Kacper Roszczyna

Projekt TKOM

Sprawozdanie Wstępne

Temat: Język maszyny reguł do inwestowania na giełdach walutowych i akcyjnych.

I. Cele

* Głównym celem projektu jest napisanie kompilatora języka maszyny reguł umożliwiającej inwestycje giełdowe.

II. Opis Gramatyki

1. Ogólny plików składowych programu:

1. Pliki konfiguracyjne definiujące dostępne produkty walutowe i produkty akcyjne.

2. Pliki konfiguracyjne zawierające listę zdarzeń, czyli zmian kursów walutowych i cen akcji.

3. Pliku programu, który zawiera listę reguł złożonych z warunków wejścia oraz akcji.

4. Pliku startowego, który zawiera początkowy stan posiadania inwestora.

1. Składnia pliku konfiguracyjnego zawierającego produkty walutowe i akcyjne. Jest to plik .xml, który w ramach zadania będzie odczytywany poprzez gotową bibliotekę i w przypadku nie wystąpienia elementów obowiązkowych lub wystąpienia elementów nadmiarowych rzucić błąd. Przykładowy plik configuracyjny.

* + Tag „*products”*: znacznik otwierający plik configuracyjny produktów. Jest to znacznik obowiązkowy.
  + Tag „*currencies”:* znacznik oznacza rozpoczęcie bloku definicji walut. Jego nieobecność w pliku wskazuje, że nie są definiowane żadne produkty walutowe. Obecność znacznika nie jest obowiązkowa. Blok może być pusty.
  + Tag „*currency”:* znacznik musi wystąpić wewnątrz bloku „*currencies”*. Gdy zostanie wykryty traktowane jest to jako początek definiowania nowego produktu walutowego.
    - Na definicję produktu walutowego składają się dwa elementy nazwa i skrót, które oznaczone są:
      * „*name”:* znacznik, który zawiera pełną nazwę produktu. Jest obowiązkowy.
      * „*abbreviation”*: skrót nazwy produktu, który musi być unikatowy ze wszystkich produktów walutowych. W przypadku powtórzenia się skrótu nazwy kompilator rzuci błąd ze wskazaniem na zduplikowaną nazwę. Znacznik jest obowiązkowy.
  + Tag „*stocks”*: znacznik oznacza rozpoczęcie bloku definicji produktów akcyjnych. Znacznik nie jest obowiązkowy, jego brak oznacza, że plik konfiguracyjny nie definiuje żadnych produktów giełdowych. W przypadku pustej sekcji oznacza brak definicji produktów.
  + Tag „*stock”*: znacznik oznacza rozpoczęcie definicji produktu giełdowego. Znacznik ten musi znajdować się wewnątrz sekcji „*stocks*”.
    - Na definicję produktu akcyjnego składają się dwa elementy: nazwa i waluta, w której można kupować akcje.
      * „*name”:* nazwa produktu. Nazwa musi być unikatowa ze wszystkich nazw produktów akcyjnych. Znacznik jest obowiązkowy. W przypadku powtórzenia się nazwy kompilator rzuci błędem i wskaże zduplikowaną nazwę.
      * „*currency*”: skrót nazwy waluty, w której można kupić akcje. Jeżeli określony skrót nie zostanie zdefiniowany w żadnym pliku konfiguracyjnym kompilator rzuci błędem ze wskazaniem na ten skrót.

<products>

<currencies>

<currency>

<name>yuan</name>

<abbreviation>YUA</abbreviation>

</currency>

<currency>

<name>yen</name>

<abbreviation>YEN</abbreviation>

</currency>

<currency>

<name>dollar USA</name>

<abbreviation>USD</abbreviation>

</currency>

</currencies>

<stocks>

<stock>

<name>CocaCola</name>

<currency>USD</currency>

</stock>

</stocks>

</products>

2. Składnia pliku zdarzeń zawierającego zmiany kursów walut i produktów akcyjnych. Jest to plik .xml, który w ramach zadania będzie odczytywany poprzez gotową bibliotekę i w przypadku nie wystąpienia elementów obowiązkowych lub wystąpienia elementów nadmiarowych rzucić błąd. Przykładowy plik zdarzeń:

