Zadania Kwalifikacyjne

WWW17 – toki pona

Piotr Masłowski

Wstęp

Tak jak jest już wspomniane na stronie warsztatów, ciężko tutaj o ścisłe wymagania. Ponieważ praktycznie żadna wiedza wstępna nie jest potrzebna – a z oczywistych przyczyn nie mogę dać zadań z samego języka – niniejsze zadania skupią się na wyjaśnieniu i sprawdzeniu zrozumienia kilku konceptów które ułatwią mi późniejsze wytłumaczenie ich odpowiedników obecnych w toki pona.

Zadanie 0 *(kradzione)*

Napisz w 3 zdaniach coś o sobie. Jakie masz doświadczenie z conlangami, językami w ogólności czy lingwistyką i dlaczego chcesz się nauczyć toki pona?

Dźwięki

W szkole podstawowej uczyliśmy się o sylabach, spółgłoskach, samogłoskach, a dodatkowo na pewno słyszeliśmy stwierdzenia typu "w tych słowach, akcent[1] przypada na trzecią sylabę od końca". Jednak czy można to jakoś ściślej opisać i głębiej zrozumieć? Czym dokładnie różnią się samogłoski od spółgłosek, albo czemu 's' i 'z' można przeciągnąć[2], a 't' i 'p' już nie?

Odpowiedziami na te pytania – jak i wiele innych – zajmuje się fonetyka. Bada ona jak tworzone są dźwięki mowy ludzkiej, jakie cechy fizyczne one mają, a także jak odbierane są one przez słuchających. Równolegle istnieje także fonologia. Ona z kolei bada użycie dźwięków w ramach języka.

[1] Akcent wyrazowy *(ang. stress)*

[2] Mam nadzieję, że wiadomo co miałem tu na myśli.

MAF | IPA

Jednym z podstawowych narzędzi wykorzystywanym w obu dziedzinach, jest Międzynarodowy Alfabet Fonetyczny. Pozwala on na precyzyjny zapis głosek i przekazywanie ich w formie tekstowej. Został opracowany pod koniec dziewiętnastego wieku przez Międzynarodowe Towarzystwo Fonetyczne i próbuje dostarczyć spójny zestaw symboli reprezentujących dźwięki języków naturalnych. Składają się na niego liczne litery – w większości zainspirowane alfabetem łacińskim i greckim – reprezentujące poszczególne głoski oraz zestaw modyfikatorów w postaci znaków diakrytycznych.

Litery MAF zwykle umieszcza się w kilku tabelach. Samogłoski opisane są przez położenie języka (wysokość i tylność) oraz kształt ust (zaokrąglenie). Tworzą więc jeden diagram gdzie oś pionowa odpowiada wysokości, a oś pozioma – tylności. Zwykle rozróżnia się tylko dwa stany zaokrąglenia, więc każdemu punktowi na diagramie mogą być przypisane po dwa symbole.

Spółgłoski są bardziej skomplikowane. W pierwszej kolejności opisuje się tak zwane spółgłoski płucne. Ich przykłady występujące w języku polskim to /m/ /p/ /b/ /n/ /f/ /v/ /k/ /g/. Jest to najliczniejszy typ spółgłosek. Opisywane są przez miejsce i sposób artykulacji oraz ich dźwięczność. Ponownie, ponieważ rozróżnia się dwa stopnie dźwięczności, wszystkie litery można umieścić w dwuwymiarowej tabeli, gdzie każda komórka zawierać może do dwóch liter. Inną grupą spółgłosek są tzw. afrykaty lub inaczej spółgłoski zwarto-szczelinowe (np. /t͡s/ – polskie ⟨c⟩). Ich wymowa polega wpierw na zatrzymaniu przepływu powietrza (podobnie jak w /t/), a następnym wypuszczeniu go (tak jak w /s/). Ponadto wyróżnić można również spółgłoski koartykułowane – takie, w których naraz pojawiają się cechy różnych spółgłosek. Ostatecznie istnieje mniejsza liczba przeróżnych spółgłosek niepłucnych. Przykładowo, zaliczają się do nich mlaski.

