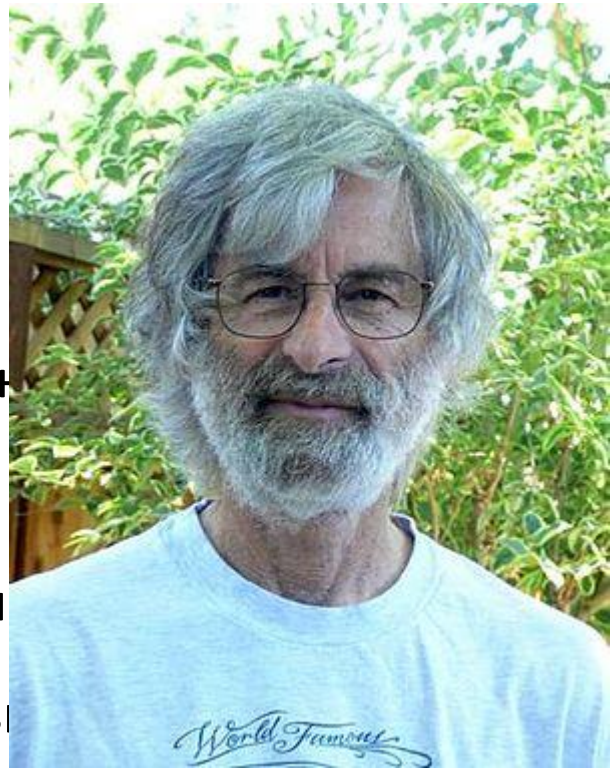
TLA+ и моделирование динамических свойств

