Propozycje rozszerzenia JoSKIPI

Michał Marcińczuk

28.09.2010 r.

wersja 3.0

Niniejszy dokument zawiera opis propozycji nowych operatorów będących rozszerzeniem języka JoSKIPI na potrzeby analizy semantycznej, w tym tworzenia i usuwania anotacji semantycznych, tworzenia i usuwania relacji semantycznych i opisu zdarzeń.

1. Dopasowanie sekwencji tokenów i indeksowanie tablicą

```
matches match( position, <list_of_conditions>)
```

Operator `match`, zaczynając od pozycji `position`, dopasowuje sekwencję tokenów spełniających listę warunków `list_of_conditions`. Po spełnieniu wszystkich warunków operator zwraca tablicę referencji (`matches`) dopasowanych elementów. Wyróżnione zostały 3 rodzaje referencji:

- `REF(TOKEN[n])` – referencja na token o indeksie *n*,
- `REF(ANNOTATION)` – referencja na anotację typu ANNOTATION,
- `REF(MATCH)` – referencja na tablicę referencji.

Każdy operator na liście `list_of_conditions` generuje jedną referencję, która jest umieszczana w tablicy wynikowej. Operatory dzielą się na 3 kategorie ze względu na typ generowanej referencji:

- operatory generujące `REF(TOKEN[n])` takie jak `inter`, `regex`, `equal`, `and` zwiększają aktualną pozycje o jeden token. Zakładam, że operatory zwracają wartość *true/false*, czyli *spełnia warunek/nie spełnia warunku*.
- operatory generujące `REF(ANNOTATION)` takie jak `is` zwiększają aktualną pozycję o długość anotacji liczoną w tokenach. Zakładam, że operatory z tej grupy zwracają referencje na anotacje, dzięki której możliwe jest obliczenie długości anotacji.
- operatory generujące `REF(MATCH)` zwiększają aktualną pozycję o łączną długość dopasowanych elementów liczoną w tokenach. Każdy operator trzeba rozważyć z osobna:
  - `optional`, `repeat`, `match` – zwracają tablicę referencji, która nie wymaga dodatkowego przetwarzania,
  - `text` – zwraca liczbę dopasowanych tokenów, którą można rzutować na tablicę referencji,
  - `or` – wynik operatorów dla których operator `or` został spełniony zostaną zaindeksowane tablicą referencji,

```
matches match( <list_of_conditions>)
```

Składnia dopuszczana wyłącznie wewnątrz sekcji `actions` operatora `apply` i `do`. Wyjaśnienie znajduje się w pkt. 3.

Przykład #1.1

Tekst:

> *[Pan]$_0$ <FIRST_NAM>[Marek]$_1$</FIRST_NAM> [Noga]$_2$ [i]$_3$ <FIRST_NAM>[Łukasz]$_4$</FIRST_NAM> [Wrona]$_5$.*

Reguła:

```
match($pos,
    is('FIRST_NAM'),
    regex(orth, '\p{Lu}\p{Ll}*')
)
```

Wynik dla `$pos=0`:

```
array()
```

Wynik dla `$pos=1`:

```
array(
    [0] = REF(TOKEN[1])
    [1] = REF(FIRST_NAM)
)
```

Analiza działania:

```
1.  Wywołanie match($pos=1, list_of_conditions)
2. Ustawienie wskaźnika $_ = $pos = 1
3. Inicjalizacja tablicy referencji $r = array()
4. Dla każdego warunku $w z list_of_conditions
    a) $w = is($_, 'FIRST_NAM')                    // $_=1
          $w jest równe REF(FIRST_NAM)
          $r[] = $w
          $_ += len(REF(FIRST_NAM))
    b) $w = regex(orth[$_], '\p{Lu}\p{Ll}*')       // $_=2
          $w jest równe TRUE
          $r[] = REF(TOKEN[$_])
          $_++
5. Zwróć wynik $r
```

Przykład #1.2

Tekst:

> *[Spółka]$_0$ [Impel]$_1$ [SA]$_2$ [i]$_3$ [Firma]$_4$ [Ambra]$_5$ [Makabra]$_6$ [SA]$_7$*

Reguła:

```
$m =
match($pos,
    and(
          inter(base[$_], { 'spółka', 'firma' }),
          regex(orth[$_], '\p{Lu}\p{Ll}*')
    ),
    repeat(
          regex(orth[$_], '\p{Lu}\p{Ll}*')
    ),
    or(
          inter(orth[$_], {SA}),
          inter(orth[$_], {Sp.})
    )
)
```

Wynik dla `$pos=0`:

```
$m[0] = TOKEN[0]
$m[1] = TOKEN[1]
$m[2] = TOKEN[2]
```

Wynik dla `$pos` od 1 do 3:

```
$m = array()
```

Wynik dla `$pos=4`:

```
$m[0] = TOKEN[4]
$m[1] = array(TOKEN[5], TOKEN[6])
$m[2] = TOKEN[7]
```

Wynik dla `$pos` od 5 do 7:

```
$m = array()
```

2. Operator `do`

```
n do(position, operator_match, <list_of_actions>)
```

Operator `do` dopasowuje sekwencję tokenów opisaną przy pomocy operatora `operator_match` od wskazanej pozycji `position`. Jeżeli operator `match` zwróci niepustą tablicę referencji, to dla otrzymanej tablicy wykonywana jest każda akcja z `list_of_actions`. Operator zwraca długość (w tokenach) sekwencji dopasowanej przez operator `operator_match`.

3. Operator `iter`

```
iter( operator_do )
```

Operator `iter` uruchamia operator `operator_do` dla kolejnych tokenów w zdaniu. **Ten operator prawdopodobnie nie będzie można zapisać czysto funkcyjnie, ponieważ jego działanie wygląda następująco:**

```
var tokens                                    // tablica tokenów zdania
pos = 0                                       // aktualna pozycja
while ( pos < len(tokens) )
      n = do(pos, match, actions)   // n - długość sekwencji
      if ( n == 0 )
            pos++
      else
            pos += n
```

Jako, że przy kolejnych wywołaniach pozycja `position` jest automatycznie wyliczana i przekazywana do operatora `do` i dalej do operatora `match`, wymagane jest definiowanej uproszczonej sygnatury operatorów `match` i `apply` bez argumentu `position`.

```
do( match, <list_of_actions>)

match( <list_of_conditions>)
```

Odwołanie w sekcji `actions` do wyniku dopasowania operatora `match` będzie możliwe przy pomocy liczb całkowitych określających indeksy dopasowanych elementów.

Przykład pełnego wywołania operatora `inter`:

```
iter(
    do(
          match(
                inter(base, {'pan'}),                 // indeks 0
                regex('\p{Lu}\p{Ll}*'),               // indeks 1
                regex('\p{Lu}\p{Ll}*')                // indeks 2
          ),
          actions(
                mark(1, 'PERSON_FIRST_NAM'),
                mark(2, 'PERSON_LAST_NAM'),
                mark(0, 2, 'PERSON')
          )
    )
)
```

4.Operator `apply`

```
apply(match, <list_of_actions>)
```

Operator do jest skróconym zapisem dla `iter` i `do`. Reguła z poprzedniego punktu zapisana przy pomocy operatora `do` będzie miała postać:

```
apply(
    match(
          inter(base, {'pan'}),                       // indeks 0
          regex('\p{Lu}\p{Ll}*'),                     // indeks 1
          regex('\p{Lu}\p{Ll}*')                      // indeks 2
    ),
    actions(
          mark(1, 'PERSON_FIRST_NAM'),
          mark(2, 'PERSON_LAST_NAM'),
          mark(0, 2, 'PERSON')
    )
)
```

5. Operator `start` i `end`

Operator `start(ref)` zwraca indeks pierwszego tokenu referencji, tj:

