А.В. Дубинин, К.В. Печаткин, В.Н. Дубинин

МОДЕЛИРОВАНИЕ И ТЕСТИРОВАНИЕ НЕДЕТЕРМИНИРОВАННЫХ АВТОМАТОВ С ИСПОЛЬЗОВАНИЕМ ЯЗЫКА MOSCOW ML В СИСТЕМЕ HOL

Россия, г. Пенза, Пензенский государственный университет

The ways of using the higher order logic-based system HOL for the study of non-deterministic automata by classical methods are proposed. A method for describing and modelling non-deterministic automata in the functional programming language Moscow ML is considered.

Введение

Конечноавтоматные модели были одними из первых дискретных формальных моделей, которые использовались в автоматике, информатике и вычислительной технике. До настоящего времени интерес к моделям данного класса не ослабевает. За прошедшие годы появилось много расширений автоматной модели, среди которых временные и гибридные автоматы, автоматы ввода/вывода, диаграммы состояний, системы переходов и др. Конечные автоматы используются в разработке, проектировании и реализации дискретных и событийных систем широкого класса, в числе которых аппаратно-программные комплексы, трансляторы и компиляторы, управляющие системы, системы промышленной автоматики и т.д. и т.п. Большой вклад в развитие теории автоматов внесли отечественные ученые Гаврилов С.А., Горбатов В.А., Шалыто А.А, Вашкевич Н.П. и другие.

Недетерминированные автоматы (НДА) в интерпретации Вашкевича Н.П. обладают рядом преимуществ в представлении и исследовании параллельных процессов по сравнению с другими автоматными моделями [1,2]. Это в первую очередь связано с возможностью нахождения автомата сразу в нескольких (локальных) состояниях (т.н. «частные события») и возможностью активации сразу нескольких (локальных) переходов под воздействием «частных входных сигналов». В определенной мере это напоминает функционирование управляющих синхронных сетей Петри. Здесь и далее под НДА будем понимать автоматную модель, предложенную Вашкевичем Н.П.

Несмотря на довольно существенный объем исследований в области НДА, методы моделирования, верификации и тестирования НДА разработаны недостаточно. В данной работе для этих целей предлагается использование языка *Moscow ML* и системы *HOL*. Следует отметить, что в работе *не рассматривается* верификация НДА методом доказательства теорем в *HOL*.

1. Технологическая цепочка исследования НДА в системе HOL

Основой предложенного метода моделирования и тестирования НДА является система HOL и входные языки этой системы. Термин *HOL* (сокращение от *Higher Order Logic*) обозначает логику высшего порядка. Главными особенностями высокоуровневой логики HOL являются: 1) возможность квантифицировать предикаты (а не только переменные); 2) в качестве аргумента предиката может выступать предикат (а не только переменная). Система *HOL* разработана для поддержки интерактивного доказательства теорем в логике высшего порядка [3,4]. Изначально она была разработана для сугубо математических целей, но впоследствии нашла себе много прикладных применений, в том числе для спецификации и верификации микропроцессора *ARM* [5].

Входным языком системы *HOL* является язык *Moscow ML* [6]. Язык *Moscow ML*

представляет собой полную реализацию *Standard ML* (*SML*), включая некоторые модули и расширения, поддерживает большую часть необходимых частей базовой библиотеки *SML*. Кроме того, по ряду пунктов язык *Moscow ML* расширяет язык *Standard ML*. Он также содержит некоторые императивные свойства, такие как ссылки на изменяемые значения, и поэтому не является чистым функциональным языком. При вычислениях использует «вызов-по-значению». Система типов основана на параметрическом полиморфизме. Язык *Moscow ML* является кроссплатформенным языком.

Процесс исследования НДА начинается с использования графического редактора *ACAD* [7]. В его основу заложены принципы представления НДА в виде сетей Петри. Результаты визуального представления НДА сохраняются в *XML*-формате, специально разработанном для этих целей. Трансляторы 1 и 2 преобразуют *XML*-описание НДА в описания на языке *Moscow ML* (*SML*-описания), которые могут быть использованы для моделирования, тестирования и детерминизации НДА. Для запуска *SML*-программы, код которой будет формироваться программой-транслятором, необходимо открыть *SML*-файл в среде функционального программирования *HOL*. После успешно выполненной компиляции программы на *Moscow ML* среда моделирования *HOL* позволит запустить данную программу на выполнение. Результаты выполнения *SML*-программы будут выводиться на монитор и записываться в выходные файлы. На рисунке 1 представлена технологическая цепочка исследования НДА с использованием системы *HOL* в классическом стиле, без использования метода доказательства теорем.

