

# Лекция 3

## модель структуры пула памяти на Alloy

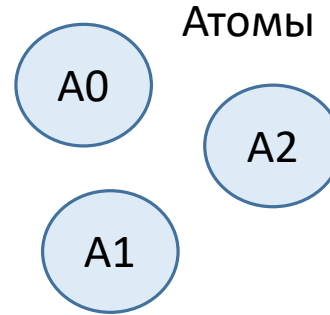
# Краткое введение в Alloy

# Сущности

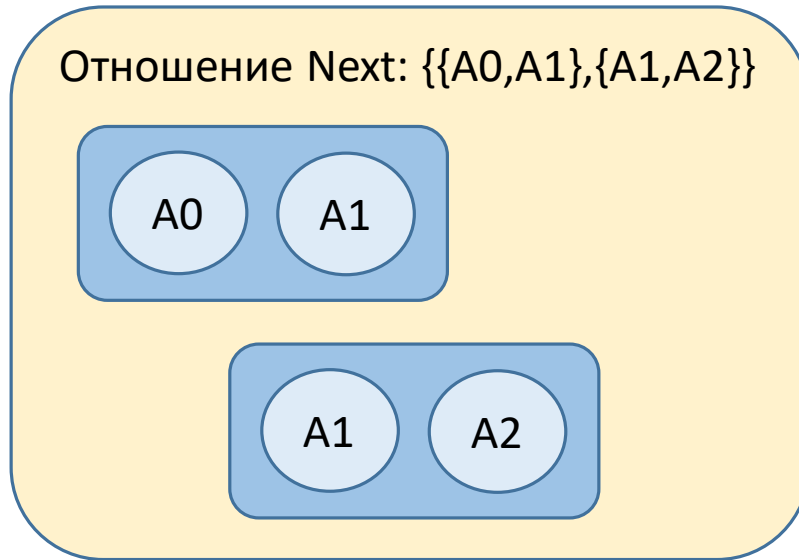
- Базовая сущность в Alloy – это отношение.
- Отношение – это множество кортежей из атомов, все кортежи одинаковой длины.
- Атом – это некий элемент из какой-либо сигнатуры
- Сигнатура – это множество элементов (которые логически как-то относятся друг к другу, похоже на множество элементов одного типа, как например, массивы в языках программирования)

# Примеры

Сигнатура A: {A0, A1, A2}



Отношение Next: {{A0,A1},{A1,A2}}



Мы не можем задавать элементы сигнатур.  
Мы можем только указывать в каких диапазонах должно находиться количество элементов.  
Alloy Analyzer сам перебирает все возможные множества элементов во всех сигнатурах и ищет модели и контр-примеры.

Сигнатура – частный случай отношения, состоящего из кортежей, в которых по одному атому.  
Поэтому имена сигнатур могут использоваться в выражениях наряду с именами отношений.

# Сигнатуры и отношения

Это пример объявления сигнатуры и отношения в Alloy

```
Sig A {  
  Next : lone A  
}
```

Отношения могут быть объявлены только в декларации сигнатуры.  
Это похоже на объявление классов и методов в языке Java, где каждый метод относится к какому-либо классу.

И первый атом в отношении будет той сигнатуры, к которой привязано отношение.  
То есть по факту Next – это бинарное отношение  $A \rightarrow A$ .

'lone' – less than or equal to **one** – это мультипликатор. Говорит о том, что для каждого уникального первого атома в отношении Next, должен быть только один или ноль уникальных вторых атомов. То есть, если в отношении уже есть {A0, A2}, то там не может быть {A0, A1} и других.

Перед самой сигнатурой тоже может быть мультипликатор

В данном случае, мы говорим, что в сигнатуре должен быть ровно один атом.

```
one Sig A {  
  Next : lone A  
}
```

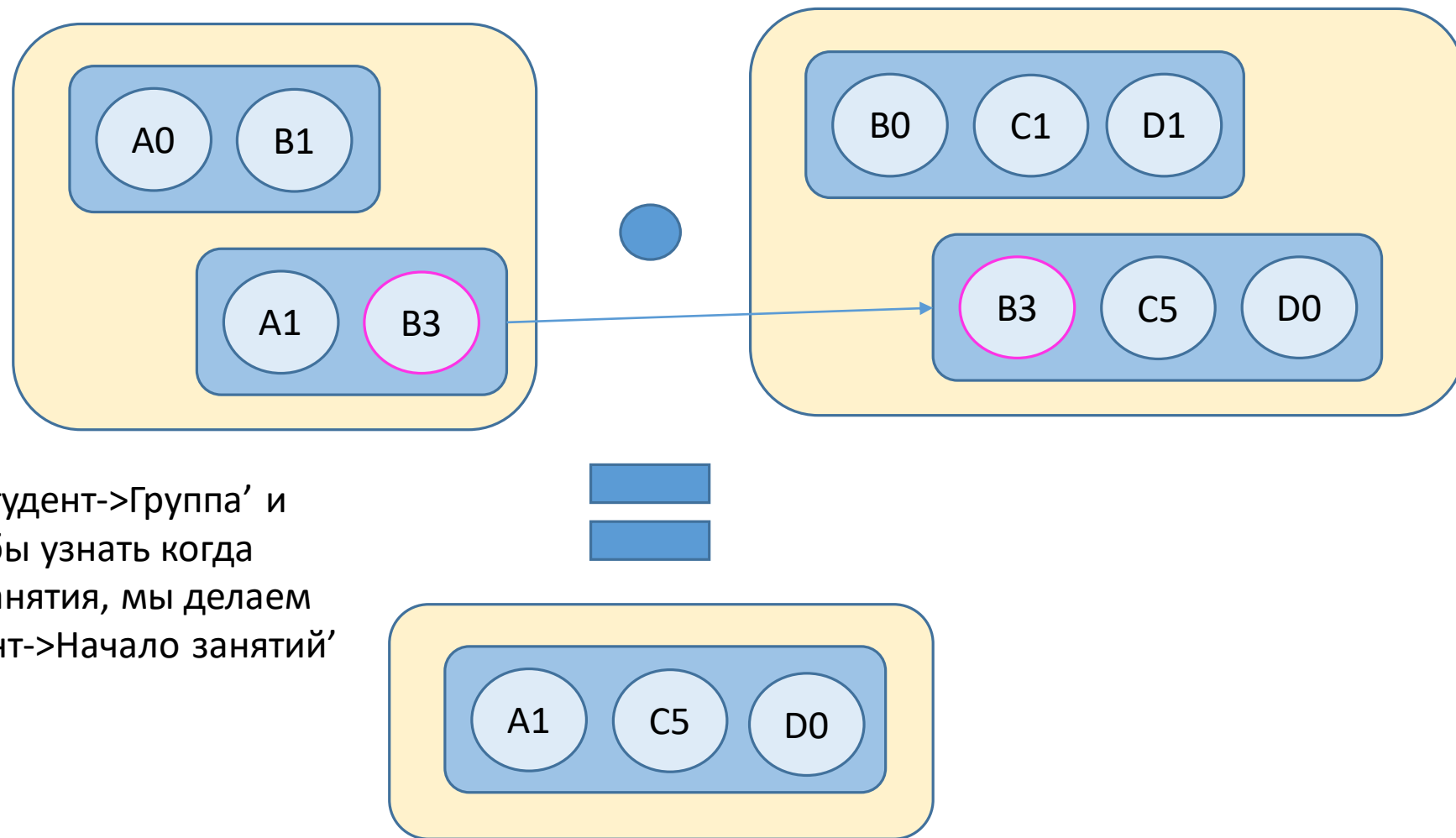
(Попробуйте посмотреть эти сигнатуры в Alloy Analyzer)

# Основные операции над отношениями

- Объединение “+”
  - Можно применять только к отношениям одинаковой арности (количество атомов в кортеже)
- Вычитание “-”
  - Тоже отношения должны быть одинаковой арности
- Join “.”
- Транзитивное замыкание бинарного отношения “^”
- Количество элементов/кортежей “#”
- Объединение с перезаписью по ключам “++”

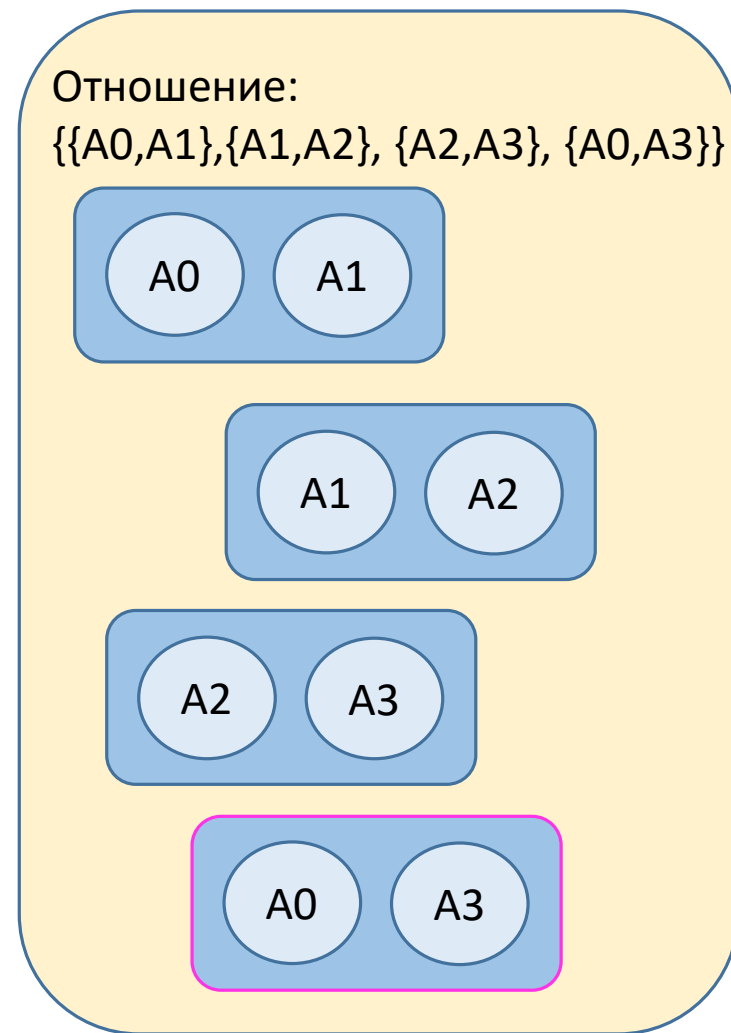
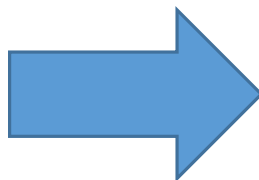
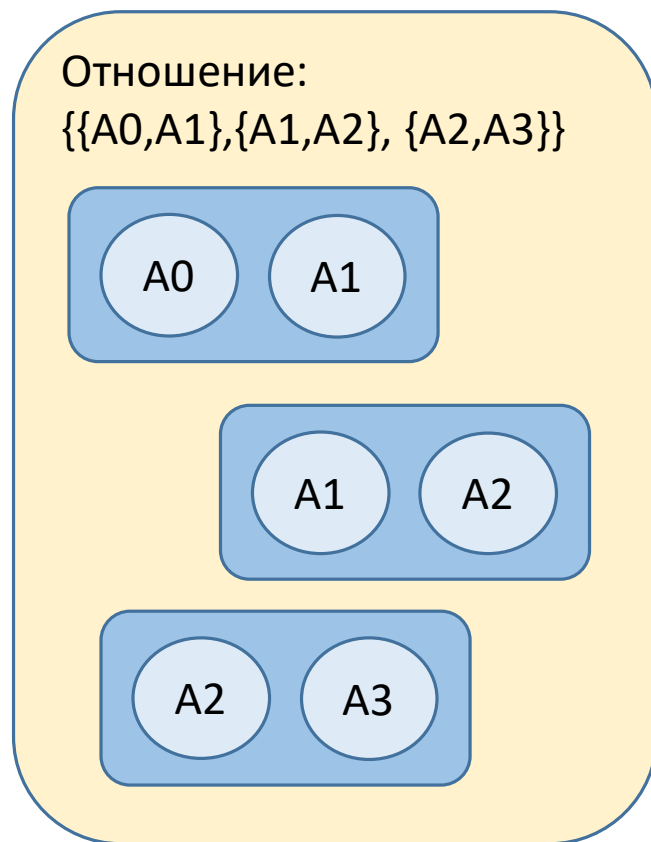
# Join “.”