- **`start(`**`REF(TOKEN[n])`**`)`** = `n`.
- **`start(`**`REF(ANNOTATION)`**`)`** = *indeks pierwszego tokenu anotacji ANNTATION*,
- **`start(`**`REF(MATCH)`**`)`** = `len(MATCH]) > 0 ? start(MATCH[0]) : -1`

Operator `end(ref)` zwraca indeks ostatniego tokenu referencji, tj:

- **`end(`**`REF(TOKEN[n])`**`)`** = `n`
- **`end(`**`REF(ANNOTATION)`**`)`** = *indeks ostatniego tokenu anotacji ANNOTATION,*
- **`end(`**`REF(MATCH)`**`)`** = `len(MATCH) > 0 ? end(MATCH[-1]): -1`
  gdzie `MATCH[-1]` *to ostatni element tablicy*

6. Dodawanie anotacji

```
mark( ref_from, ref_to, annotation_name)
```

Operator tworzy anotację typu `annotation_name` rozciągającą się od tokenu **`start(`**`ref_from`**`)`** do **`end(`**`ref_to`**`)`** włącznie, gdzie `ref_from` i `ref_to` są referencjami opisanymi w pkt. 1.

```
mark( ref, annotation_name)
```

Operator tworzy anotację typu `annotation_name` rozciągającą się od tokenu **`start(`**`ref`**`)`** do **`end(`**`ref`**`)`** włącznie, gdzie `ref` jest referencją opisanymi w pkt. 1.

```
mark( ref, annotation_name)
```

Składnia dopuszczana wyłącznie wewnątrz sekcji `actions` operatora `apply` i `do`. Tworzy anotację obejmującą całą sekwencję dopasowaną przez operator `match`.

Przykład #6.1

Tekst:

> *[Pan]$_0$ <FIRST_NAM>[Marek]$_1$</FIRST_NAM> [Noga]$_2$ [i]$_3$ <FIRST_NAM>[Łukasz]$_4$</FIRST_NAM> [Wrona]$_5$.*

Tablica referencji:

```
$m[0] = REF(TOKEN[0])
$m[1] = REF(FIRST_NAM)            // pierwszy FIRST_NAM w zdaniu
$m[2] = REF(TOKEN[2])
```

```
mark($m[0], $m[2], 'PERSON')
```

Utworzy anotację `PERSON` dla *„Pan Marek Noga”*, tj

- `$begin = start($m[0]) = start(TOKEN[0]) = 0`
- `$end = end($m[2]) = end(TOKEN[2]) = 2`

Przykład #6.2

Tekst:

> *[Spółka]$_0$ [Impel]$_1$ [SA]$_2$ [i]$_3$ [firma]$_4$ [Ambra]$_5$ [Makabra]$_6$ [SA]$_7$*

Tablica referencji:

```
$m[0] = REF(TOKEN[4])
$m[1] = REF(TOKEN[5:6])
$m[2] = REF(TOKEN[7])
```

Reguła:

```
mark($m[1], $m[2], 'COMPANY_NAM')
```

Utworzy anotację `COMPANY_NAM` dla *„Ambra Makabra SA”*, tj

- `$begin = start($m[1]) = start(TOKEN[5:6]) = 5`
- `$end = end($m[2]) = end(TOKEN[7]) = 7`

Przykład #6.2

Tekst:

> *[IBM]$_0$ [jest]$_1$ [firmą]$_2$ [z]$_3$ [branży]$_4$ [IT]$_5$*

Reguła:

```
apply(
    match(
          regex('\p[Lu}+')
    ),
    actions(
          mark('UPPER')
    )
)
```

Wynik:

> *<UPPER>[IBM]$_0$</UPPER> [jest]$_1$ [firmą]$_2$ [z]$_3$ [branży]$_4$ <UPPER>[IT]$_5$</UPPER>*

7.Dopasowanie anotacji

```
is( position, annotation_name)
```

Dopasowuje ciąg tokenów zaczynający się od pozycji `position` będący w całości oznaczony jako anotacja `annotation_name`. Po udanym dopasowaniu zwraca referencję na obiekt anotacji.

```
is( annotation_name)
```

Składnia dopuszczana wyłącznie wewnątrz sekcji `actions` operatora `apply` i `do`. Anotacja

jest sprawdzana poczynając od aktualnej pozycji ustawionej przez operator `match`.

