AKADEMIA GÓRNICZO-HUTNICZA IM. STANISŁAWA STASZICA W KRAKOWIE

**WYDZIAŁ ELEKTROTECHNIKI, AUTOMATYKI,
INFORMATYKI I INŻYNIERII BIOMEDYCZNEJ**

KATEDRA INFORMATYKI STOSOWANEJ

Praca dyplomowa magisterska

Implementacja algorytmu weryfikacji modelowej własności LTL w środowisku rozproszonym

Implementation of the distributed LTL model checking algorithm

Autor: *Marcin Szpyrka*
Kierunek studiów: *Informatyka*
Opiekun pracy: *prof. dr hab. Marcin Szpyrka*

Kraków, 2019

Uprzedzony o odpowiedzialności karnej na podstawie art. 115 ust. 1 i 2 ustawy z dnia 4 lutego 1994 r. o prawie autorskim i prawach pokrewnych (t.j. Dz.U. z 2006 r. Nr 90, poz. 631 z późn. zm.): „Kto przywłaszcza sobie autorstwo albo wprowadza w błąd co do autorstwa całości lub części cudzego utworu albo artystycznego wykonania, podlega grzywnie, karze ograniczenia wolności albo pozbawienia wolności do lat 3. Tej samej karze podlega, kto rozpowszechnia bez podania nazwiska lub pseudonimu twórcy cudzy utwór w wersji oryginalnej albo w postaci opracowania, artystycznego wykonania albo publicznie zniekształca taki utwór, artystyczne wykonanie, fonogram, wideogram lub nadanie.", a także uprzedzony o odpowiedzialności dyscyplinarnej na podstawie art. 211 ust. 1 ustawy z dnia 27 lipca 2005 r. Prawo o szkolnictwie wyższym (t.j. Dz. U. z 2012 r. poz. 572, z późn. zm.): „Za naruszenie przepisów obowiązujących w uczelni oraz za czyny uchybiające godności studenta student ponosi odpowiedzialność dyscyplinarną przed komisją dyscyplinarną albo przed sądem koleżeńskim samorządu studenckiego, zwanym dalej «sądem koleżeńskim».", oświadczam, że niniejszą pracę dyplomową wykonałem(-am) osobiście i samodzielnie i że nie korzystałem(-am) ze źródeł innych niż wymienione w pracy.

Serdecznie dziękuję . . . tu ciąg dalszych podziękowań np. dla promotora, żony, sąsiada itp.

Spis treści

1. Wstęp

Weryfikacja modelowa to dziedzina umożliwiająca sprawdzenie systemu pod kątem specyfikacji. Operacja taka jest zazwyczaj bardzo wymagająca pod kątem obliczeniowym, co skutkuje długimi czasami wykonania. Przeciwdziałać temu można na kilka sposobów, np. stosując uproszczenie modelu, czy wykorzystanie wydajniejszego procesora. Kolejna możliwość to stworzenie skalowanego systemu rozproszonego i właśnie ta metoda zostanie rozważona.

Samą specyfikację wyrazić można na wiele sposobów. W pracy wykorzystana zostanie logika LTL (ang. *linear-time temporal logic*).

1.1. Cel pracy

Celem pracy jest implementacja rozproszonego algorytmu weryfikacji modelowej w oparciu o własności logiczne czasu liniowego - LTL dla języka Alvis z wykorzystaniem frameworka Spring.

