Введение в моделирование и верификацию аппаратных и программных систем

Лекция 6: Проверка моделей на практике

Борис Юрьевич Конев

konev@liverpool.ac.uk

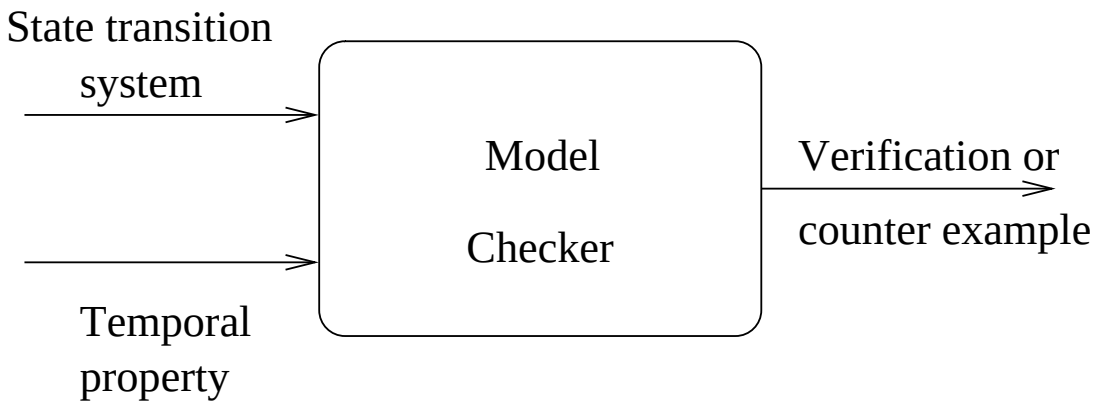
Liverpool University

Октябрь-Ноябрь 2007

Система NuSMV

- NuSMV означает *New Symbolic Model Verifier*.
- Проект с открытым кодом; систему можно бесплатно скачать с сайта `http://nusmv.irst.itc.it`
- Во многом, реализует идеи системы (с закрытым кодом) SMV, реализованной Кеном МакМилланом в Университете Карнеги Мелон (CMU).
- SMV и NuSMV используют один язык входа
- NuSMV имеет лучший интерфейс. В ней реализовано больше алгоритмов. LTL и CTL спецификации (в SMV только CTL).

NuSMV



- На вход NuSMV получает файл с описанием системы переходов и спецификацию (формулу временной логики).
- Выход: соответствует ли система спецификации
  - Если система не соответствует спецификации, NuSMV выдает путь, на котором свойство нарушается

Синтаксис программ на NuSMV

- Программа на SMV содержит один или несколько модулей (обозначаемых `MODULE`).
- Один из модулей должен называться `main`.
- Внутри модулей можно объявлять переменные (`VAR`).
- Переменным можно присваивать значения (`ASSIGN`).
- Обычно, переменным присваивают начальные значения (`init`) и значение в следующий момент времени (`next`).
- LTL свойства выделяются ключевым словом `LTLSPEC`.
- 1 означает true, а 0 означает false.
- Комментарии начинаются с --

Пример (монитор ресурсов)

```
MODULE main
VAR
  request : boolean;
  status : {ready, busy};
ASSIGN
  init(status) := ready;
  next(status) := case
                    request : busy;
                    1 : {ready, busy};
                  esac;
LTLSPEC
  G(request -> F status=busy)
```

Описание resource.smv

- Только один модуль (`main`).
- Две переменные: булева переменная `request` и переменная `status`, которая может принимать одно из двух значений, `ready` или `busy`.
- Начальное значение `request` не определено, т.е., оно может быть как true так и false.
- Следующее значение `request` не определено, то есть оно опять может быть как true так и false.
- Моделирует то, что состояние `request` определяется не программой, а средой.
- В начале исполнения, `status` имеет значение `ready`.

Следующее значение `status`

- Следующее значение `status` определяется текущим значением.
  - `request : busy;` Если текущее значение переменной есть true, то в следующий момент значением переменной `status` будет `busy`.
  - `1: {ready,busy};` Случаи анализируются сверху вниз, `1` всегда срабатывает
  - `1: {ready,busy};` в следующий момент, значение `status` может быть как `ready` так и `busy`.