Przykład #7.1

Tekst:

> *<COMPANY>[IBM]$_0$<COMPANY> [jest]$_1$ [firmą]$_2$ [z]$_3$ [branży]$_4$ [IT]$_5$*

Reguła:

```
is($pos, 'PERSON')
```

Wynik:

```
is(0, 'PERSON') = REF(COMPANY)
is(1, 'PERSON') = NULL
```

Przykład #7.2

Tekst:

> *[Pan]$_0$ <FIRST_NAM>[Marek]$_1$</FIRST_NAM> [Noga]$_2$*

Reguła:

```
apply(
    match(
          is('FIRST_NAM'),
          regex('\p{Lu}\p{Ll}*')
    ),
    actions(
          mark(1, 'LAST_NAME')
    )
)
```

Wynik:

> *[Pan]$_0$ <FIRST_NAM>[Marek]$_1$</FIRST_NAM> <LAST_NAM>[Noga]$_2$<LAST_NAM>*

8.Usuwanie anotacji

```
unmark( ref )
```

Usuwa anotację poprzez referencję.

Przykład #7.2

Reguła:

```
apply(
    match($m,
          text($_, "ul. "),
          is($_, 'PERSON_LAST_NAM')
    ),
    actions(
          unmark(1, 'PERSON_LAST_NAM')
    )
)
```

Tekst:

Budynek znajduje się przy ul. <PERSON_LAST_NAM>Mickiewicza</PERSON_LAST_NAM>.

Wynik:

Budynek znajduje się przy ul. Mickiewicza.

9. Sprawdzenie, czy anotacja jest częścią innej anotacji

```
part( ref, annotation_name)
```

TODO

10. Dopasowanie tekstu bez uwzględniania podziału na tokeny

```
text( position, tekst)
```

Operator dopasowuje sekwencję tokenów, dla których złączenie orth-ów równe jest zadanej wartości `tekst`. Operator będzie potrzebny, kiedy nie ma pewności, jak dany tekst może zostać podzielony przez tokenizator. Operator zwraca liczbę dopasowanych tokenów.

```
text( tekst)
```

Składnia dopuszczana wyłącznie wewnątrz sekcji `actions` operatora `apply` i `do`. Anotacja jest sprawdzana poczynając od aktualnej pozycji ustawionej przez operator `match`.

Przykład #7.1

Tekst:

[Sp.]$_0$ [z]$_1$ [o.o.]$_2$ [i]$_3$ [Sp]$_4$ [.]$_5$ [z]$_6$ [o.]$_7$ [o.]$_8$

Reguła:

```
text($pos, 'Sp. z o.o.')
```

Wynik:

```
text(0, 'Sp. z o.o.') = 3
text(1, 'Sp. z o.o.') = 0
text(4, 'Sp. z o.o.') = 5
```

Przykład #7.2

Dla zdania:

[Spółka] [XYZ] [Sp].[z] [o.o.]

Reguła:

```
apply(
    match(
          intern(base, { 'spółka', 'firma' }),
          regex(orth, '\p{Lu}\p{Ll}*'),
          text($_, 'Sp. z o.o.')
    ),
    actions(
          mark('COMPANY')
    )
)
```

Wynik:

<COMPANY>[Spółka] [XYZ] [Sp].[z] [o.o.]</COMPANY>

11. Zwielokrotnione dopasowanie

```
repeat( position, condition)
```

Dopasowuje sekwencję tokenów począwszy od pozycji `position` spełniających określone warunku. Operator dopasowuje najdłuższą możliwą sekwencję. Operator zwraca tablicę referencji na dopasowane elementy.

```
repeat( condition)
```

Składnia dopuszczana wyłącznie wewnątrz sekcji `actions` operatora `apply` i `do`. Sekwencja jest sprawdzana poczynając od aktualnej pozycji ustawionej przez operator `match`.

