Software Engineering Conference Russia
November 14-15, 2019. Saint-Petersburg

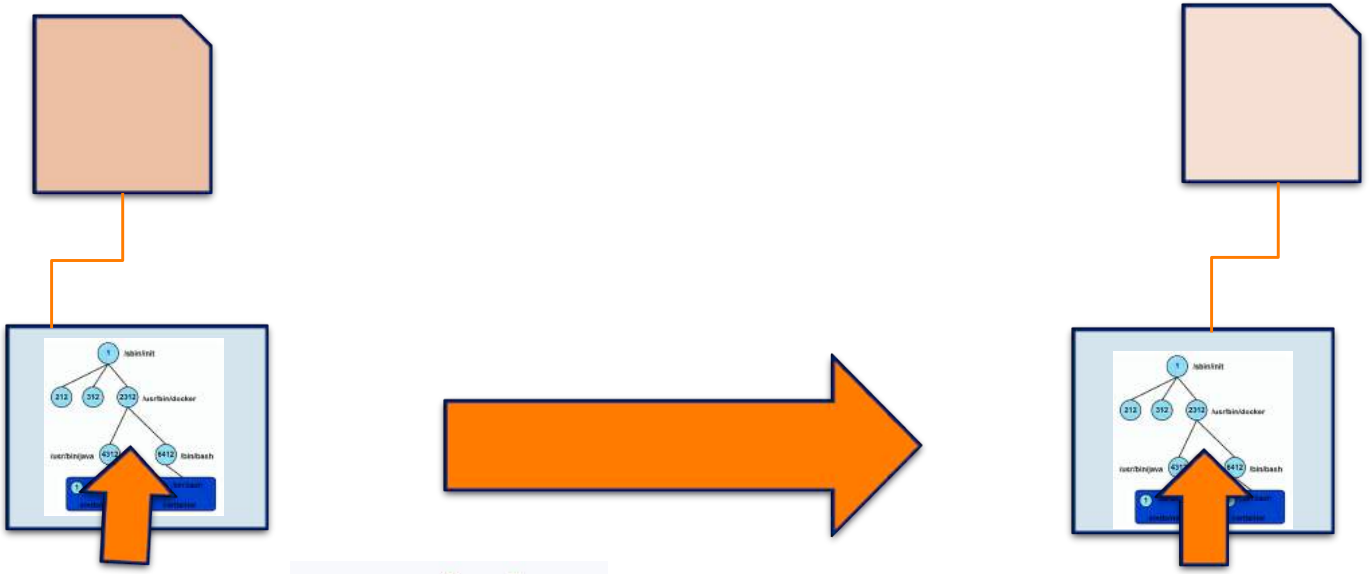
Обобщённая теория восстановления деревьев процессов: подводные камни checkpoint/restore

Николай Ефанов
МФТИ(ГУ)