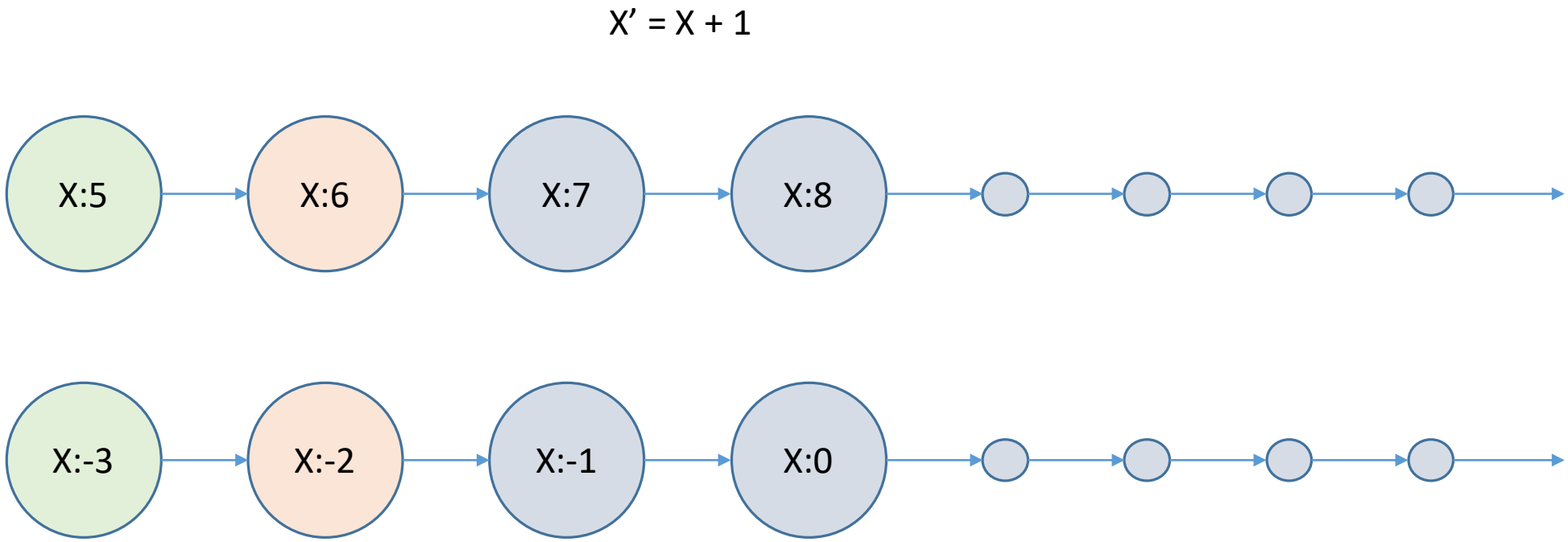
История

- TLA+ – temporal logic of actions (а “+” – это структуры данных и операции над ними)
- Автор – Лэсли Лэмпорт
- Премии:
  - 2000 — Премия Дейкстры
  - 2004 — Премия Эмануэля Пиора IEEE за вклад в развитие теории и практики параллельного программирования и отказоустойчивых вычислен
  - 2005 — Премия Дейкстры за работу *Reaching Agreement in the Presence of Faults*.
  - 2008 — Медаль Джона фон Неймана IEEE за фундаментальный вклад в теорию распределённ и параллельных вычислений.
  - 2013 — Премия Тьюринга за фундаментальный в в теорию распределённых систем.
  - 2014 — Премия Дейкстры.
  - 2019 — C&C Prize (от корпорации NEC).



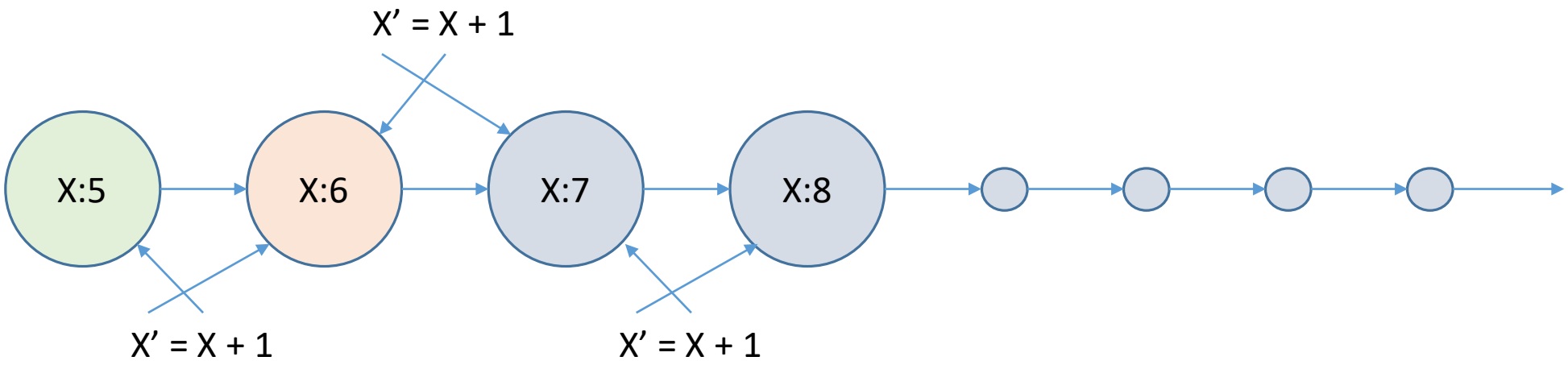
Основы TLA+

- Система моделируется множеством цепочек состояний – поведениями (цепочки бесконечные, множество цепочек в общем случае - тоже)



