

AKADEMIA GÓRNICZO-HUTNICZA IM. STANISŁAWA
STASZICA

KRAKÓW

Generator specyfikacji logicznej

Autorzy:

Marcin JĘDRZEJCZYK
Paweł OGORZAŁY

Prowadzący:

Dr inż. Radosław KLIMEK

29 sierpnia 2016

1 Cel projektu

Celem projektu jest wytworzenie programu, który dla podanego diagramu będzie w stanie go sparsować do formatu pozwalającego na wygenerowanie specyfikacji logicznej.

2 Powód tworzenia generatora

- Ręczne tworzenie specyfikacji logiki jest trudne dla niedoświadczonych w tym użytkowników.
- Formalna weryfikacja modelu oprogramowania pozwala obniżyć koszty i zwiększyć niezawodność.
- Brak takich narzędzi.

3 Ważne

- Diagram aktywności musi składać się z wcześniej zdefiniowanych wzorców, zagnieżdżanie jest dozwolone.
- Diagram aktywności składa się tylko z atomicznych aktywności, zidentyfikowanych podczas tworzenia scenariuszy przypadków użycia.
- Generator musi działać automatycznie, usuwa to błąd ludzki.

4 Specyfikacja logiczna

Automatyzacja generowania specyfikacji logicznej, rozumianej jako zestawu temporalnych formuł logicznych jest kluczowym zagadnieniem. Opiera się ono na kilku założeniach:

- Modele oprogramowania są opracowane jako przepływy pracy. Przepływ to postępujące zdarzenie, zadanie, interakcja obejmujące proces pracy.
- Przepływy pracy są opracowane przy użyciu zdefiniowanych wzorców.
- Każdy wzorzec przepływu pracy jest powiązany z zdefiniowanymi wcześniej formułami logicznymi opisującymi własności wzorca.

Elementarny zestaw formuł oznaczony $elem(a_1, \dots, a_n)$ lub po prostu $elem()$, nad atomowymi formułami a_1, \dots, a_n , gdzie formuła atomowa jest podzielona na trzy podzbiory parami rozłączne:

- Pierwszy podzbiór, który zawiera co najmniej jeden element to *argumenty wejściowe*
- Drugi podzbiór, który może być pusty to *zwykłe argumenty*
- Trzeci podzbiór, który zawiera co najmniej jeden element to *argumenty wyjściowe*

jest zbiorem formuł f_1, \dots, f_m składniowo poprawnych oraz f_1, \dots, f_m gdzie $m > 0$ są temporalnymi formułami logicznymi to $elem() = \{f_1, \dots, f_m\}$.

Wprowadźmy pojęcie formuły wejścia/wyjścia należących do klasycznej logiki. f_{en} i f_{ex} opisują logiczne okoliczności związane z otwarciem i zamknięciem wzorca. f_{en} jest spełnione, gdy pewne początkowe działania są aktywne. Na przykład $a \wedge b$ dla f_{en} oznacza, że gdy wykonanie wzorca jest zainicjowane, obie aktywności a i b są spełnione. $a \vee b$ dla f_{ex} oznacza, że gdy wykonywanie wzorca ma zostać zakończone wtedy działania a oraz b są aktywne. Podsumowując f_{en} oraz f_{ex} opisują odpowiednio pierwsze i ostatnie aktywne działania wzorca.

Wzór zestawu formuł oznaczony $pat(a_1, \dots, a_n)$ lub upraszczając $pat()$, nad formułami atomicznymi a_1, \dots, a_n jest zbiorem $pat(a_1, \dots, a_n) \equiv \{f_{en}, f_{ex}\} \cup elem(a_1, \dots, a_n)$ a jej elementy są częściowo uporządkowane w taki sposób, że f_{en} jest zawsze pierwszym elementem, f_{ex} jest zawsze drugim elementem.

Dla każdego wzorca pat , $pat.f_{en}$ i $pat.f_{ex}$ oznacza odpowiednio wejściowe i wyjściowe formuły wzorca.

