Tłumaczenie specyfikacji Estelle do języka wspomagającego formalną weryfikację

Michał Pakuła*
Instytut Informatyki, Uniwersytet Warszawski
ul. Banacha 2, 02-097 Warszawa
mpakula@mimuw.edu.pl

Streszczenie

Formalne specyfikacje oraz ich automatyczne weryfikowanie zyskują coraz większe znaczenie w procesie tworzenia oprogramowania. Estelle jest jedną z bardziej popularnych technik formalnego opisu systemów rozproszonych, szczególnie dobrze dostosowaną do specyfikowania protokołów komunikacyjnych, jednak brakuje dla niej narzędzi do automatycznej weryfikacji specyfikacji. By zapełnić tę lukę zdecydowaliśmy się opracować dla Estelle reguły tłumaczenia do języka, który umożliwia automatyczną weryfikację specyfikacji.

1. Wstęp

Celem niniejszej pracy było opracowanie i implementacja reguł tłumaczenia, które umożliwią wykorzystanie już istniejącego oprogramowania do weryfikowania specyfikacji zapisanych w Estelle. Żeby było to możliwe, musieliśmy przejść przez dwa etapy:

1. przegląd dostępnych weryfikatorów i wybór języka docelowego,

*Niniejsza praca została częściowo zrealizowana w ramach projektu badawczego KBN nr 8 T11C015 15

2. opracowanie reguł tłumaczenia z Estelle do języka docelowego i zapisanie tych reguł w formie automatycznego translatora.

Od początku wiadomo było, że przetłumaczenie pełnego języka Estelle nie jest możliwe. Wynika to z dość istotnej rozbieżności między poziomami abstrakcji na jakich operują Estelle i języki wejściowe dostępnych narzędzi. Niemniej jednak mieliśmy nadzieję, że uda się dobrać taki język docelowy i tak opracować reguły tłumaczenia, by ograniczenia w stosunku do standardu (a zarazem siły wyrazu) Estelle były jak najmniejsze. W trakcie bardziej szczegółowych badań okazało się, że lista ograniczeń jest całkiem spora, jednak nadal uważamy, że opracowane reguły i zaimplementowany translator są wystarczająco silne, by mogły znaleźć zastosowanie w praktyce.

Niniejsza praca jest zorganizowana w następujący sposób. Rozdział 2. zawiera najważniejsze informacje na temat Estelle. Rozdział 3. przedstawia metodę weryfikacji modelowej oraz narzędzia brane pod uwagę w trakcie poszukiwania najlepszego języka docelowego. W szczególności opiszemy zwięźle wybrany język — Promela oraz stowarzyszone z nim narzędzie — SPIN. Rozdział 4. prezentuje podzbiór Estelle, dla którego zostały opracowane reguły tłumaczenia oraz wyjaśnia, z czego wynikały poszczególne ograniczenia. Ponadto opisowo i schematycznie są w nim przedstawione reguły tłumaczenia. Rozdział 5., to zbiór sugestii odnośnie ewentualnych dalszych prac nad translatorem oraz porównanie początkowych zamierzeń i przekonań z faktycznymi efektami końcowymi.

2. Wprowadzenie do Estelle

Estelle [7] jest formalną techniką opisu drugiej generacji, służącą do specyfikowania rozproszonych, współbieżnych systemów przetwarzania informacji, a w szczególności do opisu protokołów komunikacyjnych. Powstała ona na podstawie doświadczeń zebranych przez jej poprzedników oraz przy współpracy z ITU-T, organizacją, która zdefiniowała również SDL.

Technika ta opiera się na rozszerzonym modelu automatów skończenie stanowych oraz na standardzie ISO Pascal, poziom 0. Za pomocą Estelle można modelować system jako hierarchiczną strukturę komunikujących się automatów, które działają współbieżnie i wymieniają informacje za pomocą kanałów komunikacyjnych oraz (w ograniczonym stopniu) poprzez zmienne dzielone. Automaty te w Estelle są nazwane modułami, ich interfejs określa się w nagłówku, a zachowanie (czyli relację przejścia) w treści modułu, za pomocą tranzycji.

Język Estelle jest stworzony do zapisu specyfikacji na wysokim poziomie abstrakcji. Pozwala między innymi na używanie nieokreślonych typów (oznaczanych przez `...`) i nieokreślonych wartości (konstrukcja `any` *NazwaTypu*).