1.2. Struktura pracy

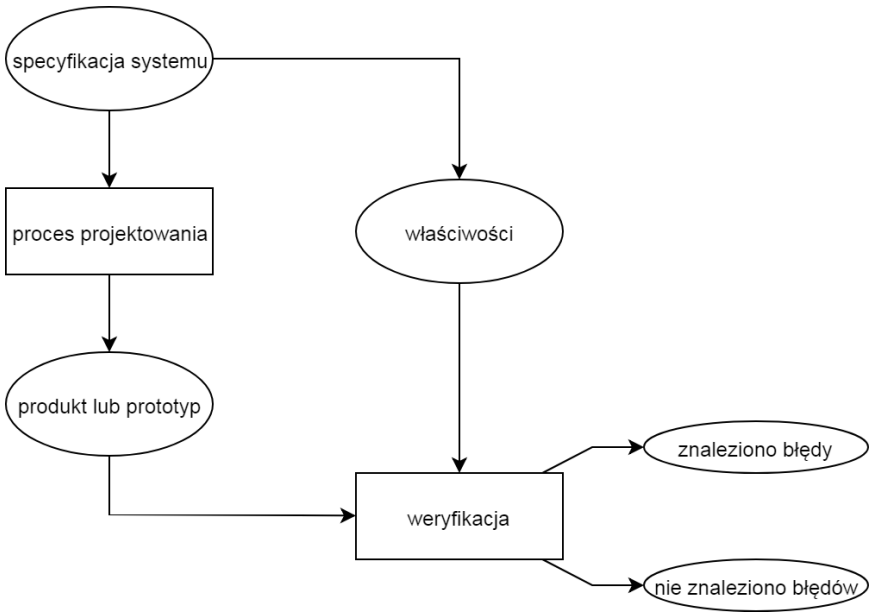
W następnym rozdziale pracy omówione zostanie zagadnienie weryfikacji modelowej w kontekście tematu pracy. Rozdział trzeci zawiera opis stworzonego rozwiązania. W czwartym rozdziale znajduje się prezentacja uzyskanych wyników. Piąty rozdział zawiera podsumowanie.

2. Weryfikacja modelowa

Systemy tworzone przez ludzi są coraz bardziej złożone oraz odgrywają coraz większą rolę w życiu każdego z nas. Błędy w oprogramowaniu skutkują stratami finansowymi, wizerunkowymi, opóźnieniami, a także utratą zdrowia i życia ludzi. Dowodzą temu następujące przykłady: nieudany start Ariane-5 (04.06.1996), błąd w procesorach Pentium II Intela, czy źle działająca maszyna do radioterapii spowodowała śmierć sześciu pacjentów w latach 1985-1987.

2.1. Weryfikacja systemu

Weryfikacja systemu ma na celu ustalenie, czy projekt posiada oczekiwane właściwości. Mogą one być dość podstawowe, np. nigdy nie dojdzie do zakleszczenia lub związane z domeną, np. nie można wypłacić więcej pieniędzy, niż jest na koncie. Specyfikacja dostarcza informacji, jak system może oraz jak nie może się zachowywać. Oprogramowanie uważa się za poprawne, jeśli spełnia wszystkie właściwości. Schemat weryfikacji został przedstawiony na rys. 2.1.