Wyrażenie logiczne W_L jest strukturą stworzoną na podstawie następujących reguł:

- każdy elementarny zestaw $pat(a_i)$, gdzie $i > 0$ i każde a_i jest formułą atomiczną, jest wyrażeniem logicznym
- każde $pat(A_i)$, gdzie $i > 0$ i każde A_i jest
 - formułą atomiczną, lub
 - wyrażeniem logicznym $pat()$

jest także wyrażeniem logicznym

Wyrażenie logiczne stworzone w powyższy sposób jest dobrze sformatowane. Prostym przykładem wyrażenia logicznego jest $w = Seq(Split(a, b, c), Cond(d, e, f))$ znaczenie wszystkich wzorców jest intuicyjne i nie jest formalnie zdefiniowane. $|w|$ oznacza długość wyrażenia logicznego, które jest liczbą wzorców w wyrażeniu. $w[i]$ oznacza wzór na i -tej pozycji w wyrażeniu logicznym w , np. $w[2] = Split$. $|w[i]|$ oznacza liczbę argumentów i -tego wzoru, np. $|w[1]| = 2$ a $|w[3]| = 3$.

Zestaw *predefiniowanych wzorców* Π jest to zestaw, który zawiera wszystkie dopuszczalne wzorce przepływu dla procesu rozwoju branego pod uwagę. Zbiór *wzorców własności* P , lub w skrócie *własności* P to zestaw logicznych atrybutów i cechy, które wzorce wymienione w Π posiadają. $'''$ to selektor odpowiedniego obiektu wzorca, np. $w[i]'P$ oznacza własność P , która zawiera formuły dla wzorca na i -tej pozycji w wyrażeniu w . $''$ jest selektorem formuły w elementarnym zbiorze, do którego odnosi się wzorec w wyrażeniu logicznym, np. $w[i]'P.f_{en}$ oznacza formułę f_{en} w elementarnym zbiorze i -tego wzorca w wyrażeniu. $'' \uparrow$ jest selektorem argumentów w wzorcu. $w[i] \uparrow a_j$ oznacza argument a_j i -tego wzorca w wyrażeniu logicznym w .

Niech w^c dla wyrażenia logicznego w będzie zagregowaną formułą wejściową (wyjściową), gdzie agregowana formuła jest wyliczana na podstawie następujących reguł:

- jeśli nie ma wzorca w miejscu jakiegokolwiek formuły atomicznej/argumentu, która składniowo należy do formuły f_{en} (lub f_{ex}) w , wtedy w^c jest równe f_{en} (w^x jest równe f_{ex}).

- jeśli istnieje wzorec, powiedzmy $t()$ w miejscu jakiegokolwiek atomicznego argumentu powiedzmy r , który składniowo należy do formuły f_{en} (lub f_{ex}) w , wtedy r jest zastępowane przez t^e (lub t^x) dla każdego przypadku.

Algorytm - Generowanie specyfikacji logicznej (A)

Input: Wyrażenie logiczne W_L , zdefiniowane wzorce przepływu(nie puste) P

Output: Specyfikacja logiczna L

```

 $L := 0$ 
for  $i := 1$  to  $|W_L|$  do
   $L2 := W_L[i]P' \setminus \{W_L[i]'P.f_{en}, W_L[i]'P.f_{ex}\};$ 
  for  $j := 1$  to  $|W_L[i]|$  do
    if  $W_L[i] \uparrow a_j$  is non-atomic then
       $agg := (W_L[i]'P \uparrow a_j())^e + "V" + (W_L[i]'P \uparrow a_j())^x;$ 
      replace in  $L2$  every pattern  $W_L[i]'P \uparrow a_j()$  by  $agg$ ;
    end if
  end for
   $L := L \cup L2$ 
end for