Ponieważ dźwięków najlepiej uczyć się ze świadomością, jak one brzmią, polecam przede wszystkim przejrzeć tabelki z przykładami słów dla poszczególnych dźwięków, znajdujące się na Wikipedii w artykułach o fonologii|fonetyce|transkrypcji|... języków które znasz. Dodatkowo, na https://www.ipachart.com/ można znaleźć interaktywne tabele MAF wraz z nagraniami.

Trochę innych wytłumaczeń:

- https://www.szkolnictwo.pl/szukaj,Mi%C4%99dzynarodowy_alfabet_fonetyczny
- https://www.komlogo.pl/index.php/encyklopedia/127-a/942-miedzynarodowy-alfabet-fonetyczny
- https://youtu.be/9uZam0ubq-Y
- http://jezykowo2.blogspot.com/2010/

Przykłady wspmnianych artykułów:

- https://en.wikipedia.org/wiki/Polish_phonology
- https://pl.wikipedia.org/wiki/Fonetyka_j%C4%99zyka_polskiego : (
- https://en.wikipedia.org/wiki/Help:IPA

- https://en.wikipedia.org/wiki/Help:IPA/Polish
- https://pl.wiktionary.org/wiki/Aneks:J%C4%99zyk_polski_-_wymowa_-_g%C5%82oski

Teksty zapisane w MAF (niestety tylko angielskie):

- https://www3.kau.se/kurstorg/files/i/82F31992160621E2D5pYFF85DE9D/IPA.Transcription.Exercises.pdf
- http://phonetic-blog.blogspot.com/search?q=%C3%B0i
- http://phonetic-blog.blogspot.com/search?q=%C9%99n
- https://fonetika.ff.cuni.cz/studium/stranky-kurzu/english-phonetics-phonology/transcription-exercises/

Warto jeszcze wspomnieć czym różni się tak zwana transkrypcja szeroka (fonemiczna) od transkrypcji wąskiej (fonetycznej). W tej pierwszej pomija się szczegóły niezbyt istotne w obrębie danego języka, a w drugiej stara się zachować wierność faktycznym dźwiękom – nawet jeśli różnice nie są zauważane przez przeciętnych użytkowników. Zwykle, dla obecnego w konkretnym języku zestawu fonemów dobiera się najbliżej im odpowiadające litery MAF – nawet jeśli ich wymowa może zmieniać się między słowami, albo nieznacznie odstawać od faktycznego znaczenia symboli. Jednakże, gdy istotne jest dokładne odwzorowanie dźwięków (na przykład w zastosowaniach fonetycznych, a nie fonologicznych) należy zastosować symbole najbliższe głoskom rzeczywiście występującym w dźwięku i w razie potrzeby uszczegółowić zapis przy użyciu znaków diakrytycznych. Transkrypcję fonetyczną umieszcza się w nawiasach kwadratowych, a fonemiczną, pomiędzy ukośnikami. Przykładowo [kɔt̪] i /kɔt/ albo [ˈt͡ʂäs̪] oraz /t͡ʂas/.

Zadanie 1

Zapisz swoje imię i nazwisko w transkrypcji fonemicznej. (Nie musisz rozróżniać alofonów – nie interesują nas różne rodzaje ⟨h⟩, ⟨ɫ⟩ czy ⟨ń⟩.)

Przykładowo “Piotr Masłowski” zapisałbym następująco: /pʲɔtr maˈswɔf.ski/

Zadanie 2

Wybierz 16 słów z dowolnych języków (ludzkich), które choć trochę znasz i zapisz je (fonemicznie) w MAF.

Przykład:

polski		angielski	
tutaj	/ˈtu.taj/	interjection	/ɪn.tɚˈd͡ʒɛk.ʃən/
dlaczego	/dlaˈt͡ʂɛ.gɔ/	convey	/kənˈveɪ/

Składnia

Jaka kolejność wyrazów ma sens? Z czego składają się zdania? Albo – które typy słów możemy razem ułożyć, żeby wspólnie miały jakieś znaczenie? Tego rodzaju zagadnieniami zajmuje się między innymi syntaktyka.

Ponieważ są to zajęcia z pojedynczego języka – a nie całej lingwistyki[3] – nie będziemy zbyt dogłębnie analizować teoretycznych aspektów składni w toki pona. Chciałbym jednak wprowadzić teraz kilka idei dotyczących kolejności i hierarchii słów w obrębie zdania. Dzięki temu będę mógł łatwo wytłumaczyć kilka konstrukcji językowych obecnych w toki pona, a wy będziecie już mieli wyrobione pewne intuicje z nimi związane.

