł, nawet jeśli jest to język bez fleskji, jakim jest angielski.
- Można zatem opisywać całkiem duże podzbiory języka naturalnego, ale nie będzie to opis pełny.
- Jest też duży kłopot z określeniem semantyki, czyli znaczenia tworzonych fraz. Między innymi ze względu na kontekstowość.

Inne typy gramatyk

- Do tej pory rozważaliśmy tylko jeden typ gramatyk, nazywany bezkontekstowym.
- Chomsky próbował też oprócz produkcji bezkontekstowych w stylu `A → w` wprowadzać produkcje kontekstowe typu
  - `uAv → uwv`

  aby umożliwić zastosowanie produkcji `A→w` tylko w pewnym kontekście (między u a v). Takie gramatyki nazywamy kontekstowymi, ale to też okazało się za mało.

Języki programowania

- Języki programowania doskonale się nadają do opisu za pomocą gramatyk.
- Zaczynamy od podania gramatyki i obwieszczamy, że program jest poprawny składniowo, jeśli się go da wyprowadzić z podanej gramatyki
- Ponadto za pomocą tego samego mechanizmu możemy określić semantykę programu, przypisując znaczenie poszczególnym produkcjom gramatycznym.

Identyfikatory

- Zaczniemy od prostej wprawki
- Określimy, które zbitki znaków stanowią poprawne identyfikatory w Pascalu
- Najpierw intuicyjna definicja w języku naturalnym:
  - poprawny identyfikator, to ciąg liter lub cyfr zaczynający się od litery. Literami są litery alfabetu łacińskiego oraz znak podkreślenia _. Cyframi są wszystkie cyfry arabskie 0,...,9.

Produkcje generujące poprawne identyfikatory pascalowe

- <identyfikator> ::= <litera>{<litera lub cyfra>}
- <litera lub cyfra> ::= <litera> | <cyfra>
- <litera> ::= a | b | c | ... | z | _ | A | B | C | ... | Z
- <cyfra> ::= 0 | 1 | 2 | 3 | ... | 9

Przykłady poprawnych i niepoprawnych identyfikatorów:

Poprawne: x, x_1, x12, ala, licznik, StanPodstawowy, stan_podstawowy, _

Niepoprawne: 12, x-y, x?, faktura5/02/2004, abc@wp.pl

Cyfry a liczby

- Bardzo częstym błędem jest mylenie cyfr i liczb.
- Cyfry są pojęciem *leksykalnym* i nie mają żadnej wartości liczbowej same w sobie.
- Liczby są pojęciem *semantycznym* i reprezentują wartości.
- Dopiero powiązanie ciągów cyfr z odpowiednimi wartościami (w tym ciągów jednoelementowych reprezentujących tzw. liczby jednocyfrowe) nadaje takim ciągom znaczenie liczbowe.

Liczby całkowite

- Liczby całkowite zdefiniujemy zgodnie z tradycją zapisując je w systemie pozycyjnym dziesiętnym:

<liczba> ::= <liczba bez znaku> |
<znak><liczba bez znaku>

<znak> ::= + | –

<liczba bez znaku> ::= <cyfra> |
<liczba bez znaku><cyfra>

<cyfra > ::= 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |

Jednoznaczność

- Zauważmy, że podana gramatyka jest jednoznaczna, czyli każde słowo z niej wyprowadzalne można otrzymać tylko na jeden sposób. Przykład wyprowadzenia liczby –124:

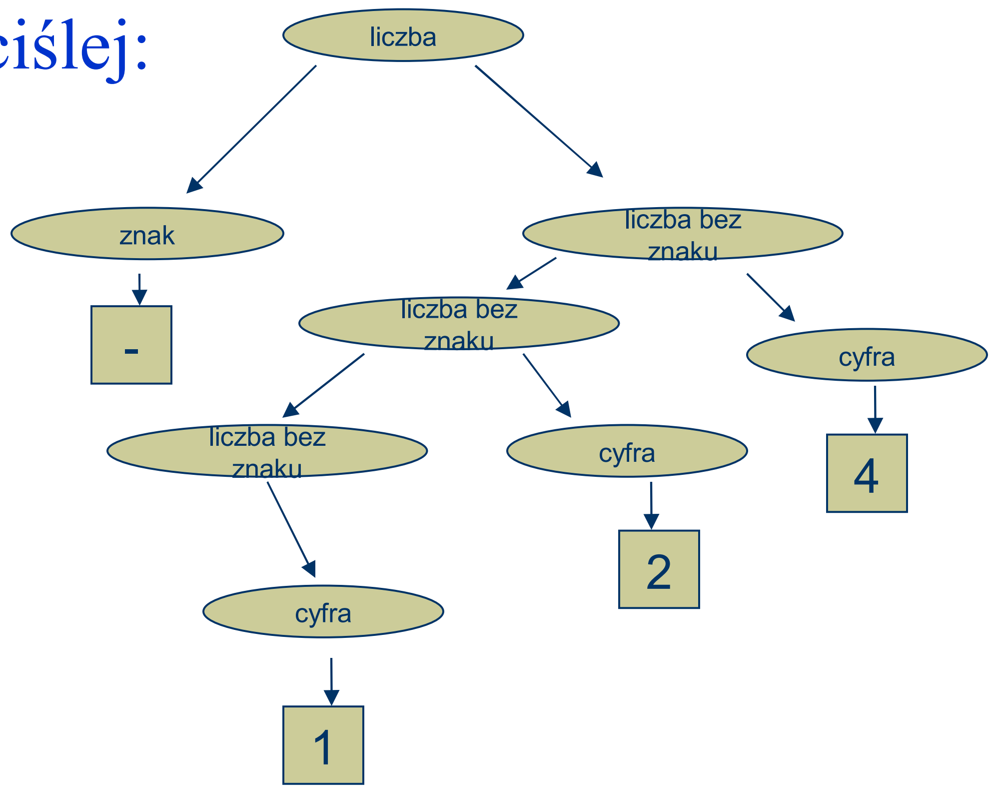
<liczba> → <znak><liczba bez znaku>

- –<liczba bez znaku> →
- –<liczba bez znaku><cyfra> →
- –<liczba bez znaku>4
- –<liczba bez znaku><cyfra>4 →
- –<liczba bez znaku>24 →
- –<cyfra>24 → –124

Jednoznaczność – w jakim sensie?

- Nieco ściślej:

Drzewo wyprowadzenia jest tylko jedno dla każdej liczby!



Niejednoznaczna wersja

- Liczby całkowite zdefiniujemy zgodnie z tradycją zapisując je w systemie pozycyjnym dziesiętnym:

<liczba> ::= <liczba bez znaku> |
<znak><liczba bez znaku>

<znak> ::= + | –

<liczba bez znaku> ::= <cyfra> |
<liczba bez znaku><liczba bez znaku>

<cyfra > ::= 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |

Semantyka liczb

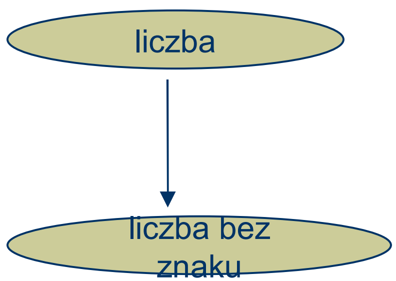
- Na razie mamy metodę tworzenia prawidłowych napisów reprezentujących liczby całkowite. Napisy te zaczynają się od znaku +/– lub nie i składają się z niepustego ciągu cyfr.
- Określenie znaczenia liczby zapisanej jako ciąg cyfr poprzedzony ewentualnym znakiem można związać z ciągiem wyprowadzeń prowadzącym jednoznacznie do powstania tej liczby.
- Niech V(x) oznacza wartość węzła x w drzewie wyprowadzenia danego ciągu cyfr.

Konwencja

- Na kolejnych slajdach przyjmujemy, że kolorem czerwonym oznaczane są obiekty semantyczne, a więc np. liczby, zaś kolrem niebieskim obiekty syntaktyczne, a więc np. cyfry lub ciągi cyfr.
- Zatem 123, to po prostu napis składający się z 3 znaków, zaś 123, to liczba o wartości CXXIII.

Reguły semantyczne

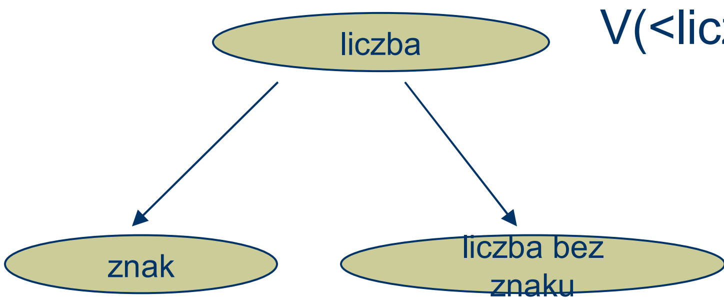
- Reguły semantyczne pozwalają zsyntetyzować wartość węzła na podstawie wartości węzłów bezpośrednio podwieszonych.