```

Specyfikacja logiczna L składa się ze wszystkich wzorców uzyskanych z wyrażenia logicznego W_L przy stałych właściwościach P

$$L(W_L, P) = \{f : f \in A(W_L, P)\}$$

5 Przykłady

Podane wzorce P :































- Sequence($f1, f4$)
 - $f1 \Rightarrow \diamond f4$
 - $\neg f1 \Rightarrow \neg \diamond f4$
 - $\Box \neg (f1 \wedge f4)$
- Concurrency($f1, f2, f3, f4$)
 - $f1 \Rightarrow \diamond f2 \wedge \diamond f3$
 - $\neg f1 \Rightarrow \neg (\diamond f2 \wedge \diamond f3)$
 - $f2 \wedge f3 \Rightarrow \diamond f4$
 - $\neg (f2 \wedge f3) \Rightarrow \neg \diamond f4$
 - $\Box \neg (f1 \wedge (f2 \vee f3))$
 - $\Box \neg ((f2 \vee f3) \wedge f4)$
 - $\Box \neg (f1 \wedge f4)$
- Branching($f1, f2, f3, f4$)
 - $f1 \Rightarrow (\diamond f2 \wedge \neg \diamond f3) \vee (\neg \diamond f2 \wedge \diamond f3)$
 - $\neg f1 \Rightarrow \neg ((\diamond f2 \wedge \neg \diamond f3) \vee (\neg \diamond f2 \wedge \diamond f3))$
 - $f2 \vee f3 \Rightarrow \diamond f4$

- $\neg(f2 \vee f3) \Rightarrow \neg \diamond f4$
- $\Box \neg(f1 \wedge f4)$
- $\Box \neg(f2 \wedge f3)$
- $\Box \neg(f1 \wedge (f2 \vee f3))$
- $\Box \neg((f2 \vee f3) \wedge f4)$
- LoopWhile(a,b,c,d)
 - $f1 \Rightarrow \diamond f2$
 - $\neg f1 \Rightarrow \neg \diamond f2$
 - $f2 \wedge c(f2) \Rightarrow \diamond c \wedge \neg \diamond f4$
 - $\neg(f2 \wedge c(f2)) \Rightarrow \neg(\diamond f3 \wedge \neg \diamond f4)$
 - $f2 \wedge \neg c(f2) \Rightarrow \neg \diamond f3 \wedge \diamond f4$
 - $\neg(f2 \wedge \neg c(f2)) \Rightarrow \neg(\neg \diamond f3 \wedge \diamond f4)$
 - $f3 \Rightarrow \diamond f2$
 - $\neg f3 \Rightarrow \neg \diamond f2$
 - $\Box \neg(f1 \wedge f2)$
 - $\Box \neg(f1 \wedge f3)$
 - $\Box \neg(f1 \wedge f4)$
 - $\Box \neg(f2 \wedge f3)$
 - $\Box \neg(f2 \wedge f4)$
 - $\Box \neg(f3 \wedge f4)$

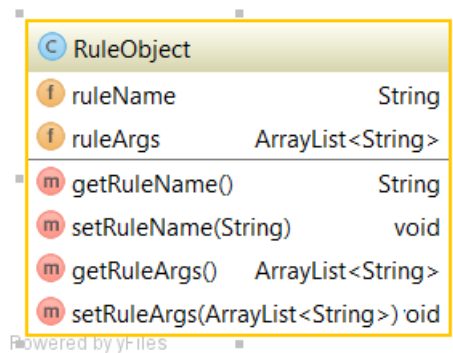
Wyjście programu dla: TUTAJ OBRAZEK UML, POTEM WYCIĄGNIĘTA FORMUŁA I WYNIK