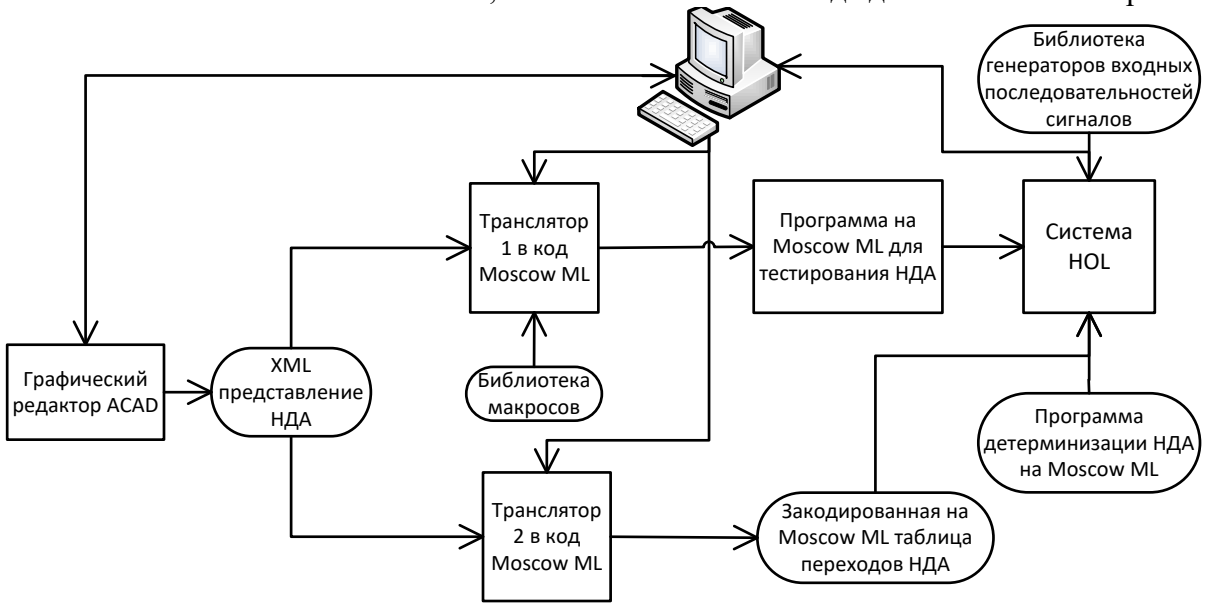

Рисунок 1 – Технологическая цепочка исследования НДА в HOL

2. Представление НДА в форме СКУ на языке Moscow ML

Как известно из [1,2], НДА можно представить аналитически с помощью системы канонических уравнений (СКУ), представленной ниже:

$$S_j^{Y_j}(t+1) = \bigvee_{i,j} X_{i,j}(t) \& S_i(t) \vee \bigvee_{j} X_{j,j}(t) \& S_j(t)$$

Для получения более полной информации по НДА и СКУ следует обратиться к [1,2]. В [8] были выделены две семантики выполнения НДА, в данной работе используется Семантика 1. СКУ можно представить функциями языка *Moscow ML*, среди которых могут быть рекурсивные, с использованием библиотеки *Moscow ML Basis Library* и механизма связанной рекурсии (*mutual recursion*).

Ниже приведен разработанный синтаксис представления СКУ на языке *Moscow ML* с помощью расширенных форм Бэкуса-Наура (РБНФ).

<СКУ>::= “fun” <описание частного события S_j >
[{“and” <описание частного события S_j >}] “;”
<описание частного события S_j>::= <значение частного события S_j на нулевом такте> ”|“ < описание частного события S_j на любом другом такте >
<значение частного события S_j на нулевом такте>::= “false” | <значение сигнала приведения автомата в исходное состояние на нулевом такте>
<значение сигнала приведения автомата в исходное состояние на нулевом такте >::= “(“ “nth” “(“ “(“ “nth” “(“ <список последовательностей входных элементарных сигналов> “,” ”0” “)” “)” “,” ”0” “)” “)”
<описание частного события S_j на любом другом такте>::= “(“ <описание события S_i, непосредственно предшествующего событию S_j> [{“orelse” <описание события S_i, непосредственно предшествующего событию S_j>}] “)”
<описание события S_i, непосредственно предшествующего событию S_j>::= “(“ <элементарный входной сигнал из алфавита [X], входящий в сочетание входных сигналов, в результате воздействия которых осуществляется переход НДА из события S_i к событию S_j> [{“andalso” “(“ <элементарный входной сигнал из алфавита [X], входящий в сочетание входных сигналов, в результате воздействия которых осуществляется переход НДА из события S_i к событию S_j>}] “andalso” <значение события S_i, непосредственно предшествующего событию S_j>[{“)” }])
<элементарный входной сигнал из алфавита [X], входящий в сочетание входных сигналов, в результате воздействия которых осуществляется переход НДА из события S_i к событию S_j >::= <инверсный сигнал> | <прямой сигнал>
<инверсный сигнал>::= “(“ “not” “(“ “nth” “(“ “(“ “nth” “(“ <список последовательностей входных элементарных сигналов> ”,” <номер такта работы автомата относительно события S_j > “)” “)” “,” <номер элементарного входного сигнала> ”)” “)” “)”
<прямой сигнал>::= “(“ “nth” “(“ “(“ “nth” “(“ <список последовательностей входных элементарных сигналов> “,” <номер такта работы автомата относительно события S_j > “)” “)” “,” <номер элементарного входного сигнала> “)” “)”
<значение события S_i, непосредственно предшествующего событию S_j>::= “(“ “SA” <значение i> “(“ <номер такта работы автомата относительно события S_i> “,“ <список последовательностей входных элементарных сигналов> “)” “)”

3. Описание программы-интерпретатора

Используемый при разработке программы-интерпретатора язык *Moscow ML* является функциональным языком программирования. Поэтому полноправными объектами, исходя из общих концепций функционального программирования, будут функции.

