Modelowanie i Analiza Systemów Informatycznych

Sprawozdanie z laboratorium

Data	Tytuł zajęć	Uczestnicy
04.06.2020 15:15	Logika Temporalna i Automaty Czasowe - konstrukcja i weryfikacja zsynchronizowanych automatów NuSMV.	Bartosz Rodziewicz (226105)

Zadanie 1

```
Zmień modelowanie poniższego automatu z pośredniego na bezpośrednie, nie zmieniając jego zachowania.
```

```
MODULE main
VAR x : 0..30;
VAR y : boolean;
ASSIGN
        init(x) := 30;
        next(x) := case
                y : x mod 2; TRUE : x / 2;
        esac;
        init(y) := TRUE;
        next(y) := y;
LTLSPEC y -> X(F(x=0))
INVARSPEC y
```

Rozwiązanie

W poleceniu zapisane zostało, aby wykonać jak najkrótszy kod automatu. Jedyne skrócenie jakie byłbym w stanie tu zrobić to usunięcie linijki `TRUE : x / 2;` w casie modelującym przejście `x`. Oryginalny automat jednak posiada tą linijkę (mimo, że ten default case nie jest potrzebny, ponieważ `y` jest zawsze `true`), więc zdecydowałem się ją zachować również w rozwiązaniu.

```
MODULE main
VAR
        x : 0..30;
        y : boolean;

INIT
        x = 30 &
        y = TRUE;

TRANS
        next(x) in case
                y : x mod 2;
                TRUE : x / 2;
        esac;
TRANS
        next(y) = y;

LTLSPEC y -> X(F(x=0))
INVARSPEC y
```

Zadanie 2

```
Wykonaj automat zamka szyfrowego, który przyjmuje dowolnej długości ciąg cyfr z zakresu od 0  do 9. Ciąg cyfr, kończący się sekwencją
159, otwiera zamek. Otwarcie zamka powoduje zatrzymanie działania automatu.
```

W poprzednich sprawozdaniach, gdzie rozwiązaniem były rysunki grafów wydawały mi się one na tyle nieczytelne, że wymagały komentarza. W tych zadaniach, mam wrażenie, że kod jest na tyle czytelny, że za dużo komentarzy nie wymaga. Tak jak w poprzednich sprawozdaniach, wszystkie nazwy zmiennych podane w treści zadania zmieniłem na angielskie, aby zachować czystość kodu. Zmienna `correctDigits` reprezentuje ile poprawnych cyfr zostało już wprowadzonych. `correctDigits` ma casa na inicjację, aby w przypadku, gdy pierwszą wylosowaną cyfrą jest 1, system poprawnie zadziałał i ustawił, że jedna poprawna cyfra została już wpisana.

```
MODULE main
VAR
	digit : 0..9;
	correctDigits : 0..3;

INIT
	correctDigits in case
		digit = 1 : 1;
		TRUE : 0;
	esac;

TRANS
	next(digit) in case
		correctDigits = 3 : digit;
		TRUE : 0..9;
	esac;

TRANS
	next(correctDigits) in case
		correctDigits = 3 : correctDigits;
		next(digit) = 1 : 1;
		next(digit) = 5 & correctDigits = 1 : 2;
		next(digit) = 9 & correctDigits = 2 : 3;
		TRUE : 0;
	esac;
```

Zadanie 3

```
Zweryfikuj poprawność działania modelu wykonanego w poprzednim zadaniu. Do każdej weryfikowanej formuły podaj: jej postać w języku
NuSMV, jej słowny opis i pełny wynik jej weryfikacji.
```

Negatywne formuły w tym zadaniu, oraz w zadaniu 5 w sprawozdaniu podane są w formie zwracającej wartość `false`. W kodzie załączonym do sprawozdania te formuły są napisane w formie `!(formuła)`, czyli zwracające `true`, aby ułatwić czytanie wyników wykonania analizy.