- $W_L = \text{Concur}(a,b,c,d)$ to:
 $L = \{a \Rightarrow \diamond b \wedge \diamond c, \neg a \Rightarrow \neg(\diamond b \wedge \diamond c), b \wedge c \Rightarrow \diamond d, \neg(b \wedge c) \Rightarrow \neg \diamond d, \Box \neg(a \wedge (b \vee c)), \Box \neg((b \vee c) \wedge d), \Box \neg(a \wedge d)\}$
- $W_L = \text{Seq}(\text{Seq}(a,b),c)$ to:
 $L = \{a \Rightarrow \diamond b, \neg a \Rightarrow \neg \diamond b, \Box \neg(a \wedge b)\} \cup$
 $\cup \{a \Rightarrow c, \neg a \Rightarrow \neg \diamond c, \Box \neg(a \wedge c)\} \cup$
 $\cup \{b \Rightarrow c, \neg b \Rightarrow \neg \diamond c, \Box \neg(b \wedge c)\}$
- Branch(Seq(a,b),c,d,e) to:
 $L =$
 $\{a \Rightarrow \diamond b, \neg a \Rightarrow \neg \diamond b, \Box \neg(a \wedge b)\} \cup \{a \Rightarrow (\diamond c \wedge \neg \diamond d) \vee (\neg \diamond c \wedge \diamond d), \neg a \Rightarrow \neg((\diamond c \wedge \neg \diamond d) \vee$
 $\vee (\neg \diamond c \wedge \diamond d)), c \vee d \Rightarrow \diamond e, \neg(c \vee d) \Rightarrow \neg \diamond e, \Box \neg(a \wedge e), \Box \neg(c \wedge d), \Box \neg(a \wedge (c \vee d)),$
 $\Box \neg((c \vee d) \wedge e)\} \cup \{b \Rightarrow (\diamond c \wedge \neg \diamond d) \vee (\neg \diamond c \wedge \diamond d), \neg b \Rightarrow \neg((\diamond c \wedge \neg \diamond d) \vee (\neg \diamond c \wedge \diamond d)),$
 $c \vee d \Rightarrow \diamond e, \neg(c \vee d) \Rightarrow \neg \diamond e, \Box \neg(b \wedge e), \Box \neg(c \wedge d), \Box \neg(b \wedge (c \vee d)), \Box \neg((c \vee d) \wedge e)\}$