Na pewno ułatwia to pisanie specyfikacji i pozwala na oderwanie się od szczegółów implementacyjnych, jednak z drugiej strony powoduje, że tego typu specyfikacje trudno weryfikować.

Dla Estelle istnieje pakiet narzędzi EDT [2], obejmujący m.in. symulator specyfikacji i generator kodu, jednak do tej pory nie powstało oprogramowanie do weryfikacji specyfikacji. Ponieważ stworzenie takiego weryfikatora jest zadaniem dość dużym i skomplikowanym a istnieje wiele sprawdzonych, dobrych narzędzi, postanowiliśmy wykorzystać jedno z nich. Ze względu na to, że Estelle bazuje na modelu automatów skończenie stanowych, naturalnym wyborem techniki weryfikacji była metoda weryfikacji modelowej, której implementacje na ogół opierają się na takim samym modelu teoretycznym.

3. Systemy weryfikacji

W trakcie poszukiwań języka docelowego dla tłumaczenia skupiliśmy się jedynie na tych językach, które są związane z metodą weryfikacji modelowej. Wybór ten był spowodowany dwiema przesłankami. Po pierwsze, metoda ta jest w pełni automatyczna, co znakomicie upraszcza proces weryfikacji. Po drugie, model rozszerzonych automatów skończenie stanowych — będący teoretycznym modelem specyfikacji Estelle — jest również bardzo często wykorzystywany w metodzie weryfikacji modelowej, co upraszcza zasady tłumaczenia.

W dalszej części rozdziału opiszemy języki, które badaliśmy na etapie poszukiwań, a na koniec przedstawimy język Promela i związane z nim narzędzie SPIN — wybrane jako docelowy system weryfikacyjny dla Estelle.

3.1. Przegląd istniejących narzędzi

Początkowo był rozpatrywany system weryfikacji specyfikacji SMV [9], [3]. Dosyć szybko okazało się jednak, że system ten, przystosowany przede wszystkim do weryfikacji rozwiązań sprzętowych, jest zdecydowanie zbyt ubogi w porównaniu z bogactwem konstrukcji Estelle.

Później nastąpiła próba wykorzystania systemu MONA [11]. MONA, to oparty na logice język programowania i zarazem narzędzie do tłumaczenia programów (a konkretniej — formuł) do automatów skończenie stanowych. Jest ona przystosowana przede wszystkim do zastosowań związanych z wyszukiwaniem wzorców i w gruncie rzeczy nie wydaje się mieć wiele wspólnego ze specyfikacją protokołów komunikacyjnych.

Kolejne narzędzie — HyTech [5] — służy do weryfikacji liniowych automatów hybrydowych. Automaty hybrydowe to model teoretyczny, łączący w sobie standardowe automaty skończenie stanowe, pozwalające na opisywanie zdarzeń dyskretnych, ze zmiennymi analogowymi, za pomocą których jest możliwe reprezentowanie zachowań o charakterze ciągłym, takich jak zmiany temperatury, czasu czy odległości. Automaty hybrydowe są używane przede wszystkim w specyfikacjach różnego rodzaju sterowników mających monitorować lub sterować działaniem urządzeń analogowych.

Już po wstępnym zapoznaniu się z możliwościami HyTecha okazało się, że nie będzie on dobrym językiem docelowym dla tłumaczenia specyfikacji Estelle. Podstawowym argumentem przeciwko był brak możliwości komunikacji pomiędzy automatami — automaty w HyTechu mogą co najwyżej synchronizować się między sobą, ale nie mogą przekazywać sobie żadnych danych. Na dodatek system typów tego języka jest zupełnie inny od systemu typów Estelle.

Zbadaliśmy także przydatność systemu Kronos [4], który jest oparty na teorii automatów z czasem i logice temporalnej z czasem. Niestety, okazał się on nieprzydatny dla naszych celów ponieważ, podobnie jak HyTech, umożliwia jedynie synchronizację między automatami ale nie pozwala na wymianę danych.

Ostatnim z odrzuconych narzędzi był Uppaal [1], również oparty na modelu automatów z czasem, rozszerzonym o możliwość korzystania ze zmiennych całkowitoliczbowych. Podobnie, jak w przypadku dwóch poprzednich systemów, także w Uppaalu automaty nie mogą przekazywać sobie żadnych danych, lecz jedynie synchronizować się za pomocą kanałów.