Запись: A.B



Например, есть отношение A: 'Студент->Группа' и B: 'Группа->Начало занятий', чтобы узнать когда какому студенту приходиться на занятия, мы делаем новое отношение C = A.B: 'Студент->Начало занятий'

# Транзитивное замыкание “ $\wedge$ ”





# Предопределённые множества и отношения

- `univ` – множество всех атомов в модели
- `none` - пустое множество
- `iden` – бинарное отношение равенства для всех элементов из `univ`, это отношение содержит пару `{element, element}`
- Задание отношений:
  - `Signature1 -> Signature2 -> Signature3` – декартово произведение, то есть все триплеты с элементами из соответствующих сигнатур
  - `{set expr1} -> {set expr2}` – декартово произведение, пары из элементов соответствующих множеств, порождаемых выражениями
  - `{a:A, b:B | predicate[a,b]}` – предикативное задание отношения  $A \rightarrow B$ , в котором пары элементов `a` и `b` удовлетворяют предикатам

# Кванторы

- $\text{all } a:A \mid \text{predicate}$
- $\text{some } a:A \mid \text{predicate}$
- $\text{one } a:A \mid \text{predicate} == \text{some } a:A \mid (\text{predicate and } (\text{all } a1:A-a \mid \text{not predicate}))$
- $\text{no } a:A \mid \text{predicate} == \text{all } a:A \mid \text{not predicate}$
- $\text{lone } a:A \mid \text{predicate} == (\text{one } a:A \mid \text{predicate}) \text{ or } (\text{no } a:A \mid \text{predicate})$

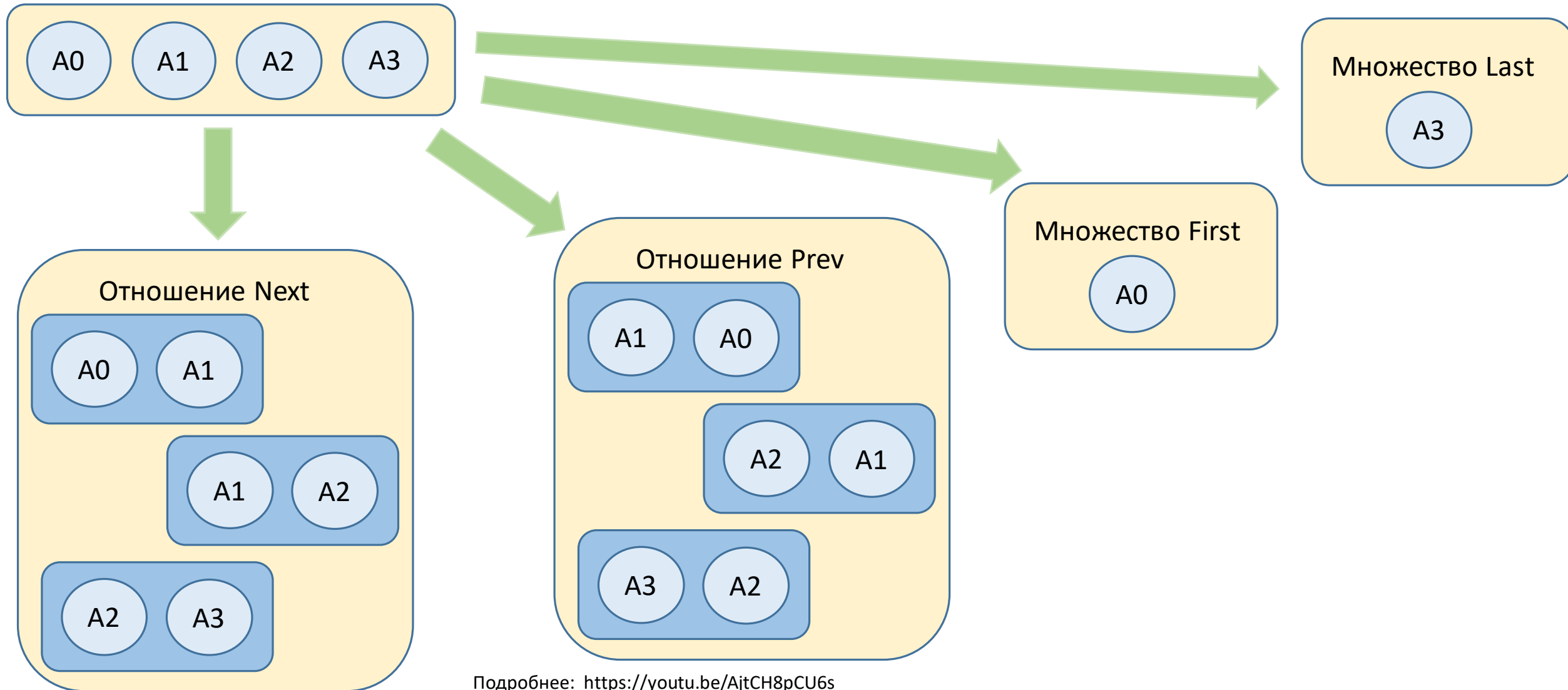
# Мультипликаторы

- Применяются при задании сигнатур: `one sig A {}`, `lone sig A {}`
- При задании отношений: `sig A {rel: one T}`, `sig A {rel: lone T}`, `sig A {rel: set T}`, `sig A {rel: some T}`
- Как предикаты над множествами: `no <set expr>`, `some <set expr>`, `lone <set expr>`, `one <set expr>`

# Модель структуры памяти и операций над ней

# Модуль order.als

Вводит строгий полный порядок на заданной сигнатуре



Подробнее: <https://youtu.be/AjtCH8pCU6s>

[https://github.com/vasil-sd/engineering-sw-hw-model-checking-lectures/blob/master/alloy\\_model/order.als](https://github.com/vasil-sd/engineering-sw-hw-model-checking-lectures/blob/master/alloy_model/order.als)

# Order

- Функции:

- fun next – возвращает следующий после данного элемент: item.next или next[item]
- fun prev - предыдущий
- fun first – первый элемент в порядке (минимальных)
- fun last – последний (максимальный)
- fun all\_greater – возвращает множество элементов, которые больше данного: item.all\_greater, all\_greater[item]
- fun all\_smaller – множество меньших заданного элемента
- fun minimum – вернуть минимальный элемент из данного множества
- fun maximum – максимальный

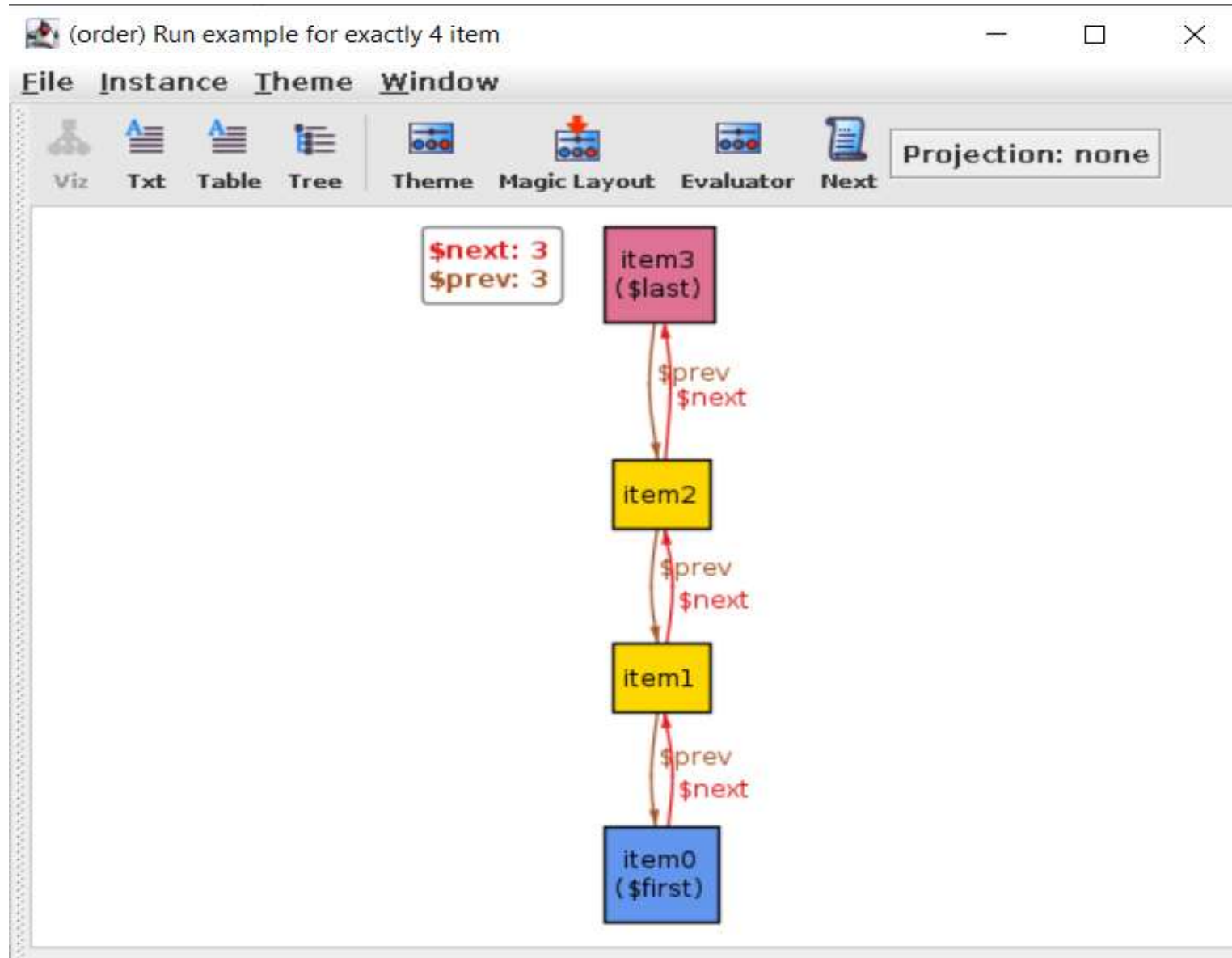
```
fun all_greater : item->item { ^this/next }
```

```
fun minimum(items : set item) : lone item { items - items.all_greater }
```