Перед описанием разработанных функций следует упомянуть о механизме разбиения на модули языка *Moscow ML*, так как возможность разложения большой программы на некоторое количество независимых модулей с четко определенным интерфейсом является крайне важной при разработке крупных программных продуктов. Модуль в языке *Moscow ML* представляет собой структуру среды, или сокращенно структуру. Как известно, среда представляет собой хранилище информации о смысле идентификаторов, объявленных в программе. Структура обладает некоторого рода типом (интерфейсом), называемым «сигнатурой», сигнатура определяет проекцию (view) структуры, то есть то, что должно быть видно в структуре.

В разработанной программе была создана структура:

```
structure ReturnMark : SIG =
struct
…
end;
```

Данная структура сопоставляется с сигнатурой SIG:

```
signature SIG =
    sig
    val sosmark : int -> bool list list
    val Spis1 : bool list -> string
    end;
```

Тем самым обеспечивается доступ только к необходимым функциям, что исключит вызов пользователем функций, которые не имеет смысла вызывать по отдельности вне структуры.

Далее перейдем к рассмотрению основных функций.

Функция: *val Spis1 : bool list -> unit*

```
fun Spis1 (nil) = ""
| Spis1 (x::xs) = if (xs = nil) then
                (Bool.toString(x)) ^ (Spis1 (xs))
                else
                (Bool.toString(x)) ^ " " ^ (Spis1 (xs));
```

Представленная функция является рекурсивной, использующей механизм образцов *(patterns)* языка *Moscow ML*, что выражается в применении оператора “|”. На вход поступает одномерный список типа *bool list*. Далее с использованием конструктора списков последовательно извлекаются элементы списка, начиная с первого, переводятся в строку, а потом эти строки конкатенируются. На выходе получаем требуемую строку, которая будет представлена значениями элементов списка.

Функция: *val generation : int -> bool list list*

```
fun generation (takt) =
    let
        …
    in
        …
    end;
```

Функция *generation* генерирует список тестовых последовательностей входных элементарных сигналов на тактах случайным образом, при этом значения сигнала x_0 (сигнала приведения автомата в исходное состояние) на тактах не будут входить в этот список, так как его поведение строго определено. На вход поступает номер такта (аргумент типа *int*), для которого и до которого нужно сгенерировать последовательности элементарных входных сигналов. На выходе мы получаем требуемый список типа *bool list list.*

Функция: *val shemsanduse : int -> bool list list.*

Данная функция создает список тестовых последовательностей входных элементарных сигналов на тактах (список типа *bool list list*), получая при этом информацию об одном элементарном входном сигнале на всех требуемых тактах

(список типа *bool list)* с помощью функций: *sh1, sh2, sh3,* которые будут рассмотрены ниже, как и функция *user*, которая тоже будет возвращать список значений на всех требуемых тактах какого-то одного элементарного входного сигнала, но при этом на каждом такте значение сигнала будет устанавливать сам пользователь, взаимодействуя с терминалом. Однако, в список тестовых последовательностей, который создается функцией *shemsanduse*, опять же не будут входить значения сигнала x_0. Также следует отметить, что функция *shemsanduse* реализует одну из составных частей интерактивного консольного интерфейса программы, используя при этом функции для работы с потоками. Данный интерфейс позволяет нам выбрать метод создания одномерного списка значений конкретного элементарного входного сигнала на тактах. Также обеспечивается интерактивность в задании значения входного сигнала на нулевом такте. На данное значение будет указывать ссылка *nachznach* **(**тип *bool ref***)**.

На полученные значения будут указывать соответствующие ссылки: *dltrue* и *dlfalse* **(**тип *int ref).* Далее, используя полученные ссылки, мы вырабатываем значения, на которые они указывают, и посылаем полученные величины на вход функции *sh3.* Следует отметить, что на вход функции *shemsanduse* также поступает номер такта (аргумент типа *int*), для которого и до которого нужно сгенерировать последовательности элементарных входных сигналов. Данное значение также поступает на входы *sh1, sh2, sh3 и user*.

Функция: *val sh1 : int -> bool list.*

Функция:*val sh2 : int -> bool list.*

На вход этих функций поступает номер такта *takt* (аргумент типа *int*), для которого и до которого нужно сгенерировать значения одного входного элементарного сигнала. Функция *sh1* создает список значений одного входного элементарного сигнала на требуемых тактах (список типа *bool list),* причем на каждом такте значение данного сигнала будет равно *true* для первой функции и *false* – для второй.

