Система автоматической верификации Spin

Евтушенко Н.В., Винарский Е.М.

*по всем вопросам писать на **vinevg2015@gmail.com** или в телеграмм @evgenii1996*

30 октября 2020 г.

Типы свойств, обычно проверяемые системой Spin

- свойство достижимости (*reachability*): может ли быть достигнуто заданное состояние системы
- свойство безопасности (*safety*): нечто плохое и нежелательное никогда не произойдёт
- свойство живости (*liveness*): при некоторых условиях нечто хорошее обязательно произойдёт
- свойство справедливости (*fairness*): нечто будет вычисляться бесконечно часто

vinevg2015@gmail.com

Возможная схема решения задачи Model Checking

Цель при верификации: удостовериться, что система обладает требуемыми свойствами. Для этого мы:

1. Строим формальную модель, описывающую поведение системы в виде композиции конечных автоматов, расширенных конечных автоматов, входо-выходных полуавтоматов, ...
2. Описываем поведение (компонентов) системы на языке Promela
3. Строим *LTL*-формулы, описывающие свойства системы
4. Анализируем результаты работы верификатора Spin:
   - Если Spin находит контрпример, то свойство не выполняется
   - **НО** если свойство не выполняется, то Spin может и не найти контрпример (по причине переполнения памяти)

Для небольших систем Spin всегда находит контрпример

vinevg2015@gmail.com

LTL-формулы в Spin

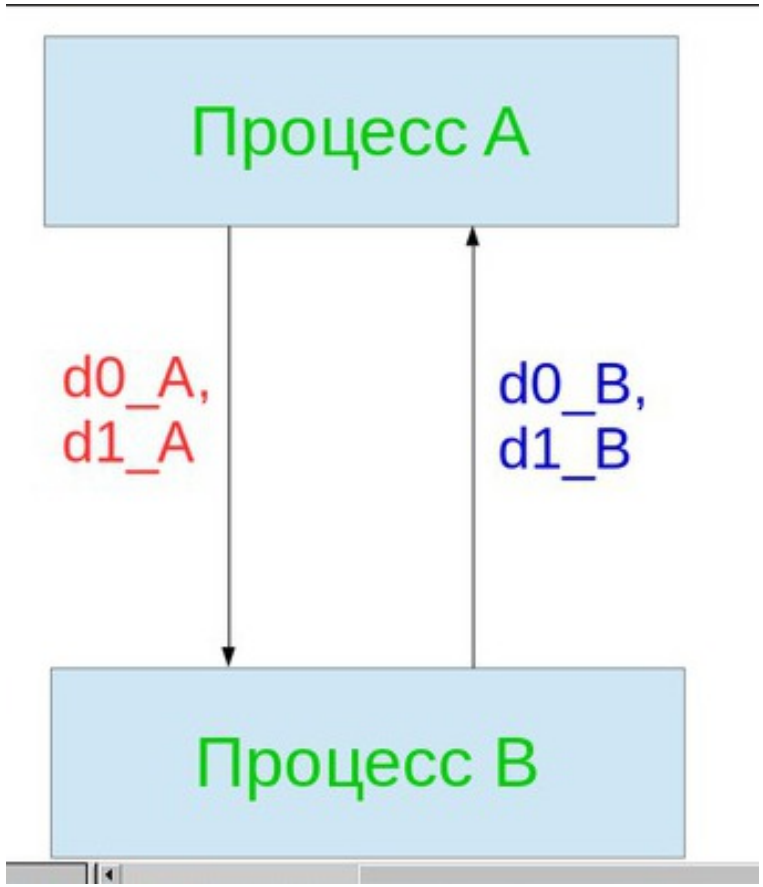
Операторы темпоральной логики в Promela

- оператор G (всегда в будущем) – []
- оператор F (когда-то в будущем) – <>
- оператор U (до тех пор, пока) – *until*
- оператор X (в следующий момент) – отсутствует в Spin
- конъюнкция – &&
- дизъюнкция – ||
- импликация – $\rightarrow$
- отрицание – !