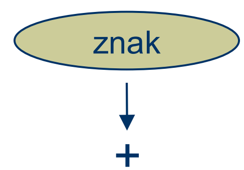
V(<liczba>) := V(<liczba bez znaku>)

Reguły semantyczne

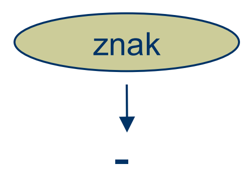
- Na podstawie kolejnych reguł będziemy w stanie zsyntetyzować końcową wartość liczby.



V(<liczba>) :=V(<znak>)* V(<liczba bez znaku>)



V(<znak>):=1



V(<znak>):=-1

Semantyka liczb

cyfra → 0 V(<cyfra>):=0

cyfra → 1 V(<cyfra>):=1

cyfra → 2 V(<cyfra>):=2

cyfra → 3 V(<cyfra>):=3

cyfra → 4 V(<cyfra>):=4

cyfra → 5 V(<cyfra>):=5

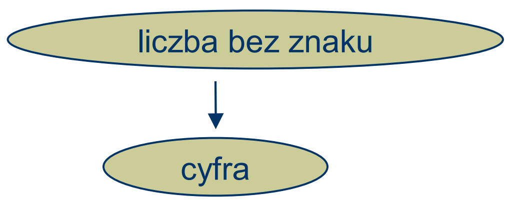
cyfra → 6 V(<cyfra>):=6

cyfra → 7 V(<cyfra>):=7

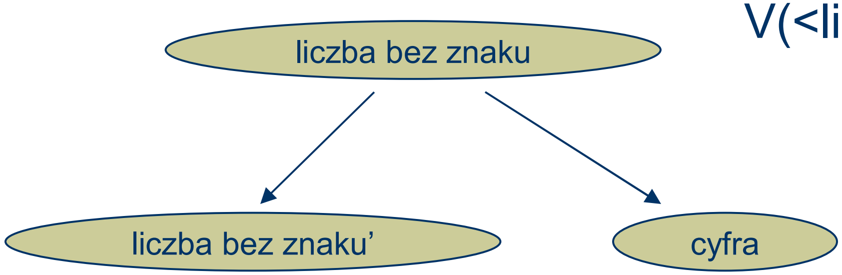
cyfra → 8 V(<cyfra>):=8

cyfra → 9 V(<cyfra>):=9

Semantyka liczb



V(<liczba bez znaku>):=V(<cyfra>)

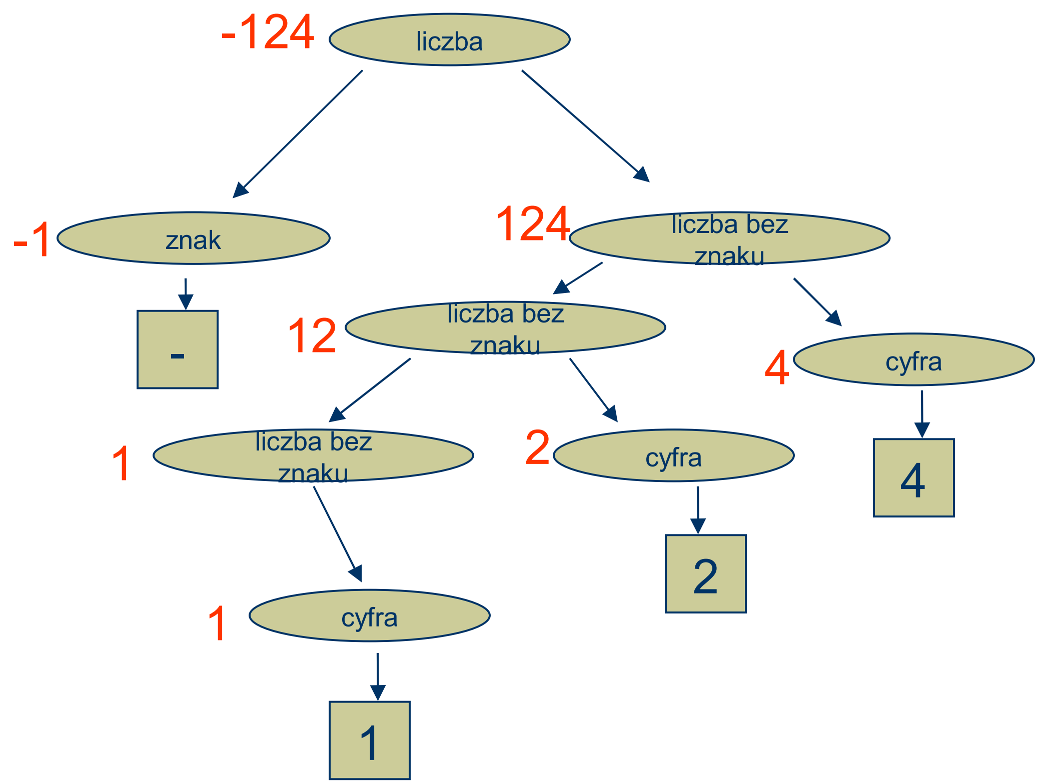


V(<liczba bez znaku>):=
10*V(<liczba bez znaku’>) + V(<cyfra>)