* *„events\_config:* tag najwyższego poziomu, jego obecność jest obowiązkowa.
* „*start”*: tag wskazujący początkowy stan systemu. Nie jest obowiązkowy, ale musi wystąpić w przynajmniej jednym pliku zdarzeń. Wewnątrz tego bloku określane są wszystkie startowe wartości walut i ich przeliczniki oraz wszystkie startowe kursy akcji. Jeżeli wartość startowa, któregokolwiek z elementów nie zostanie określona to kompilator rzuci błędem. Jeżeli żaden blok start nie zostanie znaleziony to kompilator rzuci błędem.
  + „*date”:* zawiera datę w formacie yyyy.MM.dd. Tag ten musi pojawić się przynajmniej w jednym bloku startowym. Brak obecności skutkuje błędem kompilatora.
  + „*currencies”:* blok wewnątrz, którego definiowane są kursy początkowe walut, czyli zgodne z dniem wskazanym przez pole „date”. Złożony z wielu elementów „*currency”.* Blok może być pusty, może też nie wystąpić.
    - „*currency”:* blok definiujący początkową wartość określonej waluty w przeliczeniu na fikcyjną walutę uniwersalna.
      * „*name”*: skrót nazwy waluty. Skrót musi zostać zdefiniowany w plikach konfiguracyjnych.
      * „*value”*: jest to wartość zmiennoprzecinkowa o dwóch miejscach znaczących, która oznacza ilość wydanej waluty celem nabycia jednej jednostki waluty fikcyjnej.
  + „*stocks*”: blow wewnątrz, którego definiowane są kursy początkowe akcji, czyli zgodne z dniem wskazanym przez pole „date”. Złożony z wielu elementów „*stock”.* Może być pusty lub nie występować w poszczególnych plikach, ale wszystkie kursy początkowe muszą być zdefiniowane.
    - „*stock”*: blok definiujący początkową wartość określonej akcji w walucie przypisanej do danych akcji.
      * „*name”:* nazwa akcji, musi wystąpić w plikach konfiguracyjnych.
      * „*price*”: cena w walucie przypisanej do akcji. Liczba zmiennoprzecinkowa z „.” jako separatorem, o dwóch miejscach znaczących.
  + „*events”:* blok definiujący zdarzenia zachodzące w rzeczywistości, na skutek, których będą wykonywane różne reguły zdefiniowane w programie. Sekcja może nie występować lub być pusta.
    - „*event”:*  definicja pojedynczego zdarzenia, wszystkie jej pola są obowiązkowe.
      * „*type”:*  może przyjmować dwie wartości: STOCK lub CURRENCY. Które odpowiadają zmianie kursu akcji lub zmianie kursu waluty.
      * „*name”:* nazwa akcji lub skrót nazwy waluty. Nazwa ta musi występować w plikach konfiguracyjnych.
      * „*date”:* data nastąpienia zdarzenia. Jeżeli data zdarzenia jest ustawiona wcześniej niż data startu to kompilator wyrzuci ostrzeżenie. Format daty to yyyy.MM.dd.
      * „*change”:* zmiana w punktach procentowych, w przypadku spadku poprzedzona znakiem „-”. Separatorem jest znak „.”. Dostępna jest precyzja zmiany do 4 cyfr znaczących.