Checkpoint-Restore

Host 1

Host 2



сохранить

возобновить

CRIU

BERKELEY LAB

Intel Parallel Amplifier

Intel Parallel Inspector

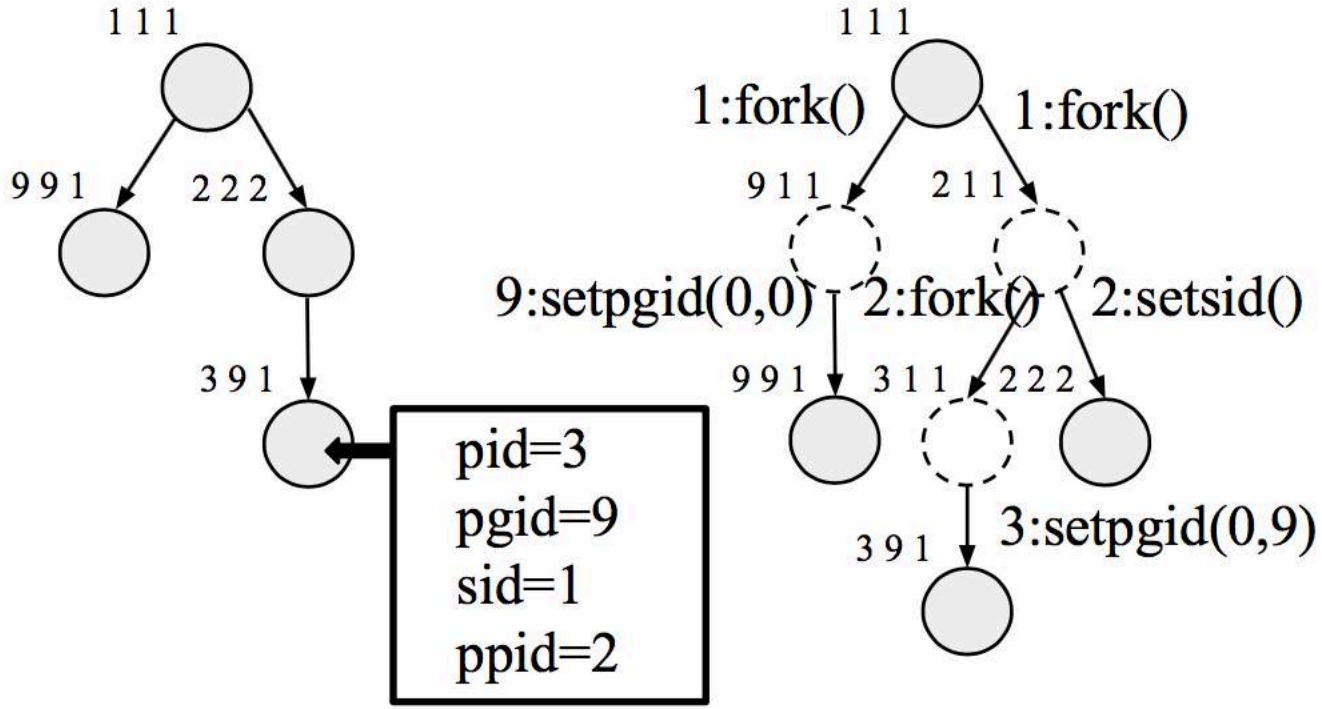
OpenVZ

DMTCP

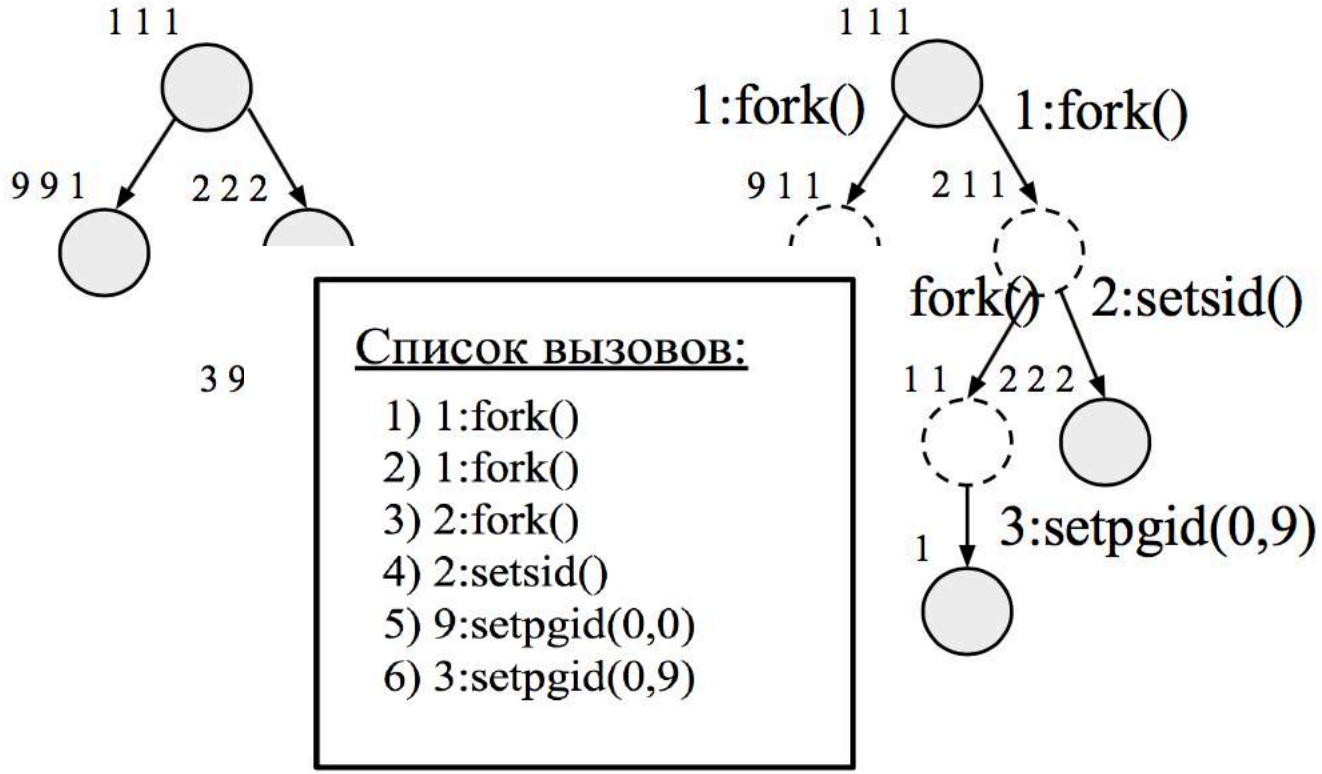
Применение

1. Живая миграция
2. Технологии контейнерной виртуализации
3. Восстановление после сбоев
4. Отложенная отладка
5. Обновления ядра ОС «на лету»
6. Ускорение загрузки приложений
7. Пауза в играх ☺

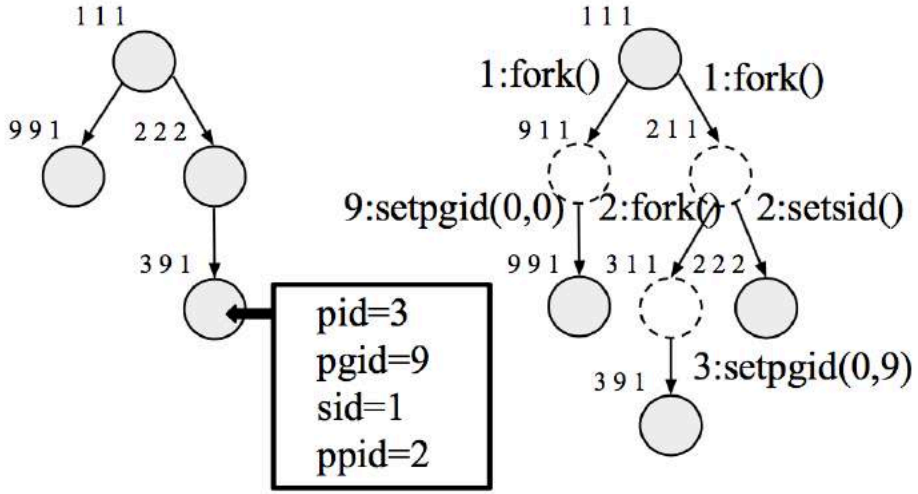
Подзадача: восстановить дерево процессов:



Подзадача: восстановить дерево процессов…решение:



Подзадача: восстановить дерево процессов…



Список вызовов:

1) 1:fork()
2) 1:fork()
3) 2:fork()
4) 2:setsid()
5) 9:setpgid(0,0)
6) 3:setpgid(0,9)

1. Профилировка

   \+ простота и истинность измерений

   \- накладные расходы и непрозрачность

2. Эвристики реконструкции

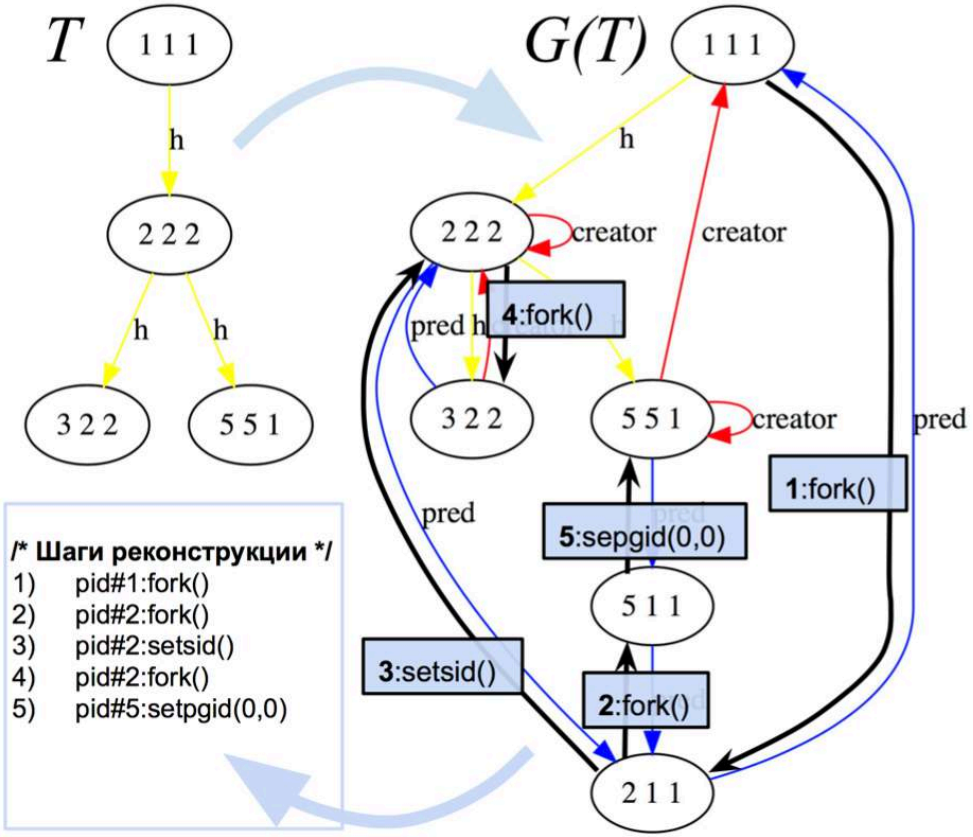
   -неполное покрытие

3. Математическое моделирование

   +должно работать

   -а какой сложности?

Формальные модели восстановления: Алгоритм AGTTM

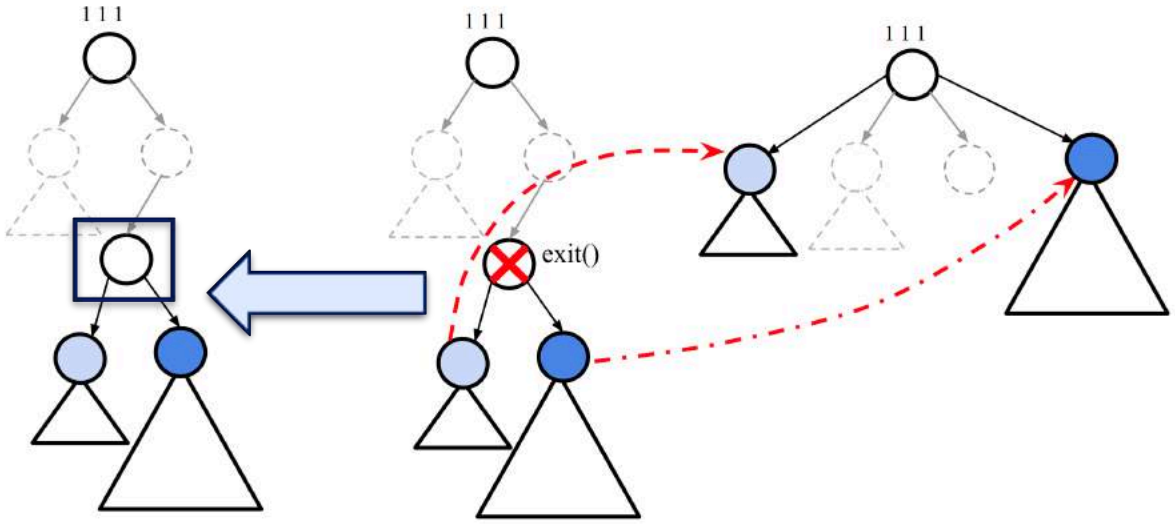


T → G(T) → ребёрное покрытие T\{root} → список команд

Формальные модели восстановления: Алгоритм AGTTM

1. **Препроцессинг:**

   Восстановить завершённые процессы, порядок в поддеревьях (обратить все «обратные reparent'ы»)



