

Dokumentacja wstępna

Opis struktur binarnych

Grzegorz Lewczuk

Projekt realizowany na przedmiot TKOM

31.03.2017

1 Opis projektu

Język deklaracyjny opisujący struktury binarne (z możliwą dokładnością do pojedynczych bitów). Projekt powinien umożliwiać zakodowanie i zdekodowanie dowolnej opisanej struktury i prezentację jej w wybranym formacie. Powinno być możliwe zdefiniowanie pól zależnych - np. pole Length i Contents.

Projekt zakłada opis struktury binarnej za pomocą języka opartego na **ASN1**, na potrzeby projektu używana będzie nazwa **MiniASN**, zważywszy na uproszczoną funkcjonalność języka względem pierwowzoru.

Opis funkcjonalności

Dopuszczone są typy zmiennych **UINT**, **BITSTRING** oraz **BOOL** pierwszy przechowuje liczbę całkowitą nieujemną, drugi ciąg bitów, trzeci wartość boolowską. Pierwsze dwa typy mogą być parametryzowane, typ zmiennej parametryzujemy liczbą bitów przeznaczonych na jej zapis. Standardowo

`UINT ::= UINT_16`

`BITSTRING ::= BITSTRING_8`

Opis struktury binarnej będzie zbiorem sekwencji zmiennych

Dodatkowo sparametryzowany będzie typ **CHOICE**[zmienna **UINT**], który będzie przyjmował wielkość w bitach odpowiednią dla pola spełniającego warunek. W strukturze **CHOICE** powinna być także wartość domyślna, gdy żadna inna nie spełni swojego warunku.*

* pokazane dokładniej w przykładowym kodzie

Możliwa będzie także deklaracja tablic parametryzowana identyfikatorem zmiennej lub liczbą zawierającą ilość elementów oraz opisem zawartości każdego elementu

2 Składnia i opis języka

Lista tokenów

'SEQUENCE' , 'CHOICE' , 'ARRAY' , 'UINT' , 'BITSTRING' ,
'BOOL' , '::=' , '_' , '[' , ']' , '{' , '}' , '==' , '<' ,
'>' , '<=' , '>=' , '!=' , ',' , 'DEFAULT' , 'AND' , 'OR' ,
'TRUE' , 'FALSE'

Gramatyka

```
data_structure = { sequence }
sequence = id '::=' 'SEQUENCE' '{' { declaration } '}'

declaration = ( variable_declaration | choice_declaration | array_declaration )
choice_declaration = id '::=' 'CHOICE' '[' id ']' '{'
    { choice_variable_declaration } choice_variable_default_declaration '}'
choice_variable_declaration = variable_declaration '[' expression ']'
choice_variable_default_declaration = variable_declaration '[' 'DEFAULT' ']'
array_declaration = id '::=' 'ARRAY' '[' ( id | number ) ']' '{' { type } '}'
variable_declaration = id type

type = ( simple_type | complex_type )
complex_type = simple_parameterized_type '_' number
simple_parameterized_type = ( 'UINT' | 'BITSTRING' )
simple_type = ( simple_parameterized_type | 'BOOL' )

expression = and_expression { 'OR' and_expression }
and_expression = simple_expression { 'AND' simple_expression }
simple_expression = ( equality_expression | relational_expression )
equality_expression = id equality_operator value
relational_expression = id relational_operator value

equality_operator = ( '==' | '!=' )
relational_operator = ( '<' | '>' | '<=' | '>=' )

value = ( number | boolean )

number = ( digit_without_zero { digit } | digit )
digit = ( digit_without_zero | '0' )
digit_without_zero = '1'..'9'

boolean = ( 'TRUE' | 'FALSE' )

id = letter { ( letter | digit ) }
letter = ( 'a'..'z' | 'A'..'Z' )
```

3 Wymagania

Wymagania funkcjonalne

1. Odczyt danych binarnych na podstawie dostarczonej struktury danych
2. Sprawdzanie poprawności struktury danych (dostarczonego kodu)
3. Poprawne zdekodowanie danych binarnych do postaci wygodnej do odczytu dla człowieka

Wymagania niefunkcjonalne

1. Jak najdokładniejsze informacje o błędach w strukturze danych

4 Sposób uruchomienia

Aplikacja uruchamiana będzie wraz z dostarczonym opisem struktury danych oraz plikiem binarnym zawierającym dane. Na ekranie powinien wyświetlić się odpowiedni wynik dekodowania zrozumiały dla człowieka.

Uruchomić będzie można komendą, np:

```
python miniASN.py struktura.miniasn dane.bin
```

6 Przykładowy kod

```
MojaSekuencja ::= SEQUENCE {  
    a      UINT  
    b      UINT_11  
    c      BITSTRING  
    d      BITSTRING_20  
    e      BOOL  
}
```

```

MojaSekwencjaChoice ::= SEQUENCE {
    a      UINT_8
    b      ::= CHOICE[a]
    {
        c      UINT[a > 0 AND a < 100]
        d      BOOL[a == 0]
        e      BITSTRING_16[a == 100 OR a == 110]
        f      DITSTRING[DEFAULT]
    }
    g      ::= CHOICE[b]
    {
        i      BOOL[b == TRUE]
        j      UINT[DEFAULT]
    }
}

```

```

MojaSekwencjaArray ::= SEQUENCE {
    a      UINT_8
    b      ::= ARRAY[5]
    {
        c      UINT
        d      BITSTRING
    }
}

```

```

MojaSekwencjaComplex ::= SEQUENCE {
    a      UINT_8
    b      ::= ARRAY[5]
    {
        c      UINT
        d      BITSTRING
    }
}

```