Po tych kilku nieudanych próbach ostatecznie zdecydowaliśmy się na język Promela i związane z nim narzędzie o nazwie SPIN [6]. Mimo, że Promela nie umożliwia operowania czasem, to w sumie jest najbliższa semantyce i składni Estelle, toteż opracowanie reguł tłumaczenia do Promeli okazało się z jednej strony najprostsze, a z drugiej pozwoliło na przetłumaczenie stosunkowo dużego podzbioru Estelle.

3.2. Promela i SPIN

Język Promela syntaktycznie przypomina popularny język C, jednak jest od niego znacznie uboższy, a niektóre instrukcje mają nieco inną składnię i odmienną semantykę. Występują w nim również elementy nie pojawiające się w C, takie jak na przykład kanały komunikacyjne. Szczegóły można znaleźć w pracy [6]. Tutaj jedynie wspomnimy, że specyfikacja w Promeli to zbiór procesów zapisany w jednym pliku. Każdy proces opisuje zachowanie pojedynczego automatu skończenie stanowego. Procesy wykonują się względem siebie asynchronicznie i mogą się komunikować poprzez tzw. kanały[1].

[1] Nazwa "kanał komunikacyjny" może być nieco myląca. Nie stanowią one narzędzia komunikacji typu "punkt — punkt". Bardziej przypominają "tablice ogłoszeń" — każdy proces może umieścić wiadomość w kanale i każdy może taką wiadomość wyjąć. Użycie nazwy "kanał" może wynikać z faktu, iż domyślnie wiadomości są w nim kolejkowane zgodnie z zasadą FIFO.

W języku Promela istnieją również specjalne konstrukcje, za pomocą których można określać własności, jakie powinna spełniać napisana specyfikacja. Istnieje wiele różnych sposóbów definiowania tych własności, jednak największą siłę wyrazu dają `never claim` i formuły logiki temporalnej LTL.

SPIN [10] jest narzędziem umożliwiającym symulację i automatyczną weryfikację specyfikacji napisanych w Promeli. Został dla niego opracowany również bardzo wygodny interfejs okienkowy, który między innymi potrafi w czytelny, graficzny sposób pokazywać kroki wykonania procesów i operacje wysłania lub odebrania komunikatów.

Najbadziej istotna z naszego punktu widzenia jest jednak możliwość automatycznej weryfikacji specyfikacji. Po zakończeniu weryfikacji SPIN wyświetla jej wynik (czyli informację, czy podane własności są spełnione przez specyfikację) oraz różne dane statystyczne (m.in. liczbę stanów modelu, osiągnięta głębokość przeszukiwania przestrzeni stanów, zużycie pamięci operacyjnej). Jeśli okaże się, że własność nie jest spełniona, można obejrzeć przykładowe wykonanie specyfikacji, które prowadzi do błędu.

4. Reguły tłumaczenia z Estelle do Promeli

Już na początku naszych badań byliśmy świadomi, że Estelle jest językiem zbyt bogatym, by dało się go w pełni przetłumaczyć do innego języka. Staraliśmy się jednak zminimalizować liczbę zastosowanych ograniczeń tak, by pozostała część była możliwie użyteczna w praktycznych zastosowaniach.

4.1. Określenie podzbioru Estelle

Zastosowane ograniczenia można podzielić na dwie grupy: ograniczenia związane z wysokim poziomem abstrakcji oraz ograniczenia wynikające z możliwości języka Promela. Do pierwszej grupy zaliczyliśmy przede wszystkim konstrukcje abstrakcyjne, które pozwalały na stosowanie nieukonkretnionych typów i stałych. Druga grupa ograniczeń zawiera: zakaz zagnieżdżania modułów, zakaz używania konstrukcji związanych z czasem, konieczność rezygnacji z atrybutów modułów oraz zakaz stosowania procedur i funkcji. Ograniczyliśy także zbiór typów danych do zbioru typów stosowanego w Promeli. Nie zostały również opracowane reguły tłumaczenia dla bogatego zestawu skrótów notacyjnych. Choć liczba zastosowanych ograniczeń nie jest mała ufamy, że rozpatrywany podzbiór Estelle ma wystarczająco dużą siłę wyrazu w praktycznych zastosowaniach.

4.2. Reguły tłumaczenia

Poniżej opisowo przedstawiamy opracowane zasady tłumaczenia. Większość z reguł będzie przedstawiona bardzo pobieżnie ze względu na ograniczoną ilość miejsca oraz stosunkowo proste zasady. Z detalami chcielibyśmy przedstawić jedynie reguły dla kanałów i tranzycji, ponieważ są one najbardziej skomplikowane, a zarazem dość istotne dla całości pracy.