Основы TLA+

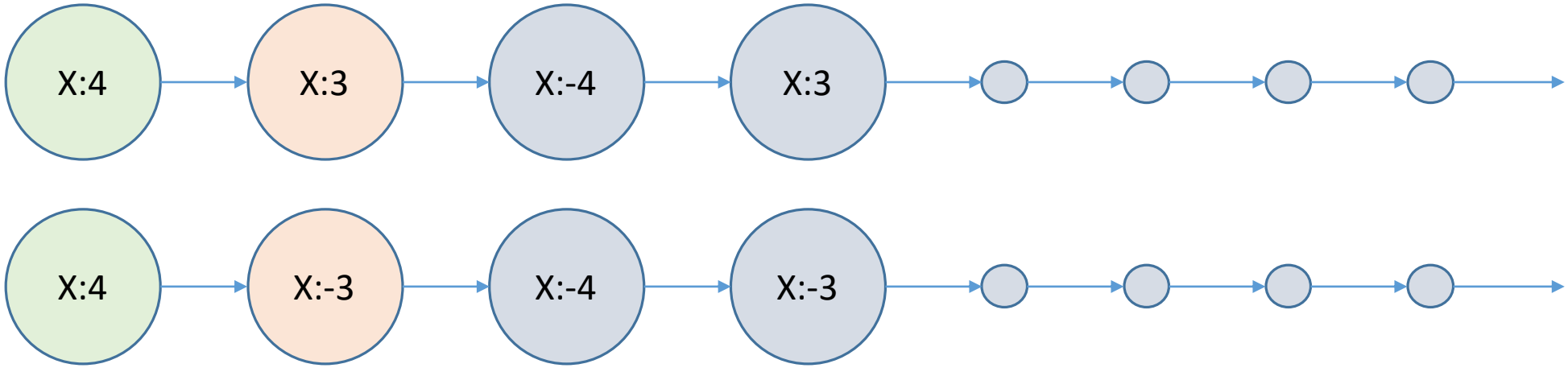
- Множество поведений задаётся логической формулой над переменными состояния: допускаются только те цепочки состояний, где TLA+ формула истинна в каждом состоянии



Основы TLA+

- Формула связывает значения текущего состояния и последующего через переменные состояния для текущего и для следующего состояний. Переменные следующего состояния обозначаются штрихом “ X’ ”

X * X + X’ * X’= 25



Основы TLA+

- Данные и структуры данных:
  - Множества
    - {1,2,3}
    - 2..7
    - {x \in {1,2,3} : x % 2 = 0}
    - {x * x : x \in {1,2,3}}
    - CHOOSE x \in {1,2,3} : x > 2
  - Функции
    - [x \in {1,2,3} |-> x * x]
    - [0 |-> 4, 1 |-> 5]
    - f[0]
  - Записи
    - [A |-> 0, B |-> 5]
    - r.A
  - Кортежи
    - <<1,2,3>>
    - t[3]

Основы TLA+

- Синтаксический “сахар”
  - LET A == Exp IN Exp
  - IF p THEN Exp1 ELSE Exp2
  - CASE
  - Двумерные OR/AND

```
/\ \/ P1
   \/ P2
   \/ P3
/\ \/ P4
   \/ /\ P5
      /\ P6
```

Основы TLA+

- Определения:
  - PrevBlock(block) == CHOOSE b \in AllBlocks: NextBlockAddress(b) = block.A
- Определения – как макросы в Си – их можно считать просто текстовыми подстановками
- Модули:
  - MODULE Name
  - CONSTANTS Param1, Param2
  - EXTENDS M1, M2, M3
  - INSTANCE M WITH Param1 <- Exp, Param2 <- Exp
  - VARIABLES V1, V2, V3