2. **Обработка**

3. **Пост-обработка**

Формальные модели восстановления: Алгоритм AGTTM

1. **Препроцессинг:**

   Восстановить завершённые процессы

2. **Обработка:** построить граф *G(T)* на базе анализа *T*

```
1  function AGTTM (IR,T,expr);
   Input  : IR,T,expr
   Output: D
2  for any Node in dfs(T.root) do
3  |  cond = f(expr(Node.attr)) // проверка текущей вершины и предка
4  |  if cond not in IR then
5  |  |  // проверка жестко наследуемых атрибутов - сессий, пространств имён
6  |  |  cond = upbranch(...)
7  |  |  if not cond in IR then
8  |  |  |  // проверка выставляемых в поддереве атрибутов - групп процессов и др.
9  |  |  |  cond = dfs(expr(Node, Current))
10 |  |  end
11 |  end
12 |  D = TR(D, Ex, IR, k(cond));
13 end
14 return D;
```

3. **Пост-обработка**

Формальные модели восстановления: Алгоритм AGTTM

1. **Препроцессинг:**

   Восстановить завершённые процессы

2. **Обработка:**

```
1  function AGTTM (IR,T,expr);
   Input  : IR,T,expr
   Output: D
2  for any Node in dfs(T.root) do
3      cond = f(expr(Node.attr)) // проверка текущей вершины и предка
4      if cond not in IR then
5          // проверка жестко наследуемых атрибутов - сессий, пространств имён
6          cond = upbranch(...)
7          if not cond in IR then
8              // проверка выставляемых в поддереве атрибутов - групп процессов и др.
9              cond = dfs(expr(Node, Current))
10         end
11     end
12     D = TR(D, Ex, IR, k(cond));
13 end
14 return D;
```

3. **Пост-обработка**

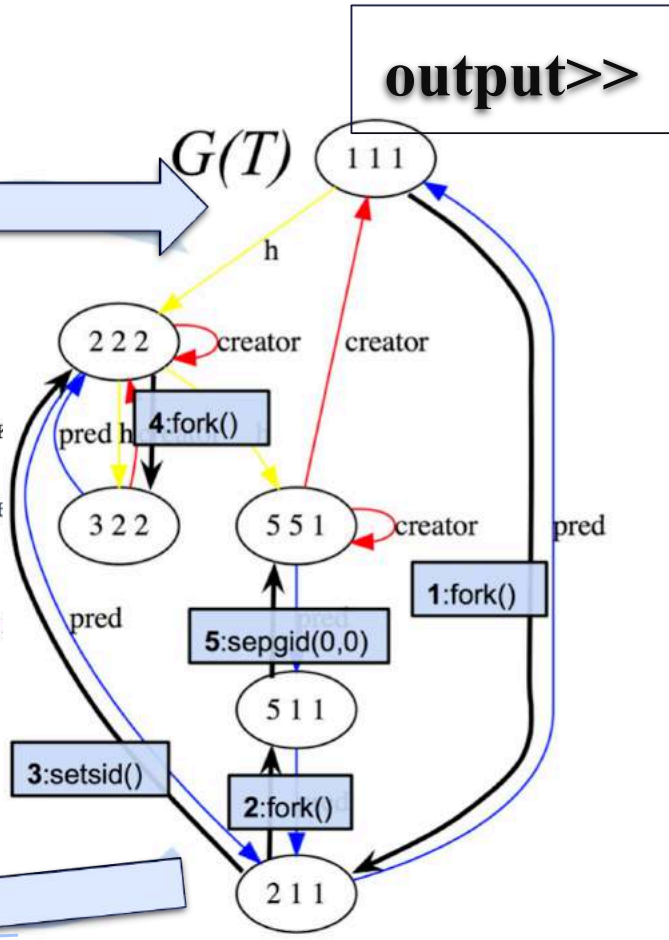
Формальные модели восстановления: Алгоритм AGTTM

1. **Препроцессинг:**

   Восстановить завершённые процессы

2. **Обработка:**

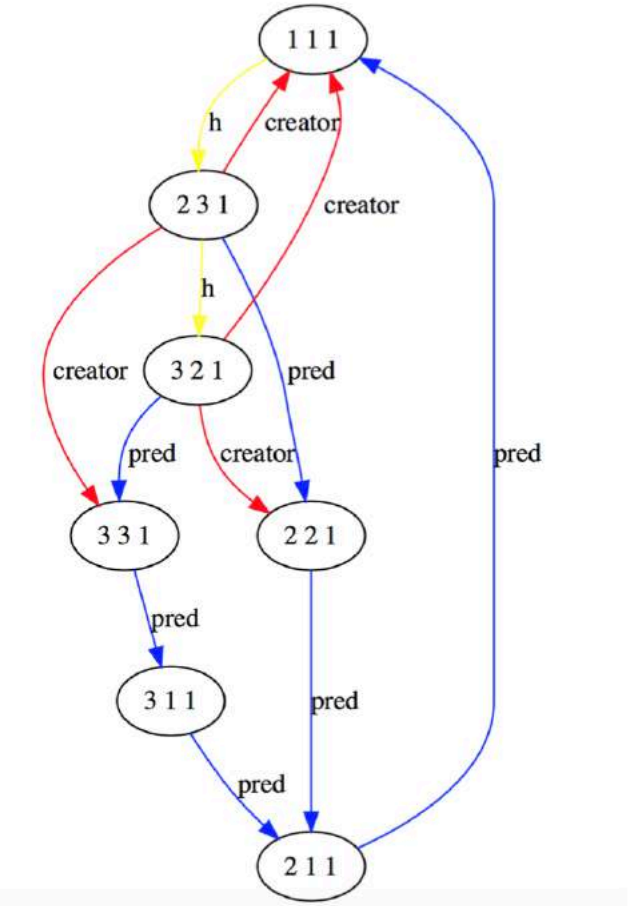
```
1  function AGTTM (IR,T,expr);
   Input  : IR,T,expr
   Output: D
2  for any Node in dfs(T.root) do
3      cond = f(expr(Node.attr)) // проверка текущей вершины и преди
4      if cond not in IR then
5          // проверка жестко наследуемых атрибутов - сессий, простра
6          cond = upbranch(...)
7          if not cond in IR then
8              // проверка выставляемых в поддереве атрибутов - групп
9              cond = dfs(expr(Node, Current))
10         end
11     end
12     D = TR(D, Ex, IR, k(cond));
13 end
14 return D;
```