Specyfikacja w Estelle to zestaw deklaracji modułów i kanałów komunikacyjnych oraz część inicjująca, określająca początkowy stan systemu. W Promeli odpowiada jej pojedynczy plik, zawierający deklaracje procesów i kanałów komunikacyjnych oraz część inicjującą. Moduł w Estelle składa się z sekcji deklaracji, sekcji inicjującej oraz zbioru tranzycji. Analogiczną strukturę nadajemy podczas tłumaczenia procesowi. Deklaracje stałych, zmiennych i typów są tłumaczone w dość naturalny sposób. Sekcja inicjalizacyjna jest tłumaczona praktycznie tak samo jak zbiór tranzycji, dla którego reguły podamy w dalszej cześci pracy. Instrukcje również są tłumaczone w dość bezpośredni sposób, chociaż niektóre z nich (zwłaszcza pętle i instrukcję wyboru) musieliśmy zapisać w nietypowy dla Promeli sposób tak, by zachować semantykę z Estelle. Przyjrzyjmy się teraz trochę bliżej najciekawszym elementom języka, dla których stworzyliśmy najbardziej skomplikowane reguły tłumaczenia.

4.2.1. Kanały

Semantyka kanałów w Estelle jest istotnie różna od semantyki kanałów w Promeli, co powodowało pewne komplikacje w opracowywaniu reguł tłumaczenia. Pierwsza podstawowa różnica polega na tym, że kanały Estelle są nieograniczone, podczas gdy w Promeli musimy z góry określić ich długość. Ponieważ na to nic nie można było poradzić, przyjęliśmy po prostu założenie, że każdy kanał będzie miał arbitralnie ustalaną długość w taki sposób, by łatwo ją było zmodyfikować w wynikowej specyfikacji.

Poważniejszy problem jest związany z następującymi kwestiami:

- deklaracja kanału w Estelle jest faktyczną deklaracją typu kanału, a instancje tego typu powstają w chwili wywołania instrukcji `connect`; dla odmiany — w Promeli nie ma deklaracji kanału jako takiej, jest jedynie jego definicja (utworzenie konkretnej instancji); w efekcie, jeśli w Promeli chce się korzystać z kilku odrębnych kanałów, ale mających taki sam typ, trzeba odpowiednio wiele razy skopiować definicję kanału, za każdym razem nadając mu inną nazwę;

- kanały w Estelle realizują komunikację typu "punkt — punkt"[2], natomiast w Promeli kanał jest tak naprawdę zwykłą kolejką prostą, do której

[2] Z dokładnością do dzielonych kolejek, o których mowa dalej.

każdy proces może wstawiać komunikaty i każdy może je z niej pobierać, odwołując się do kanału jedynie poprzez nazwę (powszechnie dostępną, bowiem jest to zmienna globalna); mechanizm ten nie tylko nie zapewnia, że komunikat trafi do adresata, ale nawet umożliwia procesowi odczytanie wiadomości, którą sam wstawił;

- w Promeli nie ma gotowego mechanizmu pozwalającego na realizację indywidualnych i dzielonych kolejek komunikatów (deklaracje `individual` i `common queue` w Estelle).

W związku z tym musieliśmy opracować specjalne zasady komunikacji, które przy wykorzystaniu standardowych kanałów Promeli pozwolą na realizację takiego mechanizmu komunikacji, jaki jest w Estelle.

Ogólnie idea polega na automatycznym numerowaniu instancji kanałów tworzonych w specyfikacji Estelle oraz wzbogacaniu każdego komunikatu w specyfikacji wynikowej o dodatkowe identyfikatory, które pozwolą na jednoznaczne określenie adresata. Dzięki opracowanym zasadom jest możliwa także realizacja semantyki deklaracji `individual` i `common queue`.

Tak jak było wspomniane wcześniej, każde wywołanie instrukcji `connect` powoduje powołanie nowej instancji kanału określonego typu. Możliwe jest stworzenie wielu instancji tego samego typu działających zupełnie niezależnie od siebie (wiadomości z różnych instancji nie mieszają się między sobą). Dlatego w specyfikacji wynikowej jest tworzona dodatkowa zmienna `chanidx`, która numeruje tworzone instancje kanałów.