<events\_config>

<start>

<date>2016.05.09</date>

<currencies>

<currency>

<name>USD</name>

<value>1.75</value>

</currency>

</currencies>

<stocks>

<stock>

<name>CocaCola</name>

<price>300.00</price>

</stock>

</stocks>

</start>

<events>

<event>

<type>STOCK</type>

<name>CocaCola</type>

<date>2016.05.10</date>

<change>17.0</change>

</event>

</events>

</events\_config>

3. Plik startowy zawiera startowy stan własności inwestora. Jest to plik XML, który opisuje ile jakiej waluty i akcji posiada inwestor. Plik jest parsowany za pomocą gotowej biblioteki i w przypadku nie wystąpienia elementów obowiązkowych lub wystąpienia elementów nadmiarowych rzucić błąd. Przykładowy plik:

* „*start”:* tag otwierający przetwarzanie, jest obowiązkowy.
  + „*currencies”:* blok wewnątrz, wewnątrz którego definiowane są . Złożony z wielu elementów „*currency”.* Blok może być pusty, może też nie wystąpić.
    - „*currency”:* blok definiujący ilość posiadanej waluty przez inwestora
      * „*name”*: skrót nazwy waluty. Skrót musi zostać zdefiniowany w plikach konfiguracyjnych.
      * „*value”*: jest to wartość zmiennoprzecinkowa o dwóch miejscach znaczących, która oznacza ilość posiadanej waluty.
  + „*stocks*”: blok wewnątrz, którego definiowane są posiadane akcje. Złożony z wielu elementów „*stock”.* Może być pusty lub nie występować w poszczególnych plikach, ale wszystkie kursy początkowe muszą być zdefiniowane.
    - „*stock”*: blok definiujący początkową liczbę posiadanych akcji
      * „*name”:* nazwa akcji, musi wystąpić w plikach konfiguracyjnych.
      * „*amount*”: liczba posiadanych akcji.

