Применение автоматного программирования и верификатора Spin для решения задачи управления пневмоподвеской автомобиля

Максим Неизвестных (neiz_max@mail.ru)

Одной из важных технологий программной инженерии является верификация программ методом model checking. Отличительная особенность данного метода – способность доказывать наличие или отсутствие заданного свойства у системы. В статье демонстрируется применение связки двух технологий: автоматного программирования и верификации программ методом model checking для решения задачи управления пневмоподвеской автомобиля.

Принцип работы пневмоподвески автомобиля

Пневматическая подвеска предназначена для регулировки дорожного просвета и ходовых характеристик автомобиля. Схема пневмосистемы автомобиля представлена на рисунке 1. У каждого колеса установлена пневмостойка. При накачивании воздуха в подушку пневмостойка удлиняется, при стравливании воздуха – укорачивается. Изменение размера пневмостойки приводит к изменению положения (высоты) кузова относительно колеса и дороги. Каждая пневмостойка имеет впускной (V_{in}) и выпускной (V_{out}) клапаны. Впускные клапаны предназначены для подачи воздуха к пневмостойкам и соединяют их с магистралью давления. Выпускные клапаны предназначены для стравливания воздуха из пневмостоек и соединяют их с магистралью сброса. Магистраль давления соединена с ресивером. Воздух в ресивер накачивается электрическим компрессором. Забираемый из атмосферы воздух подвергается предварительной фильтрации.

Система управления пневмоподвеской автомобиля

Система управления пневмоподвеской автомобиля является встраиваемой системой, к которой предъявляются серьёзные требования по безопасности. Блок управления пневмоподвеской (см. рис. 1) содержит модуль микроконтроллера (МК) и релейный модуль, который предназначен для коммутации силовой нагрузки. Модуль МК с помощью CAN-шины подключён к другим электронным системам для получения команд управления и величины скорости движения, а также для передачи прочей диагностической информации.