Przykład #11.1

Tekst:

[Spółka]$_0$ [XYZ]$_1$ [ABC]$_2$ [SA]$_3$

Operator:

```
repeat($pos,
    regex(orth, '\p{Lu}+')
)
```

Wynik dla `$pos=0`:

```
array()
```

Wynik dla `$pos=1`:

```
array(
    [0] = REF(TOKEN[1])
    [1] = REF(TOKEN[2])
    [2] = REF(TOKEN[3])
)
```

Wynik dla `$pos=2`:

```
array(
    [0] = REF(TOKEN[2])
    [1] = REF(TOKEN[3])
)
```

Wynik dla `$pos=3`:

```
array(
    [0] = REF(TOKEN[3])
)
```

Przykład #11.2

```
apply(
    match(
          inter(base, {'spółka'}),
          repeat( regex('\p{Lu}\p{Ll}*') ),
          text('SA')
    ),
    ations(
          mark(1, 2. 'COMPANY)
    )
)
```

Warianty:

```
repeat( position, condition)- domyślnie dopasuje
najdłuższą sekwencję

repeat( position, condition, number_of_repeats)- podana
liczba powtórzeń

repeat( position, condition, min_repeats, max_repeats)-
liczba powtórzeń określona przedziałem
```

12. Iteracja po powtórzeniach – do weryfikacji

```
for( group, var, action )
```

Dla ostatniego przykładu z pkt. 6, drugi element składa się z powtórzonych dopasowań, tj. , 'Marek Kowalski' i ', Iwona Wójcik' na pierwszym poziomie zagnieżdżenia. Powinna istnień możliwość iteracji po tych elementach na potrzeby zastosowania operatora mark.

Np.

```
match($m,
    is($_, 'PERSON') ,
    repeat($_
          text($_, ','),
          repeat($_,
                regex(orth[$_], '\p{Lu}\p{Ll}*')
          )
    )
),
for($m[1], $v, mark($v[1], 'PERSON'))
```

Dla zdania:

> *"W spotkaniu uczestniczył <PERSON>Pan Jarek Nowak</PERSON>, Marek Kowalski, Iwona Wójcik i Inga Wieczorek."*

zwróci wynik:

> *"W spotkaniu uczestniczył <PERSON>Pan Jarek Nowak</PERSON>, <PERSON>Marek Kowalski</PERSON>, <PERSON>Iwona Wójcik</PERSON> i Inga Wieczorek."*

13. Opcjonalne dopasowanie – do weryfikacji

```
optional( position, condition )
```

Uproszczony zapis dla operatora:

```
repeat( position, condition, 0, 1)
```

Np.

```
match( $m,
    inter( base[$_], "firma"),
    regex( orth[$_], "\p{Lu}\p{Ll}*"),
    optional( text($_, "S.A.") )
),
mark( $m[1], $m[2], "COMPANY_NAM")
```

dla tekstu:

> *Firma Intext S.A. i firma Kontra podpisały umowę.*

zwróci wynik:

> *Firma <COMPANY_NAM>Intext S.A<COMPANY_NAM>. i firma <COMPANY_NAM>Kontra <COMPANY_NAM> podpisały umowę.*

Pomimo, że w drugim dopasowaniu warunek opcjonalny nie został dopasowany, to odwołanie do indeksu `$m[2]` było poprawne. W tym przypadku operator **`mark`** pomija wszystkie puste indeksy z początku i końca zakresu.