Функция: *val sh3 : int * int * int * bool -> bool list.*

На вход рассматриваемой функции поступает номер такта *takt* (аргумент типа *int*), для которого и до которого нужно сгенерировать значения одного входного элементарного сигнала, два аргумента: *numtr и numfl*, которые имеют тип *int,* обозначающие количества идущих друг за другом тактов, на которых входной элементарный сигнал будет принимать одно и то же значение (*true* или *false*), а также значение входного элементарного сигнала на нулевом такте, которое передается в аргументе *nach* типа *bool*. Функция *sh3* на основе полученных данных и глобальных переменных создает список значений (список типа *bool list*) одного элементарного входного сигнала на требуемых тактах.

Функция: *val user : int -> bool list.*

На вход функции *user* поступает номер такта *takt* (аргумент типа *int)*, для которого и до которого нужно сгенерировать значения одного входного элементарного сигнала. Функция *user* на основе значения аргумента, глобальных переменных и информации о значениях входного элементарного сигнала на тактах, получаемой в ходе общения с пользователем посредством пользовательского интерфейса, создает список значений (список типа *bool list*) одного элементарного входного сигнала на требуемых тактах.

Функция: *val readfromfile : int -> bool list list.*

На вход функции *readfromfile* поступает номер такта *takt* (аргумент типа *int*), для которого и до которого нужно сгенерировать последовательности элементарных входных сигналов. Функция *readfromfile* на основе значения аргумента, глобальных переменных выполняет создание списка последовательностей входных элементарных сигналов, читая информацию о значениях входных элементарных сигналов, входящих в последовательности, из текстового файла. Путь к файлу с расширением **.txt* и его имя

задает сам пользователь, работая с терминалом. Происходит проверка файла на пустоту, на то, какая информация представлена в файле (список последовательностей входных элементарных сигналов/список маркировок НДА), на количество элементарных входных сигналов в последовательностях, представленных в файле, на количество тактов, которым соответствуют последовательности элементарных входных сигналов в файле. Далее посредством функций работы с потоками (в данном случае источником входного потока будет являться текстовый файл) будет обрабатываться информация, представленная в файле, в результате чего сформируется список последовательностей входных элементарных сигналов типа *bool list list*, в котором будут отсутствовать значения сигнала x_0 на тактах, что и будет возвращаться данной функцией при положительном результате проверки файла.

Функции:

*val SA0 : int * bool list list -> bool*

*val SA1 : int * bool list list -> bool*

…

На этих функциях, которые могут быть рекурсивными, строится механизм связанной рекурсии (*mutual recursion*), что выражается в применении оператора *and* между определениями функций. Это обеспечивает вызов функции, стоящей после функции, из которой производится вызов. Пример части совокупности таких функций представлен ниже:

```
 fun SA0 (0,vhsign) = (nth((nth(vhsign,0)),0))
  | SA0 (takt:int,vhsign) = (((not (nth((nth(vhsign,takt)),1))) andalso (SA0(takt-1,vhsign))) orelse (nth((nth(vhsign,takt)),0)))
 and
 SA1 (0,vhsign) = false
  | SA1 (takt:int,vhsign) = (((nth((nth(vhsign,takt)),1)) andalso (SA0(takt-1,vhsign))) orelse ((not (nth((nth(vhsign,takt)),2))) andalso (SA1(takt-1,vhsign))))
 …;
```

Данная совокупность рекурсивных функций является представлением СКУ используемого НДА. На вход одной из рассматриваемых функции, например, описывающей частное событие S_0, поступает номер такта *takt* (аргумент типа *int)* работы автомата и полный список последовательностей входных элементарных сигналов на тактах (вместе со значениями сигнала x_0 на тактах) типа *bool list list.* Кроме того, в каждой рекурсивной функции используется механизм образцов для анализа картежей входных аргументов, выражающийся в применении оператора “|”, что позволяет задать значение частного события на нулевом такте и на любом другом такте. Таким образом, используя механизм рекурсивных функций, обеспечивается движение вниз по тактам в обратном направлении переходов в графе НДА (от частного(ых) события-потомка (событий-потомков) к частному(ым) событию-предку (событиям-предкам)) до получения значения(ий) частного(ых) конечного(ых) события-предка (событий-предков) на нулевом такте. Затем происходит обратный процесс: подстановка значений функций на места вызовов этих функций, которые вместе с соответствующими значениями входных элементарных сигналов, будут образовывать одно из уравнений СКУ для определенного события на требуемом такте.

Таким образом, с получением значения(й) частного(ых) события(ий) на предшествующем такте и зависимости от соответствия значений входных элементарных сигналов требуемым значениям для осуществления какого-либо перехода, будет осуществляться переход к частному(ым) событию-потомку (событиям-потомкам). Данный процесс будет продолжаться до тех пор, пока не будет получено значение требуемого частного события типа *bool* на интересующем такте, которое и

будет являться результатом работы функции, входящей в рассматриваемую совокупность функций.