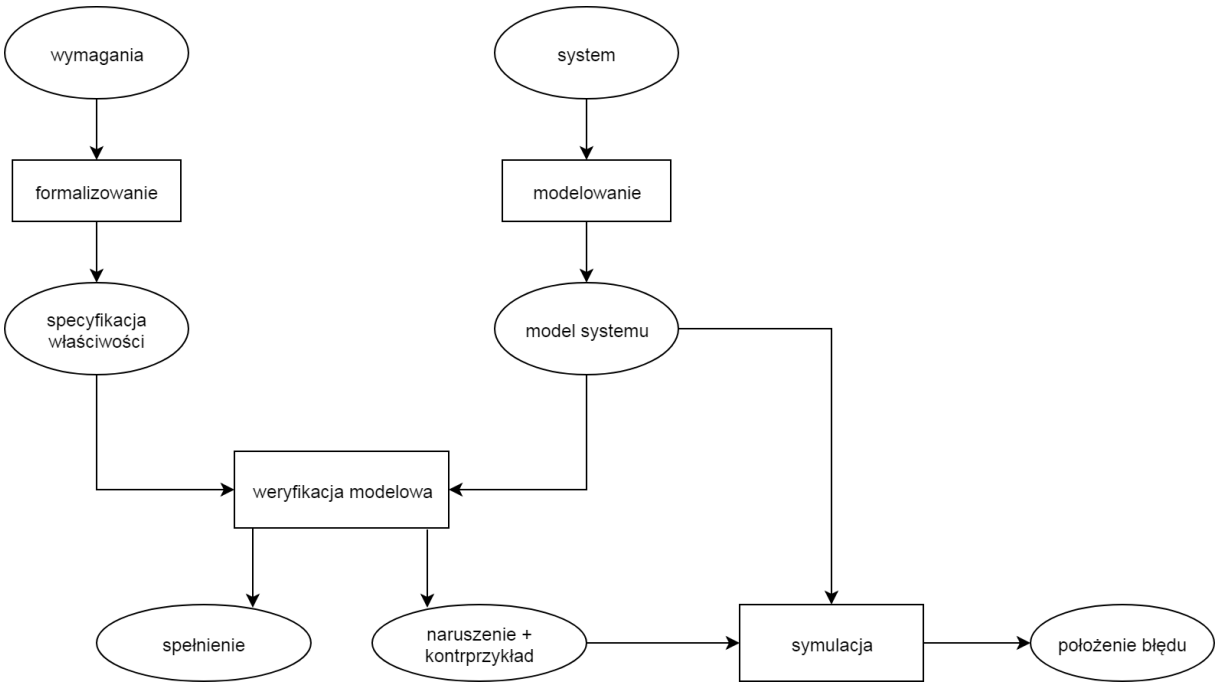
Rys. 2.1. Schemat tworzenia systemu wraz z jego weryfikacją (źródło [1]).

Podstawową formą radzenia sobie z tym problemem jest testowanie oprogramowania (testy jednostkowe, integracyjne, systemowe itp.). Polega to na uruchamianiu kodu dla różnych ścieżek wykonania, po czym porównuje się ich wyjścia z oczekiwanymi. Niestety, przetestowanie wszystkiego okazuje się praktycznie niemożliwe, zwykle sprawdzane są jedynie warunki brzegowe (stanowi to mały podzbiór wszystkich kombinacji).

2.2. Weryfikacja modelowa

Podczas tworzenia skomplikowanych systemów, kładzie się coraz większy nacisk na testowanie poprawności oprogramowania. Metody formalne mają duży potencjał na tym polu. Ich wczesna integracja (podczas procesu projektowania) dostarcza efektywnych technik weryfikacji. Intuicyjnie, metody formalne można rozważać jako matematykę stosowaną dla modelowania i analizy systemów informatycznych. Zapewniają one poprawność z matematyczną dokładnością.

Techniki weryfikacji bazujące na modelu opisują zachowanie systemu deterministycznie i kompletnie. Samo tworzenie pełnego modelu może wykryć luki lub niespójności. Po jego stworzeniu, wraz z otaczającymi algorytmami, możliwe jest eksplorowanie stanów systemu. Dzieje się to w podejściu "brute-force- przejrzane zostają wszystkie możliwe scenariusze. W ten sposób udowadnia się spełnialność właściwości. Możliwa jest również weryfikacja powiązana z czasem, np. czy system zawsze odpowie poniżej oczekiwanych 5 sekund.



Rys. 2.2. Schemat podejścia weryfikacji modelowej (źródło [1]).

Podejście to świetnie sprawdza się w wykrywaniu (częstych) błędów związanych z wielowątkowością. Typowe oczekiwane właściwości:

– osiągalność (niemożliwe jest zakleszczenie)

– bezpieczeństwo (coś niepożądanego nigdy nie wystąpi [2])

– żywotność (coś "dobrego"w końcu nastąpi [3])

– uczciwość (czy przy odpowiednich warunkach zdarzenie występuje powtarzalnie)

– właściwości czasu rzeczywistego

Model systemu zazwyczaj generuje się automatycznie z opisu w odpowiednim języku lub dialekcie wspieranym przez narzędzie. Weryfikator modelowy przeszukuje kolejne stany. Następnie sprawdzane są pod kątem właściwości. Po znalezieniu naruszenia, prezentowany zostaje kontrprzykład wraz z całą ścieżką wykonania, która do niego prowadzi. Pozwala to łatwo odtworzyć całą ścieżkę, a także znaleźć niespójność, która wymaga zmiany modelu (lub właściwości) - schemat na rys. 2.2.