vinevg2015@gmail.com

Протокол посылки-приёма сообщения

- Процесс A посылает сообщения с альтернирующим битом (по очереди) d_0, d_1
- A может послать сообщение только после получения соответствующего сообщения от B
- B может послать сообщение только после получения соответствующего сообщения от A



vinevg2015@gmail.com

Формальное описание процессов

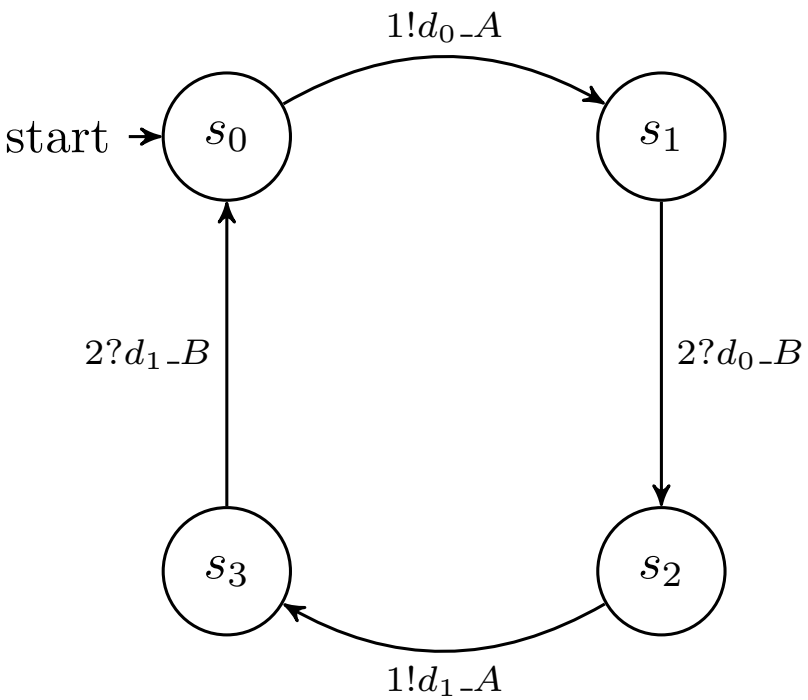


Рис.: Процесс A

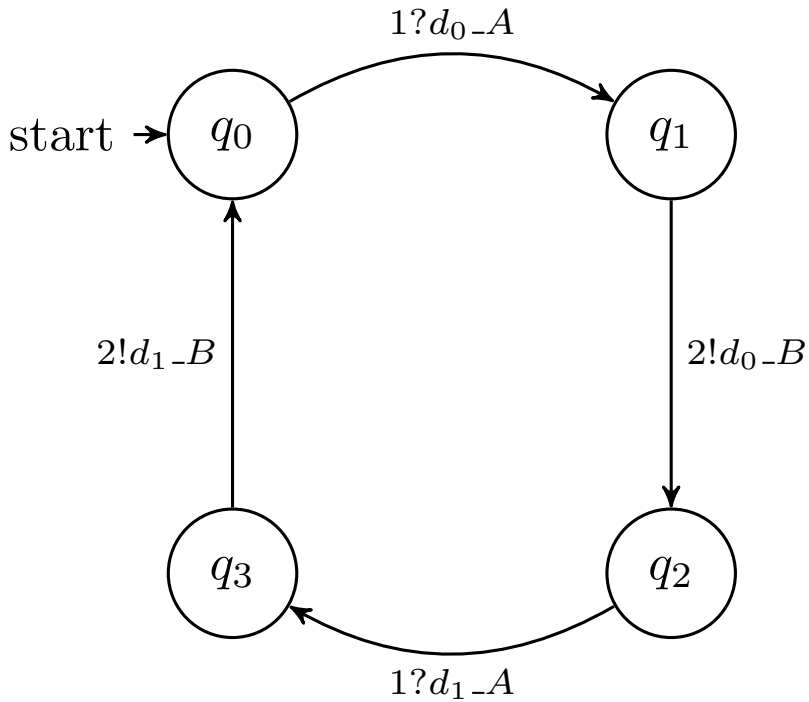


Рис.: Процесс B

Проверим, что система удовлетворяет следующим свойствам:

- $\mathsf{G}(s_0\_A) \rightarrow (\neg s_1\_A \mathsf{U} r_0\_B)$
- $\mathsf{G}(s_1\_A) \rightarrow (\neg s_0\_A \mathsf{U} r_1\_B)$
- $\mathsf{GF}(true)$

vinevg2015@gmail.com

Описание на языке *Promela*

```
1   mtype = {
2     d0_A, d1_A,
3     d0_B, d1_B
4   }
5
6   bool s0_A = false; bool s1_A = false;
7   bool r0_B = false; bool r1_B = false;
8   int state_A = 0;
9   int state_B = 0;
10
```

Рис.: типы данных

```
proctype A(chan in, out) {
  do
  :: (state_A == 0) ->
       out!d0_A
       state_A = 1
  :: (state_A == 1) ->
       in?d0_B
       state_A = 2
  :: (state_A == 2) ->
       out!d1_A
       state_A = 3
  :: (state_A == 3) ->
       in?d1_B
       state_A = 0
  od;
}
```

Рис.: процессы

Описание на языке *Promela* (2)

```
45      init {
46          chan ch1 = [0] of { mtype };
47          chan ch2 = [0] of { mtype };
48          atomic {
49              run A(ch1, ch2);
50              run B(ch2, ch1);
51          }
52      }
```

Рис.: процесс *init*

```
53
54      ltl f1 {[](s0_A -> ((!s1_A) until (r0_B)))}
55      ltl f2 {[](s1_A -> ((!s0_A) until (r1_B)))}
56      ltl f3 {[]<>(1)}
```