# Order

- Предикаты:
  - `pred less [lhs, rhs: item]`
  - `pred greater [lhs, rhs: item]`
  - `pred less_or_equal [lhs, rhs: item]`
  - `pred greater_or_equal [lhs, rhs: item]`

# Order





# Order

(order) Run example for exactly 4 item

File Instance Theme Window

Viz Txt Table Tree Evaluator Next

this/Order	First	Last	Next		Prev	
Order <sup>0</sup>	item <sup>0</sup>	item <sup>3</sup>	item <sup>0</sup>	item <sup>1</sup>	item <sup>1</sup>	item <sup>0</sup>
			item <sup>1</sup>	item <sup>2</sup>	item <sup>2</sup>	item <sup>1</sup>
			item <sup>2</sup>	item <sup>3</sup>	item <sup>3</sup>	item <sup>2</sup>

this/item	item <sup>0</sup>	item <sup>1</sup>	item <sup>2</sup>	item <sup>3</sup>
-----------	-------------------	-------------------	-------------------	-------------------

<sup>-1</sup>

# Моделирование размеров (size.als)

- Сигнатура Size
- Полный строгий порядок
- Задана сумма размеров,  $\text{Add: Size} \rightarrow \text{Size} \rightarrow \text{Size}$  – тернарное отношение: операнд, операнд, результат
- Дополнительные константы: zero, max
- Функция  $\text{Sum}[\text{LHS}, \text{RHS:Size}] : \text{Size}$  – для удобства и привычности записи
- Предикат  $\text{non\_zero}[S:\text{Size}]$  – для читаемости спецификаций
- Свойства моделей Size (сигнатура + отношения) очень близки к натуральным числам. Это необязательно было делать, но так просто привычнее и в просмотрщике моделей потом привычнее.
- Подробнее: [https://github.com/vasil-sd/engineering-sw-hw-model-checking-lectures/blob/master/alloy\\_model/size.als](https://github.com/vasil-sd/engineering-sw-hw-model-checking-lectures/blob/master/alloy_model/size.als)  
<https://youtu.be/COs4d7fsOfk>

# Size

Первый операнд

Второй операнд

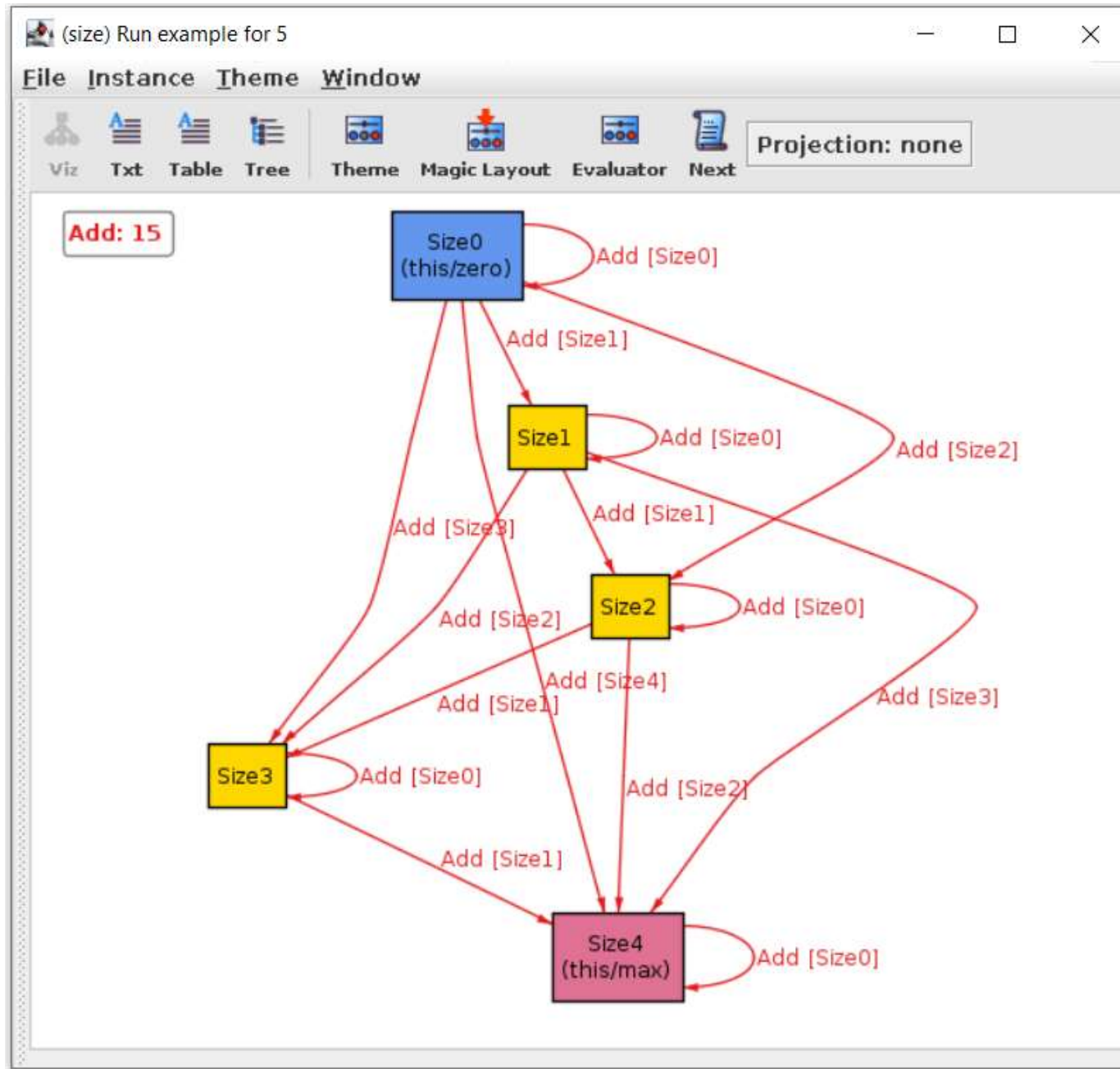
Результат

(size) Run example for 5

File Instance Theme Window		
Viz	Txt	Table Tree
Evaluator	Next	

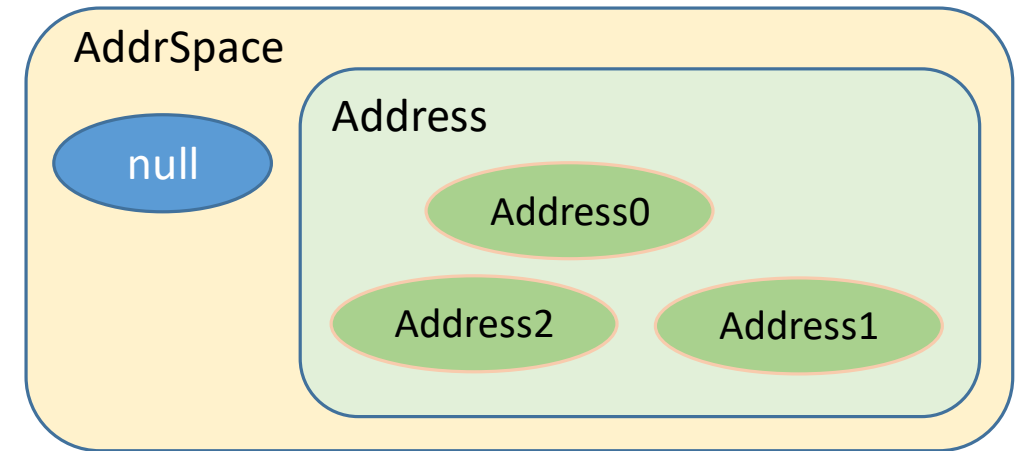
this/Size	Add	
Size <sup>0</sup>	Size <sup>0</sup>	Size <sup>0</sup>
	Size <sup>1</sup>	Size <sup>1</sup>
	Size <sup>2</sup>	Size <sup>2</sup>
	Size <sup>3</sup>	Size <sup>3</sup>
	Size <sup>4</sup>	Size <sup>4</sup>
Size <sup>1</sup>	Size <sup>0</sup>	Size <sup>1</sup>
	Size <sup>1</sup>	Size <sup>2</sup>
	Size <sup>2</sup>	Size <sup>3</sup>
	Size <sup>3</sup>	Size <sup>4</sup>
Size <sup>2</sup>	Size <sup>0</sup>	Size <sup>2</sup>
	Size <sup>1</sup>	Size <sup>3</sup>
	Size <sup>2</sup>	Size <sup>4</sup>
Size <sup>3</sup>	Size <sup>0</sup>	Size <sup>3</sup>
	Size <sup>1</sup>	Size <sup>4</sup>
Size <sup>4</sup>	Size <sup>0</sup>	Size <sup>4</sup>