Semantyka wyprowadzenia -124

Zgodnie z podanymi regułami

Tekst -124 ma zatem wartość -124



Składnia liczb całkowitych dwójkowych

- Aby zilustrować pełniej tę metodę, liczby całkowite dwójkowe zdefiniujemy bardzo podobnie, jak dziesiętne:

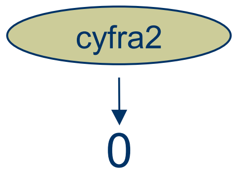
<liczba2> ::= <liczba2 bez znaku> |
<znak><liczba2 bez znaku>

<znak> ::= + | –

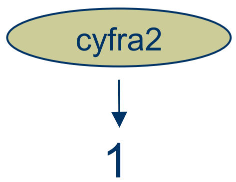
<liczba2 bez znaku> ::= <cyfra2> |
<liczba2 bez znaku><cyfra2>

<cyfra2 > ::= 0 | 1

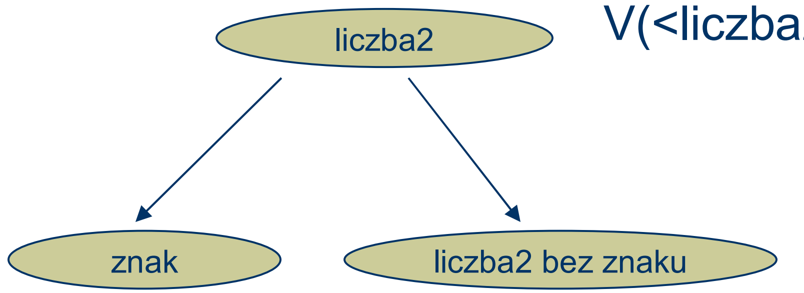
Semantyka liczb dwójkowych



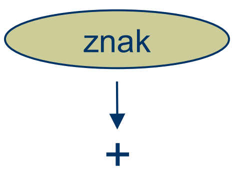
V(<cyfra2>):=0



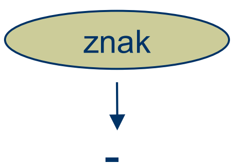
V(<cyfra2>):=1



V(<liczba2>) :=V(<znak>)* V(<liczba2 bez znaku>)

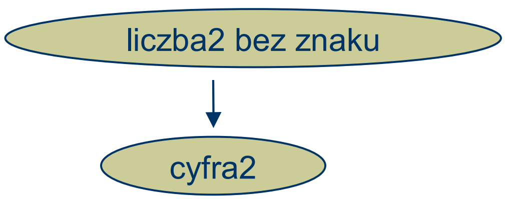


V(<znak>):=1

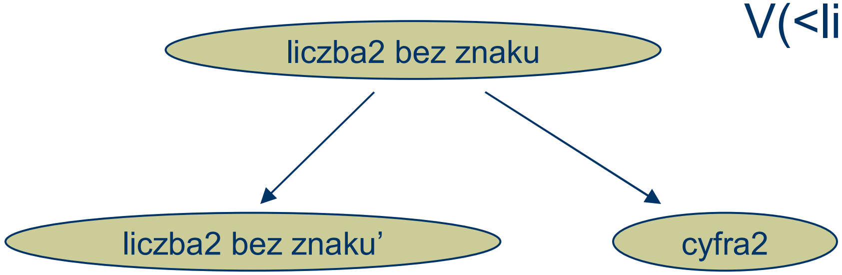


V(<znak>):=-1

Semantyka liczb dwójkowych



V(<liczba2 bez znaku>):=V(<cyfra2>)