Jako że chcemy być precyzyjni, wykorzystamy do tego język programowania. Potrzebujemy jednak, aby jego składnia była inspirowana językiem ML, a nie C. W związku z tym JavaScript, Python czy C++ odpadają, a pozostaje nam wybrać z bardziej niszowych pozycji. W wyjaśnieniach i przykładach użyty jest Haskell. Jednakże nic nie powinno stać na przeszkodzie, aby rozwiązać zadania w językach takich jak Ocaml, Elm[4], F# czy Scala. No i oczywiście, nie jest wymagana znajomość żadnego z tych języków – wszystkie aspekty, których potrzeba są wytłumaczone poniżej.

W pierwszej kolejności, przyda się nam interaktywny interpreter. Jeden dostępny online znajduje się na https://tryhaskell.org/. Alternatywnie, inny można znaleźć tutaj: https://replit.com/languages/haskell – w terminalu po prawej stronie należy wpisać `ghci`.

Arytmetyka

Zacznijmy od czegoś prostego – wpisz:

```
2 + 2
```

Jak można się było spodziewać, wynik to `4`. Wypróbujmy teraz coś delikatnie mniej oczywistego:

```
2 + 2 * 2 + 2
```

Hmm, Haskell zachował kolejność działań. Zamiast wykonać operacje od lewej do prawej, wpierw zostało wykonane mnożenie, a dopiero później dodawanie. Nie jest to nic szczególnie zaawansowanego – praktycznie każde narzędzie bardziej skomplikowane niż najprostsze z kalkulatorów, jest w stanie sobie z tym poradzić. Jednak w jaki sposób jest to zaimplementowane? Z innych języków możesz pewnie znać pojęcie pierwszeństwa operatorów. Zwykle

[3] Zbytnio się na tym nie znam. Nie przeceniajcie mnie za bardzo :)
[4] <0.19

ma się tabelkę z wypisanymi wszystkimi dostępnymi operacjami posortowanymi według kolejności ich aplikowania. W Haskellu `(+)` ma priorytet 6, a `(*)` 7. Każde wyrażenie zawierające te dwa zostanie najpierw rozbite na podwyrażenia, które trzeba dodać, a te z kolei na podpodwyrażenia, które trzeba wymnożyć.

Odejmowanie i dzielenie działają prawie identycznie. Trzeba jednak pamiętać o kolejności między kilkoma identycznymi operacjami – chcemy przecież, żeby `10 - 6 - 1` nie dawało `5`, ale `3`.

Czy oznacza to, że wszytkie operacje muszą wykonywać się od lewej do prawej? Absolutnie nie! Warto jednak, aby poszczególne grupy działań były spójne między sobą i odzwierciedlały to czego się spodziewamy. Zarówno `(*)` jak i `(/)` to tutaj `infixl 7`, a `(+)` jak i `(-)` to tutaj `infixl 6`. (Można to sprawdzić, wpisując `:info (+)`, itp. w `ghci` – niestety nie działa to gdzie indziej.) Dzięki temu, wyrażenia takie jak `2 * 3 - 10 + 40 / 3 - 1`, są interpretowane jako `(((2 * 3) - 10) + (40 / 3)) - 1`.

Prefiksy

Jak dotąd używaliśmy operatorów infiksowych – tych, które pisze się pomiędzy ich argumentami. Równie przydatne są jednak operacje prefiksowe. Jak łatwo odgadnąć, pisze się je *przed* ich argumentami. Czasami są one mniej czytelne, ale na ogół bardziej elastyczne (np. nie muszą koniecznie przyjmować dwóch argumentów). Jakie są więc przykłady operatorów prefiksowych w Haskellu? Są to po prostu zwykłe funkcje:

```
sin 7
cos 5
divMod 11 3
sum [6,4,3,1]
snd (31,"Yay!")
map log [1..4]
take 8 [0,7..100]
```

Funkcje mają pierwszeństwo nad wszystkimi operatorami – dzięki temu wyrażenia, takie jak `sin 3 - cos 5 * log 10` nie wymagają nawiasów na każdym kroku. Tak na marginesie, operatory w Haskellu to tylko funkcje, których nazwy nie są literami, a symbolami. Jeśli ktoś chce, to może ich używać prefiksowo – wystarczy otoczyć je nawiasami: `(-) 5 2`. W drugą stronę też się da – zwykłe funkcje można zmieniać w infiksy następująco: `15 `mod` 4`. (To jest ten klawisz z tyldą; koło jedynki, a nad tabem.) Tak skonstruowane operatory domyślnie rozumiane są jako `infixl 9`.