1. Kod: `COMPUTE MIN[correctDigits=0, correctDigits=3]`
   Opis: Ile ruchów potrzeba minimalnie, aby otworzyć zamek?
   Weryfikacja: 3 (czyli 4 stany)
   Wyjaśnienie: Sytuacja zakłada start z przynajmniej jedną cyfrą błędną. Jeśli od razu na start wylosowana została by 1, to odległość między stanami wynosiła by 2 (3 stany). Nie byłem jednak w stanie napisać, polecenia, które liczyło by ścieżkę minimalną od dowolnego stanu startowego.

2. Kod: `COMPUTE MAX[correctDigits=0, correctDigits=3]`
   Opis: Po ilu ruchach zamek zostanie na pewno otwarty?
   Weryfikacja: nieskończoność
   Wyjaśnienie: Może nastąpić ścieżka, która nigdy nie zakończy się poprawnym wprowadzeniem kodu.

3. Kod: `CTLSPEC EF(correctDigits=3)`
   Opis: Czy istnieje taka ścieżka, że zamek zostanie otwarty?
   Weryfikacja: **TAK**

4. Kod: `CTLSPEC AG(EF(correctDigits=3))`
   Opis: Czy z każdego miejsca automatu, zawsze istnieje taka ścieżka, że zamek zostanie otwarty?
   Weryfikacja: **TAK**

5. Kod: `CTLSPEC AG(digit >= 0 & digit <= 9)`
   Opis: Czy zawsze wprowadzona cyfra jest w przedziale [0, 9]?
   Weryfikacja: **TAK**

6. Kod: `CTLSPEC AG(correctDigits = 1 -> digit = 1)`
   Opis: Czy dla każdej ścieżki zawsze, jeśli ilość wprowadzonych dobrze cyfr wynosi 1, to ostatnio wprowadzona cyfra to 1?
   Weryfikacja: **TAK**

7. Kod: `CTLSPEC AG(correctDigits = 2 -> digit = 5)`
   Opis: Czy dla każdej ścieżki zawsze, jeśli ilość wprowadzonych dobrze cyfr wynosi 2, to ostatnio wprowadzona cyfra to 5?
   Weryfikacja: **TAK**

8. Kod: `CTLSPEC AG(correctDigits = 3 -> digit = 9)`
   Opis: Czy dla każdej ścieżki zawsze, jeśli ilość wprowadzonych dobrze cyfr wynosi 3, to ostatnio wprowadzona cyfra to 9?
   Weryfikacja: **TAK**

9. Kod: `CTLSPEC AG(correctDigits = 3 -> AX(AG(digit = 9 & correctDigits = 3)))`
   Opis: Czy dla każdej ścieżki zawsze, jeśli liczba poprawnie wprowadzonych cyfr wynosi 3, to już dla każdej ścieżki w następnym kroku, dla każdej ścieżki zawsze będzie prawdziwe, że wprowadzona cyfra to 9 i ilość poprawnie wprowadzonych cyfr to 3 (czyli, że automat się zablokuje)?
   Weryfikacja: **TAK**

10. Kod: `CTLSPEC AG(correctDigits = 3 -> EX(EF(correctDigits != 3)))`
    Opis: Czy dla każdej ścieżki, zawsze, gdy ilość poprawnie wprowadzonych liczb jest równa 3 to istnieje taka ścieżka, że od następnego kroku istnieje taka ścieżka, że kiedyś ilość poprawnie wprowadzonych liczb ulegnie zmianie?
    Weryfikacja: **NIE**