Спецификация

- LTL формулы выделяются ключевым словом `LTLSPEC`.
- NuSMV использует `G`, `F`, `X`, `U` для обозначения временных операторов.
- NuSMV использует `&`, `|`, `->`, `!` для обозначения булевых связок $\wedge$, $\vee$, $\rightarrow$ и $\neg$.
- `G(request -> F status=busy)` означает, что если приходит запрос, в некоторый будущий момент система будет занята.

Особенности NuSMV: недетерминизм

- Переменные, которым не присваивается никакое значение, моделируют произвольный вход (определяется окружением) (например, `request`).
- $\{v_1, v_2, \ldots, v_n\}$ означает недетерминированный выбор одного из значений, например, `1 : {ready, busy};`

Особенности NuSMV: переходы

- Переходы могут быть заданы как список присваиваний значений переменных в следующий момент времени в зависимости от текущего значения.
- Часто применяется разбор случаев.
  - Если условие каких-либо случаев выполнено (например, `request` истинно), выполняется стоящий выше в списке.
  - Если ни одно из условий разбора случаев не выполняется, значение переменной в следующий момент времени не определено.
- Альтернативно, переходы могут быть явно заданы пропозициональной формулой.

Запуск NuSMV (1)

- NuSMV обычно запускают из командного интерпретатора UNIX или Windows
  `NuSMV resource.smv`
- Также можно запустить NuSMV в интерактивном режиме
  `NuSMV -int resource.smv`
- Руководство пользователя доступно по адресу
  `http://nusmv.irst.itc.it/NuSMV/userman/index-v2.html`
- Также, имеется краткая инструкция по применению
  `http://nusmv.irst.itc.it/NuSMV/tutorial/index.html`.

Запуск NuSMV (2)

Запуск NuSMV на рассмотренном файле выдает.

```
*** This is NuSMV 2.2.5 (compiled on Thu May  5 16:18:21 UTC 2005)
*** For more information on NuSMV see <http://nusmv.irst.itc.it>
*** or email to <nusmv-users@irst.itc.it>.
*** Please report bugs to <nusmv@irst.itc.it>.
-- specification  G (request ->  F status = busy) is true
```

Упражнение

Скачать NySMV и проверить следующие свойства:

- `G((request & X request) -> ((X status=busy) & (X X status = busy)))`
- `(G request) -> (X G status=busy)`

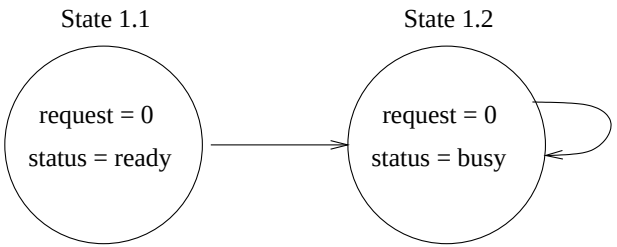
Контрмодель

Если мы попробуем другое свойство, `G status = busy`, мы получим следующее:

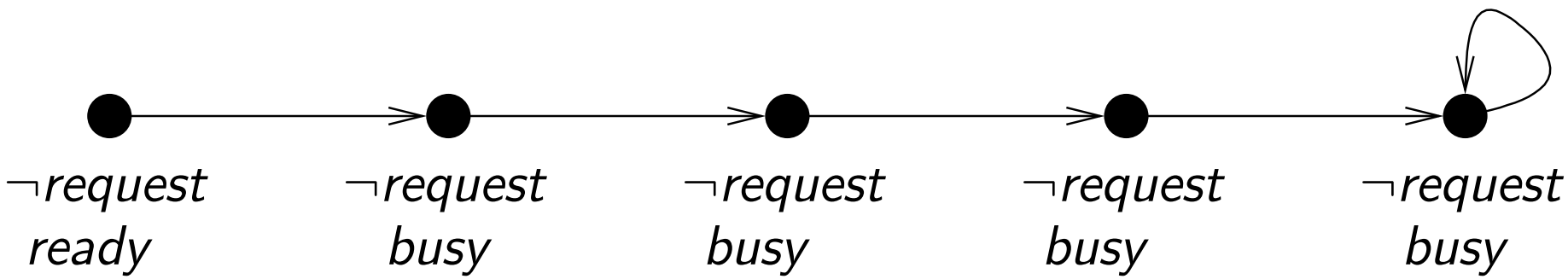
```
*** This is NuSMV 2.2.5 (compiled on Thu May  5 16:18:21 UTC 2005)
*** For more information on NuSMV see <http://nusmv.irst.itc.it>
*** or email to <nusmv-users@irst.itc.it>.
*** Please report bugs to <nusmv@irst.itc.it>.
-- specification  G status = busy is false
-- as demonstrated by the following execution sequence
Trace Description: LTL Counterexample
Trace Type: Counterexample
-> State: 1.1 <-
  request = 0
  status = ready
-- Loop starts here
-> State: 1.2 <-
  status = busy
-> State: 1.3 <-
```