Gdy translator napotka instrukcję `connect`, następuje sprawdzenie rodzaju punktów komunikacyjnych łączonych za jej pomocą. Jeśli oba punkty zostały zadeklarowane jako `individual queue`, każdemu z nich przekazywana jest aktualna wartość zmiennej `chanidx`, a sama zmienna zostaje zwiększona o 1. W przypadku, gdy tylko jeden z punktów został zadeklarowany jako `individual queue`, jemu jest przekazywana aktualna wartość zmiennej `chanidx` (po czym zmienna zwiększa się o 1), natomiast drugi punkt otrzymuje identyfikator wspólnego kanału dla danej instancji procesu[3]. W trzecim możliwym przypadku (oba punkty zadeklarowane jako `common queue`) każdy punkt otrzymuje identyfikator wspólnego kanału dla danej instancji procesu.

Dodatkowo należy wspomnieć, że każdy punkt komunikacyjny otrzymuje również swój identyfikator "końca" kanału (pierwszy punkt występujący w instrukcji `connect` dostaje 0 — lewy, drugi 1 — prawy). Jest to przydatne przede

[3] Każdej instancji procesu (utworzonej przez instrukcję `run`, która odpowiada `init` w oryginalnej specyfikacji) jest przypisywana jedna dodatkowa wartość — identyfikator wspólnego kanału. Jest ona używana dla wszystkich punktów tej instancji zadeklarowanych jako `common queue`, dzięki czemu można symulować semantykę tej deklaracji. Nie ma problemu z tym, że taka sama wartość będzie przypisana kanałom różnych typów, ponieważ i tak będą to fizycznie odrębne kanały, więc nie nastąpi pomieszanie komunikatów.

wszystkim w przypadku, gdy oba punkty są zadeklarowane jako `individual queue` i należą do tej samej instancji procesu (czyli używają takiego samego identyfikatora kanału).

4.2.2. Tranzycje

Opis sposobu tłumaczenia tranzycji rozpoczniemy od spojrzenia na listę tranzycji, a dopiero później zajmiemy się szczegółami związanymi z pojedynczą tranzycją.

Zbiór wszystkich tranzycji modułu określa sposób jego zachowania, a dokładniej relację przejścia między stanami automatu. Klauzula `from` służy do określenia stanu, w którym musi się znajdować moduł, aby dana tranzycja mogła być wykonana. Ponieważ w danej chwili moduł jest zawsze w jednym, określonym stanie, postanowiliśmy reprezentować te stany w Promeli za pomocą etykiet, a przejścia między nimi za pomocą instrukcji `goto`. W związku z tym, tłumaczenie całego zbioru tranzycji modułu odbywa się według następującego schematu:

1. dla każdego stanu jest tworzona pomocnicza lista wszystkich tranzycji, które w klauzuli `from` zawierają ten stan; należy tu pamiętać, iż Estelle umożliwia podanie więcej niż jednego stanu w klauzuli `from`, tak więc dana tranzycja może się znaleźć na kilku takich listach;

2. dla każdego stanu są wypisywane tranzycje znajdujące się na jego liście pomocniczej, zgodnie z poniższym wzorcem:

   ```
   NazwaStanu :
       if
           Tranzycja1
           Tranzycja2
           Tranzycja3
           ...
       fi;
   ```

   gdzie *TranzycjaI* jest zapisana jako opcja tak, jak to jest przedstawione poniżej.

Pojedyncza tranzycja składa się z następujących elementów:

1. stanu początkowego (klauzula `from`),

2. stanu końcowego (klauzula `to`),

3. dodatkowych warunków początkowych, które muszą być spełnione, żeby dana tranzycja mogła być wykonana (klauzule `when` i `provided`),

4. treści, składającej się z deklaracji lokalnych oraz instrukcji wykonywanych, gdy dana tranzycja zostanie uruchomiona.

Bardzo istotny jest fakt, że wykonanie pojedynczej tranzycji w Estelle jest zdarzeniem niepodzielnym, czyli nie można obserwować pośrednich wyników jej wykonania — dlatego każda opcja instrukcja wyboru `if ... fi` w poniższym schemacie jest objęta nawiasami `atomic`:

```
::  atomic { (Warunek) -> Treść }
EtykietaPomocnicza :  goto NastępnyStan ;
```

Znaczenie powyższych metasymboli jest zależne od postaci tłumaczonej tranzycji.