Zadanie 4

```
Wykonaj układ dwóch automatów (modułów), modelujących kalendarz, zsynchronizowanych ze sobą parametrami modułów.
Automat Miesiąc wyznacza bieżący miesiąc:
– bieżący miesiąc wskazywany jest przez enumeracyjną zmienną miesiąc o wartościach Styczeń, Luty, ·,
– bieżący dzień wskazywany jest przez całkowitoliczbową zmienną dzień, liczącą od 1,
– zmiana miesiąca jest powodowana przekroczeniem jego liczby dni.
Automat Rok wyznacza bieżący rok:
– typ bieżącego roku wskazywany jest przez enumeracyjną zmienną typ o wartościach Normalny, Przestępny,
– numer bieżącego roku wskazywany jest przez całkowitoliczbową zmienną rok o zakresie od 1900
do 2020,
– zmiana typu i numeru roku jest powodowana przez automat Miesiąc. Synchronizacja między automatami:
– automat Rok inkrementuje nr roku i ewentualne ustawia jego typ podczas zmiany miesiąca z grudnia na styczeń przez automat Miesiąc
(bezczasowa, natychmiastowa synchronizacja automatów).
```

Wydaje mi się, że większość kodu tutaj wydaje się być oczywista. Wyjaśnić chciałbym jednak zmienne `Year.yearType` oraz `Year.isLeapYear`. Zmienna `Year.yearType` to zmienna o której wspomina zadanie (enum - zwykły, przestępny) z jedną dodatkową wartością `none` - jest ona używana do blokowania automatów, gdy minie 2020 rok. W definicji przejść dla tej zmiennej widać, że gdy jest ostatni dzień 2020 roku to typ roku zmienia się na `none`, a gdy ta zmienna jest już na `none` ustawiona, to zostaje w tym stanie na zawsze. W definicji przejść każdej innej zmiennej jest ona używana do blokowania zmiany. Poza tym, definicja przejść dla tej zmiennej wykorzystuje definicję `Year.isLeapYear`, która jest zmienną `bool` wyliczaną z numeru aktualnego roku. Aby zapobiec rekursywnej definicji przejść zmienna `Year.year` nie używa `Year.yearType` i w warunku inkrementacji numeru roku sprawdza, czy nie ma jeszcze roku 2020.

```
MODULE main
VAR
        month : Month(year.yearType);
        year : Year(month.isLastDay);

MODULE Year(isLastDay)
VAR
        year : 1900..2020;
        yearType : { regular, leap, none};

DEFINE
        isLeapYear := (year mod 100 != 0 & year mod 4 = 0) | (year mod 400 = 0);

ASSIGN
        init(year) := 1900;
        next(year) := case
                isLastDay & year < 2020 : year + 1;
                TRUE : year;
        esac;

        init(yearType) := regular;
        next(yearType) := case
                yearType = none : none;
                isLastDay & year = 2020 : none;
                next(isLeapYear) : leap;
                TRUE : regular;
        esac;


MODULE Month(yearType)
VAR
        month : { jan, feb, mar, apr, may, jun, jul, aug, sep, oct, nov, dec };
        day : 1..31;

DEFINE
        isLastDay := month = dec & day = 31;


ASSIGN
        init(month) := jan;
        next(month) := case
                next(yearType) = none : month;
                month = jan & day = 31 : feb;
                month = feb & yearType = regular & day = 28 : mar;
                month = feb & yearType = leap & day = 29 : mar;
                month = mar & day = 31 : apr;
                month = apr & day = 30 : may;
                month = may & day = 31 : jun;
                month = jun & day = 30 : jul;
                month = jul & day = 31 : aug;
                month = aug & day = 31 : sep;
                month = sep & day = 30 : oct;
                month = oct & day = 31 : nov;
                month = nov & day = 30 : dec;
                isLastDay : jan;
                TRUE : month;
        esac;

        init(day) := 1;
```

```
        next(day) := case
                next(yearType) = none : day;
                month = next(month) & day < 31 : day + 1;
                TRUE : 1;
        esac;
```

Zadanie 5

```
Zweryfikuj poprawność działania modelu wykonanego w poprzednim zadaniu. Do każdej weryfikowanej formuły podaj: jej postać w języku
NuSMV, jej słowny opis i pełny wynik jej weryfikacji.
W szczególności zweryfikuj, czy każdy miesiąc trwa dokładnie tyle dni, ile powinien, i czy liczba dni w lutym zależy od typu roku.
```

Do testów tego automatu wykorzystałem logikę LTL, porównaniu do wszystkich poprzedni, z uwagi na to, że ten automat posiada liniowy przebieg wykonania stanów.