3. **Пост-обработка**

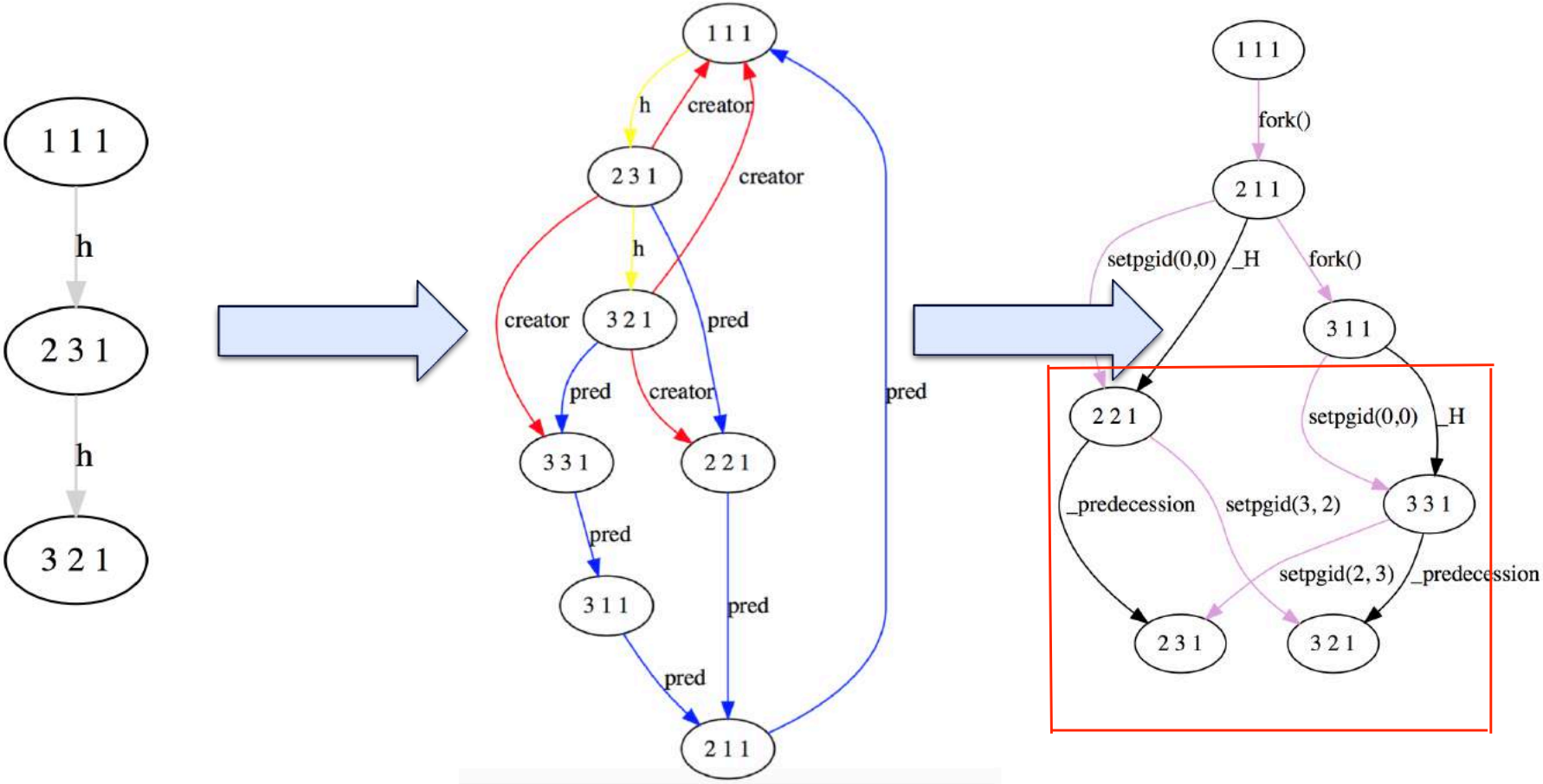
Пост-обработка графа: зачем?

```
1  function Gdep2SSA (G^dep, K);
   Input  : G^dep, K
   Output: SSA(G^dep)_correct
2  for P in getChains(G^dep) do
3      for proc in P do
4          epoch=1
5          for attr in K \ {pid} do
6              addNewVar(attr,epoch) // start
7              epoch+=1
8              if proc.attr ≠PCL(proc,attr)_pred.attr then
9                  addNewVar(attr,epoch) // новая переменная для attr
10                 epoch += 1
11                 if isCreator(PCL(proc, attr)_creator,attr) then
12                     // в модифицированном графе значение может передаваться через
                          носитель, creator - наследованное название
13                     addNode(G^dep,from=PCL(proc, attr)_creator,to=proc)
14                 end
15             end
16         end
17     end
18 end
19 return G^dep;
```



Пост-обработка графа: зачем?

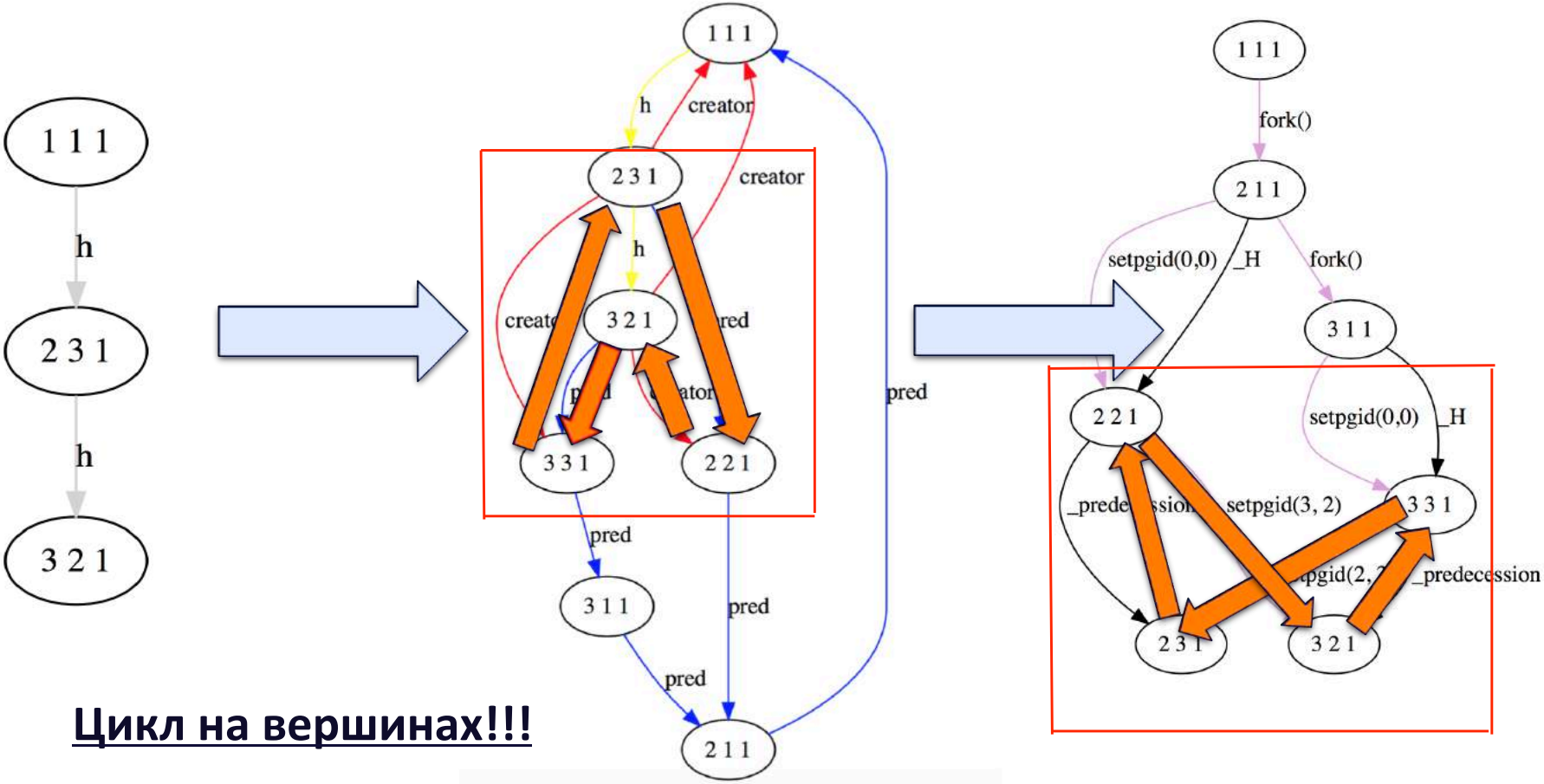
Рассмотрим дерево (1 1 1) →(2 3 1) →(3 2 1)



T *G(T)* *Граф действий*

Пост-обработка графа: зачем?