Рис.: синтаксис *LTL* формул

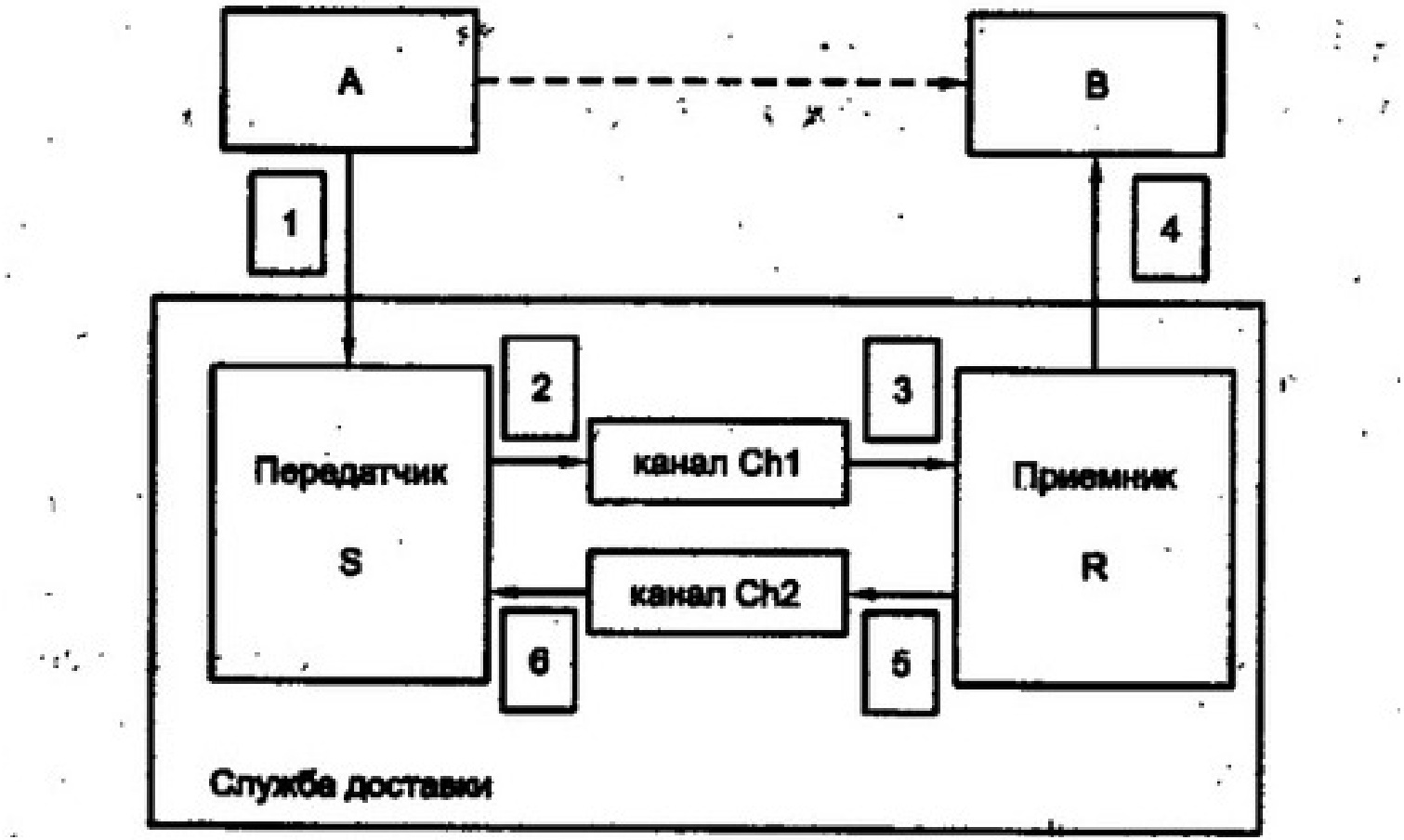
vinevg2015@gmail.com

PAR (протокол с положительными подтверждениями и повторением передачи)

- Пользователь A направляет поток сообщений пользователю B двух типов d_0 и d_1
- Для передачи используются НЕнадёжные каналы
- Сервис доставки использует передатчик S и приёмник R
- Передатчик посылает сообщения в канал и ждёт подтверждения, если передатчик не дождался подтверждения, то передатчик дублирует сообщение
- При получении сообщения, приёмник посылает подтверждение через канал
- Для исключения дублирования сообщений A не посылает подряд 2 сообщения одного типа