To nie jedyne zalety, kolejnymi mocnymi stronami weryfikacji modelowej są:

– To ogólna metoda weryfikacji, która sprawdza się zarówno przy tworzeniu oprogramowania, jak i projektowaniu sprzętu (np. procesorów).

– Wspiera częściową weryfikację - możemy sprawdzać poszczególne właściwości niezależnie, nawet gdy pełna specyfikacja nie jest gotowa.

– Wszystkie możliwości zostają sprawdzone.

– Dostarcza informacji diagnostycznych po wykryciu niespełnionej właściwości.

– Uruchomienie weryfikatora nie wymaga ekspertyzy na tym polu.

– Łatwo zintegrować to rozwiązanie z cyklem wytwarzania oprogramowania.

– Obecnie wzrasta zainteresowania tym podejściem.

Oczywiście nie brak także wad:

– Weryfikowany jest model systemu, a nie sam system. Brak tu gwarancji, że implementacja poprawnie go odtwarza.

– Sprawdzane są tylko wykorzystane wymagania. Nie można wnioskować o poprawności właściwości, które nie zostały sprawdzone (błąd niedokładnej specyfikacji [4]).

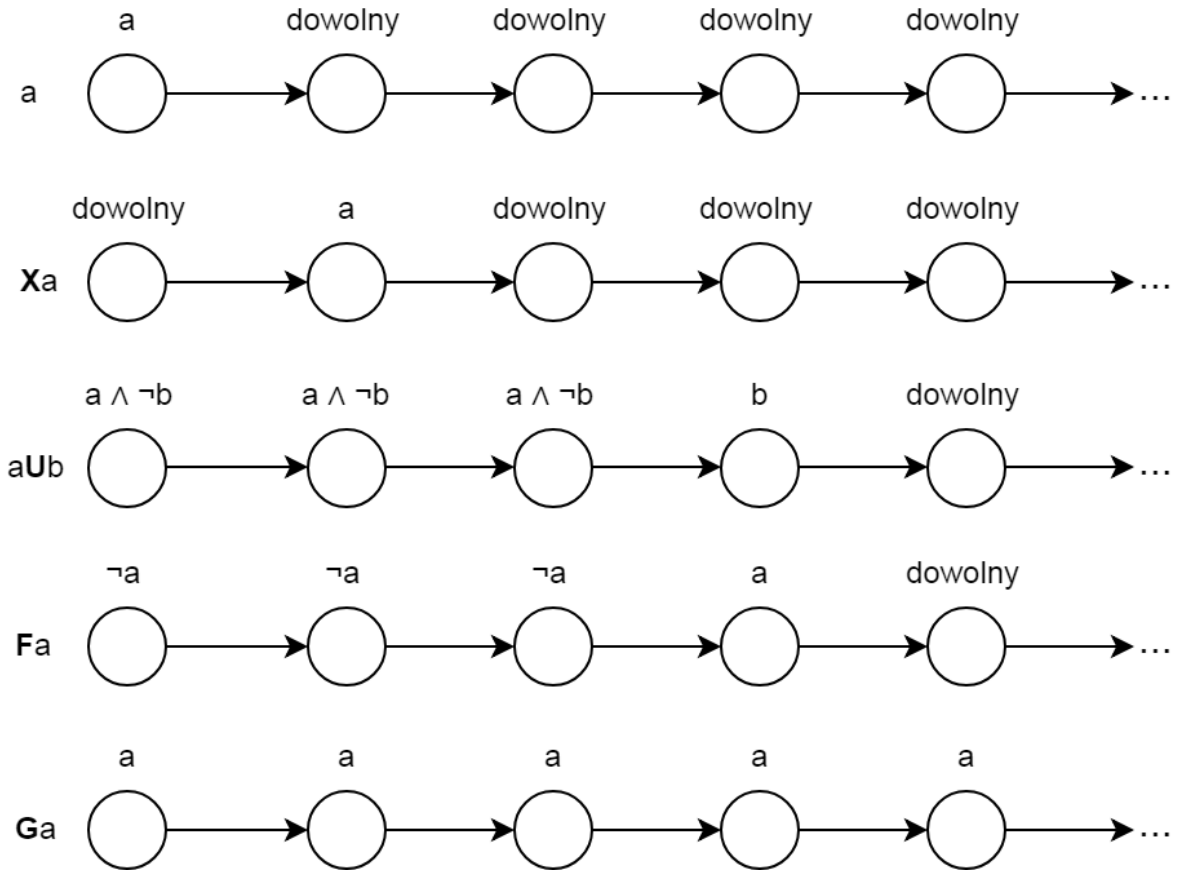
– Aplikacje oparte na dużej ilości danych mogą posiadać zbyt wiele stanów, aby je wszystkie wygenerować, a nawet ich liczba może być nieskończona.