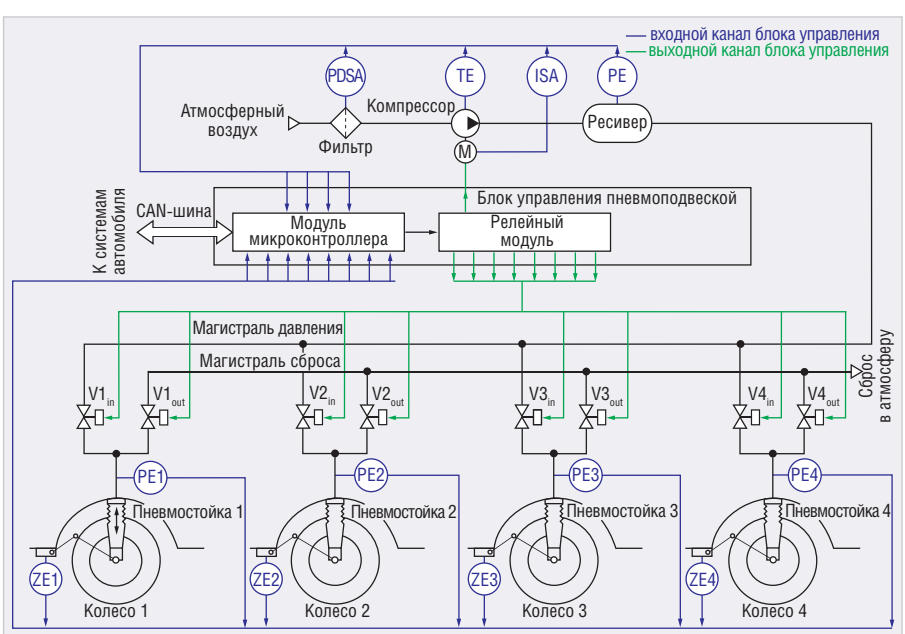

Рис. 1. Схема пневмосистемы автомобиля

Модуль МК опрашивает ряд датчиков с аналоговыми и дискретными выходами. На ресивере установлен датчик давления (PE) с аналоговым выходом 4...20 мА. Перед каждой пневмостойкой установлен датчик давления (PE1...PE4) для постоянного контроля давления непосредственно в подушке. Для контроля фактического положения кузова у каждого колеса установлен датчик положения (ZE1...ZE4), имеющий соединение с колесом посредством кривошипно-шатунного механизма. Контроль температуры компрессора осуществляется с помощью датчика температуры (TE). При высоком токе двигателя компрессора (M) срабатывает реле максимального тока (ISA), дискретный сигнал с которого поступает на модуль МК. При засорении фильтра воздуха срабатывает датчик засорения фильтра (PDSA), дискретный сигнал с которого также поступает на модуль МК. На основании всей полученной информации модуль МК через релейный модуль управляет компрессором и клапанами с электромагнитным приводом.

Пневмоподвеска может работать как в ручном, так и в автоматическом режимах. В автоматическом режиме давление ресивера поддерживается в заданных пределах, уровень кузова – согласно заданному профилю, независимо от массы и распределения груза по кузову автомобиля.

Концепция автоматного программирования

Автоматное программирование (АП) подразумевает разработку программы с применением автоматов, т.е. алгоритм проектируется как конечный автомат (КА). КА – это строго определённый математический объект [1]. В упрощённом представлении КА имеет входы, выходы, внутренние состояния, переходы между состояниями и действия, связанные с состояниями и переходами. Все компоненты, состав-

Компоненты программной системы логического управления

Компонент	Назначение
AtmHead	Управление работой всех компонентов системы
AtmMode	Переключение режимов работы пневмосистемы
AtmProfile	Переключение профиля
AtmPos1…4	Управление положением кузова у пневмостойки 1…4
AtmV1in… AtmV4in	Управление впускным клапаном 1…4
AtmV1out… AtmV4out	Управление выпускным клапаном 1…4
AtmMov1…4	Детектирование движения пневмостойки 1…4
AtmDiag1…4	Обнаружение неполадок пневмостойки 1…4
AtmCalibr	Калибровка датчиков положения кузова
AtmPres	Управление давлением ресивера
AtmDiagKompr	Обнаружение неполадок компрессора
AtmNet	Сетевое взаимодействие