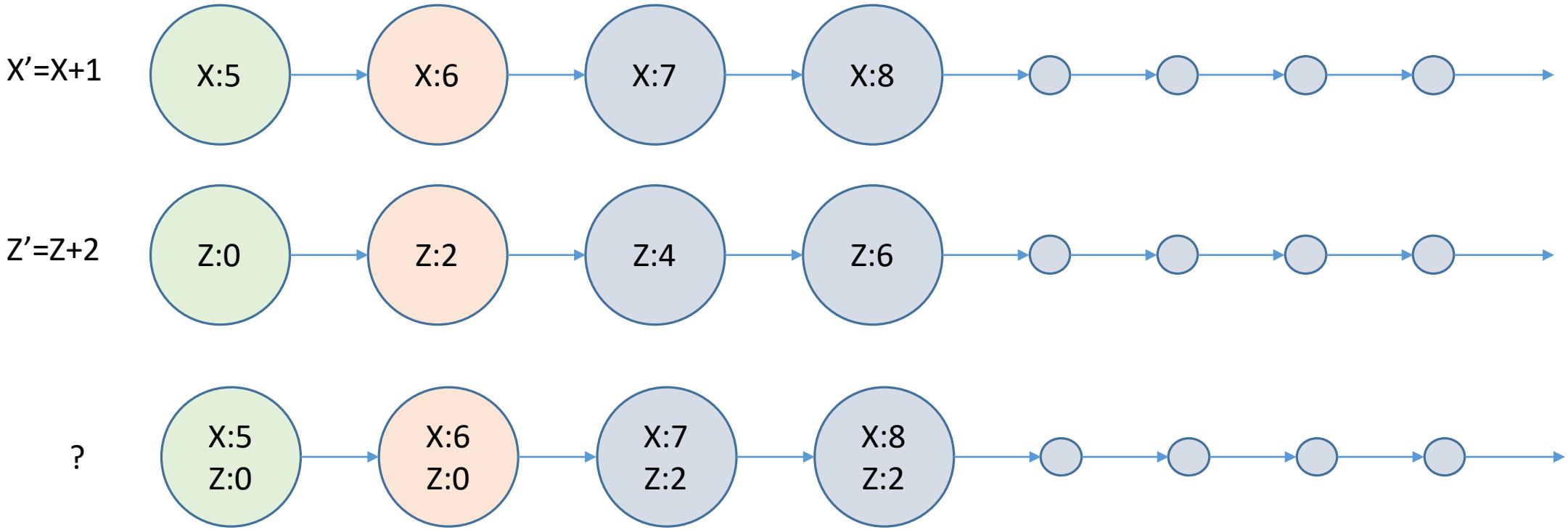
Основы TLA+

- Action (действие) – так называется логическое выражение, которое связывает два состояния, то есть включает переменные со штрихом.
- Exp’ – это выражение, где все переменные состояния заменены на эти же переменные, но только со штрихом. То есть значение выражения в следующем состоянии. Например: (x + y + 3)’ = x’ + y’ + 3
- [A]_exp – это сокращение для выражения A \/ (exp’ = exp), то есть либо action A истинно, либо exp сохранило своё значение. Обычно exp – это кортеж переменных, от которых зависит A.

TLA+, stuttering (заикание ☺)

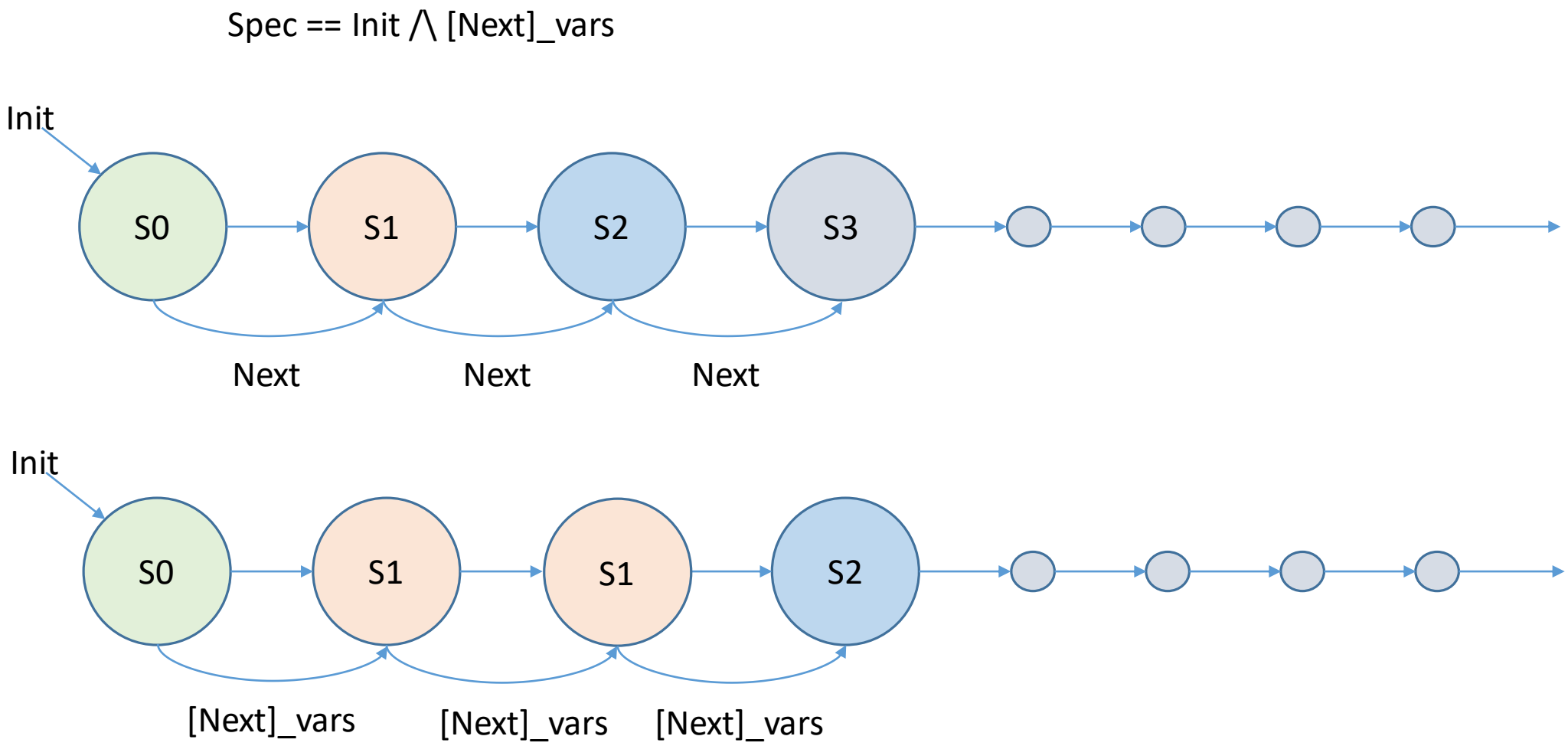
- Это повторение состояния.
- Спецификации должны быть инвариантны по отношению к stuttering
- Это нужно для того, чтобы можно было разрабатывать многокомпоненнтные (многомодульные) спецификации и легко их объединять в одну большую.