# Size

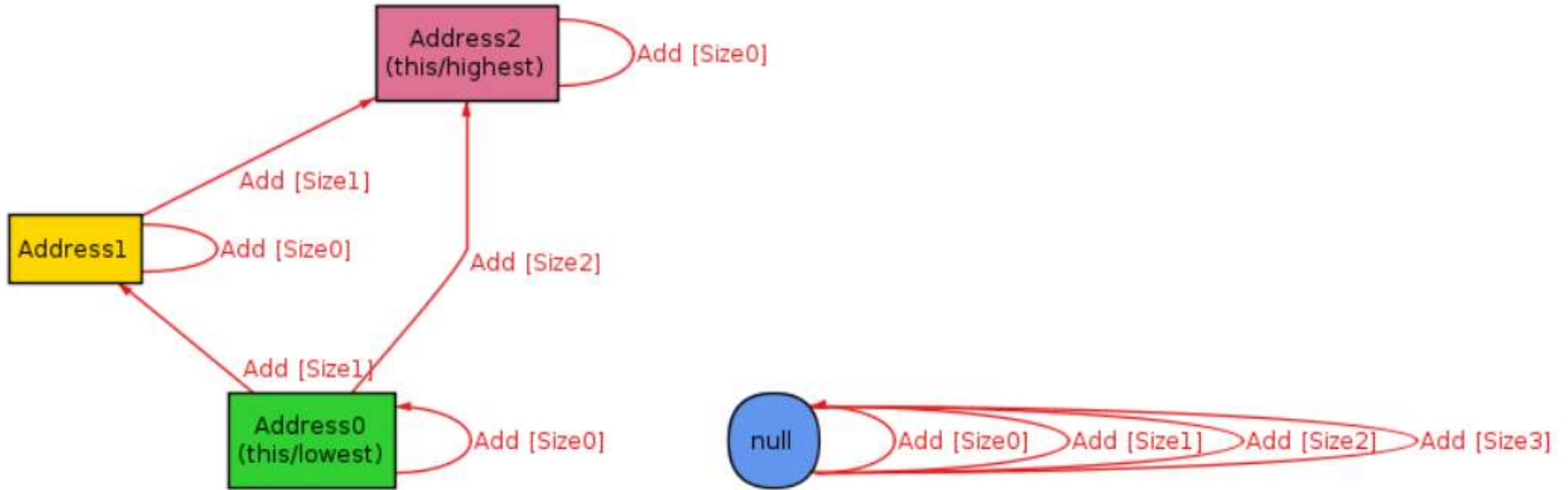


# Моделирование адресов (address.als)

- Сигнатура AddrSpace
- Выделенный адрес null
- Валидные адреса Address
- Отношение сложения адреса и размера,  $\text{Add: AddrSpace} \rightarrow \text{Size} \rightarrow \text{AddrSpace}$
- Сложения адресов нет
- Выделенные элементы lowest – самый младший адрес, highest – самый старший
- Функции:  $\text{Sum}[A:\text{AddrSpace}, S:\text{Size}]: \text{AddrSpace}$ ,  $\text{Distance}[\text{From}, \text{To}]: \text{Size}$
- Предикат  $\text{not\_null}[A:\text{AddrSpace}]$



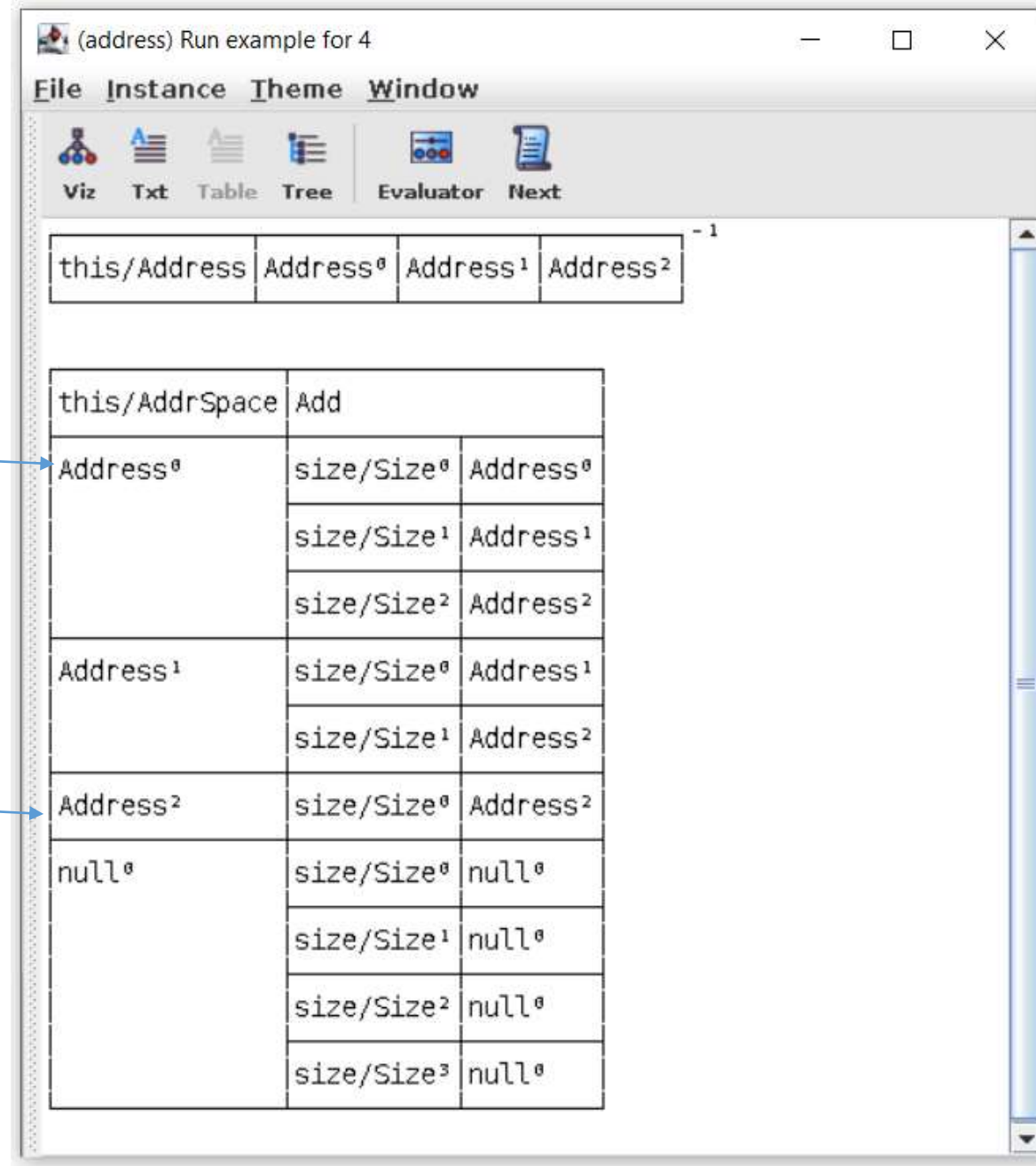
# Address



# Address

lowest

highest



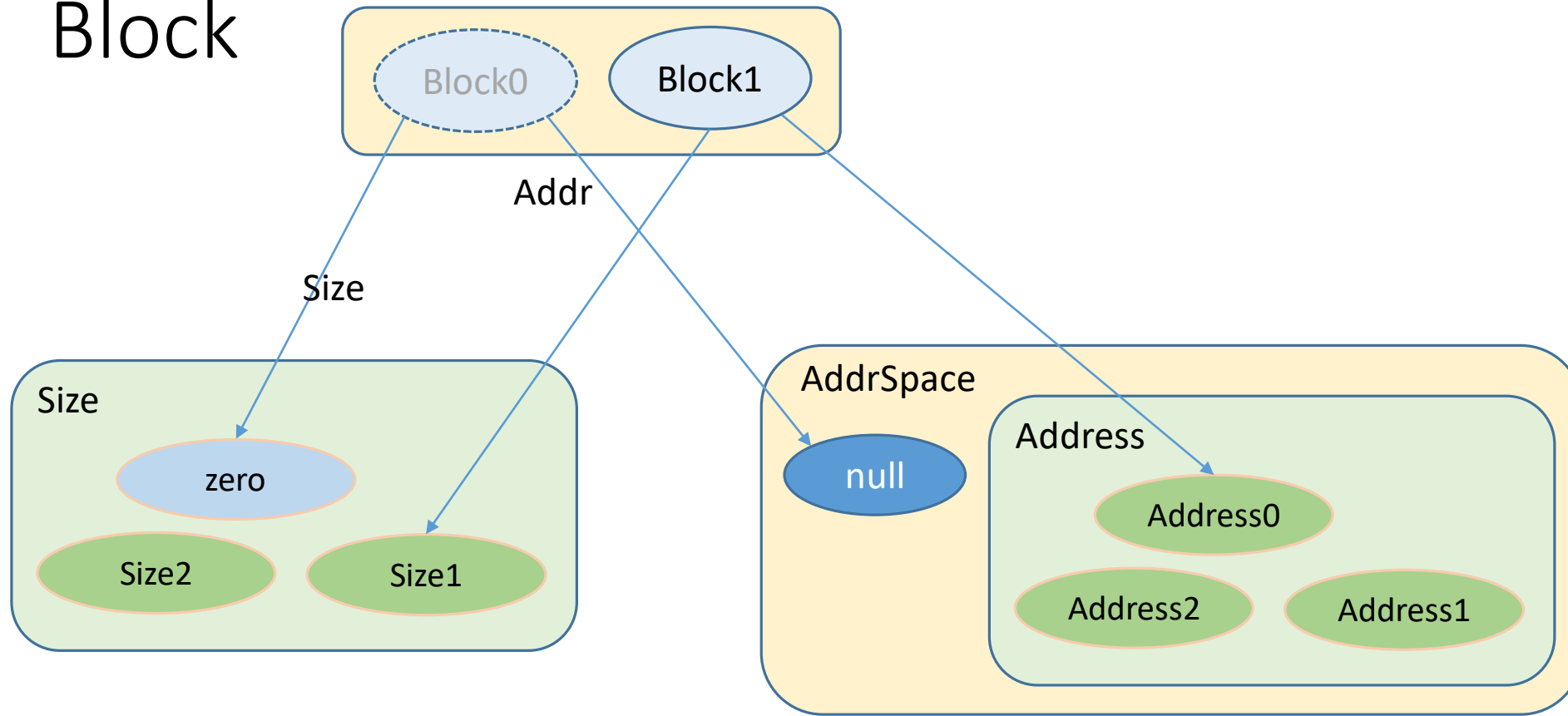
this/Address	Address <sup>0</sup>	Address <sup>1</sup>	Address <sup>2</sup>
this/AddrSpace	Add		
Address <sup>0</sup>	size/Size <sup>0</sup>	Address <sup>0</sup>	
	size/Size <sup>1</sup>	Address <sup>1</sup>	
	size/Size <sup>2</sup>	Address <sup>2</sup>	
Address <sup>1</sup>	size/Size <sup>0</sup>	Address <sup>1</sup>	
	size/Size <sup>1</sup>	Address <sup>2</sup>	
Address <sup>2</sup>	size/Size <sup>0</sup>	Address <sup>2</sup>	
null <sup>0</sup>	size/Size <sup>0</sup>	null <sup>0</sup>	
	size/Size <sup>1</sup>	null <sup>0</sup>	
	size/Size <sup>2</sup>	null <sup>0</sup>	
	size/Size <sup>3</sup>	null <sup>0</sup>	