Рассмотрим дерево (1 1 1) →(2 3 1) →(3 2 1)



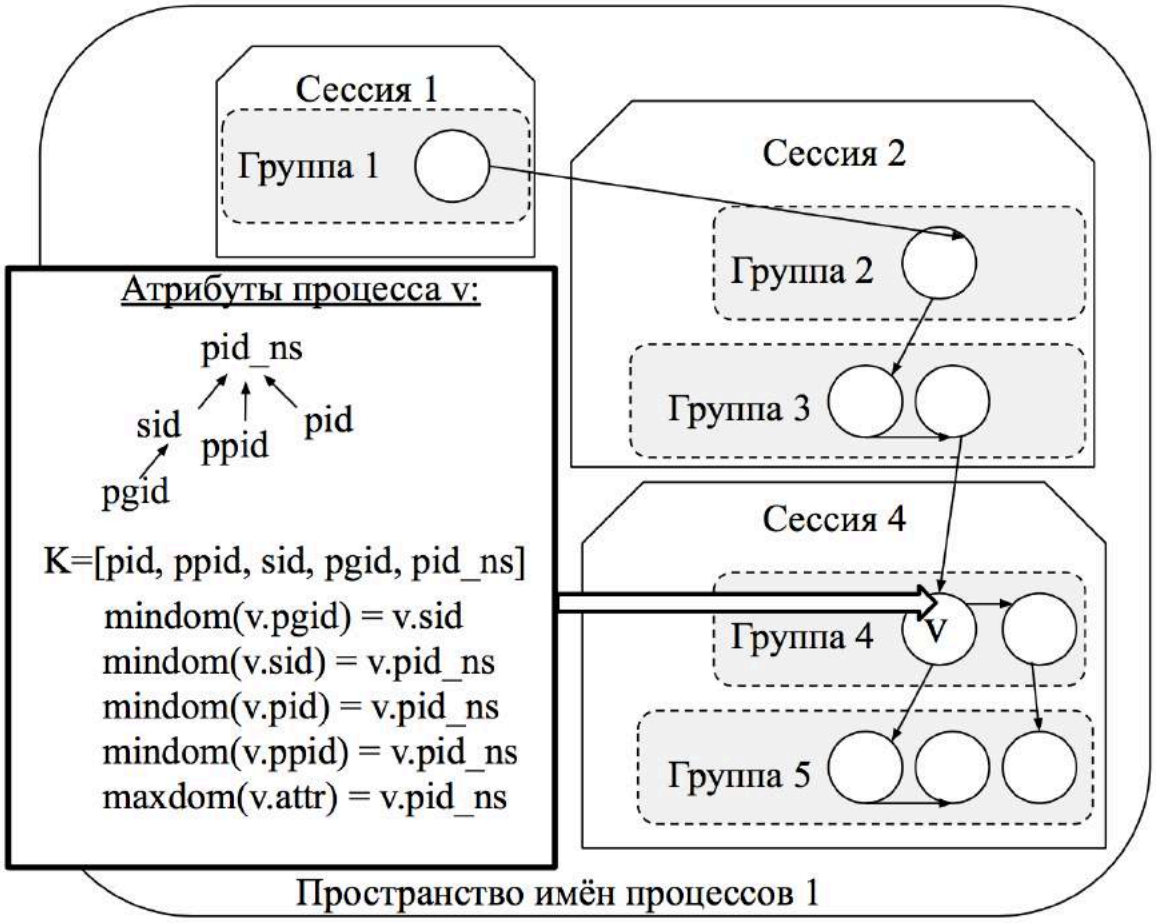
Цикл на вершинах!!!

T *G(T)* *Граф действий*

Анализ зависимостей

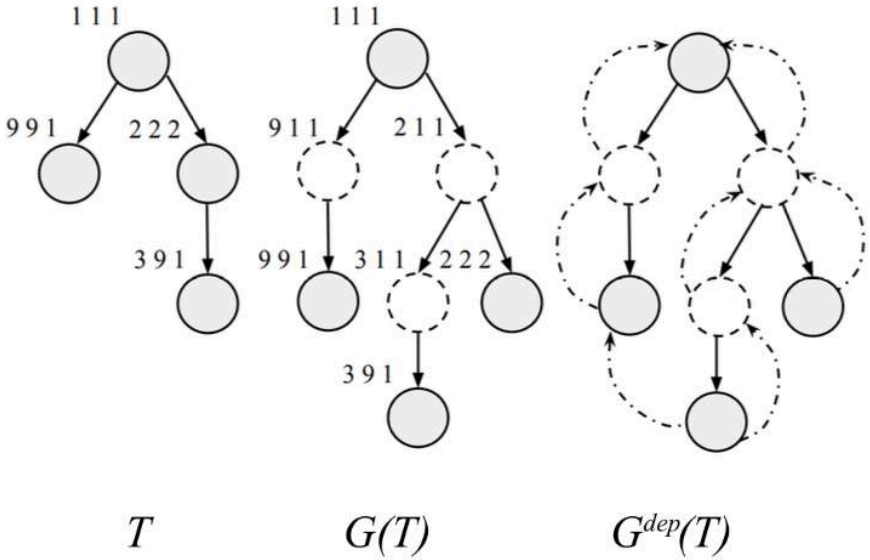
Давайте разбираться…

Исследуем зависимости между атрибутами более формально



Анализ зависимостей

Семантика: зависимость (u,v): для реализации u нужно v.



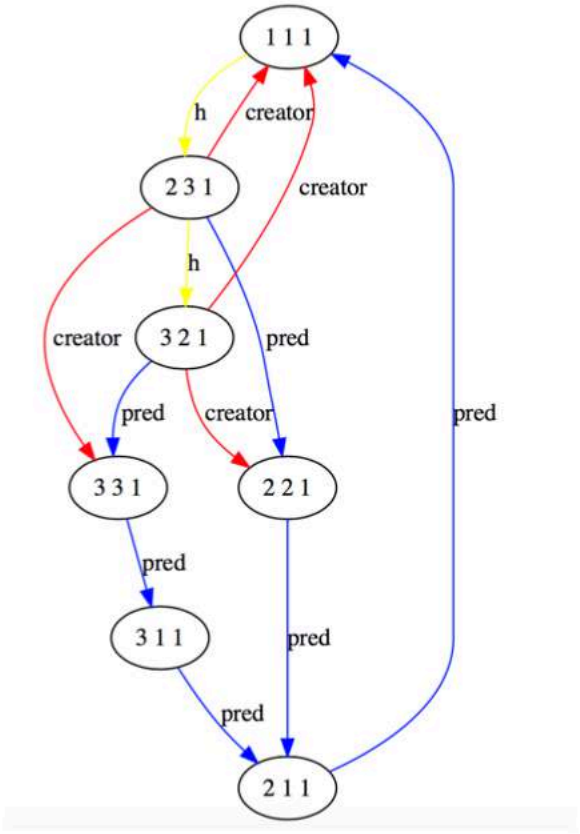
T $G(T)$ $G^{dep}(T)$

Задан частичный порядок на вершинах G(T)!

Теорема 1. *V^+ с отношением зависимости образует полную верхнюю полурешетку.*

Для любого подмножества состояний существует единственный ближайший прародитель.