TLA+, stuttering



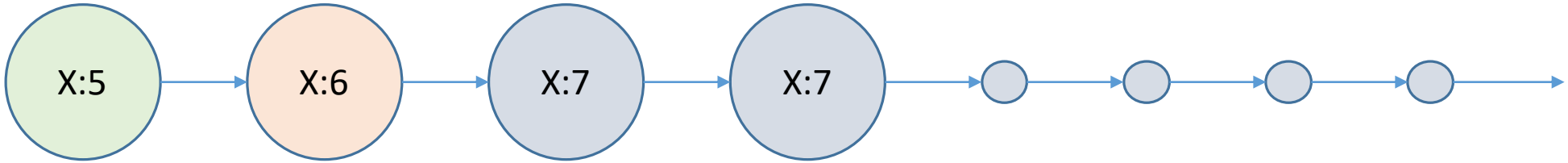
[X'=X+1]_<<X> /\ [Z'=Z+2]_<<Z>>

TLA+, типовая спецификация



TLA+, fairness

X=5 /\
[X'=X+1]_<<X>>



Хотим проверить: *<>(X > 10), когда-нибудь X будет больше 10*

И не получится, потому что наша спецификация допускает, что X может бесконечно застрять на 7, например.

Fairness: если событие может произойти, то оно обязательно произойдёт в будущем.

ENABLED(A) – action A может произойти, то есть, следующее состояние может быть таким, что A станет истинным

WF_vars(A) – условие слабой "честности", если A бесконечно долго разрешено (ENABLED), то оно обязательно произойдёт (бесконечно часто): <>[](ENABLED(<<A>>_vars)) => []<>(<<A>>_vars).
То есть weak fairness запрещает бесконечно длиннные цепочки stuttering, когда может быть сделан переход с изменением переменных состояния.

Переформулируем спецификацию:
Init == X=5
Next == X' = X + 1
vars == <<X>>
Spec == Init /\ [Next]_vars /\ WF_vars(Next)

Ограничения TLC

- Язык TLA+ весьма выразительный
- Далеко не всё, что написано на TLA+ можно промоделировать
- Спецификации должны быть основаны на описании конечных автоматов
- Основное, что нужно запомнить: переменная со штрихом должна входить в формулу только одна и с левой части равенства.
  То есть все выражения в спецификации, включающие переменные следующего состояния должны быть вида: X’ = Exp, где в Exp не должно быть X’ (и, желательно, других переменных со штрихом).