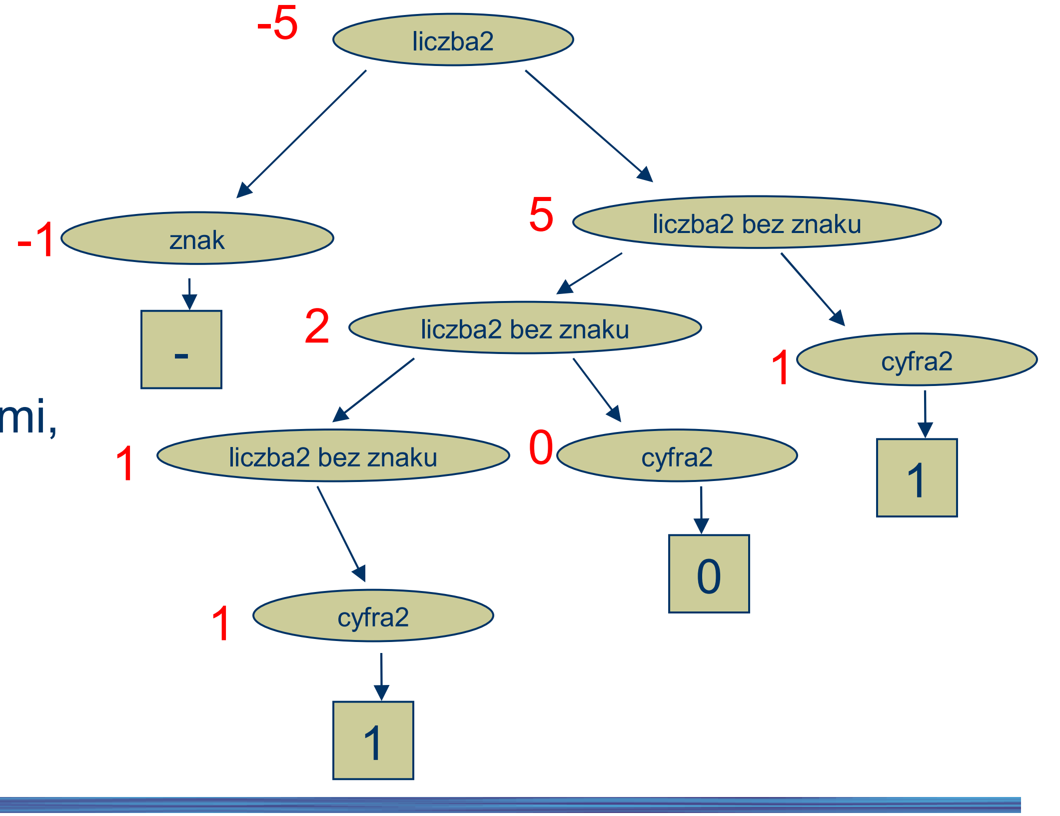


V(<liczba2 bez znaku>):=
2•V(<liczba2 bez znaku’>) + V(<cyfra2>)

Semantyka wyprowadzenia -101 dwójkowo

Przykład

Tekst -101 ma zatem, zgodnie z podanymi regułami, wartość -5



Podsumowanie semantyki liczb

- Na podstawie reguł semantycznych związanych z produkcjami gramatyki jesteśmy w stanie określić wartość pewnych napisów, które będziemy interpretowali jako liczby całkowite
- Uwaga: przykład z systemem dwójkowym był tylko w celu prezentacji techniki. W rzeczywistości w Pascalu liczby dwójkowe w takiej postaci nie występują, bo myliłyby się z dziesiętnymi, np napis -101 można by zinterpretować, jako wartość -5 lub -101.

Wyrażenia arytmetyczne (wersja uproszczona)

- Chodzi o takie zdefiniowanie składni wyrażeń, aby ich znaczenie nie odbiegało od tradycji, w szczególności żeby w sposób z nią zgodny interpretować takie wyrażenia:
  - 4+5*6 (34) a nie (54)
  - 5-3-1 (1) a nie (3)
  - -2+3 (1) a nie (-5)
- ... a przy okazji żeby umożliwić nawiasowanie pozwalające wybrać dowolną kolejność obliczania.
- Musimy przy tym mieć jednoznaczną semantykę!

Dzielenie w liczbach całkowitych

- W liczbach całkowitych dzielenie wykonujemy z resztą.
- Uznajemy, że przy dzieleniu dwóch liczb m przez n są po prostu dwie odpowiedzi: wynik dzielenia i reszta z dzielenia, np
  - 7 dzielone przez 2 to 3 i reszty 1
- Wynik dzielenia m przez n oznaczamy m div n
- Resztę z dzielenia m przez n oznaczamy m mod n
- Zatem
  - 7 div 2 = 3, 7 mod 2 = 1
  - Dla wszystkich liczb parzystych p jest p mod 2 = 0, dla wszystkich liczb nieparzystych n jest n mod 2 = 1.
- Dla dzielenia liczb rzeczywistych używamy znaku /. Zatem 7.0/2.0=3.5. Stałe rzeczywiste piszemy z kropką.
- Nie ma czegoś takiego, jak dzielenie liczb całkowitych 7/2.