W przypadku najbardziej ogólnym, czyli gdy występuje zarówno klauzula `when`, jak i `provided`, kod wynikowy jest najbardziej skomplikowany. *Warunek* ma wówczas postać:

`NazwaIP??[eval(NazwaIP_rcv),KK,_,_]`

czyli sprawdza tylko, czy jest możliwość wczytania jakiejkolwiek interakcji przeznaczonej dla danego punktu komunikacyjnego. Drugi element komunikatu ma różną wartość w zależności od rodzaju punktu komunikacyjnego. Jeśli *NazwaIP* jest typu `common queue`, *KK* jest zmienną ulotną `_`, w przeciwnym przypadku pojawia się napis postaci `eval(NazwaIP_side)` — wartość zmiennej pomocniczej, na której jest trzymany identyfikator końca kanału dla tego punktu komunikacyjnego.

Metasymbol *Treść* jest konstruowany według ponizszego schematu:

```
Makrodef
NazwaIP??<eval(NazwaIP_rcv),KK,int_ident_aux,NazwaIP_params>;
Deklaracje;
if
::  (int_ident_aux == Interakcja) ->
    if
    ::  (WarunekProvided) ->
            NazwaIP??eval(NazwaIP_rcv),KK,Interakcja,NazwaIP_params;
            WłaściwaTreść ;
    ::  !(WarunekProvided) -> goto AktualnyStan ;
    fi;
UsunięcieMakrodef
    goto EtykietaPomocnicza ;
::  (int_ident_aux != Interakcja) -> goto AktualnyStan ;
fi
```

Pierwsza instrukcja odczytuje treść komunikatu nie usuwając go z kanału. Taka operacja jest konieczna do uzyskania identyfikatora interakcji. Następnie sprawdzamy, czy identyfikator ten (wczytany na zmienną `int_ident_aux`) jest taki sam, jak identyfikator określony w klauzuli `when`. Jeśli nie, jest wykonywany skok z powrotem do aktualnego stanu.

Jeśli jednak okaże się, że odczytany identyfikator interakcji jest równy identyfikatorowi określonemu w `when`, jest wykonywane sprawdzenie warunku `provided`[4]. Jeśli warunek ten okaże się prawdziwy, komunikat jest wyjmowany z kanału i wykonuje się właściwa treść tranzycji, w przeciwnym przypadku następuje skok z powrotem do aktualnego stanu.

W podanym wyżej schemacie występują również metasymbole *Makrodef* oraz *UsunięcieMakrodef*. Oznaczają one użycie makropoleceń `#define` i `#undef` w sposób umożliwiający posługiwanie się w *WarunkuProvided* oraz w treści tranzycji takimi samymi nazwami parametrów interakcji, jak w oryginalnej specyfikacji.

Zwróćmy uwagę na użycie *EtykietyPomocniczej*[5]. Etykieta ta jest umieszczana bezpośrednio za sekwencją `atomic` reprezentującą pojedynczą tranzycję. Po etykiecie występuje instrukcja skoku do następnego stanu, określonego w klauzuli `to`. Jeśli prześledzimy schemat tłumaczenia treści tranzycji to okaże się, że skok do *EtykietyPomocniczej* następuje z wnętrza poprzedzającej ją sekwencji `atomic` tylko wtedy, gdy dana tranzycja została wykonana.

Oczywiście można wykonać skok z wnętrza `atomic` od razu do właściwej etykiety reprezentującej stan docelowy, jednak zauważyliśmy, że takie zapisanie specyfikacji będzie mało wygodne w późniejszym użyciu. Zazwyczaj formuły poddawane weryfikacji badają wykonanie poszczególnych tranzycji, co w Promeli można wyrazić za pomocą konstrukcji *NazwaProcesu*`[`*ID*`]@`*NazwaEtykiety*. Dzięki umieszczeniu w kodzie wynikowym *EtykietPomocniczych* konstruowanie formuł jest łatwiejsze i praktycznie eliminuje konieczność ręcznej modyfikacji specyfikacji.

5. Podsumowanie

Celem niniejszej pracy było zautomatyzowanie weryfikacji specyfikacji napisanych w Estelle. Aby to osiągnąć dokonaliśmy przeglądu istniejących narzędzi posługujących się metodą weryfikacji modelowej i wybraliśmy naszym zdaniem najlepsze — system SPIN. Następnie opracowaliśmy reguły tłumaczenia

[4] Zauważmy, że dzięki umieszczeniu zmiennej *NazwaIP*`_params` w ostatnim polu odczytywanego komunikatu, na zmiennej tej znajdują się już wartości przesłanych parametrów, z których potencjalnie może korzystać *WarunekProvided*.