– Podatność na problem eksplozji przestrzeni stanów [5].

2.3. Logika LTL

Logika LTL to jednak z logik temporalnych. Opiera się na liniowej strukturze czasu. Jej składnia i semantyka pozwala precyzyjnie opisywać właściwości systemu. "Temporalnaóznacza umiejscawianie zdarzeń relatywnie względem innych, to pewna abstrakcja ponad czasem. Z tego powodu niewykonalne jest sprawdzenie, czy maksymalne opóźnienie między zdarzeniami wynosi $500ms$.

Składnia LTL - spis skladni, Intuitive semantics of temporal modali (obrazek)



Rys. 2.3. Intuicyjna semantyka temporalnych modalności (źródło [1]).

2.4. Ignore

TODO

weryfikacja modelowa, przestrzen stanow, LTL (co to, opis skladni, operatorow, przyklad z opisem), automaty, automaty buchiego (przykladowe obrazki automatow z podanej formuly LTL, opis ze potrzebna negacja i szukanie spelnialnosci) weryfikacja hardware i software alogrytmy wykorzystywane w przeszukiwaniu i weryfikacji co to i po co on-the-fly Alvis, co to, po co

[6] [7] przejrzec publikacje profesora

w czesci implementacji: opis weryfikowanego systemu

A time-optimal on-the-fly parallel algorithm for model checking of weak LTL properties, in: Formal Methods and Software Engineering

3. Implementacja

4. Prezentacja wyników

5. Podsumowanie

Bibliografia

[1] Joost-Pieter Katoen Christel Baier. *Principles of Model Checking*. MIT Press, 2008.

[2] Fred B. Schneider Bowen Alpern. "Recognizing safety and liveness". In: *Distributed Computing* 2.3 (1987), pp. 117–126.

[3] Fred B. Schneider Bowen Alpern. "Defining liveness". In: *Information Processing Letters* 21.4 (1985), pp. 181–185.

[4] William K. Lam. *Hardware Design Verification: Simulation and Formal Method-Based Approaches*. Prentice Hall, 2005.

[5] Edmund M. ClarkeWilliam KlieberMiloš NováčekPaolo Zuliani. "Model Checking and the State Explosion Problem". In: *LASER Summer School* (2011), pp. 1–30.

[6] Petr Ročkai Jiří Barnat Luboš Brim. "On-the-fly parallel model checking algorithm that is optimal for verification of weak LTL properties". In: *Science of Computer Programming* 77.12 (Oct. 2012), pp. 1272–1288.

[7] Antti Valmari Jaco Geldenhuys. "More efficient on-the-fly LTL verification with Tarjan's algorithm". In: *Theoretical Computer Science* 345.1 (Nov. 2005), pp. 60–82.