Składnia wyrażeń (wersja uproszczona)

<wyrażenie> ::= <składnik> | –<składnik> |
<wyrażenie>+<składnik> |
<wyrażenie>–<składnik>

<składnik> ::= <czynnik> | <składnik> * <czynnik> |
<składnik> div <czynnik> |
<składnik> mod <czynnik>

<czynnik> ::= <stała> | <zmienna> | (<wyrażenie>)

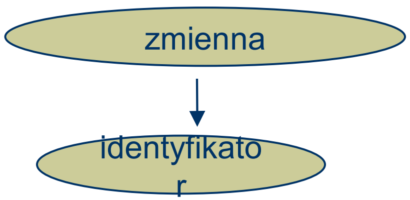
<zmienna> ::= <identyfikator>

<stała> ::= <liczba bez znaku>

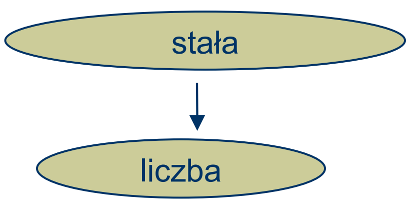
Zmienne

- Zmienne w programach mają wartości typu tego, którego zostały zadeklarowane.
- Określenie która zmienna ma jaką wartość nazywa się wartościowaniem zmiennych. Jest to funkcja, która każdej zmiennej przyporządkowuje jej wartość; będziemy ją oznaczali literą V. Tak więc V(z), to wartość zmiennej z przy wartościowaniu V.
- Jedną z możliwych wartości jest jej brak, albo inaczej mówiąc nieokreśloność i taką uniwersalną wartość może przyjąć każda zmienna. Mówimy wtedy też, że taka zmienna jest niezainicjalizowana. Może się zdarzyć, że zmienna utraci swoją wartość. Nie chodzi o zmianę wartości, ale o wykonanie na niej operacji, której wynik jest nieokreślony. Jeśli zmienna ma wartość nieokreśloną, to oznaczamy to przez V(z)=?.
- Typy zmiennych mogą być rozbudowane. Poza zmiennymi prostymi, takimi jak zmienne liczbowe, logiczne, znakowe, będziemy mieli też typy złożone takie jak tablice (w szczególności napisy), rekordy, zbiory, pliki...

Semantyka wyrażeń – zmienne i stałe

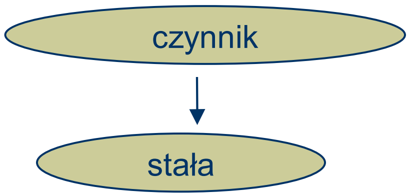


V(<zmienna>):=V(<identyfikator>)

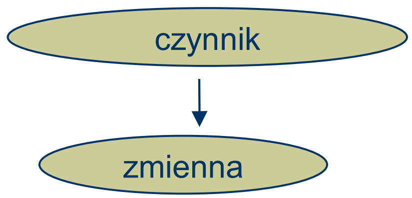


V(<stała>):=V(<liczba>)

Semantyka wyrażeń - czynnik

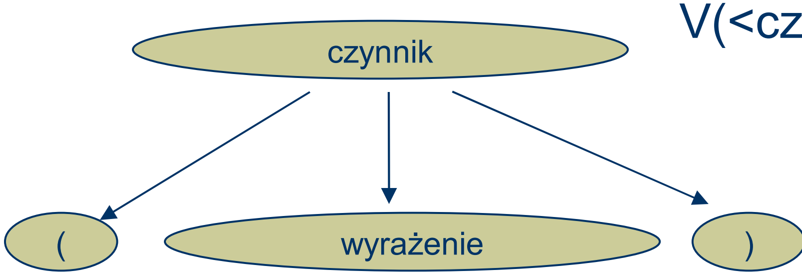


V(<czynnik>):=V(<stała>)

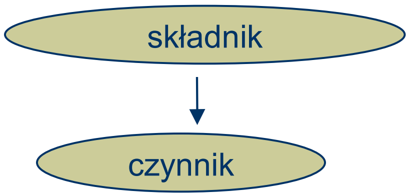


V(<czynnik>):=V(<zmienna>)