Диаграмма зависимостей



G(T)

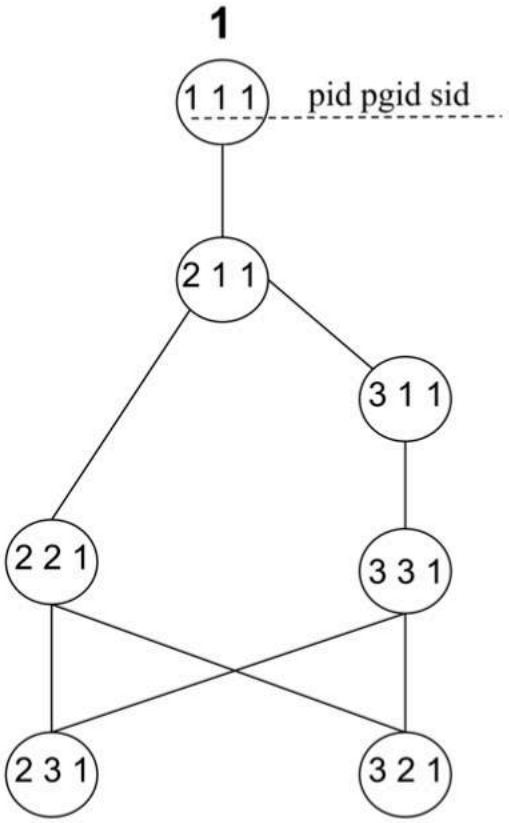
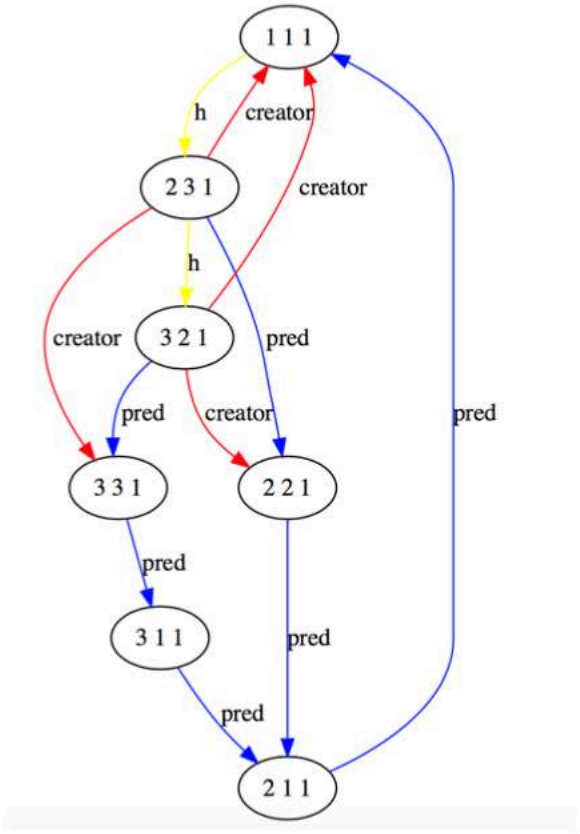


Диаграмма Хассе *G(T).nodes*

Диаграмма зависимостей



G(T)

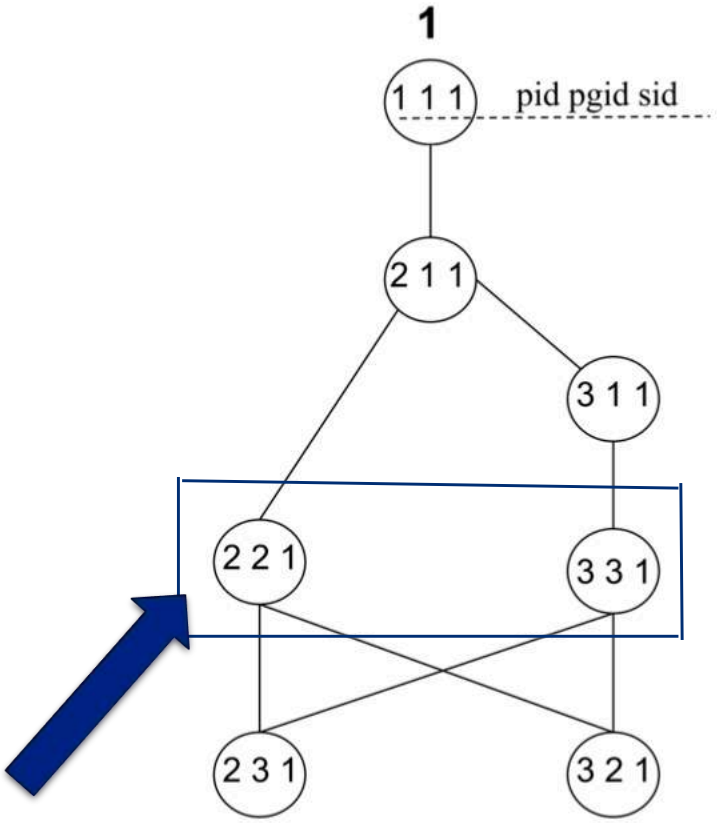
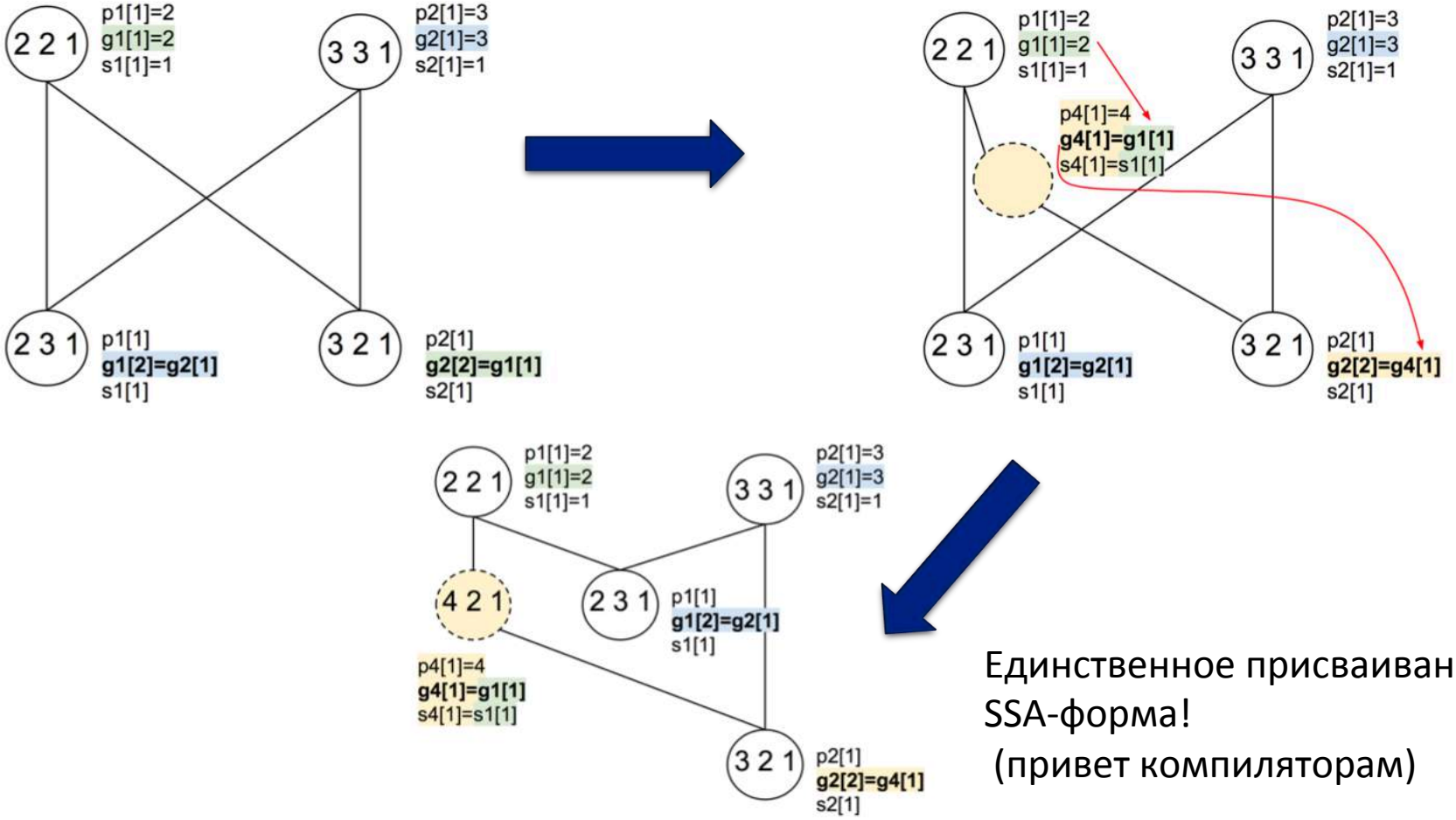


Диаграмма Хассе *G(T).nodes*

Это же не полурешётка ☹

Надо исправлять!