<start>

<currencies>

<currency>

<name>USD</name>

<value>1.75</value>

</currency>

</currencies>

<stocks>

<stock>

<name>CocaCola</name>

<amount>300</amount>

</stock>

</stocks>

</start>

4. Plik programu

* Korzystanie z atrybutów modelu:
  + Inwestowanie i formułowanie reguł inwestycyjnych jest oczywiście zależne od naszego stanu posiadania oraz tego jakie są ceny akcji lub kursy walut. Model świata inwestycyjnego można podzielić między akcje (stock) i waluty (currency).
  + Liczby zmiennoprzecinkowe definiowane są z separatorem „ , ”.
  + *currency :* słowo kluczowe, które pozwala odnieść się do poszczególnych atrybutów. Atrybuty od słowa klucza oddzielone są przez „.” i skrótem nazwy waluty.
  + *have:* słowo kontekstowe wskazujące stan posiadania inwestora. Stosowane w kombinacjach z ze słowami *currency* i *stock*
    - *currency.[<skrót>].rate :* aktualny kurs waluty, czyli ile trzeba zapłacić tej waluty za jednostkę fikcyjnej waluty.
    - *currency.[<skrót>].have.amount :* aktualna ilość posiadanej waluty, liczba z dwoma miejscami po przecinku
    - *currency.[<skrót>].have.value :* aktualna wartość w posiadanej waluty w fikcyjnej walucie
    - Każdy z atrybutów przyjmuje dodatkowy argument, którym może być data lub liczba całkowita. Oznaczają one wartość z dnia podanego lub tyle dni wstecz. W przypadku podania wartości dodatniej minus jest automatycznie dodany. Podana data musi mieć format yyyy.MM.dd. Argumenty otoczone są nawiasami okrągłymi.
      * Przykład: currency.[YUA].value(2016.08.10) – kurs YUA z dnia 10.08.2016.
      * Przykład 2: currency.[YUA].value(10) – kursu YUA 10 dni temu.
  + *stock:* słowo kluczowe pozwalające się odnieść do atrybutów akcji giełdowych. Atrybuty od słowa klucza oddzielone są przez „.” i nazwą akcji.
    - *stock.[<stock\_name>].value :* aktualna wartość pojedynczej akcji w walucie, w której występuje ten produkt na rynku
    - *stock.[<stock\_name>].globalValue* : aktualna wartość pojedynczej akcji w jednostkach waluty fikcyjnej
    - *stock.[<stock\_name>].have.value :* aktualna suma wartości wszystkich akcji, które posiada inwestor w walucie produktu
    - *stock.[<stock\_name>].have.globalValue*: aktualna suma wartości wszystkich akcji, które posiada inwestor w walucie fikcyjnej
    - *stock.[<stock\_name>].have.amount:* liczba akcji posiadanych przez inwestora
    - Każdy z atrybutów przyjmuje dodatkowy argument, którym może być data lub liczba całkowita. Oznaczają one wartość z dnia podanego lub tyle dni wstecz. W przypadku podania wartości dodatniej minus jest automatycznie dodany. Podana data musi mieć format yyyy.MM.dd. Jeżeli format daty się nie zgadza to jest rzucany błąd. Argumenty otoczone są nawiasami okrągłymi.
      * Przykład: stock.[CocaCola].value(2016.08.10) – kurs Coca Coli z dnia 10.08.2016.
      * Przykład 2: stock.[CocaCola].value(10) – kursu Coca Coli 10 dni temu.
* Sekcja konfiguracji:
  + *config –* słowo kluczowe poprzedzające nazwę pliku konfiguracyjnego, czyli takiego, który definiuje waluty i akcje. Plik musi znajdować się w tym samym katalogu co główny plik reguł. Brak pliku skutkuje ostrzeżeniem.
    - Przykład: config [example\_config.xml]
    - Blok konfiguracyjny: config [<plik\_konfiguracji.xml>]
    - Może wystąpić wiele bloków konfiguracyjnych ale musi się pojawić co najmniej 1.
  + *events –* słowo kluczowe poprzedzające nazwę pliku ze zdarzeniami, czyli takiego, który definiuje zdarzenia zaszłe w symulacji. Plik musi znajdować się w tym samym katalogu co główny plik reguł. Brak pliku skutkuje ostrzeżeniem.
    - Przykład: events [example\_events.xml]
    - Blok zdarzeń: events [<plik\_zdarzen.xml>]
    - Może wystąpić wiele bloków zdarzeń ale musi się pojawić co najmniej 1.
  + *start –* słowo kluczowe poprzedzające nazwę pliku z początkowym stanem posiadania inwestora. Plik musi znajdować się w tym samym katalogu co główny plik reguł. Brak pliku skutkuje ostrzeżeniem.
    - Przykład: start [start\_config.xml]
    - Blok startowy: start [<plik\_startu.xml>]
    - Może wystąpić wiele bloków startu, ale musi pojawić się co najmniej 1.