# Моделирование блоков памяти

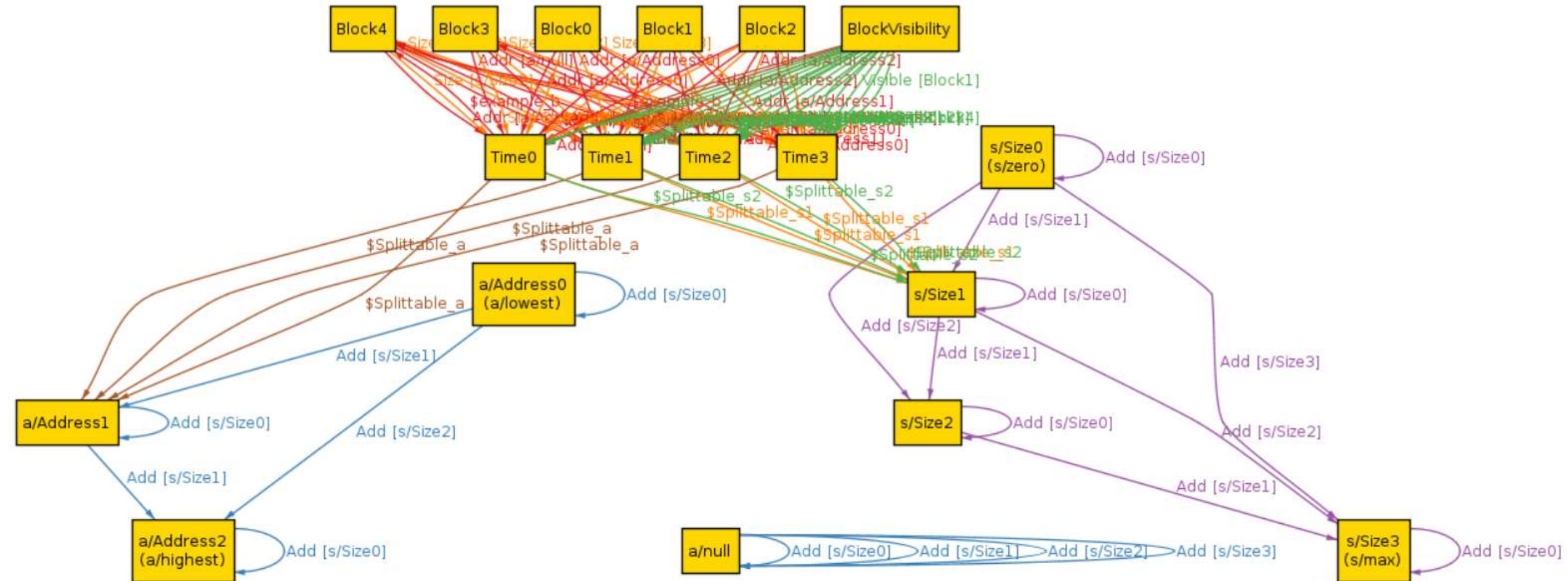
- Сигнатура Block
- Добавлена динамика через сигнатуру Time
- Отношения Addr: Block  $\rightarrow$  AddrSpace  $\rightarrow$  Time и Size: Block  $\rightarrow$  size/Size  $\rightarrow$  Time
- Видимые блоки – принимают участие в структуре памяти в заданный момент времени, невидимые – не принимают.
- Функции:
  - fun VisibleBlocks[T:Time] : set Block
  - fun InvisibleBlocks[T:Time]: set Block
  - fun NextBlockAddr[B: Block, T: Time] : AddrSpace
- Предикаты:
  - pred Visible[B: Block, T:Time]
  - pred Invisible[B: Block, T:Time]
  - pred InBlock[A:Address, B:Block, T:Time]
  - pred Splittable[B: Block, T: Time]
  - pred BlocksAreTheSameExcept[now: Time, Bs: set Block]



# Block



# Block

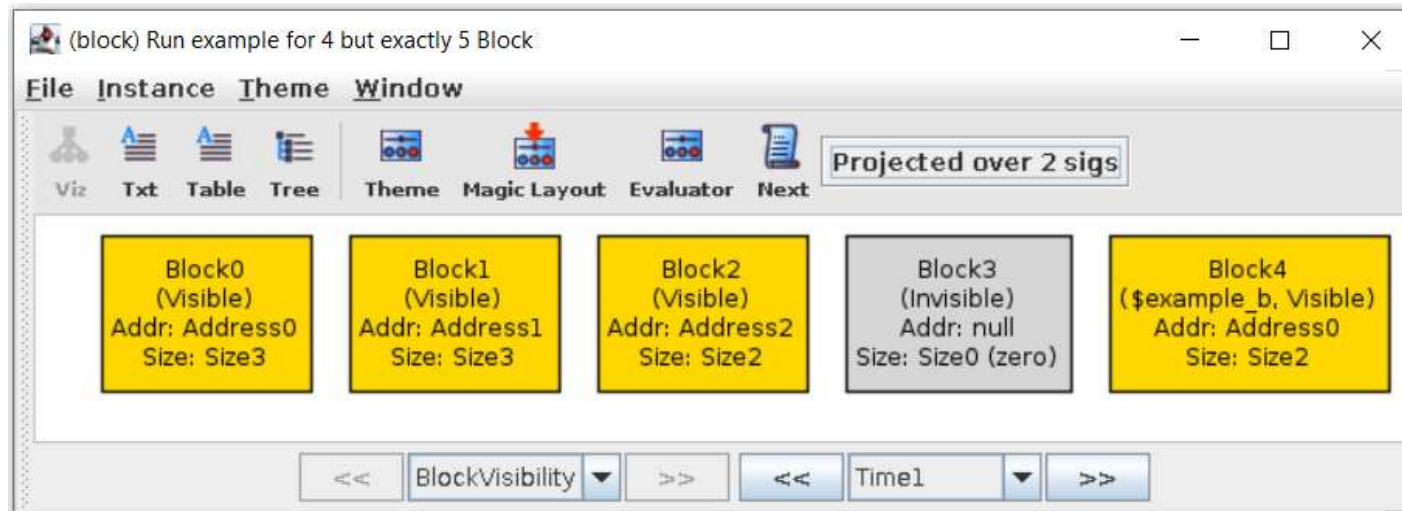
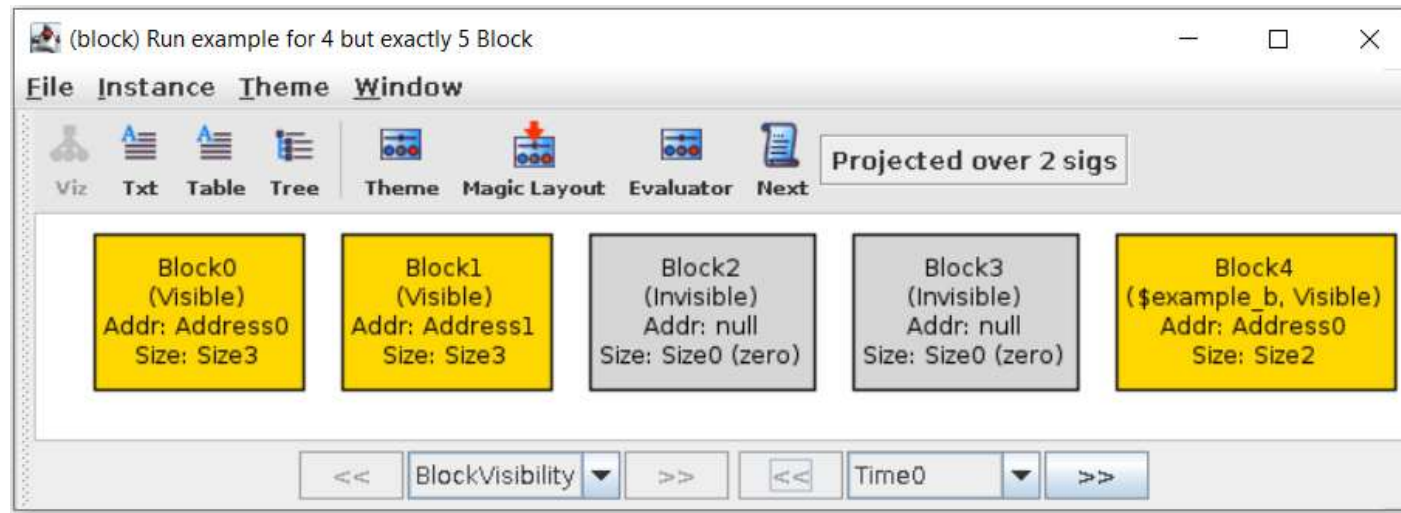


# Block



```
example: run {  
  all t: Time - first | one b : Block | t.BlocksAreTheSameExcept[b]  
} for 4 but exactly 5 Block
```

# Block



# Модель структуры памяти (memory.als)

- Классификация блоков:
  - Highest: Block  $\rightarrow$  Time – самый верхний, который упирается в верхнюю границу памяти
  - Lowest: Block  $\rightarrow$  Time – самый нижний, начинается с самого младшего адреса
  - Middle: Block  $\rightarrow$  Time – все, которые между ними
- Функции:
  - fun highest[T: Time] : set Block
  - fun lowest[T: Time] : set Block
  - fun middle[T: Time] : set Block
- Отношения соседства (сосед сверху, сосед снизу):
  - Above: Block  $\rightarrow$  Block  $\rightarrow$  Time
  - Below: Block  $\rightarrow$  Block  $\rightarrow$  Time
- Функции:
  - fun AllAbove[Bbelow: Block, T: Time] : set Block
  - fun AllBelow[Babove: Block, T: Time] : set Block
- Предикаты:
  - pred Neighbors[T: Time, Bs: set Block]

# Memory (основные свойства структуры)

Множество проверяемых блоков

Заданный момент времени

Для всех пар  
различных блоков  
из заданного  
множества

```
-- Все заданные блоки в заданный момент времени не перекрываются
pred NoOverlapping[Bs: set Block, T: Time] {
  all disj b1, b2: Bs
  | no a: Address
  | a.InBlock[b1, T] and a.InBlock[b2, T]
}
```

Нет адреса из множества  
всех валидных адресов

Который бы одновременно  
принадлежал адресам  
первого и второго блока

```
pred InBlock[A:Address, B:Block, T:Time] {
  greater_or_equal[A, B.Addr.T]
  less[A, B.NextBlockAddr[T]]
}
```