Функция: *val sosmark : int -> bool list list.*

На вход представленной выше функции поступает номер такта *takt (*аргумент типа *int)*, для которого и до которого нужно сгенерировать последовательности элементарных входных сигналов. Данная функция, получая неполный список последовательностей входных элементарных сигналов от функций: *generation, shemsanduse и readfromfile*, добавляет к нему значения сигнала x_0 на рассматриваемых тактах и выводит полный полученный список в терминал. Еще можно отметить, что функция *sosmark* перехватывает исключения от функции *readfromfile* и генерирует исключение, если неполный список входных сигналов пуст, и количество событий в автомате не равно 1. Также данная функция обеспечивает выбор пользователем метода генерации неполного списка тестовых последовательностей входных элементарных сигналов посредством работы с терминалом. Далее, после создания полного списка тестовых последовательностей входных элементарных сигналов, функция *sosmark* позволяет печатать неполный (без значений сигнала x_0) список последовательностей входных сигналов в текстовый файл. Наконец, с использованием полученного полного списка тестовых последовательностей входных элементарных сигналов и списка функций *funspis*, членами которого являются рекурсивные функции, примеры которых представлены выше, создается список маркировок НДА типа *bool list list* на нужных нам тактах. Данный список и будет результатом работы функции *sosmark*.

Функция: *val main : unit -> unit*

Функция *main* является точкой входа в программу. Вызов функции должен выполнять сам пользователь для осуществления запуска всей программы. Данная функция позволяет задать номер такта, для которого и до которого необходимо сгенерировать список последовательностей входных элементарных сигналов и список маркировок автомата. Далее происходит вызов функции *sosmark* с передачей ей номера интересуемого такта. На возвращенный функцией список маркировок будет указывать ссылка *result* типа *ref bool list list***.** Также функция *main* обрабатывает исключение *Nospisvh***,** генерируемое функцией *sosmark*, когда список, возвращаемый функцией *readfromfile*, пуст и число состояний автомата не равно 1. Когда же список маркировок автомата успешно получен, он выводится в терминале.

4. Задание входных тестовых последовательностей

Получение входных тестовых последовательностей с использованием генератора случайных чисел.

Генерирование случайных тестовых входных последовательностей на требуемых тактах, в которые не будут входить значения сигнала x_0, происходит в функции *generation*. Рассмотрим данный процесс более подробно.

Весь процесс основан на использовании объекта *gen* типа *generator*, который создается с помощью вызова функции *newgen* библиотеки *Random*. Для получения значений входных элементарных сигналов, входящих в каждую входную тестовую последовательность, необходимо объявить внутри функции список типа *bool list,* в нашем демо-случае на 10 элементов, который будет виден только в данной функции. В нашем случае к этому списку будет привязан идентификатор *vhod*.

Далее в циклах происходит заполнение списка последовательностей входных элементарных сигналов значениями из списка *vhod* с использованием функции *range (0,9) gen* на каждой итерации вложенного цикла, которая берет в качестве аргумента кортеж (0,9), членами которого могут быть совершенно любые значения типа *int*, и вырабатывает функцию, берущую в качестве аргумента объект типа *generator* и

вырабатывающую целое число. Полученный неполный список последовательностей входных элементарных сигналов будет дополнен недостающими значениями в функции *sosmark*.

Получение входных тестовых последовательностей с использованием шаблонов последовательностей.

Данная методика создания списка последовательностей входных элементарных сигналов основана на том, что с помощью готовых схем, которые реализуются функциями *sh1, sh2 и sh3*, формируются списки значений входных элементарных сигналов на требуемых тактах работы автомата. При этом схему, которую необходимо будет использовать, выбирает сам пользователь, взаимодействуя с пользовательским интерфейсом, реализуемым в функции *shemsanduse*.

Потом данные списки вместе со списками, возвращаемые функцией *user*, заполняют двумерный список значений входных элементарных сигналов, на который будет указывать ссылка *spisoklv2***,** объявленная в функции *shemsanduse*. Далее данный список преобразуется уже в неполный список последовательностей входных элементарных сигналов (без значений сигнала x_0), который потом будет дополнен недостающими значениями в функции *sosmark*.

Как уже отмечалось, пользователь задает значение сигнала на нулевом такте, а также число тактов, идущих друг за другом, на которых сигнал будет принимать значения, *false* или *true*.

Получение входных тестовых последовательностей через терминал или текстовый файл.

Альтернативный способ получение входных тестовых последовательностей - с помощью задания значений элементарного входного сигнала пользователем через терминал (клавиатуру). Список значений входного элементарного сигнала на требуемых тактах формируется пользователем путем ввода значения каждого элемента (*true* или *false*) в консоль в ответ на сообщения программы.

Другой способ получение входных тестовых последовательностей - из текстового файла, построенного по определенному формату. Каждая строка файла задает набор элементарных входных сигналов на определенном такте. Например, следующая строка задает значения четырех элементарных входных сигналов на пятом такте.

Tact 5: false false true false

Для чтения из файла используется функция *readfromfile*. При обработке из получаемых строк извлекаются подстроки с помощью функции *substring* библиотеки *Basis Library* языка *Moscow ML,* которые будут иметь значения *true* или *false*. Далее происходит анализ этих подстрок, и, в зависимости от их значений, будут добавляться соответствующие значения в списки типа *bool list*, каждый из которых будет представлять одну из последовательностей входных элементарных сигналов.