1. Kod: `LTLSPEC F(month.day < 1)`
   Opis: Czy jest możliwość, że kiedyś będzie miesiąc z dniem mniejszym niż 1?
   Weryfikacja: **NIE**

2. Kod: `LTLSPEC F(year.yearType = regular & month.month = feb & month.day > 28)`
   Opis: Czy jest możliwość, że kiedyś wydarzy się miesiąc luty w roku zwykłym mający więcej niż 28 dni?
   Weryfikacja: **NIE**

3. Kod: `LTLSPEC G(year.yearType = regular & month.month = feb -> month.day <= 28)`
   Opis: Czy zawsze spełniona jest implikacja, że jeśli rok jest zwykły i jest miesiąc luty to ilość dni w tym miesiącu będzie równa mniejsza 28?
   Weryfikacja: **TAK**
   Wyjaśnienie:

4. Kod: `LTLSPEC F(year.yearType = leap & month.month = feb & month.day > 29)`
   Opis: Czy jest możliwość, że kiedyś wydarzy się miesiąc luty w roku przestępnym mający więcej niż 28 dni?
   Weryfikacja: **NIE**

5. Kod: `LTLSPEC G(year.yearType = leap & month.month = feb -> month.day <= 29)`
   Opis: Czy zawsze spełniona jest implikacja, że jeśli rok jest przestępny i jest miesiąc luty to ilość dni w tym miesiącu będzie równa mniejsza 29?
   Weryfikacja: **TAK**

6. Kod: `LTLSPEC G(month.month in { jan, mar, may, jul, aug, oct, dec } -> month.day <= 31)`
   Opis: Czy zawsze spełniona jest implikacja, że jeśli jest miesiąc styczeń, marzec, maj, lipiec, sierpień, październik lub grudzień to ilość dni w tym miesiącu będzie równa mniejsza 31?
   Weryfikacja: **TAK**

7. Kod: `LTLSPEC G(month.month in { apr, jun, sep, nov } -> month.day <= 30)`
   Opis: Czy zawsze spełniona jest implikacja, że jeśli jest miesiąc kwiecień, czerwiec, wrzesień lub listopad to ilość dni w tym miesiącu będzie równa mniejsza 30
   Weryfikacja: **TAK**

8. Kod: `LTLSPEC G(year.year in { 1904, 1908, 1996, 2000, 2008 } -> year.yearType = leap)`
   Opis: Czy zawsze spełniona jest implikacja, że jeśli jest rok 1904, 1908, 1996, 2000 lub 2008 (kilka przykładowych lat przestępnych) to typ roku to rok przestępny?
   Weryfikacja: **TAK**

9. Kod: `LTLSPEC G(year.year in { 1900, 1902, 1999, 2001, 2019 } -> year.yearType = regular)`
   Opis: Czy zawsze spełniona jest implikacja, że jeśli jest rok 1900, 1902, 1999, 2001 lub 2019 (kilka przykładowych lat zwykłych) to typ roku to rok zwykły?
   Weryfikacja: **TAK**

10. Kod: `LTLSPEC G(year.year = 2020 -> year.yearType = leap | year.yearType = none)`
    Opis: Czy zawsze spełniona jest implikacja, że jeśli jest rok 2020 to typ roku to rok przestępny, bądź `none`?
    Weryfikacja: **TAK**
    Wyjaśnienie: Wynika, to z zastosowanej metody na blokowanie automatu po osiągnięciu końca roku 2020, opisanej w zadaniu 4.