# Memory (основные свойства структуры)

Множество проверяемых блоков

Заданный момент времени

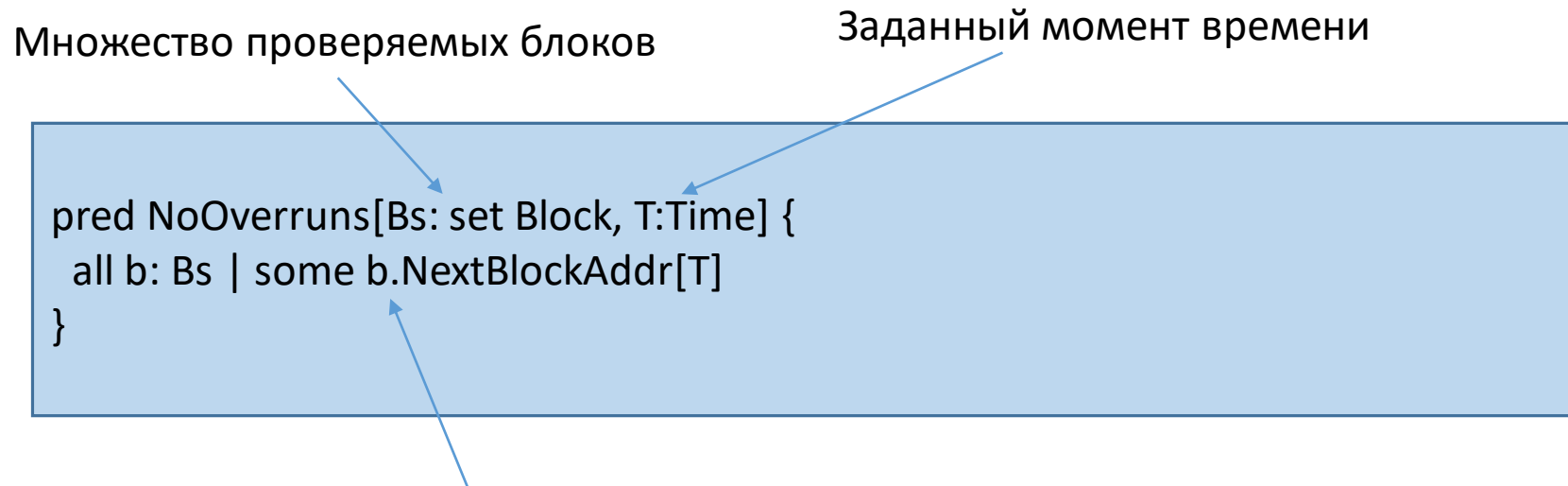
Должен быть  
самый верхний  
и  
самый нижний

```
pred NoHoles[Bs: set Block, T: Time] {  
  highest[T] in Bs  
  lowest[T] in Bs  
  all b: Bs & middle[T] | one b.Above[T] and one b.Below[T]  
  #highest[T].Below[T] = #lowest[T].Above[T]  
}
```

Для всех  
посерединке  
должно быть ровно  
по одному соседу  
сверху и снизу

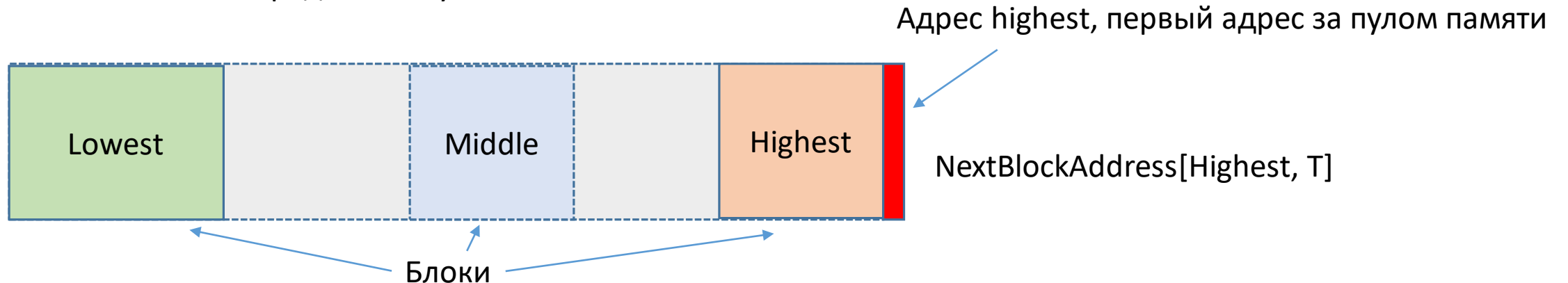
Это для обработки случая, когда у нас только единственный блок,  
размером со всю область памяти, подробнее в видео об операции  
join (отладка модели)

# Memory (основные свойства структуры)



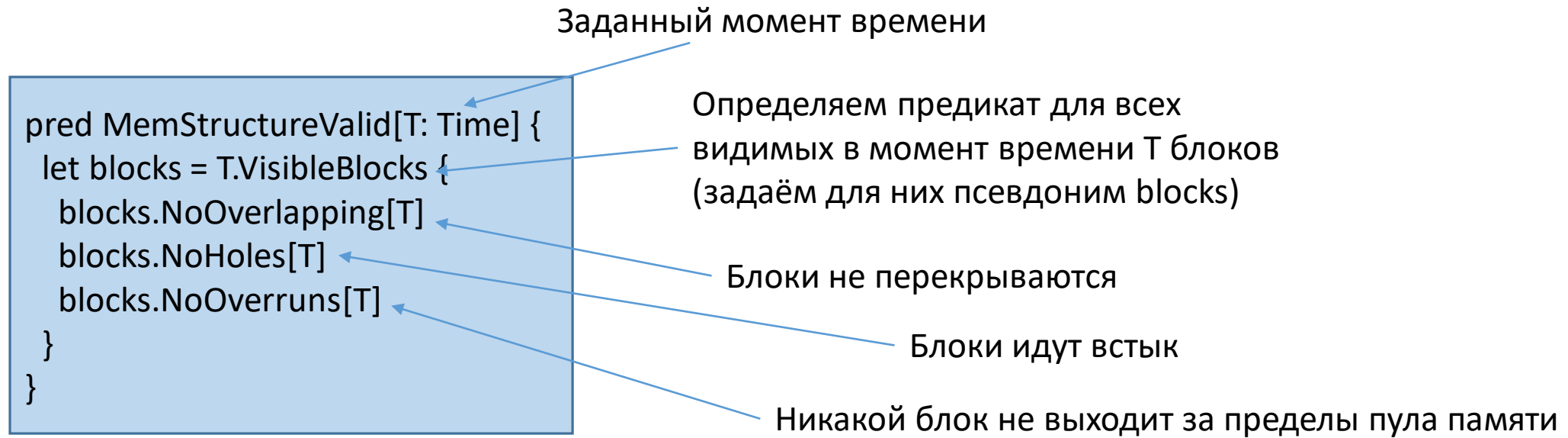
У каждого блока есть следующий за ним адрес памяти.

Таким образом у нас наибольший адрес, считается первым адресом за пределами пула памяти.