[5] *EtykietaPomocnicza* jest postaci `auxlabel`*i*, gdzie *i* to kolejna liczba całkowita — każda tranzycja ma własną, unikatową etykietę pomocniczą.

specyfikacji w Estelle do Promeli — języka wejściowego SPIN-a — i zaimplementowaliśmy je w automatycznym translatorze e2p.

Uważamy, że nasze podejście do weryfikacji specyfikacji zapisanych w Estelle jest lepsze od poprzedniego, zaprezentowanego w pracy [8]. Dzięki zautomatyzowaniu całego procesu weryfikacji, e2p wraz ze SPIN-em mogą być stosowanie w praktyce, do weryfikacji rzeczywistych systemów.

Opracowane reguły i narzędzie do automatycznego tłumaczenia wydają nam się całkiem dobrą podstawą do dalszych prac umożliwiających weryfikację specyfikacji Estelle za pomocą SPIN-a. Już w obecnej postaci mogą być wykorzystywane w praktyce, choć zdajemy sobie sprawę, że jest jeszcze kilka miejsc, w których można by je poprawić lub rozszerzyć.

Jednym z ciekawszych rozszerzeń byłaby możliwość tłumaczenia konstrukcji związanych z czasem. Niestety, aktualnie nie jest to możliwe do realizacji w SPIN-ie i wykracza poza ramy tej pracy. Liczymy również, że w kolejnej wersji tego narzędzia zostanie wprowadzona obsługa funkcji dzięki czemu możliwe będzie tłumaczenie funkcji z Estelle.

Warto również będzie dopracować implementację narzędzia e2p. Tak jak wspominaliśmy wcześniej, zarówno jego konstrukcja, jak i interfejs użytkownika są maksymalnie uproszczone. Do praktycznych zastosowań najlepiej byłoby stworzyć zintegrowane środowisko, łączące rozbudowany translator ze SPIN-em oraz ewentualnie z EDT. Dla wielu użytkowników zapewne cenne byłoby dopracowanie e2p w zakresie kontroli błędów składniowych w wejściowej specyfikacji. Aktualnie nasze narzędzie nie jest odporne na tego typu błędy, co w praktyce oznacza konieczność korzystania z komercyjnego pakietu EDP.

W czasie pracy ze SPIN-em zostały zauważone pewne błędy w jego działaniu. Zostały one zgłoszone autorom i w większości przypadków szybko poprawione. Mamy nadzieję, że obecnie weryfikacja odbywa się bez błędów.

Bibliografia

[1] J. Bengtsson, K. G. Larsen, F. Larsson, P. Petterson, W. Yi. *UPPAAL: a tool suite for automatic verification of real-time systems*, LNCS 1066, 232–243, 1996.

[2] S. Budkowski et al. *The Estelle Development Toolset.* Institut National des Télécommunications, Evry, France, `http://www-lor.int-evry.fr/edt/`

[3] *CMU — School of Computer Science, Formal Methods — Model Checking.* `http://www.cs.cmu.edu/~modelcheck/`

[4] C. Daws, A. Olivero, S. Tripakis, S. Yovine. *The tool KRONOS*, LNCS 1066, 208–219, 1996.

[5] T. Henzinger, P. Ho, H. Wong-Toi. *HyTech: the next generation*, LNCS 1019, 41–71, 1995.

[6] G. J. Holzmann. *Design and Validation of Communication Protocols*. Prentice Hall, 1991. Internet: `http://cm.bell-labs.com/cm/cs/what/spin/Doc/Book91.html`

[7] *ISO/IEC 9074:1997(E), Estelle: A Formal Description Technique Based on a Finite State Transition Model.* International Standards Organization, 1997.

[8] A. Janowska, P. Janowski. *Verification of Estelle Specifications Using TLA+*. Proc. of the 1st Int. Workshop on the Formal Description Technique Estelle, 109–130, 1998.

[9] K. L. McMillan. *The SMV language*. Cadence Berkeley Labs, Cadence Design Systems, 1999.

[10] *Spin — Formal Verification*. `http://netlib.bell-labs.com/netlib/spin/whatispin.html`

[11] *The MONA Project*. `http://www.brics.dk/~mona/`