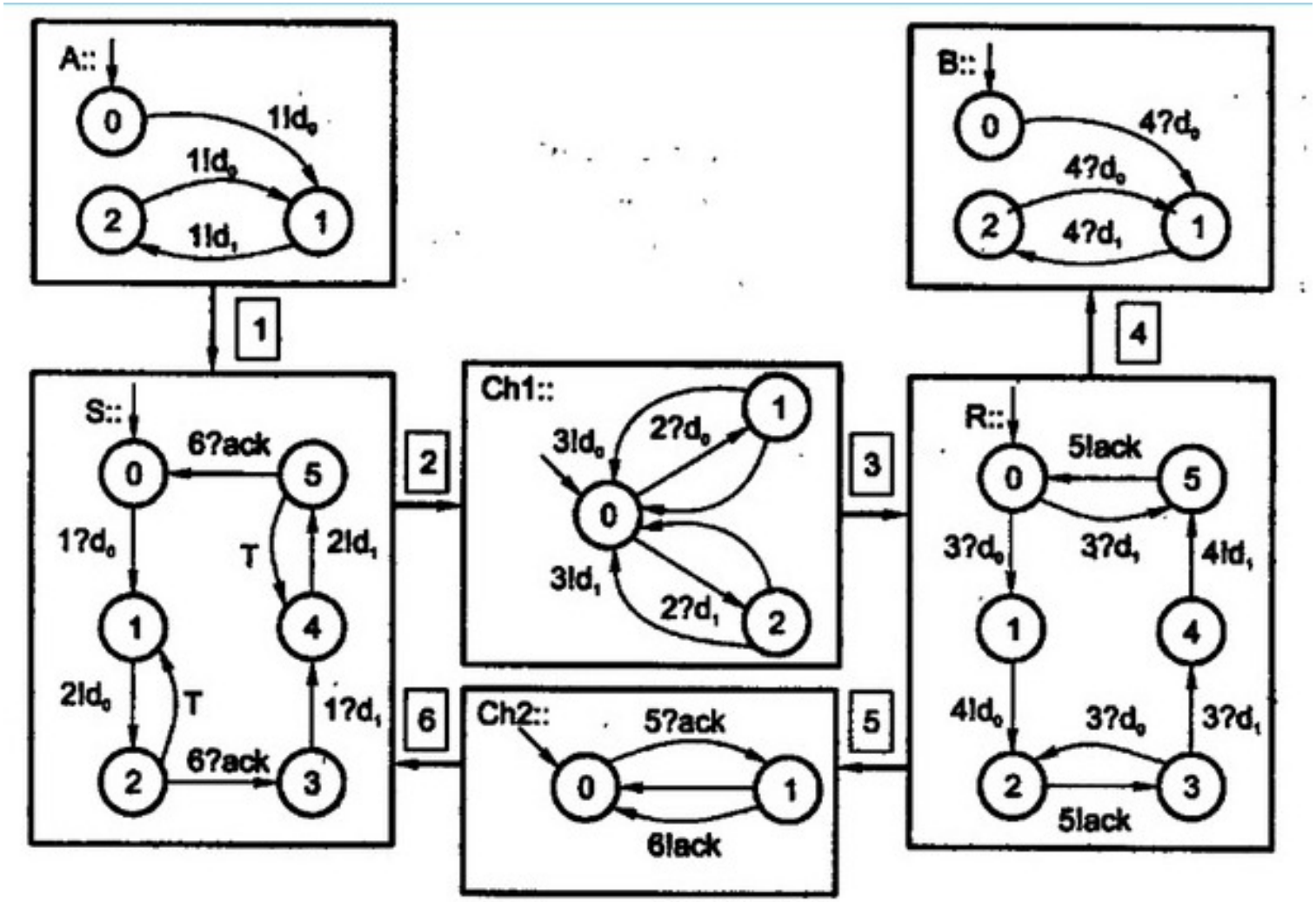
vinevg2015@gmail.com

Топология сети протокола *PAR*



vinevg2015@gmail.com

Конечно-автоматное представление протокола *PAR*



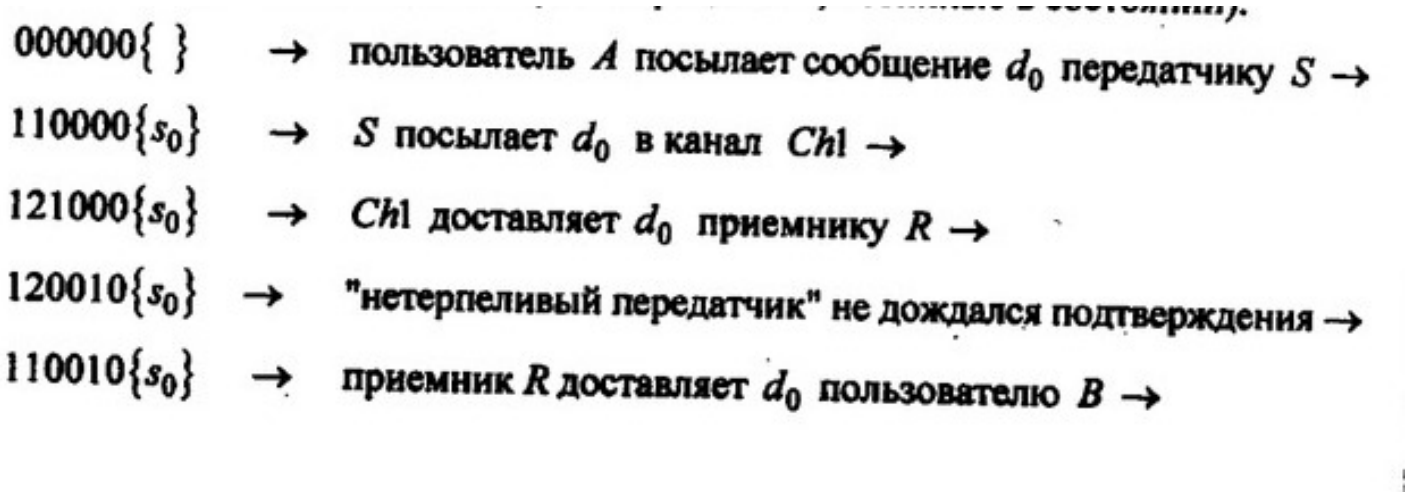
Проверяемые свойства

1. Все сообщения, посланные пользователем A должны быть доставлены B без потерь и дублирования и быть получены в том же порядке, в каком они отправлялись
$\mathbb{G}(s_0 \to \neg s_1 \mathbb{U} r_0) \wedge \mathbb{G}(s_1 \to \neg s_0 \mathbb{U} r_1) \wedge \mathbb{G}(r_0 \to \neg r_1 \mathbb{U} s_0) \wedge \mathbb{G}(r_1 \to \neg r_0 \mathbb{U} s_1)$
2. Если A хочет послать сообщение, то он сможет это сделать
$\mathbb{G}((s_0 \to \mathbb{F} s_1) \wedge (s_1 \to \mathbb{F} s_0))$

vinevg2015@gmail.com

Контрпример (1)