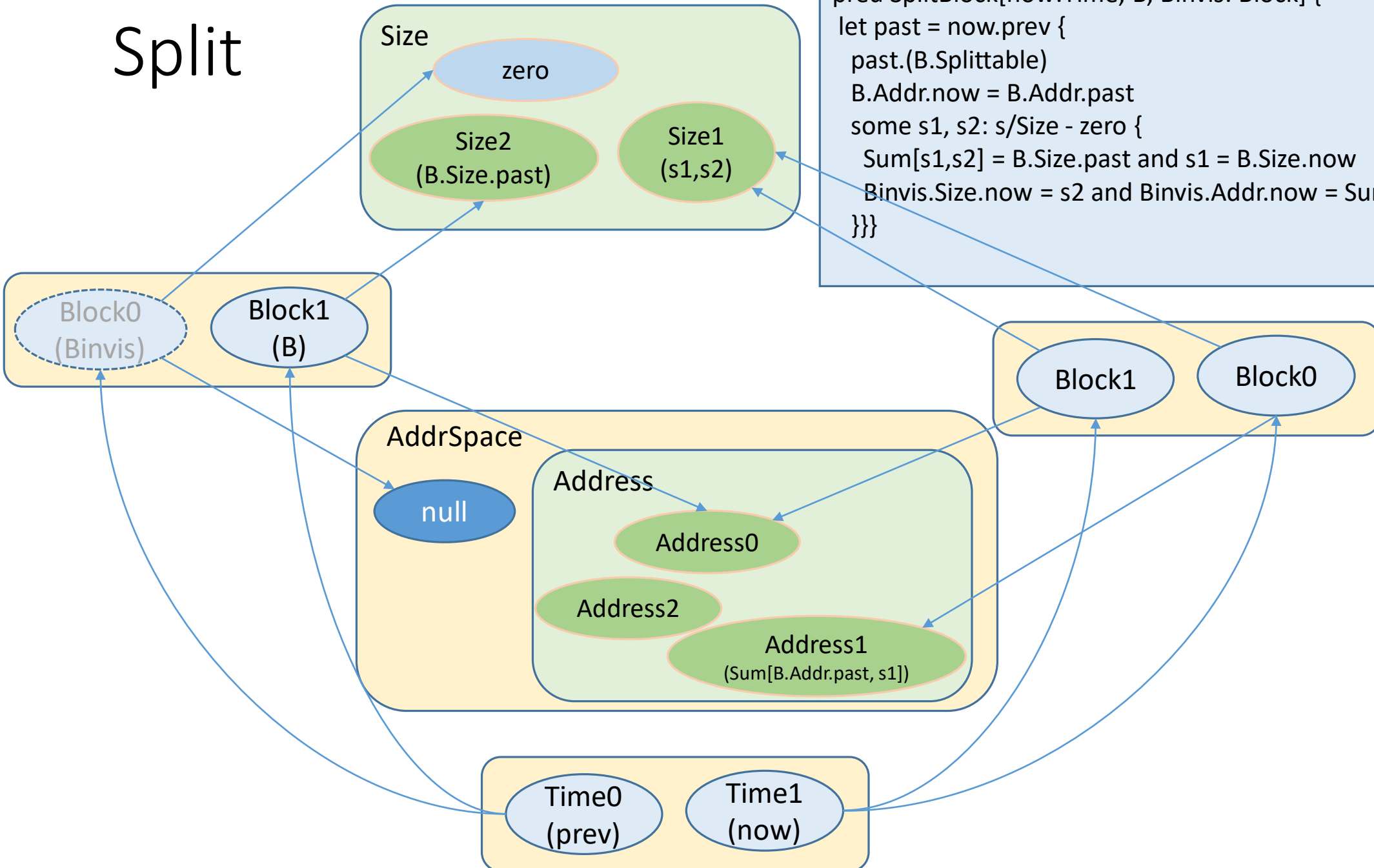
# Memory, инвариант корректности



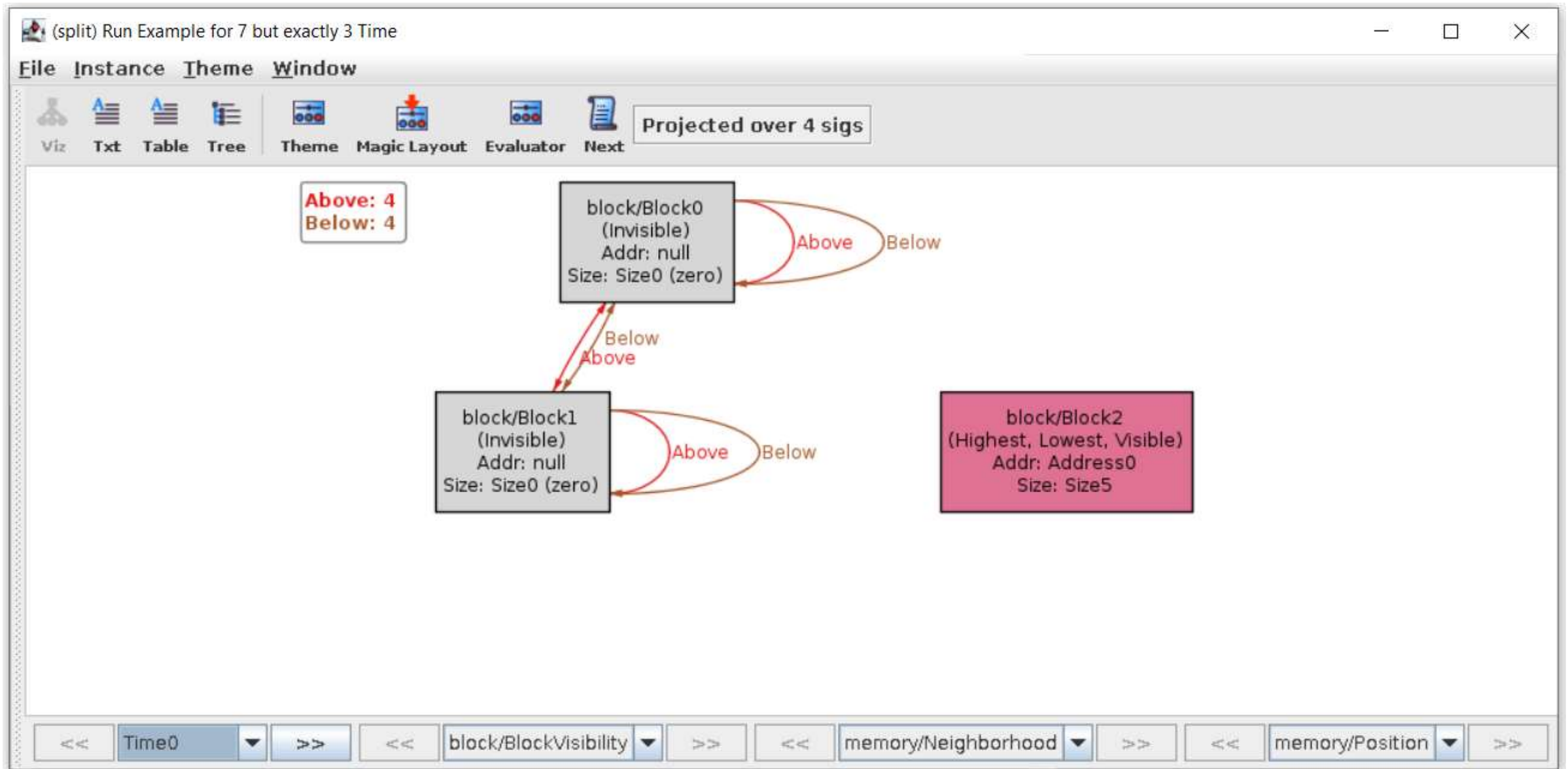
# Операция Split (split.als)

- Моделируем операцию разделения блока памяти на два при выполнении (malloc)
- Предикат: `pred SplitBlock[now:Time, B, Binvis: Block]`
- Проверка:
  - `assert SplitIsCorrectlyDefined`
  - `CheckSplit: check SplitIsCorrectlyDefined for 9 but exactly 3 Time`
- Подробнее: [https://github.com/vasil-sd/engineering-sw-hw-model-checking-lectures/blob/master/alloy\\_model/join.als](https://github.com/vasil-sd/engineering-sw-hw-model-checking-lectures/blob/master/alloy_model/join.als)  
[https://youtu.be/t\\_ho\\_HMeym0](https://youtu.be/t_ho_HMeym0)