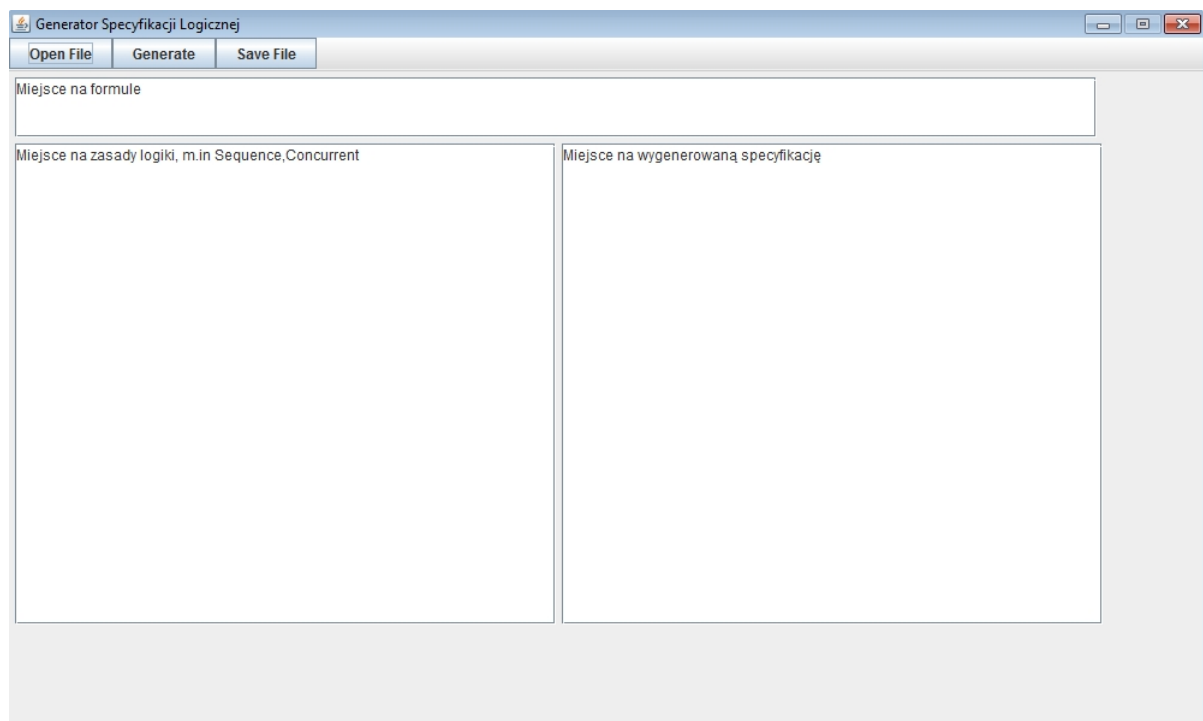
6 Struktura programu

C MainWindow	
 LOOKANDFEEL	String
 mainFrame	JFrame
 generatedOutput	JTextArea
 loadedLogicRules	JTextArea
 formulaField	JTextArea
 loadedFileLines	ArrayList<String>
 text	String
 fileChooser	JFileChooser
 menuBar	JMenuBar
 generateB	JButton
 loadFileB	JButton
 saveFileB	JButton
 rulesPanel	JScrollPane
 outputPanel	JScrollPane
 formulaPane	JScrollPane
 ruleAtt	Hashtable<String, String[]>
 ruleLogic	Hashtable<String, String[]>
 initLookAndFeel()	void
 loadListeners()	void
 parseFileIntoRules(File)	void
 generateSpecLog()	String
 parseWL(String)	ArrayList<RuleObject>
 isAtomic(String)	boolean
 checkFormulaField(String, ArrayList<RuleObject>)	int
 getCos(String[])	String
 getF_en(String)	String
 getF_ex(String)	Object
 getL2(String, ArrayList<String>)	String
 resetLoadedData()	void
 getArgs(String)	ArrayList<String>

Powered by yFiles



7 Wygląd GUI



Rysunek 1: Interfejs graficzny

WCZYTYWANIE ZASAD LOGIKI
 OPCJA ZAPISU-OBRAZEK
 GENEROWANIE

8 Działanie programu

8.1 Jak korzystać z programu

1. Przygotuj plik wzorców z wzorcami i ich logiką.
2. Wczytaj odpowiedni plik wzorców, wciskając przycisk "Open file" i korzystając z okna dialogowego. (wczytane zasady pojawią się w dużym polu tekstowym po lewej)

3. Przygotuj formułę do przeparsowania i wprowadź ją do "Miejsca na formułę" (pierwsze pole tekstowe od góry)
4. Żeby wygenerować specyfikację, wciśnij przycisk "Generate". (specyfikacja pojawi się w dużym polu tekstowym po prawej)

Uwaga! Pamiętaj o poprawności zapisu reguł w pliku i formuły w polu tekstowym. Nie przestrzeganie zasad struktury pliku uniemożliwi poprawne działanie programu. Zaś błędy w formule, które nie zostały przewidziane mogą zakłamać wynik programu.

Przykład poprawnego zapisu pliku i formuły:

TODO screeny i listingi

8.2 Zabezpieczenia

Pole formuły jest zabezpieczone przed następującymi błędami:

- Błędna ilość argumentów wzorca,
- Błędna nazwa wzorca,
- Występowanie znaków niedopuszczalnych
- Poprawne nawiasowanie-TODO
- Zbłąkane litery, np. Sequence(a,Seqyence(b,c)d) d to zbłąkana litera

9 Literatura

Radosław Klimek: From Extraction of Logical Specifications to Deduction-Based Formal Verification of Requitements Models. Strony 61-75.

Radosław Klimek: Elicitation of logical specifications from RUP-like processes for formal verification.