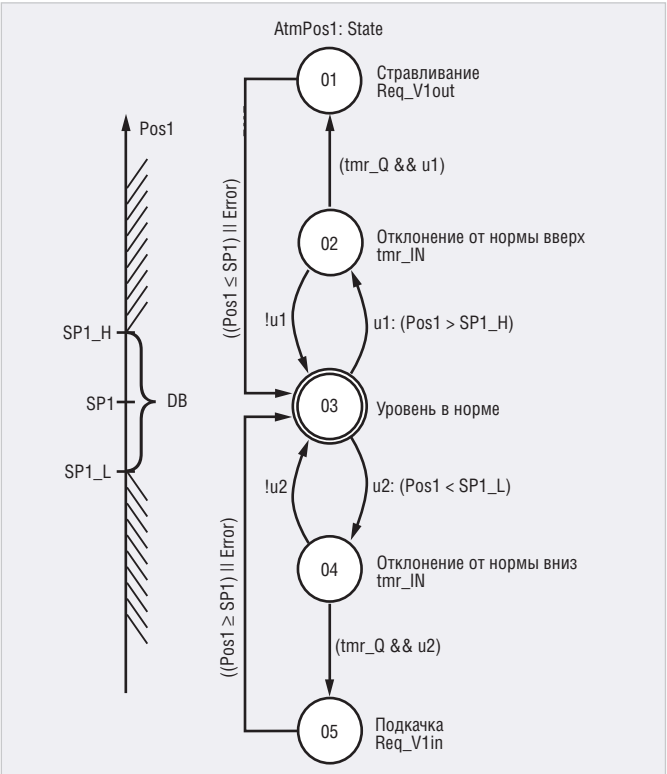

Рис. 2. Граф переходов автомата AtmPos1

ляющие КА, конечны. В общем случае любой КА работает следующим образом. На основании полученного входного сигнала и внутреннего состояния автомат переходит в новое состояние и совершает действия, связанные с этим переходом и новым состоянием. Действия оказывают влияние на выходные сигналы. Таким образом, с помощью КА можно описать сколь угодно сложную систему, где выходы являются реакцией на входы с учётом предыстории входных сигналов. Автомат, описывающий сложную реагирующую систему, также может оказаться громоздким и сложным объектом. В этом случае реагирующую систему описывают совокупностью взаимодействующих более простых автоматов. При этом автоматы можно рассматривать как параллельно работающие процессы [2], что является удобной и естественной для программных систем интерпретацией.

Автомат удобно представлять в виде графа переходов. Граф переходов автомата содержит всю необходимую для понимания его работы информацию. Первыми работами в области АП задач логического управления были публикации [3, 4]. Технология АП позволяет повысить безопасность разрабатываемого программного обеспечения для систем управления [5, 6].

Автоматная реализация компонентов

Задача логического управления пневмоподвеской разбита на несколько подзадач. Для выполнения каждой из них отводится свой компонент, имеющий автоматную реализацию (см. табл.).

Рассмотрим автоматную реализацию компонента AtmPos1. Граф переходов автомата изображён на рисунке 2. На приведённом рисунке использованы следующие условные обозначения:

- Pos1 – фактическое положение кузова у пневмостойки 1;
- SP1 – заданное положение кузова у пневмостойки 1;
- SP1_H – верхнее допустимое отклонение кузова у пневмостойки 1;
- SP1_L – нижнее допустимое отклонение кузова у пневмостойки 1;
- DB – зона нечувствительности;
- Sate – состояние автомата;
- Error – ошибка работы пневмостойки или клапана;
- Req_V1out – запрос открытия клапана $V1_{out}$;
- Req_V1in – запрос открытия клапана $V1_{in}$;
- tmr – таймер задержки реакции на отклонение кузова;
- tmr_IN – сигнал запуска таймера (вход таймера);
- tmr_Q – сигнал срабатывания таймера (выход таймера).

Автомат AtmPos1 имеет пять состояний: 01 – стравливание, 02 – отклонение от нормы вверх, 03 – уровень в норме, 04 – отклонение от нормы вниз, 05 – подкачка. Номер состояния автомата AtmPos1 хранится в переменной State. Состояние 03 является стартовым (обозначено двумя окружностями). Если фактическое и заданное положения кузова отличаются незначительно, т.е. значение Pos1 находится в зоне нечувствительности DB, то автомат продолжает оставаться в состоянии 03. Если выполняется условие u1 (Pos1>SP1_H), т.е. произошло отклонение вверх, то автомат переходит в состояние 02, при этом запускается таймер задержки реакции на отклонение кузова (tmr_IN). Если до момента срабатывания таймера tmr условие u1 перестало выполняться, т.е. кузов вернулся в нормальное положение, то автомат возвращается в состояние 03 и таймер tmr прекращает работу. Если же время истекло (сработал выход таймера tmr_Q) и условие u1 продолжает сохраняться, то автомат переходит в состояние 01 и устанавливает запрос на открытие выпускного клапана (Req_V1out). Автомат AtmV1out отрабатывает данный запрос и открывает клапан. После открытия выпускного клапана кузов начинает снижаться. Когда положение кузова достигает заданного значения (Pos1≤SP1) или происхо-