Что это значит

- Если переменная не упомянута, ее значение не изменяется.
- Следующий путь через систему переходов является контрпримером.

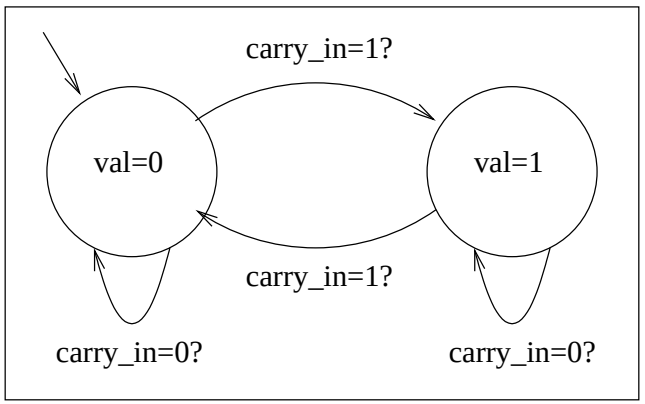


($\neg request$ означает $request = 0$, а $busy$ означает $status = busy$).



Моделирование более чем одного процесса

- Рассмотрим модель счетчика до трех в двоичной записи: от 000 до 111 (и далее по циклу).
- Для этого скомбинируем три счетчика от 0 до 1



- Бит *carry _in* есть бит переноса от другого счетчика (для комбинирования)

На языке NuSMV

```
MODULE main
VAR
  bit0 : counter_cell(1);
  bit1 : counter_cell(bit0.carry_out);
  bit2 : counter_cell(bit1.carry_out);

LTLSPEC
  G F bit2.carry_out

MODULE counter_cell(carry_in)
VAR
  value : boolean;
ASSIGN
  init(value) := 0;
  next(value) := (value + carry_in) mod 2;
DEFINE
  carry_out := value & carry_in;
```

Пояснения

- Два модуля `main` и `counter_cell`.
- `counter_cell` реализует систему переходов, описанную выше. Имеет один входной параметр (`carry_in`).
- В модуле `main` создается три экземпляра `counter_cell` названные `bit0`,`bit1`,`bit2`.
- Входной параметр `bit0` равен 1, в то время как для `bit1` и `bit2` значение параметра `carry_out` есть `bit0` и `bit1`, соответственно.
- Осуществляется синхронная композиция модулей (режим по умолчанию для NuSMV). То есть, имеются глобальные часы и каждый модуль выполняет одну инструкцию к в каждый момент времени. (Синхронизированная композиция.)