- s_k – Передатчик послал сообщение с номером k
- r_k – Приёмник получил сообщение с номером k



Контрпример (2)

110021$\{s_0, r_0\}$ → "нетерпеливый передатчик" повторно посылает d_0 в канал $Ch1$ →

121021$\{s_0, r_0\}$ → приемник R посылает в канал $Ch2$ подтверждение приема d_0 →

121131$\{s_0, r_0\}$ → передатчик S получает из $Ch2$ подтверждение приема d_0 →

131031$\{s_0, r_0\}$ → канал $Ch1$ повторно доставляет d_0 приемнику R →

130021$\{s_0, r_0\}$ → передатчик S принимает от A новые данные d_1 →

240021$\{s_1, r_0\}$ → S передает эти данные в канал $Ch1$ →

252021$\{s_1, r_0\}$ → R передает в $Ch2$ подтверждение повторного получения d_0 →

252131$\{s_1, r_0\}$ → канал $Ch1$ теряет сообщение d_1 →

250131$\{s_1, r_0\}$ → S получает подтверждение от $Ch2$ (считает, что d_1 дошло) →

200031$\{s_1, r_0\}$ → S принимает от A новые данные d_0 →

110031$\{s_0, r_0\}$ → S посылает d_0 в канал $Ch1$; здесь нарушение свойства $\mathbf{G}(s_1 \Rightarrow (\neg s_0 \mathbf{U} r_1))$ →

vinevg2015@gmail.com

Верификация протокола PAR

Для удобства реализации расширим алфавит:

- d_0_A, d_1_A,
- d_0_Sender, d_1_Sender, $TIME\_TRANSITION$,
- $d_0\_Channel\_1$, $d_1\_Channel\_1$, $LOSS\_Channel\_1$,
- $d_0\_Receiver$, $d_1\_Receiver$, $ACK\_Receiver$,
- $ACK\_Channel\_2$, $LOSS\_Channel\_2$,