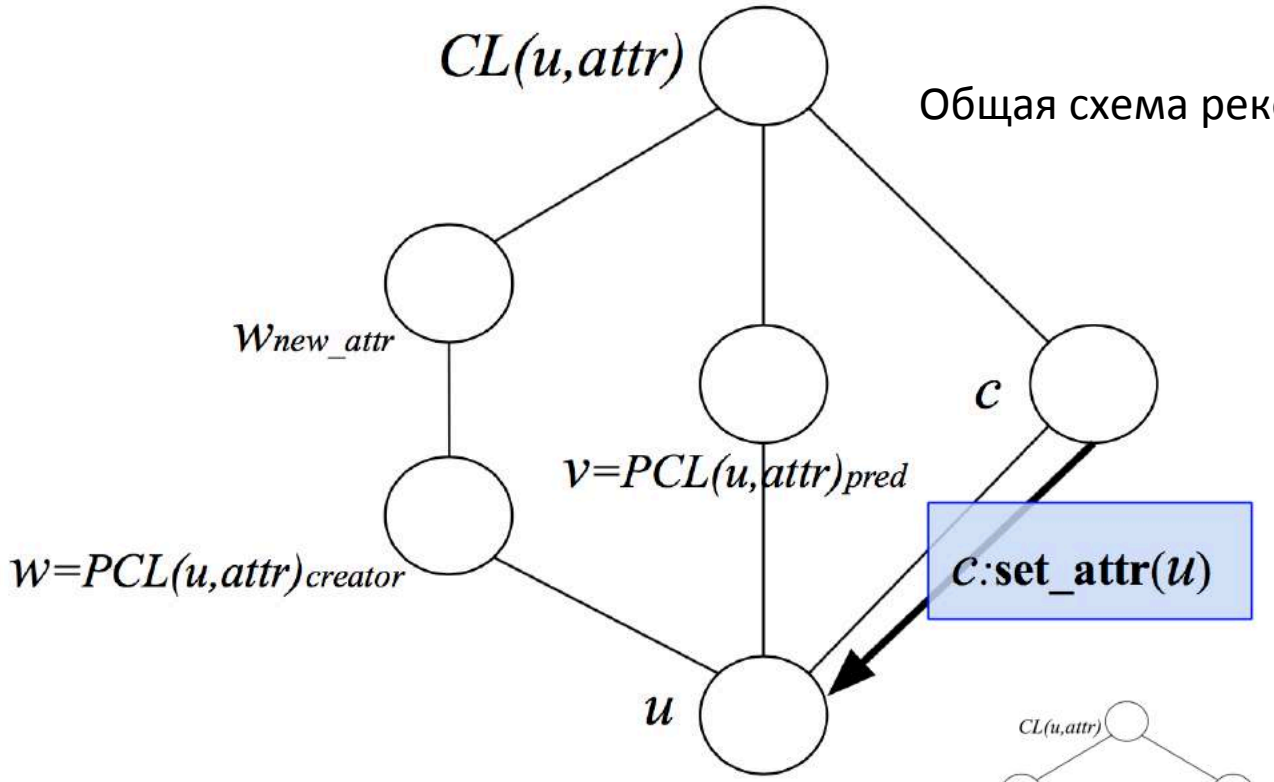
Промежуточный носитель



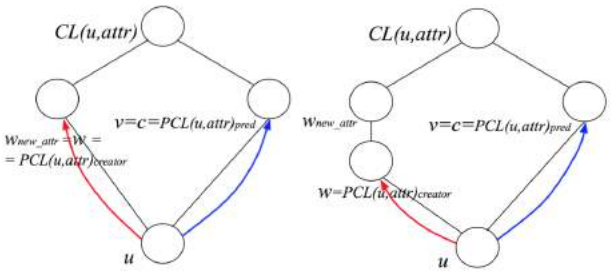
Единственное присваивание-SSA-форма!
(привет компиляторам)

Теперь полурешётка☺

Обобщаем

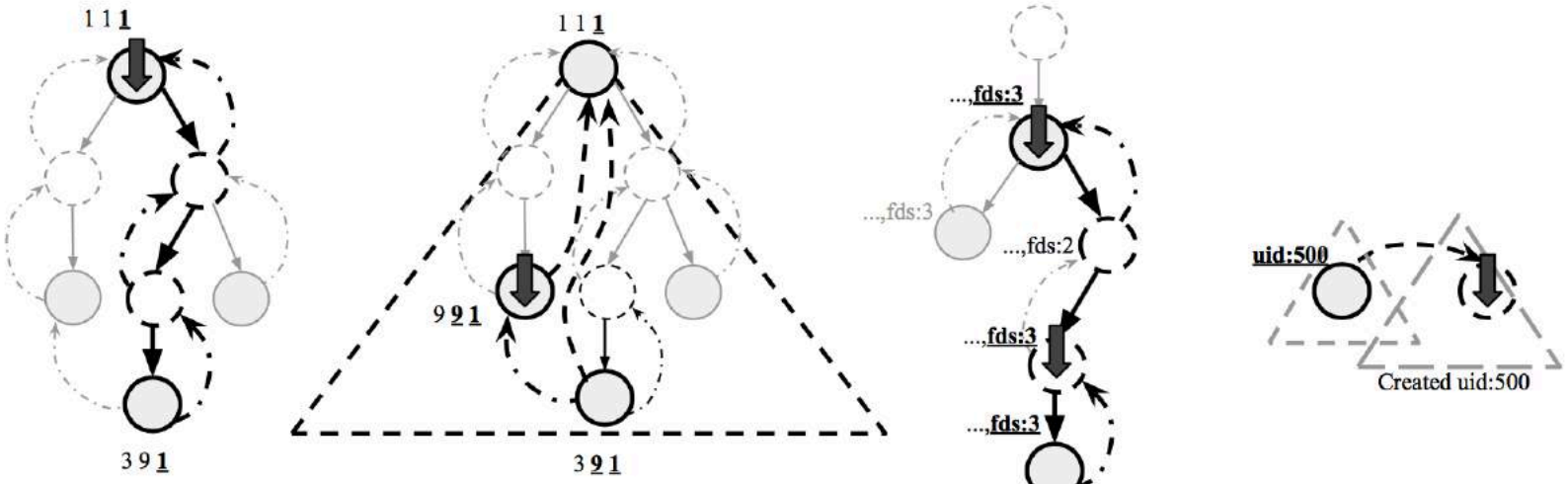


Общая схема реконструкции



$$\begin{cases} w \le w_{new} = PCL(u,attr_i)_{creator} \le CL(u,attr_i), \\ v = PCL(u,attr_i)_{pred} \le CL(u,attr_i), \\ c = PCL(u,attr_i)_{syscaller} \le CL(u,attr_i), \\ u \le v, u \le w, u \le c. \end{cases}$$

Нужно больше обобщений…



- “конечная” вершина
- “промежуточная” вершина
- точка создания ресурса (задания значения атрибуту)
- иерархические связи и системные вызовы
- зависимости по непосредственно предшествующим вершинам
- зависимости от дополнительных атрибутов

Нужно больше обобщений…

Лемма 1. $\forall G(T) = (V^+, E^+) : T = (V, E)$ - *дерево процессов с атрибутами* $K = \{pid, sid, pgid\} \rightarrow \frac{|V^+|}{|V|} \leq O(1)$.

Лемма 2. $\forall G(T) = (V^+, E^+) : T = (V, E)$ - *дерево процессов с атрибутами* K:K *не содержит мягко-наследуемых атрибутов, и* $|K| \ll |V| \rightarrow \frac{|V^+|}{|V|} \leq O(1)$, *иначе* $\frac{|V^+|}{|V|} \leq O(K)$.

Лемма 3. $\forall G(T) = (V^+, E^+) : T = (V, E)$ - *дерево процессов с атрибутами* $K \rightarrow \frac{|V^+|}{|V|} \leq O(|K|)$.