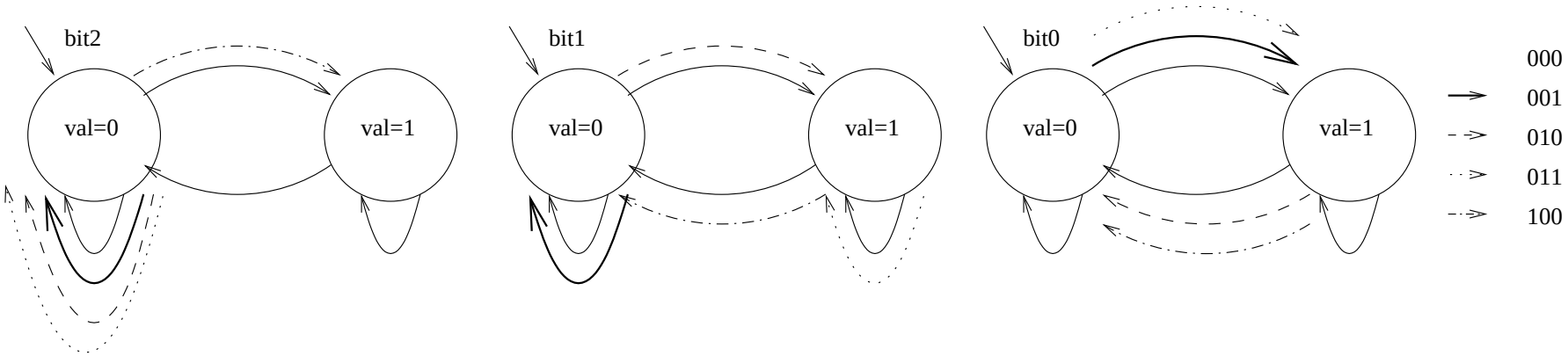
Оператор Define

- Оператор DEFINE аналогичен оператору #define в языке C
  Например, при наличии кода

  ```
  DEFINE
    carry_out := value & carry_in;
  ```

  каждое вхождение carry_out заменяется на ее определение value & carry_in

Синхронное исполнение



$next(value) := (value + carry\_in) mod 2$ $carry\_out := value \& carry\_in$

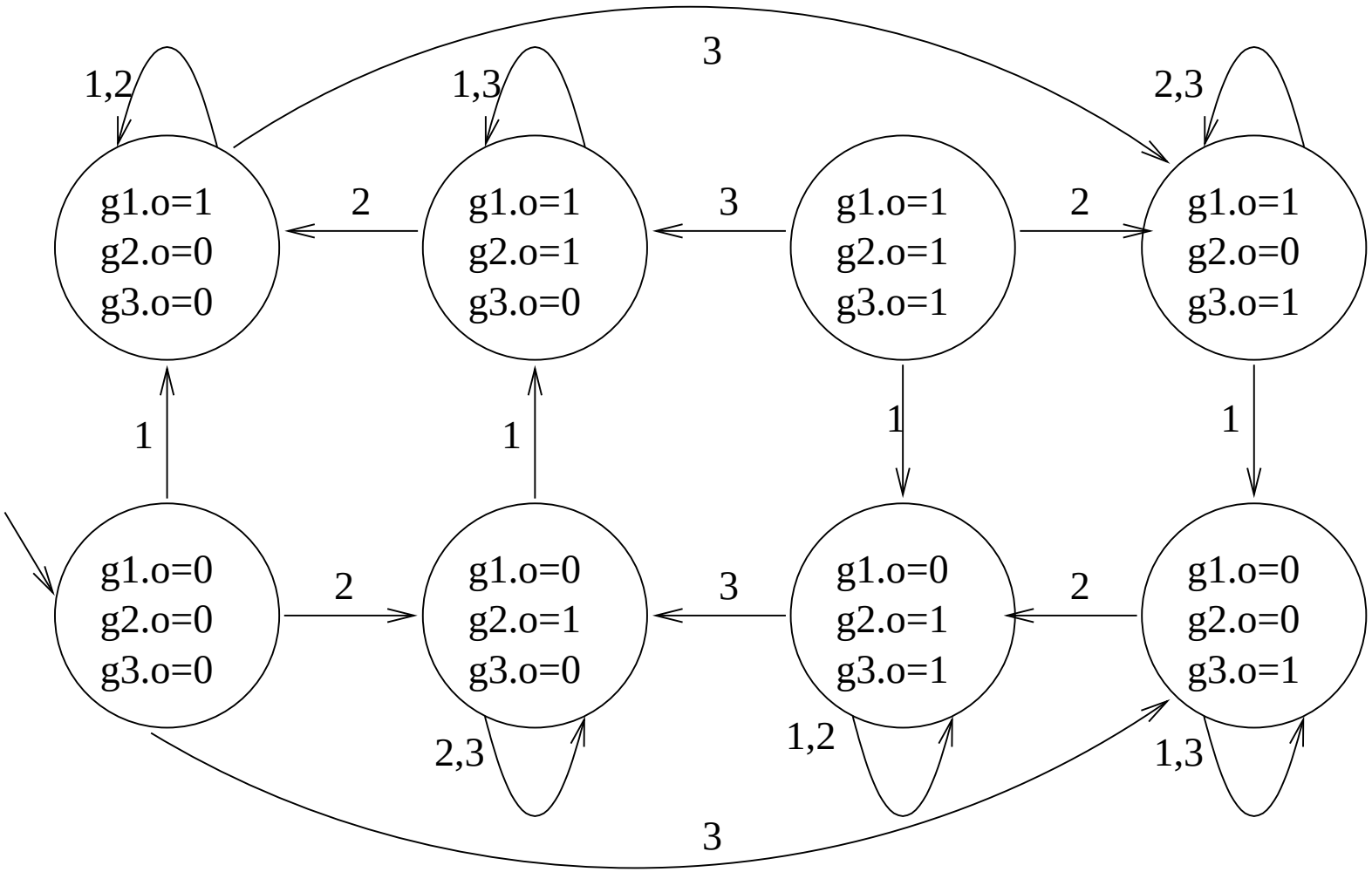
Next	bit2	bit1		bit0	
	next	next	carry_out	next	carry_out
001	0 + 0 =0	0 + 0 =0	0 & 0 =0	0 + 1 =1	0 & 1 = 0
010	0 + 0 =0	0 + 1 =1	0 & 1 =0	1 + 1 =0	1 & 1 = 1
011	0 + 0 =0	1 + 0 =1	1 & 0 =0	0 + 1 =1	0 & 1 = 0
100	0 + 1 =1	1 + 1 =0	1 & 1 =1	1 + 1 =0	1 & 1 = 1