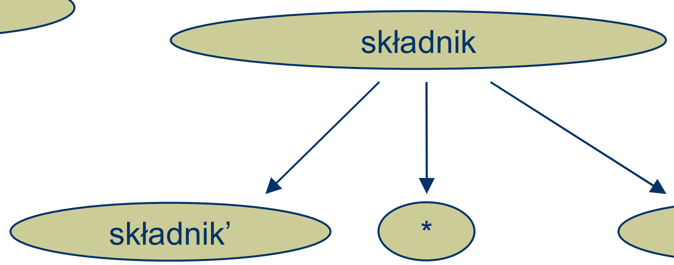
V(<czynnik>):=V(<wyrażenie>)



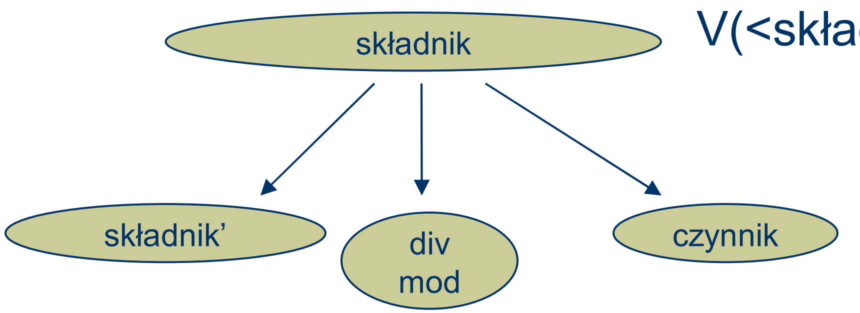
Semantyka wyrażeń - składnik



V(<składnik>):=V(<czynnik>)

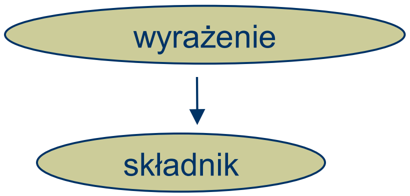


V(<składnik>):=
V(<składnik'>) • V(<czynnik>)

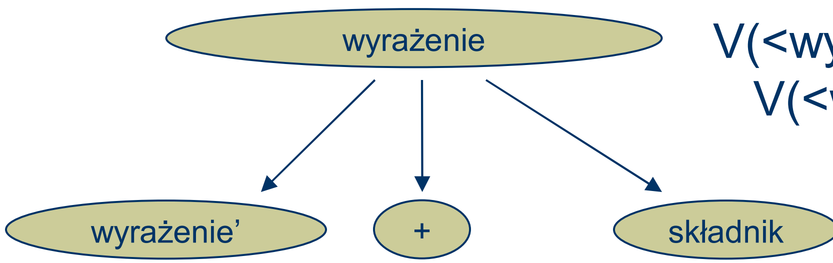


V(<składnik>):= V(<składnik'>) div/mod V(<czynnik>)
jeśli V(<czynnik>) jest różny od 0;
błąd w przeciwnym razie

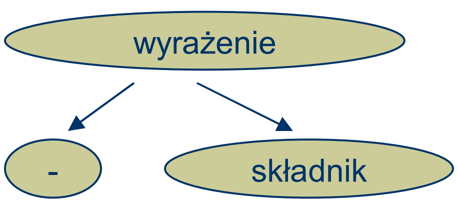
I w końcu: wyrażenie



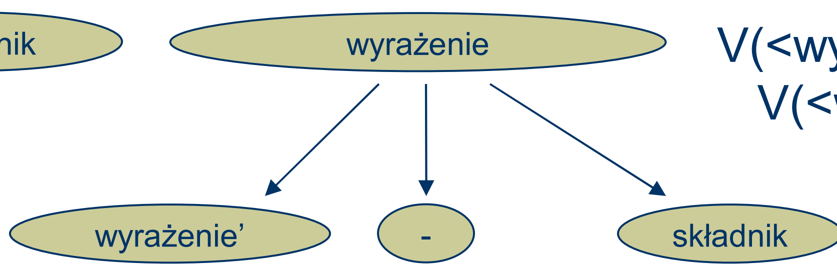
V(<wyrażenie>):=V(<składnik>)



V(<wyrażenie>):=
V(<wyrażenie’>) + V(<składnik>)



V(<wyrażenie>):= –V(<składnik>)



V(<wyrażenie>):=
V(<wyrażenie’>) – V(<składnik>)

Jednoznaczność

- Podana gramatyka jest jednoznaczna. Każde wyrażenie ma tylko jedno możliwe wyprowadzenie i w związku z tym jednoznaczną semantykę!
- Jest to niezwykle istotne, bo dzięki temu nie ma problemu z różnymi interpretacjami tego samego wyrażenia.