* Sekcja reguł:
  + Rozpoczyna ją pierwsza zadeklarowana reguła. Reguły można deklarować na dwa sposoby:
    - *rule [<plik\_reguly.invrl>],* gdzie plik\_reguly.invrl jest plikiem deklarującym regułę zgodnie z drugim typem deklaracji. Można zawrzeć dowolną liczbę takich deklaracji.
    - Pełną deklaracją reguły.
* Pełna deklaracja reguły.
  + Rozpoczęta słowem kluczowym: *rule.*
  + Ograniczona za pomocą nawiasów {}
  + Każde pole reguły jest zakończone „;”
  + Nazwy pól reguły od wartości oddzielone są za pomocą „:”
  + Tak więc ogólnie przyjmują postać: *nazwa\_pola: wartosc\_pola;* Pola dla reguł są obowiązkowe
    - Pierwszym polem każdej reguły jest *id*. Jest to wartość liczbowa, która jednoznacznie identyfikuje regułę. Nie mogą wystąpić dwie reguły o jednakowym id. Jeżeli tak się stanie to kompilacja zostanie przerwana z błędem.
      * Przykład: id: 10;
    - Drugim polem każdej reguły jest *priority.* Określa ona priorytet sprawdzenia warunków wejścia do reguły i wykonania jej akcji. Priorytet jest dodatnią liczbą całkowitą. W przypadku gdy ten warunek nie zostanie spełniony to kompilacja zostanie przerwana z błędem. Reguły sprawdzane są od najwyższego do najniższego priorytetu.
      * Przykład: priority: 55;
    - Trzecim polem każdej reguły jest *condition*. Pole to zawiera wartość logiczną prawda/fałsz, która jest określana na podstawie warunków logicznych. Puste pole jest traktowane jako prawda logiczna. Pole zakończone jest średnikiem
      * Warunek logiczny: Wyrażenie, które przyjmuje dwie wartości prawda lub fałsz. Powstaje dzięki operatorom relacji.
      * Wspierane operatory relacji: <, >, >=, <=, ==, !=. Operatory porównują dwie wartości i zwracają logiczną prawdę, jeżeli porównanie zachodzi lub fałsz gdy nie zachodzi. Porównywać można jedynie wartości liczbowe. Porównanie dwóch wartości, które nie są liczbami będzie skutkowało otrzymaniem logicznego fałszu.
        + Przykład: 5 < 3 (zwróci fałsz), 3< 5 (zwróci prawda)
      * Warunki logiczne można łączyć za pomocą operatorów logicznych. Wspierane operatory logiczne: || logiczny OR, && logiczny AND, ! Logiczny NOT.
        + Przykład 5 < 3 || 3 < 5 (zwróci prawda)
      * Wspierane są operatory matematyczne: +, -, \*, /. Przy czym operator dzielenia działa jak dzielenie liczb zmiennoprzecinkowych. (eg. 5/2 = 2,5).
      * Warunki logiczne mogą być grupowane za pomocą nawiasów okrągłych „()”. Wtedy za całe wyrażenie w nawiasach wstawiana jest finalna wartość logiczna, która je prezentuje.
      * Specjalne mechanizmy:
        + Wykonanie reguły tylko, jeżeli inna reguła (musi mieć niższy priorytet została wykonana). Służy do tego następująca konstrukcja: *rule.<id\_reguly\_wykonanej>.executed.* Żeby uzyskać warunek przeciwny wystarczy wstawić „!” przed warunek: *!rule.<id\_reguly\_wykonanej>.executed.*
        + Obserwacja trendów jako warunek wykonania. Można obserwować tylko zmianę danych liczbowy. Do zdefiniowania trendu służy konstrukcja: *<inc/desc> <obser\_wartość> <by> <operator\_relacji ze zbioru: <, >, <=, >=> <x%/-y%> in <number\_of\_days>.* Aby pokazać przykład: *inc stock.CocaCola.value by < 3% in 4.* Interpretacja tego jest: Zwróć prawdę jeżeli w ciągu ostatnich 4 dni całkowita wartość wzrosła o nie więcej niż 3%, w przeciwnym wypadku zwróć fałsz.
    - Ostatnim polem każdej reguły jest pole *action.* Pole to zawiera spis czynności, które mają się wykonać jeżeli spełniony zostanie warunek wejścia, czyli jeżeli po ewaluacji pole *condition* będzie zawierać wartość logicznej prawdy.
      * Pole rozpoczyna się słowem *action* zakończonym „ : ”.
      * Pole zawiera listę akcji oddzielonych przecinakami.
      * Budowa pojedynczej akcji:
        + Akcje zaczynają się od czynności: *buy/sell* – oznacza to nabyj lub sprzedaj
        + Potem następuje identyfikacja obiektu do nabycia lub sprzedaży, wykonuje się ją po nazwie. Czyli w przypadku produktów giełdowych: *stock.[<stock\_name>],* natomiast w przypadków walutowych *currency.[*<*currency\_name>]*.
        + W przypadku sprzedaży następnie mogą nastąpić trzy różne konstrukcje:

*part <1-100>:* sprzedaj część swoich swoich odpowiadającą liczbie procentów wskazanej

*amount <liczba/ALL>:* sprzedaj określoną liczbę akcji lub wszystkie w przypadku użycia słowa ALL

*for <liczba dodatnia>:* sprzedaj tyle akcji aby uzyskać z nich co najmniej tyle waluty w jakiej była ta akcja co podana liczba

*for <liczba dodatnia> absolute:* sprzedaj tyle akcji aby uzyskać z nich co najmniej tyle waluty fikcyjnej co liczba dodatnia podana

* + - * + W przypadku nabycia mogą nastąpić następujące konstrukcje:

dla walut:

*amount <liczba dodatnia całkowita/MAX> for <currency.[<currency\_name>]/ANY> :* nabyj określoną ilość waluty za inną walutę lub za dowolną walutę, brana jest wtedy losowo waluta, którą posiadamy, jeżeli jej nie wystarczy to losowana jest kolejna, aż do wyczerpania walut. Słowo kluczowe MAX oznacza nabycie takiej ilości jaka jest tylko możliwa.

dla produktów giełdowych:

*amount <liczba dodatnia całowita/MAX> for <currency.currency<name>/ANY/OWN>:* Kup wskazaną liczbę akcji, MAX - maksymalną możliwą liczbę akcji za wskazaną walutę, ANY – dowolnie wylosowaną walutę, OWN – walutę przypisaną do produktu akcyjnego.

Waluty, za które kupowane są akcje można listować za pomocą składni: {<currency.currency\_name>,<currency1.currency\_name> …}. Wtedy do wykonania akcji zostaną użyte wszystkie wskazane waluty w podanej kolejności. Przykładowo jeżeli kupujemy akcje Microsoftu za USD i YUA i skończą się na drodze kupna dolary to reszta akcji będzie dokupowana w USD.

* + - * + Akcje zawsze się kończą powodzeniem lub nie. Jeżeli, którakolwiek z akcji na liście się nie uda to nastąpi „fallthrough”, czyli żadna z akcji nie zostanie wykonana. Reguła zostanie oznaczona jako niewykonana i program będzie kontynuował działanie.
    - Dodatkowe uwagi:
      * Dozwolone są komentarze:
        + jednoliniowy rozpoczęty „//”

Przykładowy program:

config [example\_config.xml] //comment multiple configs allowed

events [example\_events.xml]

start [start\_config.xml]

rule [rule1.invrl]

rule {

id: 1;

priority:10;

condition: currency.[YUA].rate > 300

|| (stock.[CocaCola].value <= 12

&& stock.[CocaCola].have.amount > 15)

|| rule.[15].executed

|| inc stock.[CocaCola].value by 5% in 5;

actions: sell stocks.[CocaCola] amount 75,

sell stock.[NukaCola] part 75,

sell stock.[NukaCola] amount ALL,

sell stock.[CocaCola] for 10000,

sell stock.[CocaCola] for 10 absolute,

buy currency.[YUA] amount 77 for currency.[USD],

buy currency.[YUA] amount 800 for ANY,

buy currency.[YUA] amount MAX for ANY,

buy stock.[CocaCola] amount 10 for OWN,

buy stock.[CocaCola] amount MAX for ANY,

buy stock.[CocaCola] amount MAX for currency.[YUA],

buy stock.[CocaCola] amount MAX for {currency.[YUA], currency.[USD]};

}

III) Formalny opis gramatyki

<program> ::= <declarations><rule\_defs>

<rule\_defs> ::= „”|<rule\_def>|<rule\_def><rule\_defs>

<rule\_def> ::=<rule><opt\_format\_char> „{„

<id\_decl>

<prio\_decl>

<cond\_decl>

<actions\_decl>

„}”<new\_lilne>

<id\_decl> ::= <opt\_format\_char><id>”:”<opt\_whitespace><number>”;”

<prio\_decl> ::= <opt\_format\_char><priority>”:”<opt\_whitespace><number>”;”

<declarations> ::= <config\_decl><events\_decl><start\_decl><rules\_decl>

<config\_decl> ::= <config\_decl\_line> | <config\_decl\_line><config\_decl>;

<config\_decl\_line> ::= <config><decl\_line\_ending>;

<events\_decl> ::=<events\_decl\_line> | <events\_decl\_line><events\_decl>;