5. Описание интерфейса программы

Программа – интерпретатор для тестирования НДА обладает интуитивно понятным консольным пользовательским интерфейсом, который позволяет работать с программой любому пользователю, знакомому с основами теории НДА.

Интерфейс данной программы строится на диалогах выбора и диалогах ввода. Диалог выбора позволяет пользователю выбирать режим, который будет использоваться для генерации входных последовательностей, позволять или не позволять программе записывать результат в файл и т.д. Пользователь может легко узнать этот тип диалога по наличию в сообщении от программы списка каких-то

действий, команд и т.д.

Диалог ввода позволяет пользователю вводить название и путь к файлу, число следующих друг за другом тактов, на которых входной элементарный сигнал будет принимать одно и то же значение и т.д. Пользователь может легко узнать этот тип диалога по наличию в программном сообщении требования ввести какую-то информацию без указания списка, из которого можно что-то выбрать. При работе с данными диалогами пользователь может использовать только клавиши клавиатуры.

Для успешной компиляции программы необходимо установить на компьютер пользователя систему *HOL-4 Kananaskis 5* совместно с компилятором языка *Moscow ML* [4]. Программа-интерпретатор НДА хранится в файле с расширением *.sml*. Для ее компиляции и возможности выполнения данную программу необходимо загрузить в систему с помощью операции *use*, аргументом которой будет выступать следующая строка "*[Путь к файлу\\]Название файла.sml*". После этого система выполнит компиляцию кода в *sml*–файле. Если компиляция прошла успешно, то функциональные объекты и структуры будут созданы, о чем будет свидетельствовать соответствующие сообщения.

Далее, чтобы запустить программу, необходимо вызвать функцию *main*, которая будет являться своеобразной точкой входа в программу. Для этого необходимо прописать в консоли строку *main()*. Если вызов прошел успешно, то в консоли будет выведено приглашение на ввод номера такта, на котором и до которого пользователь, хочет получить маркировки НДА.

Далее, следуя диалогам, представленным выше, пользователь сначала должен указать номер такта, после соответствующего сообщения, потом с помощью различных методов, представленных программой, получить список последовательностей входных сигналов, который будет выведен на терминале. На основе полученных данных при работе с программой и СКУ НДА, прописанной в коде программы, пользователь получает список маркировок рассматриваемого автомата, который тоже будет представлен пользователю данной программы в консоли. На этом алгоритм тестирования заканчивается.

6. Пример тестирования НДА с использованием программы-интерпретатора

В качестве тестируемого автомата был выбран НДА Мура, представленный следующим графом, взятым из работы [1]:

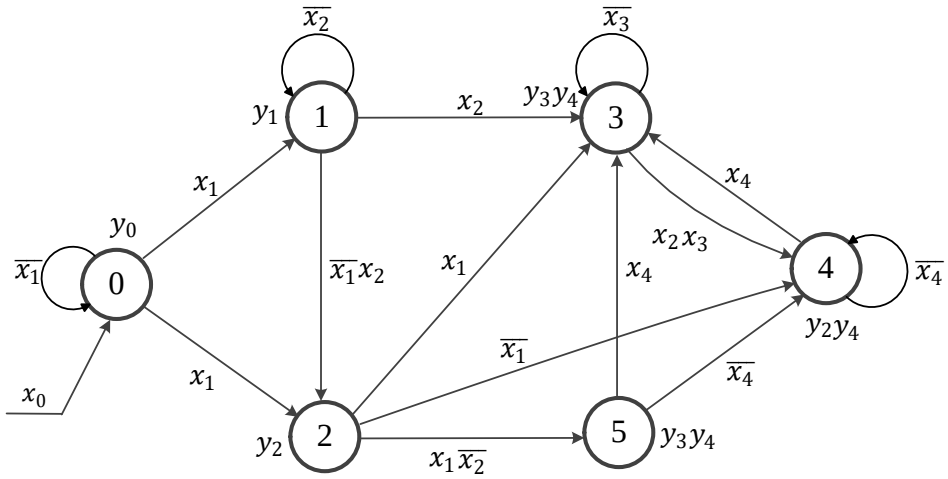

Рисунок 2 – Пример недетерминированного автомата

Данный автомат будет иметь СКУ следующего вида:

$$S_0^{y_0}(t+1) = S_0(t)\&\overline{x_1}(t) \vee x_n(t)$$

$$S_1^{y_1}(t+1) = S_0 x_1 \vee S_1 \overline{x_2}$$
$$S_2^{y_2}(t+1) = S_0 x_1 \vee S_1 \overline{x_1} x_2$$
$$S_3^{y_3 y_4}(t+1) = S_1 x_2 \vee S_2 x_1 \vee S_3 \overline{x_3} \vee (S_4 \vee S_5) x_4$$
$$S_4^{y_2 y_4}(t+1) = S_2 \overline{x_1} \vee S_3 x_2 x_3 \vee (S_4 \vee S_5) \overline{x_4}$$
$$S_5^{y_1 y_3}(t+1) = S_2 x_1 \overline{x_2}$$

Судя по СКУ рассматриваемого автомата, НДА Мура будет иметь 6 частных событий (локальных состояний) и 5 элементарных (частных) входных сигналов.