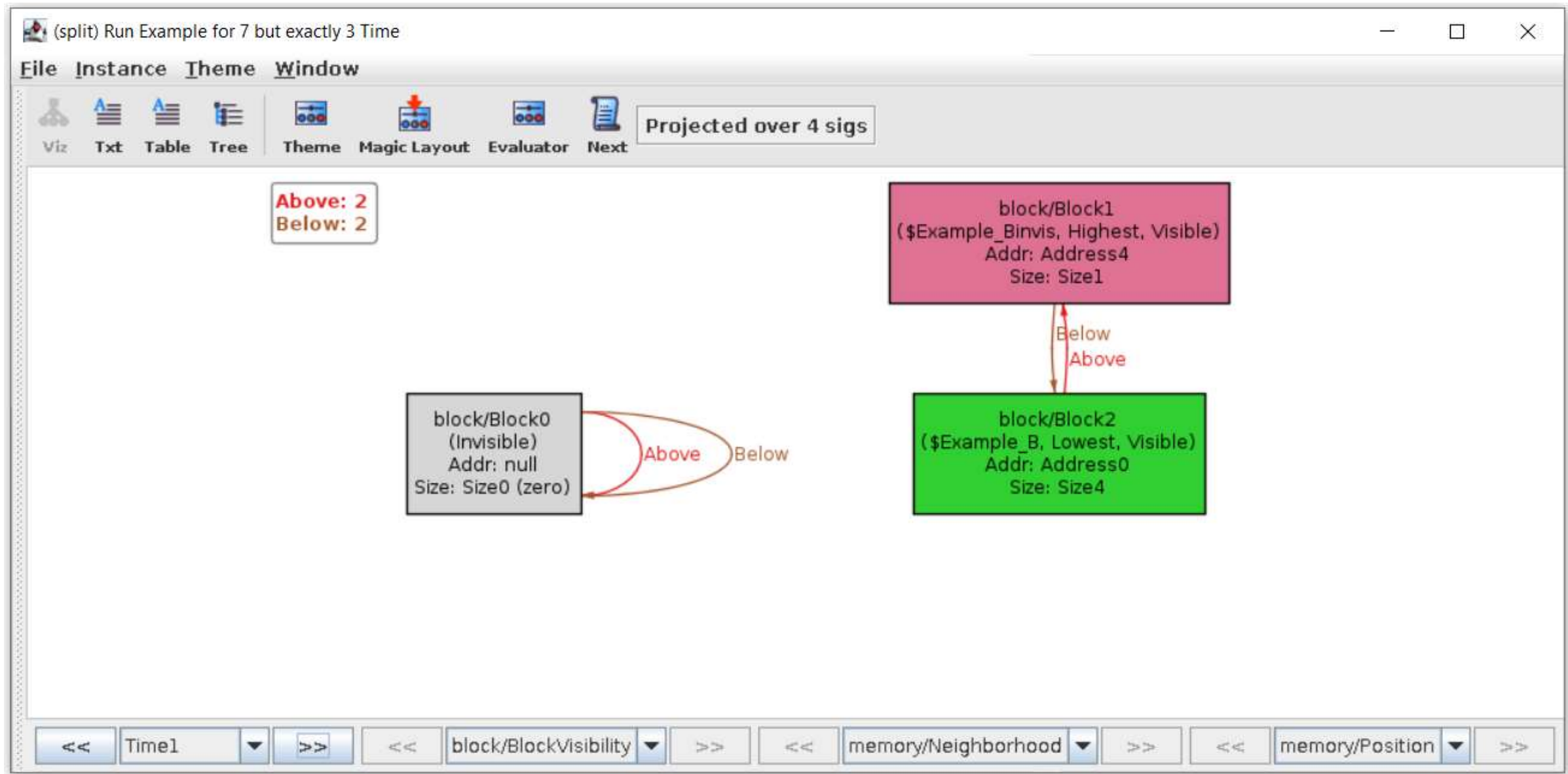
# Split



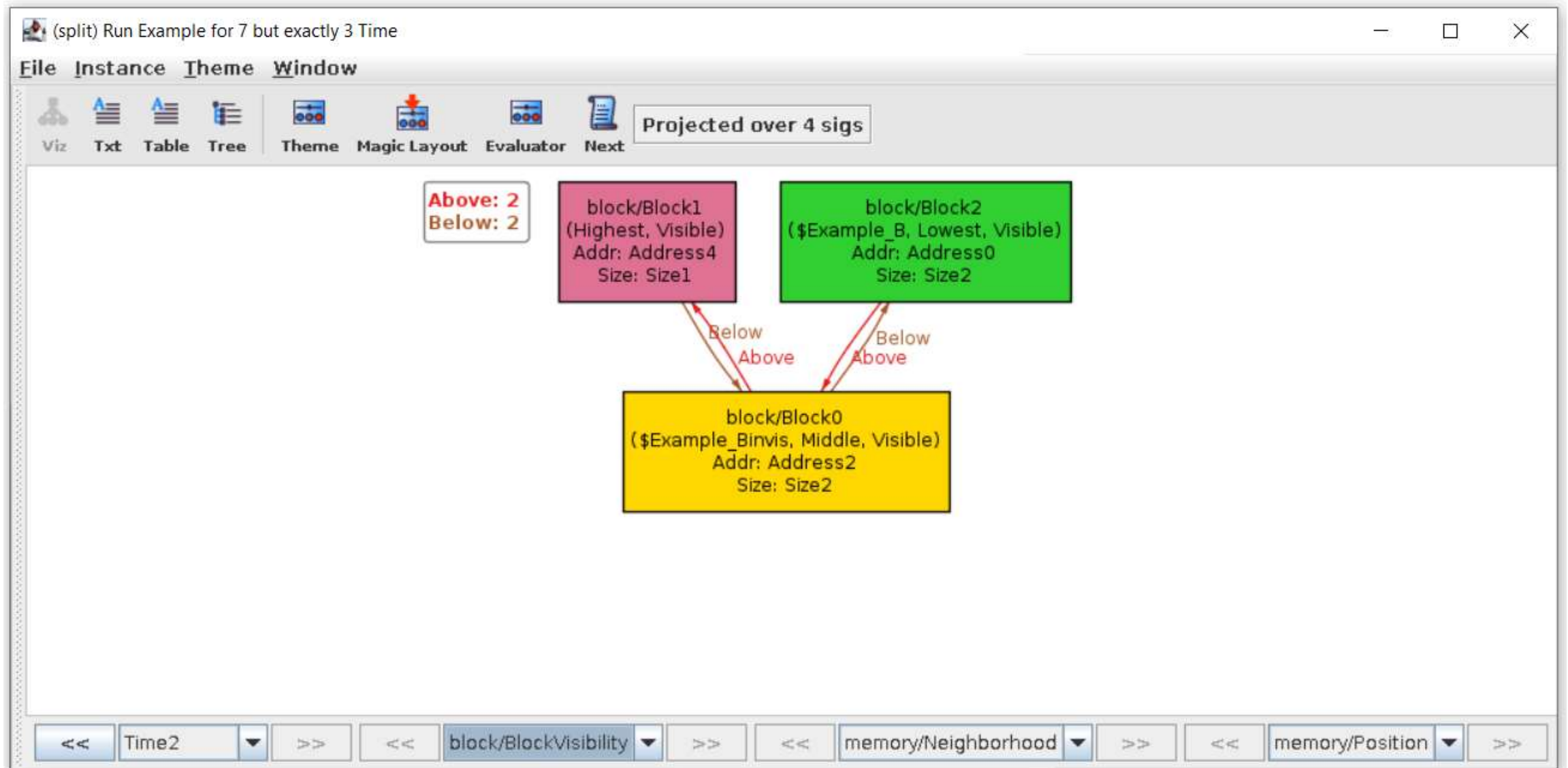
# Split



# Split



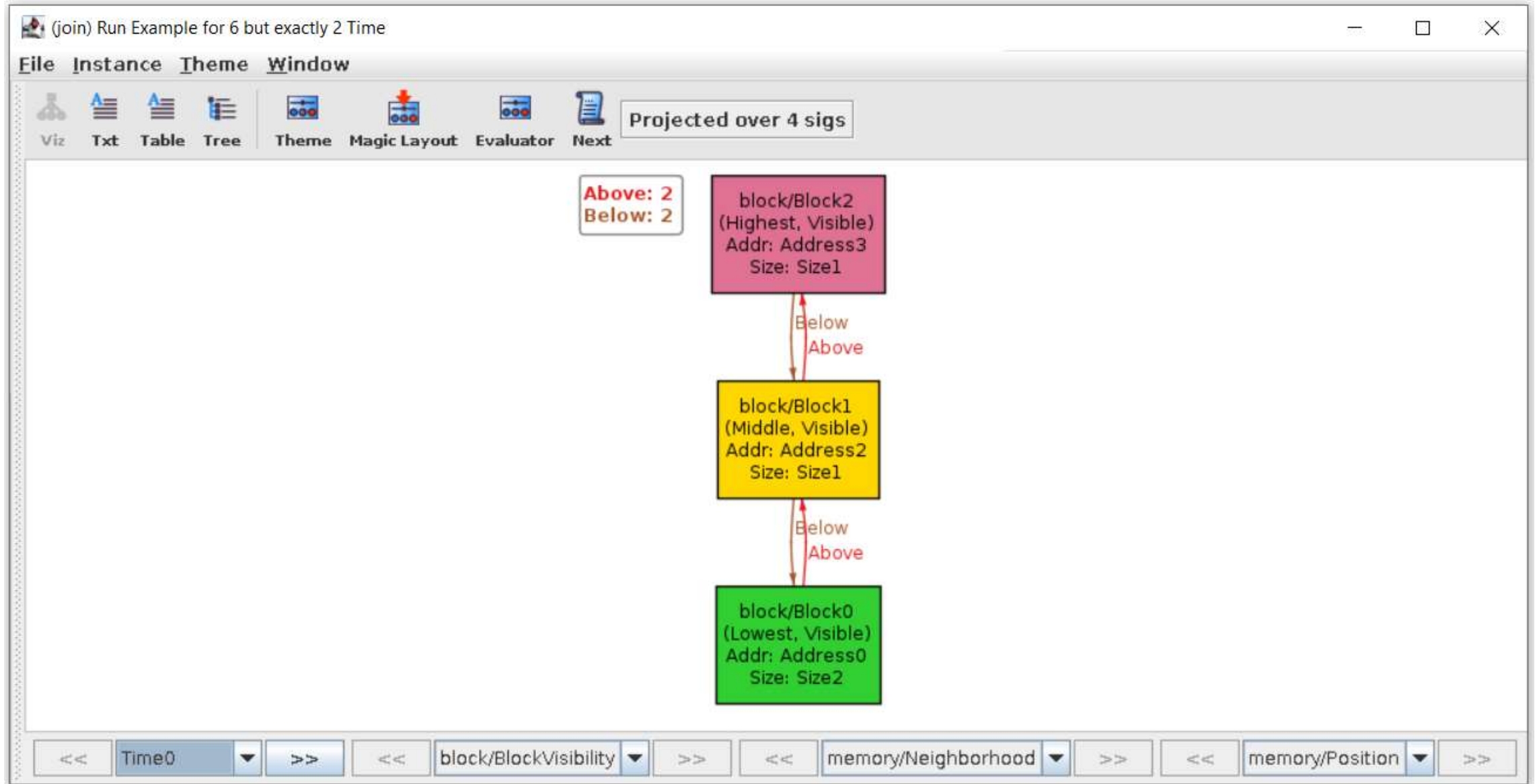
# Split



# Операция Join

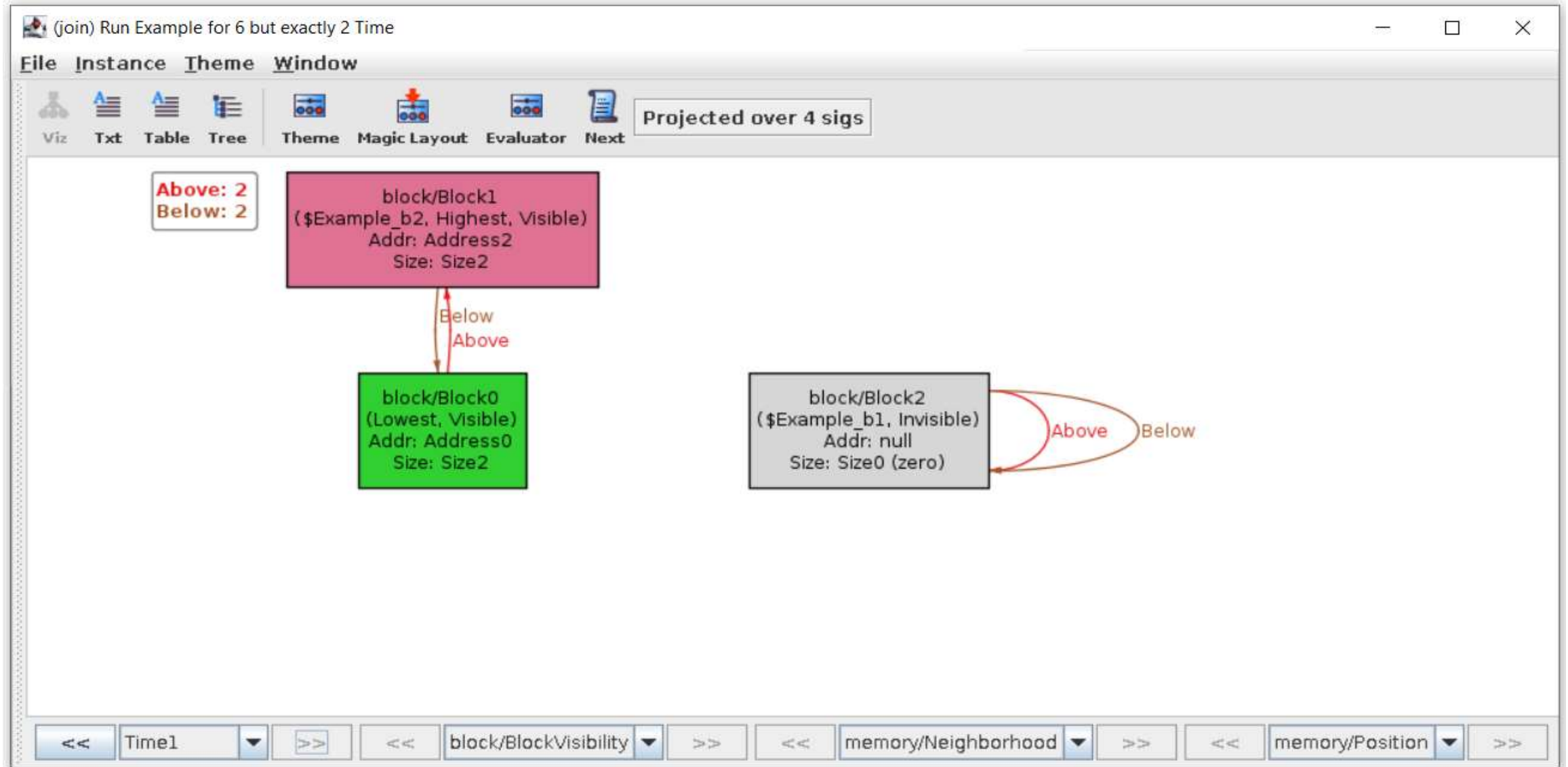
- Моделируем операцию объединения двух свободных блоков при освобождении блока памяти (free)
- Предикаты:
  - `pred UpdateBlocks[now: Time, Bbelow, Babove: Block]` – предикат объединения блоков, блоки должны быть переданы в правильном порядке
  - `pred JoinBlocks[now: Time, B1, B2: Block]` – это предикат-обёртка для переупорядочивания блоков
- Проверка корректности:
  - `assert JoinIsCorrectlyDefined`
  - `CheckJoin: check JoinIsCorrectlyDefined for 7 but exactly 2 Time`
- Подробнее: [https://github.com/vasil-sd/engineering-sw-hw-model-checking-letures/blob/master/alloy\\_model/join.als](https://github.com/vasil-sd/engineering-sw-hw-model-checking-letures/blob/master/alloy_model/join.als)  
[https://youtu.be/zf\\_oiDN63VY](https://youtu.be/zf_oiDN63VY)

# Join





# Join



# ССЫЛКИ

- Исходники модели и видео лежат тут: <https://github.com/vasil-sd/engineering-sw-hw-model-checking-lectures>
- Тут много примеров моделей на Alloy:  
<https://github.com/AlloyTools/models>
- Статьи Hillel Wayne про Alloy:  
<https://www.hillelwayne.com/tags/alloy/>