`infixr 0`

Pomimo, że Haskell nie wymaga nawiasów do używania funkcji, czasami nie możemy się bez nich obejść. Klasycznym przykładem jest oczywiście `(2 + 2) * 2`. Co jednak zrobić z następującym wyrażeniem?

```
snd (sum (take 10 (map (*2) [1..100])) `divMod` 10)
```

Nie jest ono zbyt czytelne. Zbyt dużo razy zagnieżdża ono funkcje jedne w drugich, przez co ciężko odnaleźć, gdzie poszczególne podwyrażenia się zaczynają, a gdzie kończą.

```
funkcja (funkcja (funkcja (funkcja argument)) `funkcja` argument)
```

To jest trochę czytelniejsze, jednak całkowicie bezużyteczne – wyrzucamy wszystkie szczegóły, a zostawiamy tylko składnię. Można teraz szybko zauważyć, co jest czego argumentem, ale straciliśmy całe znaczenie, które przecież normalnie jest celem wszelkich obliczeń.

Czy da się z tym jednak jakoś poradzić? Otóż w bibliotece standardowej Haskella dostępny jest dość nietypowy operator. Zdefiniowany jest on następująco:

```
fun $ arg = fun arg
```

Na pierwszy rzut oka jest on dość bezużyteczny – z lewej strony bierze funkcję, z prawej jej argument i aplikuje jedno na drugim. Po co więc nam operator, który jedyne co robi, to znika? Okazuje się jednak, że gdy ma on najniższy priorytet, to można go wykorzystywać zamiast nawiasów. Tak jak plus nie potrzebuje zwykle nawiasów naokoło swoich argumentów, tak `($)`, mający najniższe pierwszeństwo możliwe, potrzebuje ich jeszcze rzadziej. Zobaczmy więc, czy nasz początkowy przykład staje się czytelniejszy:

```
snd (sum (take 10 (map (*2) [1..100])) `divMod` 10)

snd $ (`divMod` 10) $ sum $ take 10 $ map (*2) [1..100]
```

Zrobiło się to delikatnie dłuższe, jednak możemy teraz w miarę łatwo odczytać, co po kolei się dzieje: Otrzymujemy drugi element dzielenia z resztą przez 10 sumy pierwszych dziesięciu liczb całkowitych pomnożonych przez dwa. I tak na marginesie, zwykle używa się tu jeszcze `(.)` – operatora złożenia funkcji.

Zadanie 3

Określ, jaka jest kolejność działań w następujących wyrażeniach:

0) `2 + 2 * 2 - 3 / 7 * 8 - 9 + 20 / 3 / 5 * 1`
1) `0 * 1 + 2 ✳ 3 - 4 ☃ 5 ♛ 6 ♫ 7 - 8 ☂ 9`
2) `sin 2 ⊡ cos 5 ⊞ tan (exp 2) ⚄ 7 ⚃ 0 ⚅ log (5 ⚃ 2) ⚂ 9`
3) `"abc" ♧ "xyz" ♧ "x" ♧ "DDD" ♧ "t" ♧ "ee" ♧ "" ♧ "l"`
4) `1 * 5 ☃ 2 + 4 - 7 ♫ 0 - 3 ✳ 8 ☂ 6 ♛ 9`

jeśli wiadomo, że:

```
infixr 5 ✳
infixl 8 ☃
infixl 4 ♛
infixl 4 ♫
infixr 5 ☂

infixr 1 ⊡
infixl 2 ⚃
infixr 3 ⚂
infixr 4 ⚄
infixr 5 ⚅
infixl 6 ⊞

infixr 7 ♧
infixr 6 ♧
infixr 5 ♧
infixl 4 ♧
infixl 3 ♧
infixl 2 ♧
infixl 1 ♧
```