В данном примере будем рассматривать работу автомата на десятом и предшествующих тактах. Для начала сгенерируем случайным образом входную тестовую последовательность. Пример такой последовательности представлен в таблице1:

Таблица 1. Тестовая последовательность из наборов элементарных сигналов

N	**x_0**	**x_1**	**x_2**	**x_3**	**x_4**
0	true	true	true	false	true
1	false	true	true	false	true
2	false	false	true	true	true
3	false	false	false	true	false
4	false	false	true	true	false
5	false	false	true	true	true
6	false	true	true	false	true
7	false	true	false	false	true
8	false	false	true	true	false
9	false	false	true	true	false
10	false	false	true	true	true

Далее с помощью программы-интерпретатора НДА вычислим последовательность маркировок НДА на тактах 0..10 (см. таблицу 2) на основе входной тестовой последовательности сигналов, представленной в таблице 1.

Таблица 2. Последовательность достигнутых маркировок НДА

N	**S_0**	**S_1**	**S_2**	**S_3**	**S_4**	**S_5**
0	true	false	false	false	false	false
1	false	true	true	false	false	false
2	false	false	true	true	true	false
3	false	false	false	false	true	false
4	false	false	false	false	true	false
5	false	false	false	true	false	false
6	false	false	false	true	false	false
7	false	false	false	true	false	false
8	false	false	false	false	true	false
9	false	false	false	false	true	false
10	false	false	false	true	false	false

На рисунке 3 приведены временные диаграммы входных элементарных сигналов и локальных состояний, описывающие результаты тестирования НДА в более понятной визуальной форме. Следует отметить, что для удобства восприятия значения частных событий НДА (локальных состояний) сдвинуты на один такт вперед. Например, на

рисунке 3 в такте 2 представляются как сигнал $\overline{x_0(t)}\overline{x_1(t)}x_2(t)x_3(t)x_4(t)$, так и целевое состояние $\overline{S_0(t+1)}\overline{S_1(t+1)}S_2(t+1)S_3(t+1)S_4(t+1)\overline{S_5(t+1)}$.