Table 1: Time complexity of reconstruction

Features	Basic	Cross-Attributed
fork	O(V)	O(V)
exit(reordering)	$O(\|V\|^2)$	$O(\|V\|^2)$
setsid,setpgid(credentials)	$O(\|V\|^2)$	$O(\|V\|^2)$
nested namespaces	$O(\|K\|\|V\|^2)$	$O(\|K\|\|V\|^2)$
MI-attributes(file descriptors, memory mappings,signals etc)	$O(\|K\|\|V\|^2)$	$O(\|K\|^2\|V\|^2)$

Кроме того…

Критерий корректности дерева

Теорема 2.1. *Если T - корректное дерево процессов, то множество его вершин V дополнимо до верхне полурешёточно упорядоченного по зависимостям V^+ : $(\forall x,y \in V^+ \; \exists k : \forall n > k, n \leq |K| : attr_k, attr_n \in K, depth(attr_n) < |K| - k)$ & $(x.attr_n = y.attr_n) \Rightarrow (x \sqcup y = PCL(x,attr_k)|PCL(y,attr_k)|CL(x,attr_k))$ & $(CL(x,attr_k) = CL(y,attr_k))$, где CL,PCL - введённые выше операторы замыкания и предзамыкания на V^+, и единственной неподвижной точкой CL является создатель минимально доминирующего атрибута для $attr_k$.*

Теорема 2.2. *Пусть V^+ - множество вершин графа G^{dep}, проверяемого на соответствие графу реконструкции с зависимостями, и два множества $V = \{v \in V^+ : v \neq x \sqcup y, \forall x,y \in V^+ \setminus \{v\}$ & $y.pid = v.pid\}, E = \{(x,y) : x,y \in V$ & $y.ppid = x.pid\}$ - непустые. Пусть $\forall u \in V^+ \setminus \{v_{init}\} \; \forall attr_i \in K \; \exists (Gen(u.attr_i) \subseteq V^+) \mid (Gen(u.attr_i) = \emptyset \iff u.attr_i = none)$. Тогда (V,E) - корректное дерево процессов, а G^{dep} - граф реконструкции (V,E) с зависимостями.*

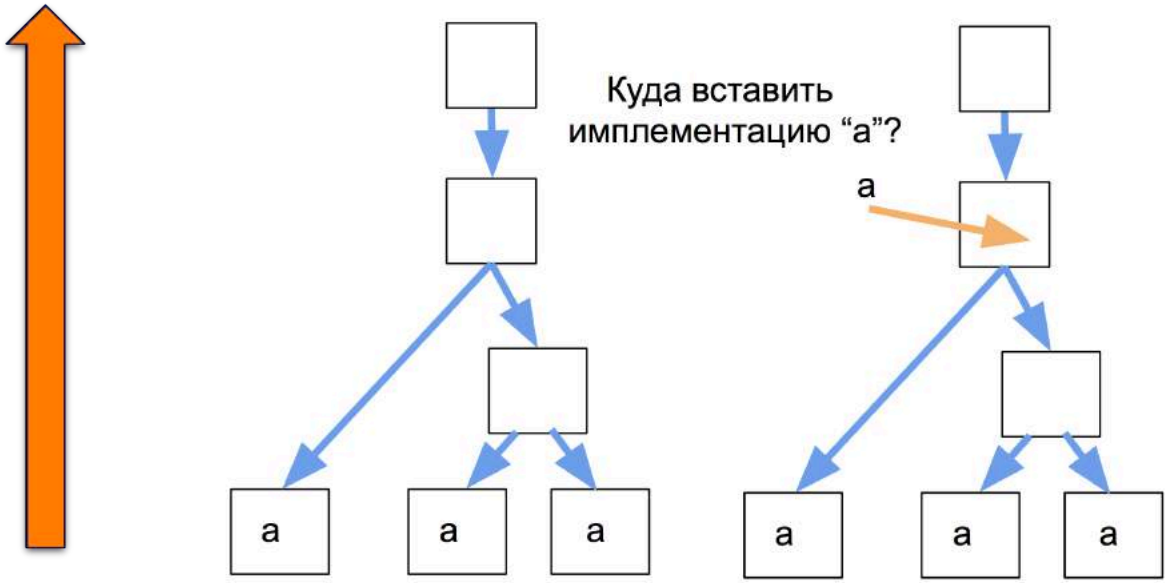
Смысл: быстрая генерация корректных тестов

Где это ещё применить?

Disclaimer: везде, где состояния + зависимости = полурешётка.

- **Реинжиниринг иерархий классов в ООП:**

подъемы/спуски полей/методов для классов/суперклассов:



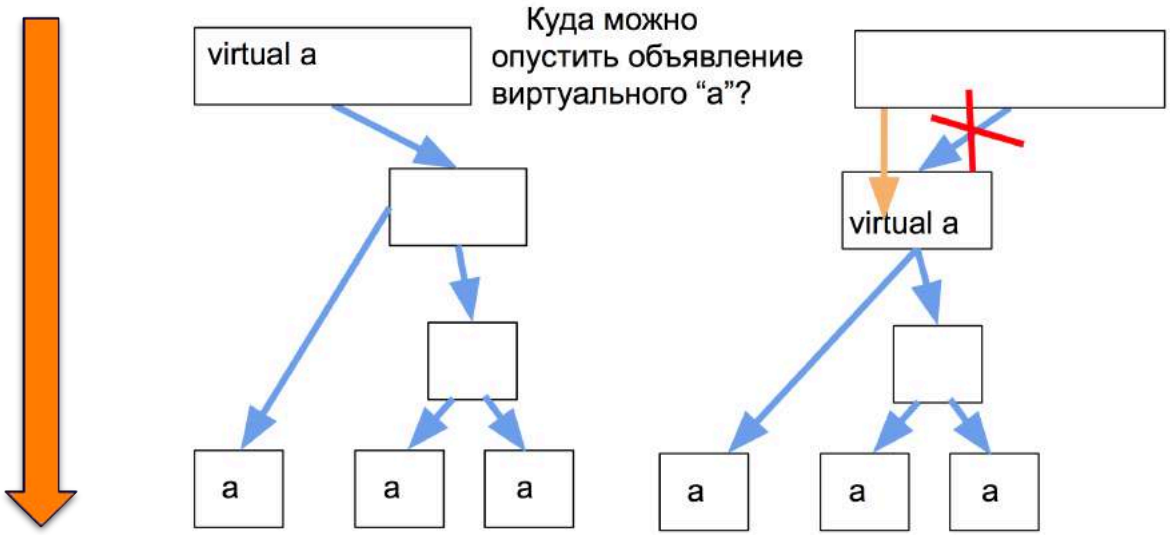
- **Реверс-инжиниринг иерархий классов в ООП:**
- **Генерация тестов для ОС, систем виртуализации, etc**

Где это ещё применить?

Disclaimer: везде, где состояния + зависимости = полурешётка.

- **Реинжиниринг иерархий классов в ООП:**

подъемы/спуски полей/методов для классов/суперклассов:



- **Реверс-инжиниринг иерархий классов в ООП:**
- **Генерация тестов для ОС, систем виртуализации, etc**

Где это ещё применить?

Disclaimer: везде, где состояния + зависимости = полурешётка.

- **Реинжиниринг иерархий классов в ООП**
- **Реверс-инжиниринг иерархий классов в ООП:**

  Проект Marx:

  http://www.cs.ucy.ac.cy/~eliasathan/papers/ndss17.pdf

  Основная идея: эвристический анализ таблиц vtable из бинарных файлов соответствующего ABI, выстраивание частичных порядков на записях.

- **Генерация тестов для ОС, систем виртуализации, etc**

Выводы:

1. Обобщённая модель позволяет:
   - Гарантированно восстанавливать деревья процессов
   - Исправлять аномалии реконструкции
   - Справляться с множеством различных ресурсов
   - Служить генератором тестов (быстрее, чем перебор)

   Открытые вопросы:

   - Любые ли отношения между атрибутами допустимы

   (дисклеймер: скорее нет – затруднительно описываются файловые блокировки, другие несравнимые объекты)

   - Интеграция в индустрию (Вероятно, CRIU?)

Contact

1. Ефанов Николай, МФТИ
2. https://github.com/nefanov
3. nefanov90@gmail.com