<events\_decl\_line> ::= <events><decl\_line\_ending>;

<start\_decl> ::= <start\_decl\_line> | <start\_decl\_line><start\_decl>;

<start\_decl\_line> ::= <start><decl\_line\_ending>;

<rules\_decl> ::= <rule\_decl\_line> | <rule\_decl\_line><rules\_decl>

<rule\_decl\_line> ::= <rule><decl\_line\_ending>;

<decl\_line\_ending> ::= <opt\_whitespace><given\_name><opt\_comment><new\_line>;

<given\_name> ::= „[„ <name> „]”;

<name> ::= <name\_character> | <name\_chacter><name>;

<opt\_comment> ::= „” | <comment>

<comment> ::= <comment\_start><comment\_text>;

<comment\_text> ::= <comment\_character> | <comment\_character><comment\_text>;

<name\_character> ::= <letter> | <digit> | <nonbracket\_symbol> | <opt\_whitespace>;

<comment\_character>::= <letter> | <digit> | <opt\_whitespace>

<character> ::= <letter> | <digit> | <symbol>;

<opt\_format\_char> ::= <opt\_whitespace> | <new\_line>;

<new\_line> ::= „\r\n” | „\n”;

<opt\_whitespace> ::= „\t” <opt\_whitespace> | „ „ <opt\_whitespace> | „”

<letter> ::= "A" | "B" | "C" | "D" | "E" | "F" | "G" | "H" | "I" | "J" | "K" | "L" | "M" | "N" | "O" | "P" | "Q" | "R" | "S" | "T" | "U" | "V" | "W" | "X" | "Y" | "Z" | "a" | "b" | "c" | "d" | "e" | "f" | "g" | "h" | "i" | "j" | "k" | "l" | "m" | "n" | "o" | "p" | "q" | "r" | "s" | "t" | "u" | "v" | "w" | "x" | "y" | "z"

<digit> ::= "0" | <nonzero\_digit>;

<nonzero\_digit> ::= "1" | "2" | "3" | "4" | "5" | "6" | "7" | "8" | "9";

<number> ::= <digit> | <nonzero\_digit><number>

<symbol> ::= <nonbracket\_symbol> | <brackets>;

<nonbracket\_symbol>::= "|" | " " | "-" | "!" | "#" | "$" | "%" | "&" | "(" | ")" | "\*" | "+" | "," | "-" | "." | "/" | ":" | ";" | "<**" | "=" | "**>" | "?" | "@" | "\" | "^" | "\_" | "`" | "{" | "|" | "}" | "~";

<brackets> ::= "[" | "]";

<comment\_start> ::=”//”

<space> ::= „ „

<rule> ::=”rule”

<start> ::=”start”

<events> ::=”events”

<config> ::=”config”

<id> ::=”id”

<priority> ::=”priority”

<condition> ::=”condition”

<actions> ::=”actions”

<stock> ::=”stock”

<currency> ::=”currency”

<have> ::=”have”

<value> ::=”value”

<amount> ::=”amount”

<max> ::= „MAX”

<for> ::=”for”

<any> ::=”ANY”

<own> ::=”OWN”

<absolute> ::=”absolute”

<all> ::=”ALL”

IV) Realizacja

Zadanie obejmuje realizację dwóch elementów:

* parsera
* leksera

Realizacja zadania:

Projekt będzie zrealizowany z użyciem języka Python. Pliki konfiguracyjne, wydarzeń i startowe zostaną sparsowane za pomocą gotowych bibliotek. Parsowanie języka maszyny zasad jest celem projektu i zostaną zaimplementowane elementy wymienione powyżej.

Projekt testów:

Aby zapewnić działanie dostarczonego kompilatora każdy obiekt wchodzący w jego skład, oraz wszystkie niezbędne funkcje zostaną przetestowane jednostkowo asercjami. Zapewni to, że składniki programu działają. Zastosowanie testów jednostkowych dodatkowo wymusi wyższą jakość kodu, gdyż „dobry kod to testowalny kod”.