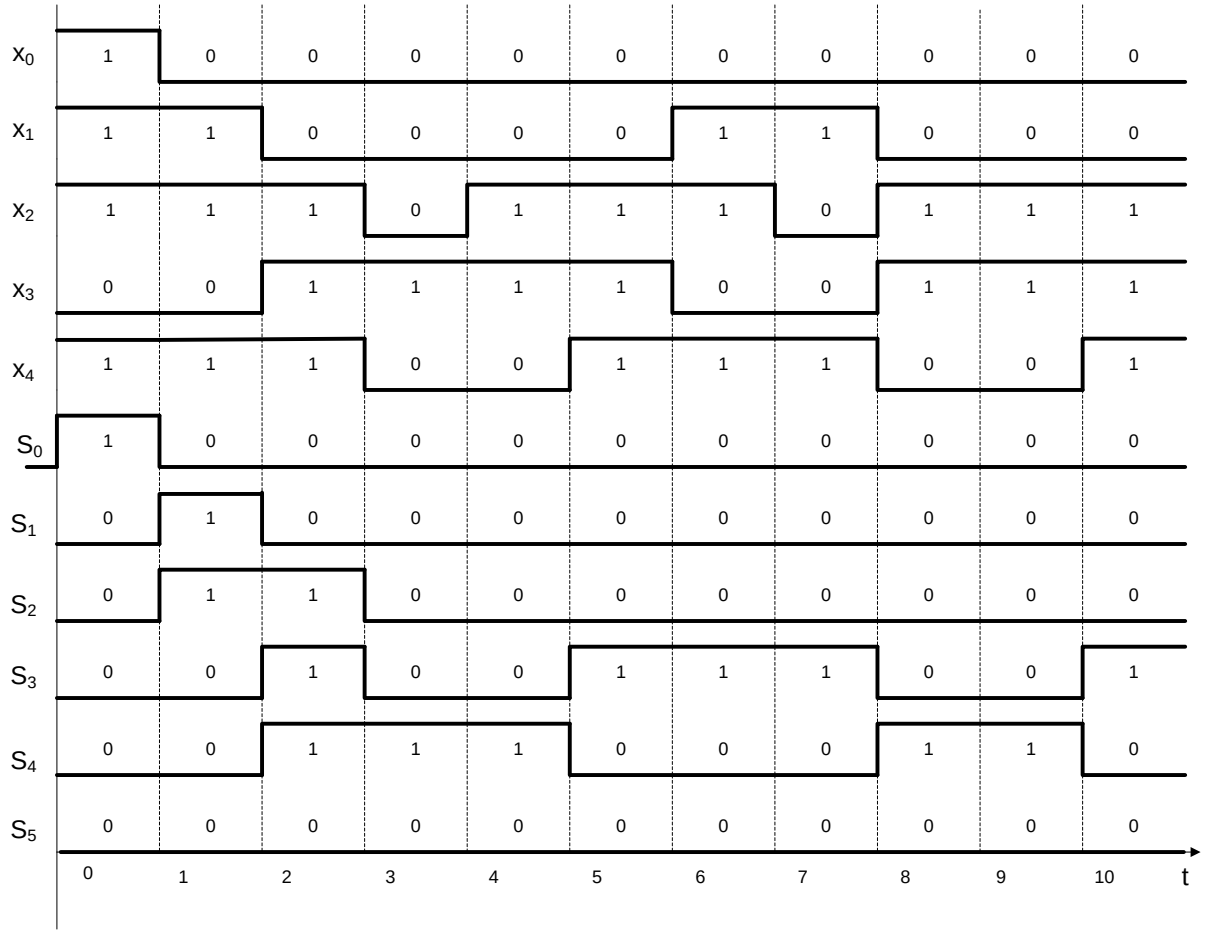

Рисунок 3 – Временные диаграммы функционирования НДА

Заключение

В данной работе были получены следующие результаты:

1) предложена технологическая цепочка исследования НДА с использованием системы высокоуровневой логики *HOL* и входящих в нее языков. Данная цепочка включает визуальный интерфейс для формирования НДА и перевода его в формат *XML* (ранее разработанная система *ACAD*), тестирование и детерминизацию НДА с использованием языка функционального программирования *Moscow ML*;

2) на языке *Moscow ML* разработана интерактивная программа-интерпретатор НДА, предназначенная для исследования НДА методом тестирования. В дополнение к данному интерпретатору на языке *Moscow ML* разработана библиотека генераторов тестовых последовательностей для входных сигналов. Программа–интерпретатор НДА выполняется в системе *HOL*, работающей под управлением ОС *Windows XP/Vista/7/10*. Новизна решения заключается в использовании языка функционального программирования *Moscow ML*, что позволяет легко трансформировать системы канонических уравнений (СКУ) НДА в языковую форму.

Разработанные методы и средства исследования НДА на основе системы *HOL* могут использоваться при проектировании цифровых схем, элементов и систем вычислительной техники и автоматики, и, в частности, при верификации как аппаратного обеспечения, так и параллельных вычислительных систем на

архитектурном уровне. Использование НДА в проектировании систем данного класса позволит формально описать предмет проектирования на основе автоматного подхода и проверить корректность и правильность его функционирования с использованием формального аппарата исследований.

Направлениями дальнейших исследований являются:

1) разработка метода формальной верификации НДА на основе доказательства теорем в *HOL*. До сих пор для формальной верификации НДА использовался только метод *Model cheching* и система *SMV* [9];

2) представление НДА с помощью арифметических полиномов и решение проблемы достижимости состояний в НДА на их основе.

3) разработка системы рекурсивных функций для детерминизации НДА на языке функционального программирования *Moscow ML*. Ранее для детерминизации НДА использовался табличный подход [1,2] и графотрансформационный подход [10].

1. Вашкевич Н.П. Недетерминированные автоматы в проектировании систем параллельной обработки: учеб. пособие. - Пенза: изд-во Пенз.гос.ун-та, 2004. – 280 с.
2. Вашкевич Н.П., Бикташев Р.А. Недетерминированные автоматы и их использование для реализации систем параллельной обработки информации : моногр. – Пенза : Изд-во ПГУ, 2016. – 394 с.
3. Introduction to HOL A theorem proving environment for higher order logic / Eds M.J.C. Gordon,T.F. Melham. - Cambridge University Press, 1993. - 496 p.
4. HOL Interactive Theorem Prover web site [Электронный ресурс]. - Режим доступа: https://hol-theorem-prover.org/
5. Fox A. Formal Specification and Verification of ARM6 // Theorem Proving in Higher Order Logics. TPHOLs 2003. Lecture Notes in Computer Science, vol 2758. Springer, Berlin, Heidelberg. – pp.25-40.
6. Moscow ML web site [Электронный ресурс] - Режим доступа: https://mosml.org/
7. Лепилин И.Б., Вашкевич Н.П., Дубинин В.Н. Графический редактор сетевого представления недетерминированных автоматов // Хроники объединенного фонда электронных ресурсов "Наука и образование". – 2010. - № 2. – С. 20. - Режим доступа: http://ofernio.ru/portal/newspaper/ofernio/2010/11.doc
8. Дубинин В.Н., Сенокосов И.В., Войнов А.С., Дроздов Д.Н., Вяткин В.В. Функционально-блочная реализация недетерминированных конечных автоматов // Труды Международной научно-технической конференции "Современные информационные технологии", Пенза, 2017, вып. 25. – С. 5-18.
9. Вашкевич Н.П., Дубинин В.Н. Формализованное описание и верификация дискретных событийных систем с параллельными процессами // Вопросы радиоэлектроники, сер.ЭВТ, 2008, Вып.5 – С.51-65.
10. Войнов А.С., Сенокосов И.В., Дубинин В.Н., Климкина Л.П. Детерминизация недетерминированных автоматов на основе графотрансформационного подхода // Труды Международной научно-технической конференции “Современные информационные технологии”, Пенза, 2016, вып.23. – С. 6-17.