Листинг 1

```
//Работа автомата AtmPos1
void AtmPos1()
{
       //условия переходов автомата:
       tr_03_02 = (Pos1 > SP1_H);
       tr_02_03 = !tr_03_02;
       tr_02_01 = (tmr_Q && tr_03_02);
       tr_01_03 = ((Pos1 <= SP1) || (Error));
       tr_03_04 = (Pos1 < SP1_L);
       tr_04_03 = !tr_03_04;
       tr_04_05 = (tmr_Q && tr_03_04);
       tr_05_03 = ((Pos1 >= SP1) || (Error));
       //переходы автомата:
       switch (State)
       {
          case 1:
          if (tr_01_03) State = 3;
          break;
          case 2:
          if (tr_02_03) State = 3;
          else
          if (tr_02_01) State = 1;
          break;
          case 3:
          if (tr_03_02) State = 2;
          else
          if (tr_03_04) State = 4;
          break;
          case 4:
          if (tr_04_03) State = 3;
          else
          if (tr_04_05) State = 5;
          break;
          case 5:
          if (tr_05_03) State = 3;
          break;
       }
       //выходы автомата:
       if (State == 1) Req_V1out = true;
       else            Req_V1out = false;
       if ((State == 2) || (State == 4)) tmr_IN = true;
       else                              tmr_IN = false;
       if (State == 5) Req_V1in = true;
       else            Req_V1in = false;
}
```

дит ошибка (Error), автомат переходит в состояние 03 и снимает запрос Req_V1out. Автомат AtmV1out отрабатывает запрос и закрывает клапан, после чего фактическое положение кузова фиксируется на заданном уровне. Аналогично отрабатывается и отклонение кузова вниз. Таким образом, работа автомата AtmPos1 стабилизирует положение кузова у пневмостойки 1. Условия переходов автомата должны быть выстроены так, чтобы из одного состояния не выходило более одной дуги с истинным условием (свойство детерминированности автомата), иначе будет неясно, в какое из состояний следует переходить.

Программная реализация автоматов

Программа для микроконтроллера AVR реализована на языке программирования C. Рассмотрим фрагмент кода программы, отражающий работу автомата AtmPos1 (см. листинг 1).

Состояние автомата определяется управляющей переменной `State` [6]. Для переключения состояний автомата используется инструкция выбора `switch` [3, 4]. Запрос открытия выпускного клапана (Req_V1out) устанавливается только в состоянии 01, запрос открытия впускного клапана (Req_V1in) – только в состоянии 05. Таймер работает, если автомат находится либо во 2-м, либо в 4-м состоянии.

Верификация программ методом model checking

Верификация представляет собой проверку соответствия программы установленным требованиям [7]. Тестирование является одним из методов верификации, однако оно позволяет лишь найти ошибки, но не позволяет доказать их отсутствие. Метод model checking (англ. – проверка модели) является формальным методом верификации, который построен с использованием строгого математического аппарата. Данный метод позволяет доказывать отсутствие ошибок, а именно нежелательных свойств, что особенно важно для систем с повышенными требованиями к безопасности. Метод требует построения модели системы и формализации требований к ней. Под системой понимаются программа и объект управления. Совместимость автоматного программирования и верификации методом проверки модели заключается в том, что автоматная программа очень просто преобразуется в свою модель [8]. Модель объекта управления также может быть представлена в виде конечных автоматов [2]. Модель описывает множество всех возможных поведений системы. Требования к модели должны быть выражены на формальном языке темпоральной логики (логики, учитывающей временной аспект). Высказывание данного языка описывает некоторое множество допустимых или недопустимых поведений модели. Верификатор автоматически проверяет истинность формального высказывания для данной модели, или, другими словами, выполнимость для неё темпорального свойства.

Верификатор Spin

Spin – инструмент, предназначенный для выполнения верификации методом model checking, разработанный в Bell Labs. Данный верификатор успешно применяется в космической, авиационной, телекоммуникационной, медицинской и других отраслях [9]. Spin поддерживает два языка: язык моделирования Promela [9, 10], предназначенный для описания модели системы и LTL (Linear Temporal Logic) – язык линейной темпоральной логики [7–11], предназначенный для описания спецификаций требований к модели. Spin имеет интерфейс командной строки, в связи с чем гораздо удобнее пользоваться графической оболочкой XSpin [10].