Композиция модулей

- Поддерживается как синхронная так и асинхронная композиция модулей
- Синхронная композиция означает, что каждая компонента системы делает один шаг вычислений
  - Режим по умолчанию
- Асинхронная композиция означает, что только одна (произвольно выбранная) компонента может делать шаг в каждый момент времени. Т.о., одна компонента может сделать несколько шагов до того, как другая компонента сделает шаг.
- Чтобы скомбинировать модули асинхронно, следует использовать ключевое слово `process` при порождении экземпляра модуля.

Пример: кольцо инверторов

Асинхронное исполнение



Кольцо инверторов

```
MODULE inverter(input)
  VAR
  output : boolean;

  ASSIGN
   init(output) := 0;
   next(output) := !input;

MODULE main
  VAR
    g1: process inverter(g3.output);
    g2: process inverter(g1.output);
    g3: process inverter(g2.output);
```

Пояснения

- Порождается три экземпляра инвертирующего процесса, работающие асинхронно (обозначается ключевым словом `process`).
- Один из экземпляров выбирается недетерминированным образом и выполняется оператор присваивания.
- Если переменной не присваивается значение, ее значение не изменяется.
  Например, если `g1.output = 0` `g2.output = 0` `g3.output = 0` и `g2` выбирается для исполнения, значения `g1.output` и `g3.output` не меняются (сохраняют значение 0).
- Заметим, что это, в частности, означает, что какой-либо из процессов может вообще никогда не исполниться.

Где справедливость?

Справедливое исполнение (когда каждый процесс исполняется бесконечно часто) можно вынудить.

- В NuSMV используется ключевое слово `FAIRNESS`.
- `FAIRNESS` φ, где φ пропозициональная формула заставляет программу проверки моделей рассматривать только такие пути, на которых φ становится истинной бесконечно часто.
- В NuSMV у каждого процесса (неявно) имеется специальная переменная `running`, которая имеет значение 1 тогда и только тогда, когда процесс исполняется.
- Добавив `FAIRNESS running` в код инвертора, мы заставим каждый экземпляр инвертора работать бесконечно часто.

Условие справедливости (fairness)

Нетрудно убедиться, что формула

```
LTLSPEC G F (g1.output = 1) &
        G F (g2.output = 1) &
        G F (g3.output = 1)
```

верна при условии, что в код инвертора добавлено `FAIRNESS running`.
Однако если условие справедливости убрать, свойство нарушается.

Условие compassion

Для проверки LTL моделей NuSMV предоставляет более сильные условия справедливости вида “если на каком-то пути φ становится истинной бесконечно часто, то и φ становится истинной бесконечно часто на этом пути.

В NuSMV используется ключевое слово `COMPASSION`.

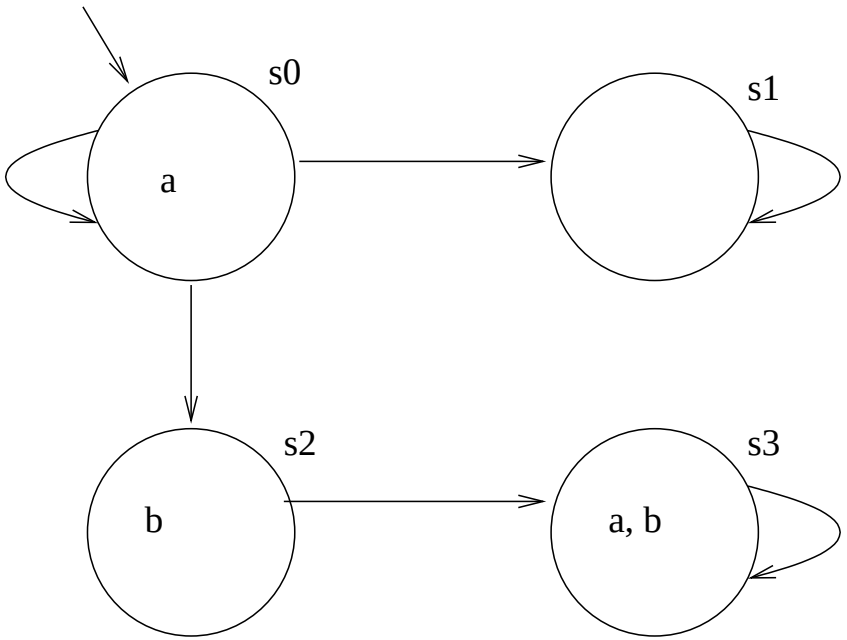
`COMPASSION` (φ, φ), где φ и φ суть пропозициональные формулы.

Проверка CTL свойств

- Синтаксис языка описания моделей идентичен языку описания моделей для проверки LTL свойств.
- CTL формулы отмечаются ключевым словом `SPEC` (а не `LTLSPEC`).
- Пропозициональные связки такие же `&`, `|`, `->`, `!`
- Для унарных пар операторов используются `AG`, `AF`, `AX`, `EG`, `EF`, `EX`
- Для бинарных операторов в NuSMV используются `A` `[`φ `U` ψ`]` и `E` `[`φ `U` ψ`]` (обратите внимание на *квадратные скобки*).

Пример

Рассмотрим систему переходов



и свойства

- $\mathbf{E}(a\mathbf{U}b)$;
- $\mathbf{E}(a\mathbf{U}b) \wedge \neg b$;
- $\mathbf{AXE}(a\mathbf{U}b)$.

Входной файл NuSMV

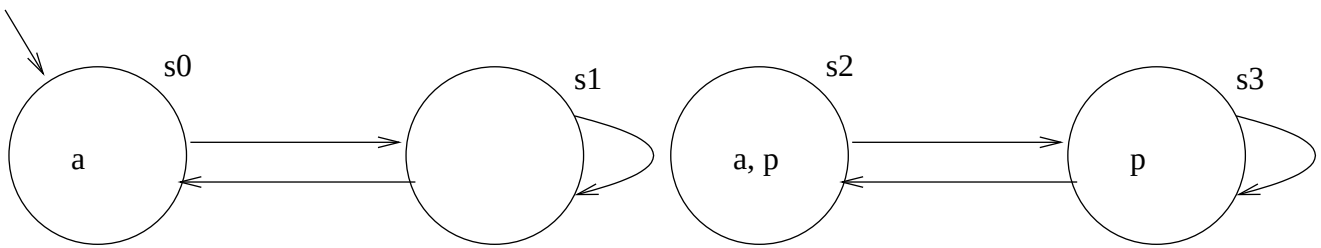
```
MODULE main
VAR
  state : {s0,s1,s2,s3};
ASSIGN
   init(state) := s0;
   next(state) :=
        case
        state = s0: {s0, s1, s2};
        state = s1: s1;
        state = s2: s3;
        state = s3: s3;
        esac;
DEFINE
     a:= (state = s0) | (state = s3);
     b:= (state = s2) | (state = s3);
SPEC      E [a U b]
SPEC      E [a U b] & !b
SPEC      AX E [a U b]
```

Результат работы NuSMV

```
NuSMV test3.smv
*** This is NuSMV 2.2.5 (compiled on Thu May  5 16:18:21 UTC 2005)
*** For more information on NuSMV see <http://nusmv.irst.itc.it>
*** or email to <nusmv-users@irst.itc.it>.
*** Please report bugs to <nusmv@irst.itc.it>.
-- specification E [ a U b ]  is true
-- specification (E [ a U b ]  & !b) is true
-- specification AX E [ a U b ]  is false
-- as demonstrated by the following execution sequence
Trace Description: CTL Counterexample
Trace Type: Counterexample
-> State: 1.1 <-
  state = s0
  b = 0
  a = 1
-> State: 1.2 <-
  state = s1
  a = 0
```

Контрмодель

- Как и в случае проверки LTL моделей, NuSMV пытается построить контрмодель, если свойство не выполняется.
- Заметим, что это не всегда возможно в CTL.
- Утверждения о существовании пути не могут быть опровергнуты предъявлением единственного пути.
- Рассмотрим контрмодель для свойства $\mathbf{EF}p$ и модели переходов



Справедливость

- Так же как и в случае LTL, возможно ограничить рассмотрение только такими путями через систему переходов, на которых некоторое условие становится истинным бесконечно часто. Назовем такие пути справедливыми путями.
  - Позволяет исключить пути, на которых один из процессов вообще не исполняется, на которых один из процессов никогда не выходит из критической секции и т.п.
- Такой же синтаксис как и в случае LTL: `FAIRNESS` φ, где φ — пропозициональная формула.

О справедливости в CTL и LTL

- Свойство справедливости выразимо в логике LTL.
  - Чтобы проверить свойство φ в присутствии свойства справедливости φ (т.е. φ должно быть верно бесконечно часто), достаточно проверить свойство $(\mathbf{GF}\varphi) \Rightarrow \varphi$ без каких-либо ограничений на систему переходов.
- Это не возможно в CTL т.к. $(\mathbf{GF}\varphi) \Rightarrow \varphi$ не является CTL формулой и, как мы видели, добавление **A** или **E** будет означать что-то другое.
  - Например $(\mathbf{AGAF}\varphi) \Rightarrow \varphi$ означает если все пути справедливые то φ верно, а не φ верно на всех справедливых путях.
- NuSMV не поддерживает свойство compassion для CTL.

Проверка справедливых CTL моделей

- Пусть $C = \{\psi_1, \ldots, \psi_n\}$ — множество условий справедливости. Путь $s_0, s_1, \ldots$ справедлив по отношению к C тогда и только тогда, когда для каждого i существует бесконечно много j т.ч. $s_j \models \psi_i$.
- $\mathbf{A}_C$ и $\mathbf{E}_C$ означают кванторы по справедливым путям.
- Все CTL операторы (с условием справедливости) могут быть выражены через $\mathbf{E}_C\mathbf{U}$, $\mathbf{E}_C\mathbf{G}$, $\mathbf{E}_C\mathbf{X}$.
- Заметим, что

$$\begin{aligned} \mathbf{E}_C\varphi\mathbf{U}\psi &\equiv \mathbf{E}\varphi\mathbf{U}(\psi \wedge \mathbf{E}_C\mathbf{G}\mathbf{true}) \\ \mathbf{E}_C\mathbf{X}\varphi &\equiv \mathbf{E}\mathbf{X}(\varphi \wedge \mathbf{E}_C\mathbf{G}\mathbf{true}) \end{aligned}$$

  (путь справедлив если его суффикс справедлив)
- Значит, надо только научиться обрабатывать $\mathbf{E}_C\mathbf{G}\varphi$.

Динамический алгоритм для CTL с условием справедливости. $\mathbf{E}_C\mathbf{G}\varphi$

$C = \{\psi_1, \ldots, \psi_n\}$ — множество условий справедливости.

- Рассмотрим вершины, помеченные φ
- Найдем максимальные компоненты сильной связности ограниченного графа
- Удалим компоненту если она не содержит узла, помеченного ψ_i для некоторого i. Назовем такую компоненту справедливой.
- Используя поиск “назад”, найдем вершины ограниченного графа, из которых существует путь в вершину справедливой компоненты.