11. Kod:
    `COMPUTE MAX[year.year = 2020 & year.yearType = leap & month.month = jan & month.day = 1, year.year = 2020 & year.yearType = leap & month.month = dec & month.day = 31]`
    Opis: Jaka jest odległość pomiędzy 1 stycznia 2020, a 31 grudnia 2020?
    Weryfikacja: 365 (czyli rok ma 366 dni)
    Wyjaśnienie: Test używany, do sprawdzenia, czy najpierw przeleci cały rok 2020 poprawnie, a dopiero potem przejdzie na typ roku `none`, który zablokuje automat.

12. Kod: `COMPUTE MAX[year.year = 1900 & month.month = jan & month.day = 1, year.year = 1900 & month.month = dec & month.day = 31]`
    Opis: Jaka jest odległość pomiędzy 1 stycznia 1900, a 31 grudnia 1900?
    Weryfikacja: 364 (czyli 1900 ma 365 dni)

13. Kod: `LTLSPEC G(year.year = 2020 & year.yearType = leap & month.month = dec & month.day = 31 -> X(G(year.yearType = none)))`
    Opis: Czy zawsze prawdziwa jest implikacja, że jeśli jest 31 grudnia 2020 i typ roku jest przestępny, to od następnego kroku na zawsze typ roku będzie `none` (automat zablokowany)?
    Weryfikacja: **TAK**

14. Kod: `LTLSPEC G(year.year >= 1900 & year.year <= 2020)`
    Opis: Czy zawsze rok jest w przedziale [1900, 2020]?
    Weryfikacja: **TAK**

15. Kod: `LTLSPEC G(year.yearType = regular -> X(F(year.yearType = leap)))`
    Opis: Czy zawsze prawdziwa jest implikacja, że jeśli jest rok zwykły, to od następnego kroku prawdziwe jest, że kiedyś nastąpi rok przestępny?

Weryfikacja: **TAK**
Wyjaśnienie: Weryfikacja w drugą stronę `leap -> F(regular)` jest nie możliwa z uwagi na rok 2020 i fakt, że jest to rok przestępny.

16. Kod: `LTLSPEC G(year.yearType = regular)`
    Opis: Czy zawsze jest zwykły rok?
    Weryfikacja: **NIE**

17. Kod: `LTLSPEC G(F(year.year = 2020))`
    Opis: Czy zawsze, kiedyś osiągnięty zostanie rok 2020?
    Weryfikacja: **TAK**

18. Kod: `LTLSPEC G(year.year = 2020 -> X(G(year.year = 2020)))`
    Opis: Czy zawsze prawdziwa jest implikacja, że jeśli mamy rok 2020, to od następnego kroku, zawsze będzie już rok 2020?
    Weryfikacja: **TAK**

19. Kod: `LTLSPEC G(month.month = jan -> X(F(month.month = dec)))`
    Opis: Czy zawsze prawdziwa jest implikacja, że jeśli mamy styczeń, to od następnego kroku kiedyś wydarzy się grudzień?
    Weryfikacja: **TAK**

20. Kod: `LTLSPEC G(year.yearType = leap -> year.isLeapYear)`
    Opis: Czy zawsze prawdziwa jest implikacja, że yearType jest ustawiony na przestępny, wtedy gdy definicja isLeapYear wylicza, że mamy rok przestępny?
    Weryfikacja: **TAK**

21. Kod: `LTLSPEC G(year.isLeapYear -> year.yearType = leap | year.yearType = none)`
    Opis: Czy zawsze prawdziwa jest implikacja, że jeśli definicja isLeapYear wylicza, że powinien być rok przestępny to `yearType` jest ustawony na przestępny, bądź `none`?
    Weryfikacja: **TAK**
    Wyjaśnienie: Z uwagi na specyficzną definicję roku 2020, konieczne jest uwzględnienie typu `none` w tej implikacji.