Модель и исследуемое свойство, описанные на языках Promela и LTL соответственно, подаются на вход верификатора Spin. Верификатор автоматически осуществляет проверку выполнимости заданного свойства. Принципы работы верификаторов изложены в [7, 8, 11]. После проверки свойства Spin выдаёт один из 3 ответов:

1. Свойство выполняется.
2. Свойство не выполняется (в этом случае приводится пример поведения модели, когда свойство нарушается).
3. Недостаточно вычислительных ресурсов для верификации (т.е. необходимо упростить модель).

Модель системы

Рассмотрим фрагменты модели системы, описанной на языке Promela, отражающие работу автомата AtmPos1.

Множество состояний автоматов определяется с помощью объявления перечислимого типа mtype:

```
mtype = { S0, S1, S2, S3, S4, S5 };
```

Объявим и инициализируем переменную для хранения состояния автомата AtmPos1:

```
mtype State = S0;
```

Объявим и инициализируем переменные для хранения условий переходов автомата AtmPos1 (см. листинг 2).

Опишем работу автомата AtmPos1 в отдельном файле:

```
#define AtmPos1 "D:\Spin\PnevmoPodveska\AtmPos1.pml"
```

Первая часть файла *AtmPos1.pml* определяет условия переходов автомата (см. листинг 3).

Вторая часть файла *AtmPos1.pml* определяет переключения состояний автомата (см. листинг 4).

Переключения автомата моделируются с помощью оператора ветвления. Между ключевыми словами `if` и `fi` располагаются индикаторы начала последовательности ::, которые указывают на альтернативные ветви последовательностей операторов. Если автомат находится в нулевом состоянии, то страж `(State == S0)` выполним, остальные стражи заблокированы. За выполнимым стражем `(State == S0)` следует пустой оператор `skip`, поэтому происходит выход из оператора ветвления без какого-либо действия. Если автомат находится в 1-м состоянии, то страж `(State == S1)` выполним, остальные стражи заблокированы. За выполнимым стражем `(State == S1)` следует оператор ветвления `if`. Если условие перехода из состояния S1 в состояние S3 истинно, то страж `(tr_01_03)` выполним, поэтому за ним выполняется оператор присваивания `State = S3`. На этом последовательность операторов заканчивается и происходит выход из 2-го, а затем и из 1-го операторов ветвления. Если во время выполнения страж `(tr_01_03)` заблокирован, то происходит выполнение альтернативной ветви `else` с пустым оператором `skip` и переключения состояния автомата не происходит.

Третья часть файла *AtmPos1.pml* определяет действия автомата в состояниях (см. листинг 5).

Периодический запуск работы автомата AtmPos1 будет производиться из процесса с названием `model` (см. листинг 6):

Процесс работает в бесконечном цикле `do`, внутри которого атомарно (ключевое слово `atomic`) запускаются все автоматы модели, в том числе и автомат AtmPos1. Атомарность выполнения действий автоматов говорит о том, что верификатор не будет рассматривать их промежуточные вычисления.

Недетерминированность входных сигналов моделируется следующим образом:

```
if
:: PEmax = true
:: PEmax = false
fi;
```

Оператор ветвления используется без стражей. В этом случае обе ветви выполнимы, поэтому верификатор недетерминированным образом выбирает любую из них. Данная конструкция позволяет на любой итерации работы модели установить предельно высокое давление ресивера.

Спецификация требований

Спецификация требований – набор формализованных требований, подлежащих проверке на модели. Рассмотрим формализацию нескольких требований (свойств).

Ограниченность работы компрессора по времени, ϕ_1. Всегда, если компрессор включился, то рано или поздно он выключится:

```
LTL: □ (Kompr → (◊ ¬ Kompr)),
```

где □ – темпоральный оператор «всегда», ◊ – темпоральный оператор «рано или поздно», → – импликация (если.., то...), ¬ – логическое отрицание («не»), `Kompr` – бит включения компрессора.

Бесперебойность работы компрессора до набора давления, ϕ_2. Всегда, если компрессор включился, то он будет работать без отключений до тех пор, пока не нормализует давление в ресивере:

```
LTL: □ (Kompr → (Kompr U PE_sp)),
```

где `U` – темпоральный оператор «до тех пор, пока», `PE_sp` – фактическое давление ресивера не ниже заданного.

Соблюдение блокировок работы компрессора, ϕ_3. Всегда, если компрессор работает, то температура компрессора, ток компрессора, давление ресивера в норме и фильтр воздуха не засорён:

```
LTL: □ (Kompr → ((¬TEmax) ∧ (¬ISA) ∧ (¬PEmax) ∧ (¬PDSA))),
```

где ∧ – конъюнкция (логическое «и»), `TEmax` – температура компрессора выше установленного предела, `ISA` – ток двигателя компрессора выше установленного предела, `PEmax` – давление ресивера выше установленного предела, `PDSA` – фильтр воздуха засорён.

Работа впускных клапанов при достаточном давлении ресивера, ϕ_4. Выполнение этого требования контролируется для того, чтобы избежать перетока воздуха из пневмостоек в ресивер и, как следствие, опускания кузова автомобиля. Всегда, если хоть один впускной клапан открыт, то давление ресивера не ниже установленного минимума:

Листинг 2

```
bool tr_03_02 = false;
bool tr_02_03 = false;
bool tr_02_01 = false;
bool tr_01_03 = false;
bool tr_03_04 = false;
bool tr_04_03 = false;
bool tr_04_05 = false;
bool tr_05_03 = false;
```

Листинг 3

```
tr_03_02 = (Pos1 > SP1_H);
tr_02_03 = !tr_03_02;
tr_02_01 = tmr_Q;
tr_01_03 = ((Pos1 <= SP1) || (Error));
tr_03_04 = (Pos1 < SP1_L);
tr_04_03 = !tr_03_04;
tr_04_05 = tmr_Q;
tr_05_03 = ((Pos1 >= SP1) || (Error));
```

Листинг 4

```
if
:: (State == S0) -> skip
:: (State == S1) ->
    if
    :: (tr_01_03) -> State = S3
    :: else skip
    fi
:: (State == S2) ->
    if
    :: (tr_02_03) -> State = S3
    :: (tr_02_01) -> State = S1
    :: else skip
    fi
:: (State == S3) ->
    if
    :: (tr_03_02) -> State = S2
    :: (tr_03_04) -> State = S4
    :: else skip
    fi
:: (State == S4) ->
    if
    :: (tr_04_03) -> State = S3
    :: (tr_04_05) -> State = S5
    :: else skip
    fi
:: (State == S5) ->
    if
    :: (tr_05_03) -> State = S3
    :: else skip
    fi
fi;
```

Листинг 5

```
if
:: (State == S1) -> Req_V1out = true
:: else       Req_V1out = false
fi;
if
:: ((State == S2) || (State == S4))
-> tmr_IN = true
:: else       tmr_IN = false
fi;
if
:: (State == S5)-> Req_V1in = true
:: else          Req_V1in = false
fi;
```

Листинг 6

```
active proctype model()
{
      do
      :: atomic {
      ...
      #include AtmPos1
      ...
      }
      od
}
```

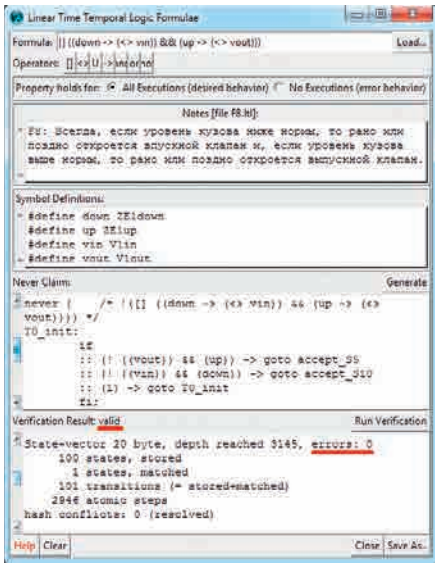

Рис. 3. Проверка свойства ϕ_8 в верификаторе Spin

`LTL: □ ((V1in ∨ V2in ∨ V3in ∨ V4in) → (¬PEmin))`,

где ∨ – дизъюнкция (логическое «или»), `V1in` – бит включения электромагнита впускного клапана пневмостойки 1, `PEmin` – давление ресивера ниже минимального предела.

Взаимоблокировка впускного и выпускного клапанов, ϕ_5. Никогда впускной и выпускной клапаны не открыты одновременно:

`LTL: □ ¬ (V1in ∧ V1out)`,

где `V1out` – бит включения электромагнита выпускного клапана пневмостойки 1.

Ограниченность работы впускного клапана по времени, ϕ_6. Всегда, если впускной клапан открылся, то рано или поздно он закроется:

`LTL: □ (V1in → (◊ ¬V1in))`.

Постоянство действия автоматической защиты от разрыва подушки пневмостойки с помощью сброса давления, ϕ_7. Всегда, если давление в пневмостойке аварийно высокое, то впускной клапан закрыт, а выпускной клапан открыт:

`LTL: □ (PE1max → (¬V1in ∧ V1out))`,

где `PE1max` – давление пневмостойки 1 выше установленного предела.

Наличие отклика системы управления на отклонения уровня кузова, ϕ_8. Всегда, если уровень кузова ниже нормы, то рано или поздно откроется впускной клапан и если уровень кузова выше нормы, то рано или поздно откроется выпускной клапан:

`LTL: □ ((ZE1down → ◊V1in) ∧ (ZE1up → ◊V1out))`,

где `ZE1down` – отклонение вниз положения кузова у пневмостойки 1, `ZE1up` – отклонение вверх положения кузова у пневмостойки 1.

Запрет подъёма кузова на высокой скорости, ϕ_9. Всегда, если фактическая скорость автомобиля выше заданной, то все впускные клапаны закрыты:

`LTL: □ (SEmax → (¬V1in ∧ ¬V2in ∧ ¬V3in ∧ ¬V4in))`,

где `SEmax` – скорость автомобиля выше установленного предела.

Свойства ϕ_7 и ϕ_9 непосредственно влияют на безопасность работы системы и движения автомобиля.

Верификация

Выполним проверку свойства ϕ_8 из спецификации требований (см. рис. 3). Для этого в XSpin необходимо открыть вкладку *Run → LTL Property manager* [10]. В поле *Formula* введём LTL-формулу, которую необходимо проверить. В поле *Property holds for* выберем *All Executions*, что означает проверку выполнимости формулы на всех вычислениях. Свойство ϕ_8 является желаемым и должно выполняться на всех возможных вычислениях. В поле *Symbol Definitions* определяются атомарные высказывания LTL-формулы. Процесс верификации запускается нажатием кнопки *Run Verification*. В результате в поле *Verification Result* будет выведено *valid*, что означает выполнимость формулы. Здесь же отображается количество обнаруженных ошибок (*errors* 0). Остальные свойства спецификации также выполнимы. При проверке свойства ϕ_8 должны быть установлены следующие параметры моделируемой системы: автоматический режим всегда включён, нештатные ситуации всегда отсутствуют; а, например, при проверке свойства ϕ_3 – наоборот, нештатные ситуации TEmax, ISA, PEmax, PDSA могут устанавливаться недетерминированным образом, что позволит корректно проверить работу блокировок компрессора.

Заключение

В статье описан процесс разработки алгоритма управления пневмоподвеской автомобиля с применением автоматного подхода. Декомпозиция задачи управления позволила рассматривать программу как систему взаимодействующих автоматов. На основе разработанной системы автоматов выполнено построение модели на языке Promela и программы на языке C. Согласно требованиям к разрабатываемой системе создана спецификация требований на языке LTL. На завершающем этапе проведена верификация системы методом model checking.

АП позволяет систематизировать логику работы программы, избежать множества ошибок, а также облегчает сам процесс разработки, т.к. графы переходов легче понимать и анализировать, чем программный код. Таким образом, использование АП упрощает моделирование и программирование системы.

Верификация программ методом model checking позволяет находить редкие критические ошибки, которые могли быть пропущены при тестировании. Совместное применение технологии АП и верификации методом model checking снижает количество ошибок в программах и повышает надёжность и безопасность встраиваемых систем.

Литература

1. *Карпов Ю.Г.* Теория автоматов: учебник для вузов. – СПб: ИД ПИТЕР, 2003. – 208 с.
2. *Зюбин В.Е.* Программирование информационно-управляющих систем на основе конечных автоматов: учеб.-метод. Пособие – Новосибирск: Новосиб. гос. ун-т., 2006. – 96 с.
3. *Шалыто А.А.* Алгоритмизация и программирование задач логического управления. – СПб: СПбГУ ИТМО, 1998. – 56 с.
4. *Шалыто А.А.* Switch-технология. Алгоритмизация и программирование задач логического управления. – СПб: Наука, 1998. – 628 с.
5. *Шалыто А.А.* Автоматное проектирование программ. Алгоритмизация и программирование задач логического управления. Известия РАН. Теория и системы управления. 2000. № 6. С. 63–81.
6. *Шалыто А., Туккель Н.* Программирование с явным выделением состояний. Мир ПК. 2001. № 8. С. 116–121. № 9. С. 132–138.
7. *Карпов Ю.Г.* MODEL CHECKING. Верификация параллельных и распределённых программных систем. – СПб: БХВ-Петербург, 2010. – 560 с.
8. *Вельдер С.Э., Лукин М.А., Шалыто А.А., Яминов Б.Р.* Верификация автоматных программ. – СПб: Наука, 2011. – 244 с.
9. www.spinroot.com
10. *Шошмина И.В., Карпов Ю.Г.* Введение в язык Promela и систему комплексной верификации Spin: учебное пособие. – СПб: СПбГПУ, 2009. – 66 с.
11. *Кларк Э.М., Грамберг О., Пелед Д.* Верификация моделей программ: Model Checking. – М.: МЦНМО, 2002. – 416 с.

Доломант

ЗАО «НАУЧНО-ПРОИЗВОДСТВЕННАЯ ФИРМА «ДОЛОМАНТ»

ОТВЕТСТВЕННАЯ ЭЛЕКТРОНИКА ДЛЯ ЖЕСТКИХ УСЛОВИЙ ЭКСПЛУАТАЦИИ

100% РОССИЙСКАЯ КОМПАНИЯ

ЗАКАЗНЫЕ РАЗРАБОТКИ

Разработка электронного оборудования по ТЗ заказчика в кратчайшие сроки

- Модификация КД существующего изделия
- Разработка спецвычислителя на базе COM-модуля
- Конфигурирование модульного корпусированного изделия
- Сборка магистрально-модульной системы по спецификации заказчика
- Разработка изделия с нуля

КОНТРАКТНОЕ ПРОИЗВОДСТВО

Контрактная сборка электроники уровней: модуль / узел / блок / шкаф / комплекс

- ОКР, технологические консультации и согласования
- Макеты, установочные партии, постановка в серию
- Полное комплектование производства импортными и отечественными компонентами и материалами
- Поддержание складов, своевременное анонсирование снятия с производства, подбор аналогов
- Серийное плановое производство
- Тестирование и испытания по методикам и ТУ
- Гарантийный и постгарантийный сервис

WWW.DOLOMANT.RU • (495) 739-0775