14. Odpytanie Słowosieci *(wishlist)*

Możliwość sprawdzenia, czy dane słowo jest synonimem i/lub hiponimem, np.

```
synonym($_, "umowa#n#1")    // synonim

kind_of($_, "umowa#n#1")    // synonim lub hiponim
```

15. Utworzenie relacji

```
link( ref_from, ref_to, annotation_name)
```

Tworzy relację między anotacjami wskazanymi przez referencje `ref_from` i `ref_to` o nazwie `annotation_name`.

Przykład #11.2

```
apply(
    match(
          is('PERSON'),
          text('pracuje w'),
          is('COMPANY')
    )
    actions(
          link(0, 2, "EMPLOYEE_OF")
    )
)
```

16. Usunięcie relacji – do weryfikacji

Usunięcie relacji między anotacjami, np.:

```
unlink($a[0], $b[0], "MEMBER_OF")
```

17. Łączenie operatora `llook`, `rlook` z `match` – do weryfikacji

Mechanizm do znajdowania sekwencji tokenów względem innej sekwencji tokenów, np:

```
match($m,
    inter(base[$_],"podpisać"),
    inter(base[$_],"umowa")
),
llook(
    match($a,
          is($_, "COMPANY")
    )
),
rlook(
    match($b,
          is("COMPANY”)
    )
)
$e = mark($m[0], $m[1], "EVENT_AGREEMENT_SIGNUP"),
link($e, $a[0], "AGREEMENT_SIDE"),
link($e, $b[0], "AGREEMENT_SIDE")
```

18. Operator `exists`

Sprawdza, czy istnieje w badanym kontekście sekwencja o podanych warunkach.

TODO

19. Kontekst zdania i dokumentu

Możliwe powinno być określenie kontekstu działania operatora (zdanie, dokument, okno), w szczególności dla operatorów `rlook`, `llook`, `exists`.

TODO

Przykładowe reguły

Zaznacz wszystko co zaczyna się z dużych liter:

```
apply(
    match(
        regex("\p{Lu}{Ll}*")
    ),
    actions(
        match("UPPERCASE")
    )
)
```

Oznaczenie nazw miejscowości ze słowników:

```
apply(
    match(
        and(
            inter(nmb, {"sg"}),
            inter(flex, {"subst"}),
            inter(base, {"miasto", "miasteczko", "miejscowość"})
        ),
        and(
            is("CITY_GAZ_BASED"),
            inter(cas, {"nom", cas[0]})
        )
    ),
    actions(
        mark(1, "GAZ_CITY")
    )
)
```

Oznaczenie nazw miejscowości po słowach kluczowych:

```
apply(
    match(
        and(
            inter(nmb, {"sg"}),
            inter(flex, {"subst"}),
            inter(base, {"miasto", "miasteczko", "miejscowość"})
        ),
        repeat( is("UPPERCASE") )
    ),
    actions(
        mark(1, "KEY_CITY")
    )
)
```

Oznaczenie nazw regionów na podstawie słownika:

```
apply(
    match(
          and(
                inter(nmb, {"sg"}),
                inter(flex, {"subst"}),
                inter(base, {"region"})
          ),
          and(
                is("REGION_GAZ_BASED"),
                inter(cas, {"nom", cas[0]})
          )
    ),
    actions(
          mark(1, "GAZ_REGION")
    )
)
```

Analogicznie oznaczenie REGION dla słów „gmina” , „powiat”, „województwo”.

Oznaczenie dla nazw zaczynających się od słowa Morze, Ocean, Jezioro z dużych liter.

```
apply(
    match(
          and(
                is("UPPERCASE"),
                inter(flex, {"subst"}),
                is("INT_PROOF_GEOG_NAME")
          ),
          inter(flex, {"adj"})
    ),
    actions(
          mark(1, "GEOG_NAME")
    )
)
```

Oznaczenie skrótów ulic, placów itp.

```
apply(
    match(
          inter(base, {"ul", "pl", "al"}),
          optional( text(".") )
    ),
    actions(
          match("URABN_ABBREV")
    )
)
```