PlusCal

- Метаязык над TLA+
- Позволяет в более привычном виде описывать алгоритмы (очень похоже на обычные программы)
- Транслируется в TLA+
- Можно описывать как последовательные алгоритмы, так и параллельные в виде множества процессов
- Нужно быть внимательным к расстановке меток в процессах, так как метки задают атомарные блоки операций и при неправильной расстановке меток, алгоритм может не соответствовать реализации.

Пример алгоритма на PlusCal (c-диалект)

```
(* --algorithm AsyncInterface {

variables
  val \in 0..100,
  rdy \in 0..1,
  ack \in 0..1;

process (Send = "send")
{
  s00: while (TRUE) {
  s01:   await rdy = ack;
  s02:   val :=CHOOSE v \in 0..100: TRUE;
         rdy := 1 - rdy;
         print <<ack, rdy, val, "Send">>;
         assert (rdy # ack);
       }
};

process (Recv = "recv")
{
  r00: while (TRUE) {
  r01:   await rdy # ack;
  r02:   ack := 1 - ack;
       }
};

} \* end algorithm
*)
```

Трансляция в TLA+

```
\* BEGIN TRANSLATION
CONSTANT defaultInitValue
VARIABLES val, rdy, ack, pc

vars == << val, rdy, ack, pc>>

ProcSet == {"send"} \cup {"recv"}

Init == (* Global variables *)
        /\ val \in 0..100
        /\ rdy \in 0..1
        /\ ack \in 0..1
        /\ pc = [self \in ProcSet |-> CASE self = "send" -> "s00"
                                        [] self = "recv" -> "r00"]
```

Трансляция в TLA+

```
s00 ==
    /\ pc["send"] = "s00"
    /\ pc' = [pc EXCEPT !["send"] = "s01"]
    /\ UNCHANGED << val, rdy, ack >>

s01 ==
    /\ pc["send"] = "s01"
    /\ rdy = ack
    /\ pc' = [pc EXCEPT !["send"] = "s02"]
    /\ UNCHANGED << val, rdy, ack>>

s02 ==
    /\ pc["send"] = "s02"
    /\ val' = 44
    /\ rdy' = 1 - rdy
    /\ pc' = [pc EXCEPT !["send"] = "s00"]
    /\ UNCHANGED << ack>>

Send == s00 \/ s01 \/ s02
```

Трансляция в TLA+

```
r00 ==
    /\ pc["recv"] = "r00"
    /\ pc' = [pc EXCEPT !["recv"] = "r01"]
    /\ UNCHANGED << val, rdy, ack>>

r01 ==
    /\ pc["recv"] = "r01"
    /\ rdy # ack
    /\ pc' = [pc EXCEPT !["recv"] = "r02"]
    /\ UNCHANGED << val, rdy, ack>>

r02 ==
    /\ pc["recv"] = "r02“
    /\ ack' = 1 - ack
    /\ pc' = [pc EXCEPT !["recv"] = "r00"]
    /\ UNCHANGED << val, rdy>>

Recv == r00 \/ r01 \/ r02
```

Трансляция в TLA+

```
Next == Send \/ Recv

Spec == Init /\ [][Next]_vars

\* END TRANSLATION
```

Полезные ссылки

- Основная книга по TLA+: Specifying Systems
- Видеокурс по TLA+: ссылка
- Сайт Hillel Waine Learn TLA+
- Руководства по PlusCal:
  - pluscal.pdf
  - c-manual.pdf (C-подобный синтаксис)
  - p-manual.pdf (Паскаль-подобный синтаксис)
  - Проверка многопоточных алгоритмов с помощью PlusCal
- Примеры спецификаций: TLA+ Examples
- Список статей с примерами спецификаций: List of TLA+ examples
- Статьи Рона Преслера: Ron Pressler articles