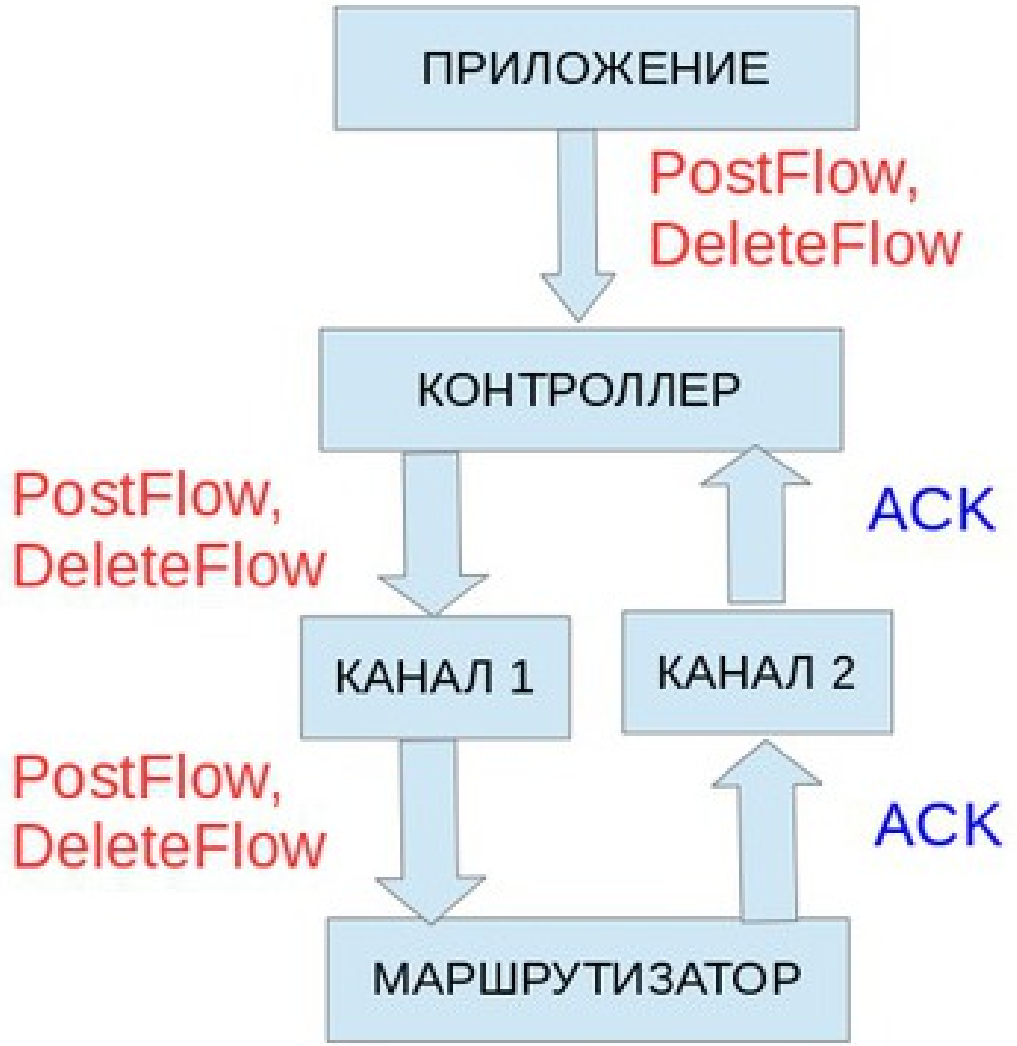
vinevg2015@gmail.com

Упрощённая модель SDN структуры

- Маршрутизатор обеспечивает передачу данных через между узлами сети посредством специальных правил (*flow entry*) передачи данных.
- Приложение устанавливает такие правила для маршрутизатора, используя контроллер (сигналы *PostFlow, DeleteFlow*)
- Контроллер "пропихивает"полученные сигналы *PostFlow, DeleteFlow* в маршрутизатор
- Маршрутизатор посылает подтверждения о получении сигнала в контроллер
- Каналы НЕ надёжные и сообщения могут в них потеряться

vinevg2015@gmail.com

Топология сети



vinevg2015@gmail.com

Задание для самостоятельной работы

Пусть

- *flow_entry_cont* – количество правил в маршрутизаторе на конкретный момент времени "с точки зрения приложения и контроллера"
- *flow_entry_switch* – реальное количество правил в маршрутизаторе на конкретный момент времени

Для удобства, как и в примере PAR, расширяем алфавит $mtype = \{PostFlow\_App, DeleteFlow\_App, PostFlow\_Cont, DeleteFlow\_Cont, TIME\_TRANSITION, PostFlow\_Channel1, DeleteFlow\_Channel1, LOSS\_Channel1, ACK\_Channel2, LOSS\_Channel2, ACK\_Switch\}$

vinevg2015@gmail.com

Задание для самостоятельной работы (2)

Компоненты упрощённой модели SDN в виде конечных автоматов и расширенных конечных автоматов есть в репозитории
Необходимо

- Описать компоненты на языке Promela
- Проверить следующие свойства, написав подходящие *LTL*-формулы, и найти соответствующие контрпримеры:
  - Количество правил в коммутаторе и количество правил в коммутаторе “с точки зрения приложения и контроллера” должно быть неотрицательным
  - В системе не должно быть “тупиковых” ситуаций (deadlock)

vinevg2015@gmail.com

Полуавтоматы, моделирующие приложение и контроллер

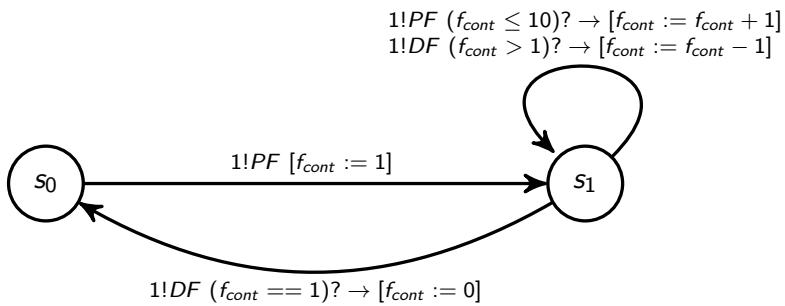


Рис.: Приложение

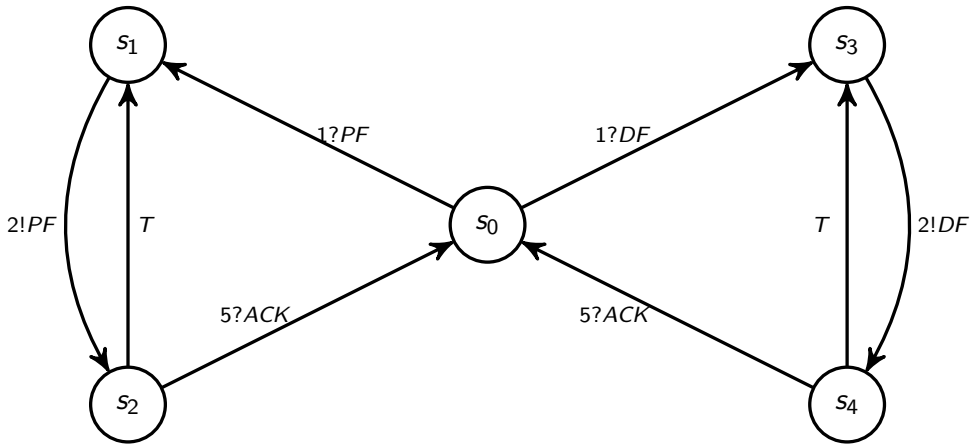


Рис.: Контроллер

Полуавтоматы, моделирующие каналы и коммутатор

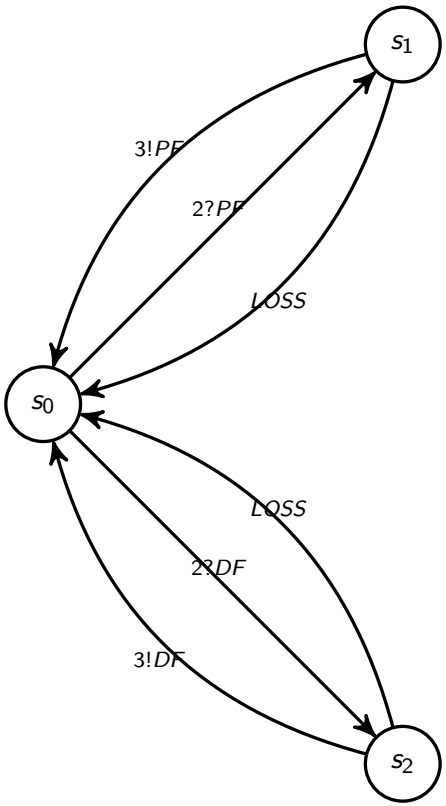


Рис.: Канал 1

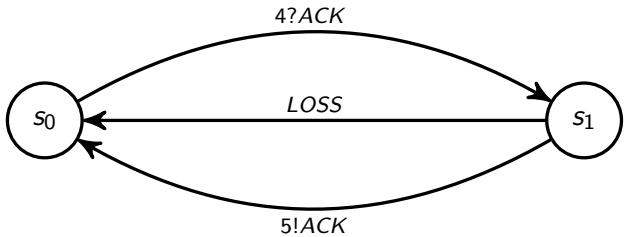


Рис.: Канал 2

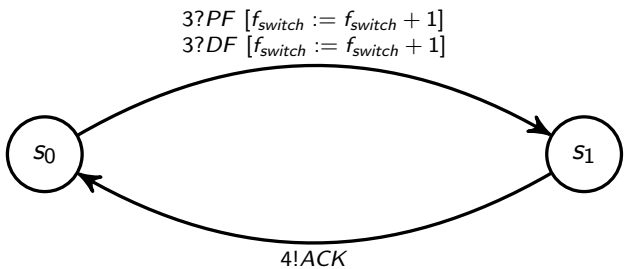


Рис.: Коммутатор