Przykład niejednoznacznej gramatyki

- Przykładowo taka gramatyka:

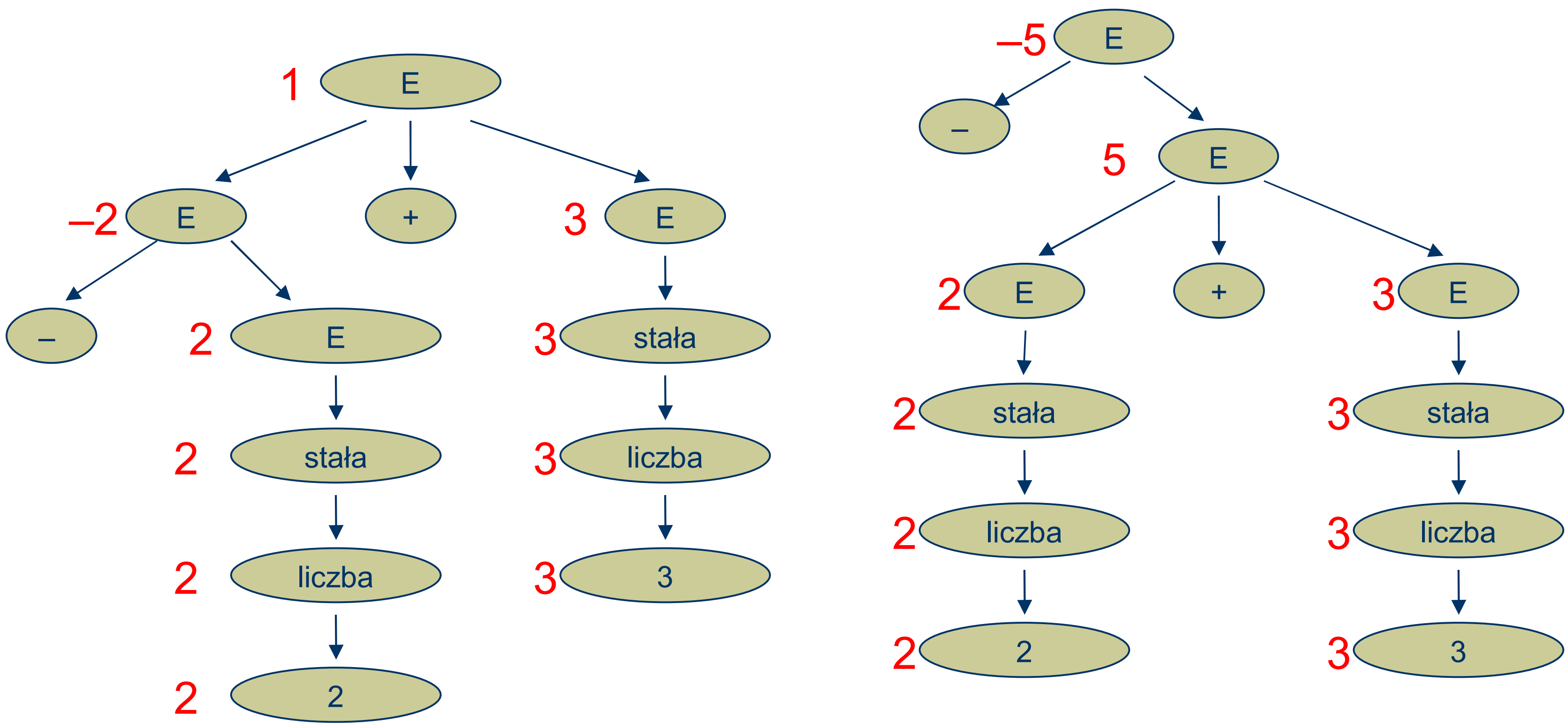
E → <stała> | <zmienna> | –E | E+E | E–E| E div E | E mod E| (E)

Byłaby niedobra, bo na przykład -2+3 miałoby dwa wyprowadzenia i określenie wartości uzależnione od konkretnego wyprowadzenia dałoby różne wyniki (1 i -5)

- Zakładamy tu próbę określenia semantyki w możliwie najnaturalniejszy sposób (np dla produkcji E→E’+E’’ określamy V(E) = V(E’)+V(E’’), itd)
- Zatem gramatyka ta, choć wygenerowałaby dokładnie ten sam język, nie nadawałaby się do zdefiniowania semantyki, która powinna być jednoznaczna.

Problem z niejednoznacznością

- Dwa różne wyprowadzenia w ostatniej gramatyce:



Jednoznaczne wyprowadzenie

- W naszej podstawowej gramatyce wyprowadzenie wyrażenia -2+3 można zrobić tylko na 1 sposób: