



61-я научно-практическая конференция МФТИ

October 22
Moscow/Dolgoprudnyi

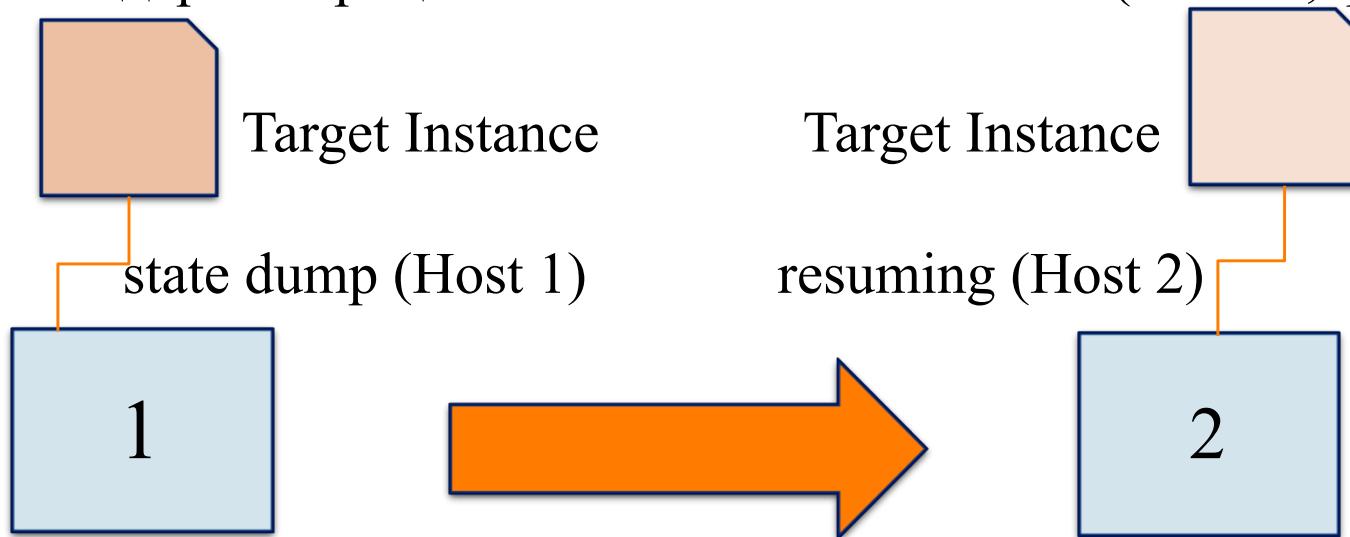
Атрибутная грамматика деревьев процессов и
восстановление порождающих цепочек
системных вызовов

Николай Ефанов

Цель построения модели сохранения/ восстановления и прикладные задачи

Задача: сохранение и восстановление состояния процессов и окружения из пространства пользователя. Строгая модель сможет предоставить:

1. Улучшение процедуры восстановления в Checkpoint-Restore (CRIU, etc).
2. Сокращение накладных расходов при живой миграции.
3. В идеале: дерево процессов + полезный контекст (память, файлы и др.).



Задача и ограничения:

1. Восстановить последовательности системных вызовов, порождающие входное дерево процессов D . Вызовы хранятся в дереве T :

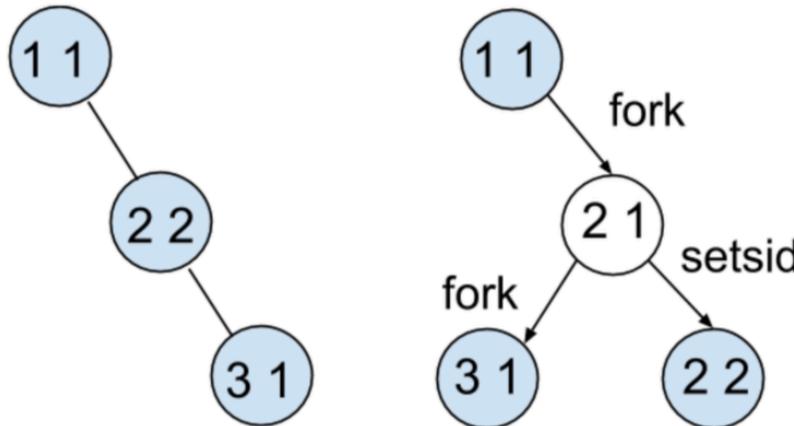
$$D = \{V, E\}, \rightarrow T = \{V^+, E^+\}, V^+, E^+ - \text{хранят цепочки}$$

$$|E \setminus (E \cap E^+)| \geq 0$$

$$V = V \cap V^+$$

2. Ограничения:

- Только Linux
- Ввод корректен. Иначе: Ошибка разбора.
- Системные вызовы:
 - ✓ **Fork**: создать процесс-потомок
 - ✓ **Setsid**: создать новую сессию
 - ✓ **Setpgid**: установить группу вызвавшему: setpgid(0,pgid)
 - ✓ **Exit**: завершить процесс -> присвоить потомков Init-процессу
 - ✓ → Базис трансформаций дерева процессов



Анализ задачи: комбинаторные оценки

Число деревьев, полученных с **fork** (из формулы Кейли для деревьев*):

$$F(n) = \sum_{i=2}^{n-1} i^{i-2}$$

И это с учётом ‘нечувствительности’ к перестановкам идентификаторов!

Добавив абстрактный вызов **k call**, изменяющий идентификатор **k**:

$$F(n, k_call) = F(n) \sum_{m=0}^{n-1} \binom{n}{m} (m+1)^{n-m-1}$$

Прямая генерация – не лучшая идея!

* Ефанов Н.Н. Комбинаторные и групповые свойства деревьев процессов Linux // Сборник трудов XV международной конференции «Алгебра, теория чисел и дискретная геометрия», Тула, 28-31 мая 2018 г., С. 180-183.

Предшествующая работа: грамматика строк

Строчная нотация + Набор правил:

- Процессы представляются как “ $p, g, s [children]$ ” и рекурсивно перечисляются
- Набор правил переписывания строк:
 - Пример правила: **fork**(* * * [*]) --> * * *[* \2 \3 [] \4].
 - Левые части могут содержать контексты – «если конфигурация такая, тогда...»
 - Правило **exit** удаляет данные{1 * * [*], **exit**(* * * [*])} --> 1 \1 \2 [\3 \7].
- Грамматика не является контекстно-свободной (**setpgid**)
- Это грамматика Типа 0 по Хомскому (см. **exit**)
- Эвристики в анализе позволяют решить задачу за $O(P(n))$

Предшествующая работа: грамматика строк

- «BPSF»: Двухстадийный $O(N \log(N) \log(S) \log(P))$ -time анализатор:
 - Стадия 1: бесконтекстный анализ --> промежуточное состояние в стеке
 - Стадия 2: разбор контекста --> ответ
 - +AVL-структуры поиска параметров: логгирование p, g, s в ходе работы

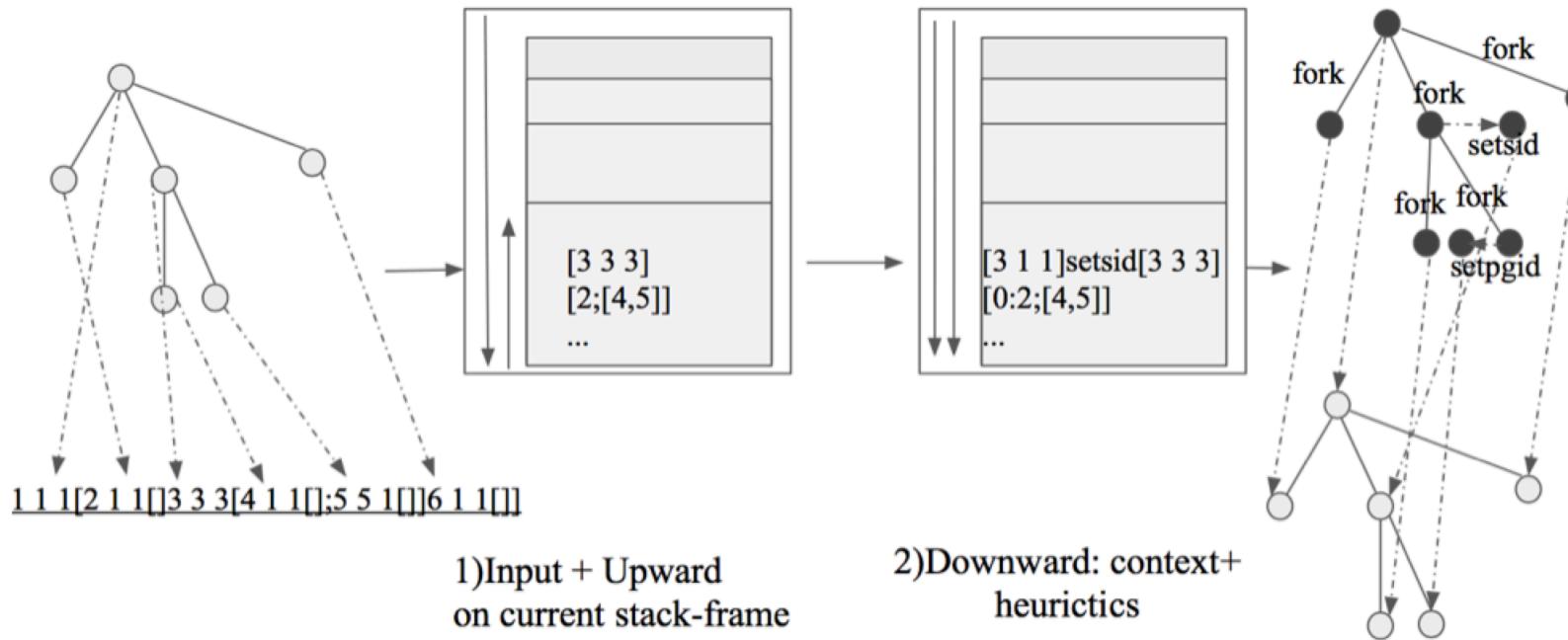


Схема «Дерево процессов→Строка→Стек с кадрами+Структуры поиска→Дерево»

Предшествующая работа: грамматика строк

- «BPSF»: Двухстадийный $O(N \log(N) \log(S) \log(P))$ -time анализатор:
 - Стадия 1: бесконтекстный анализ --> промежуточное состояние в стеке
 - Стадия 2: разбор контекста --> ответ
 - +AVL-структуры поиска параметров: логгирование p, g, s в ходе работы

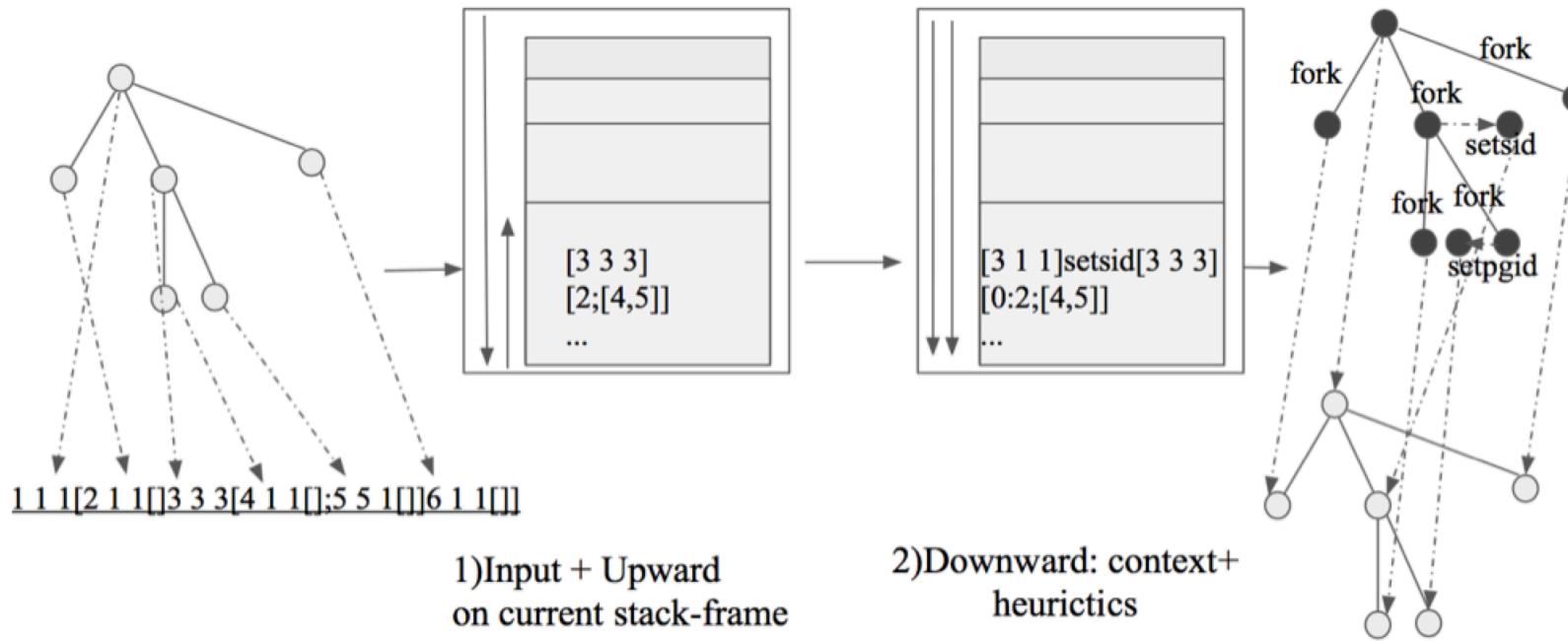
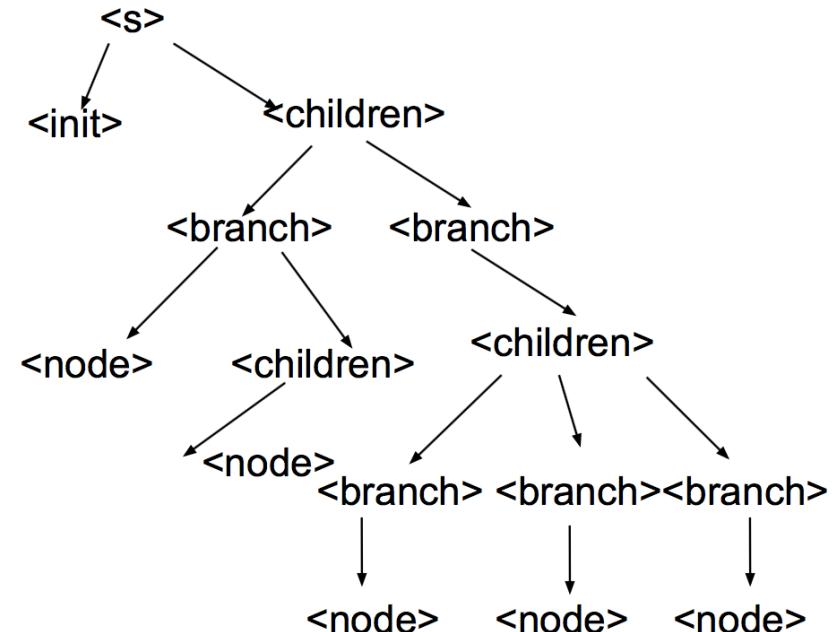


Схема «Дерево процессов → Стока → Стек с кадрами + Структуры поиска → Дерево»
! Некоторая избыточность преобразований. Какой выход?

КС-грамматика $\{A, N, P, \langle s \rangle\}$

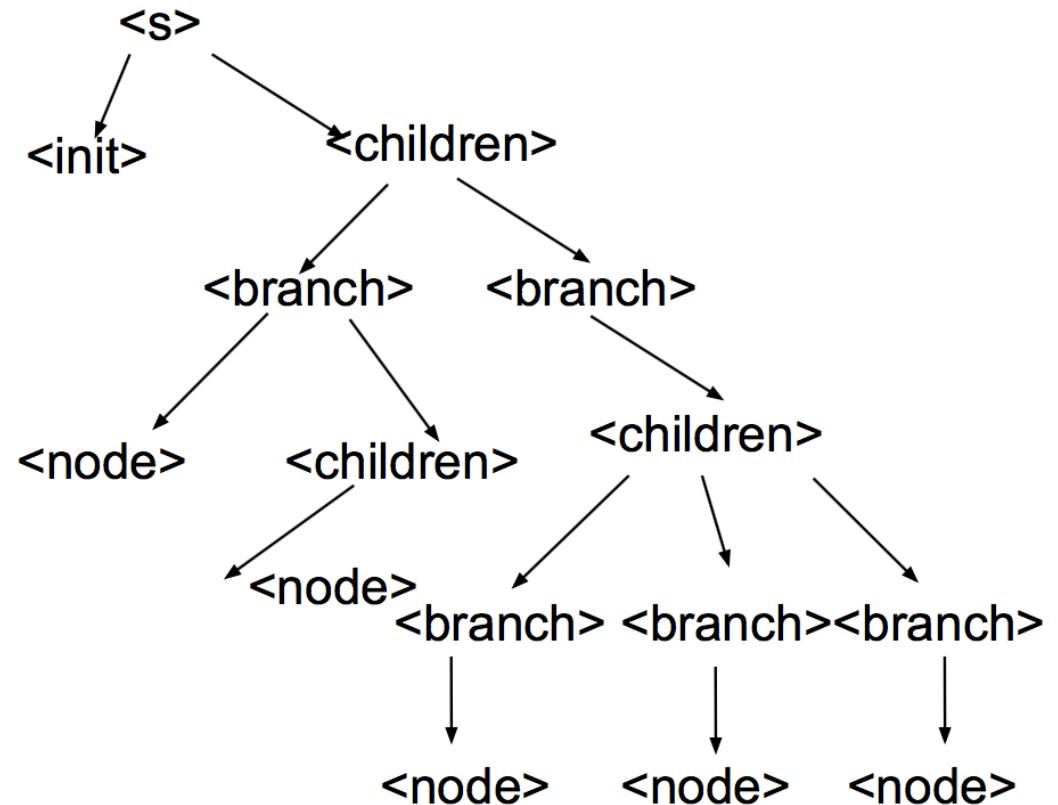
- A – терминалы
- $N = \{\langle s \rangle, \langle init \rangle, \langle children \rangle, \langle branch \rangle, \langle node \rangle\}$
- $\langle s \rangle$ - стартовый нетерминал
- P – набор правил:
 - $\langle s \rangle = \langle init \rangle \langle children \rangle | \langle init \rangle$
 - $\langle init \rangle = "1..1"$
 - $\langle children \rangle = \{\langle branch \rangle\}$
 - $\langle branch \rangle = \langle node \rangle \langle children \rangle | \langle node \rangle$
 - $\langle node \rangle = \text{строка терминалов}$



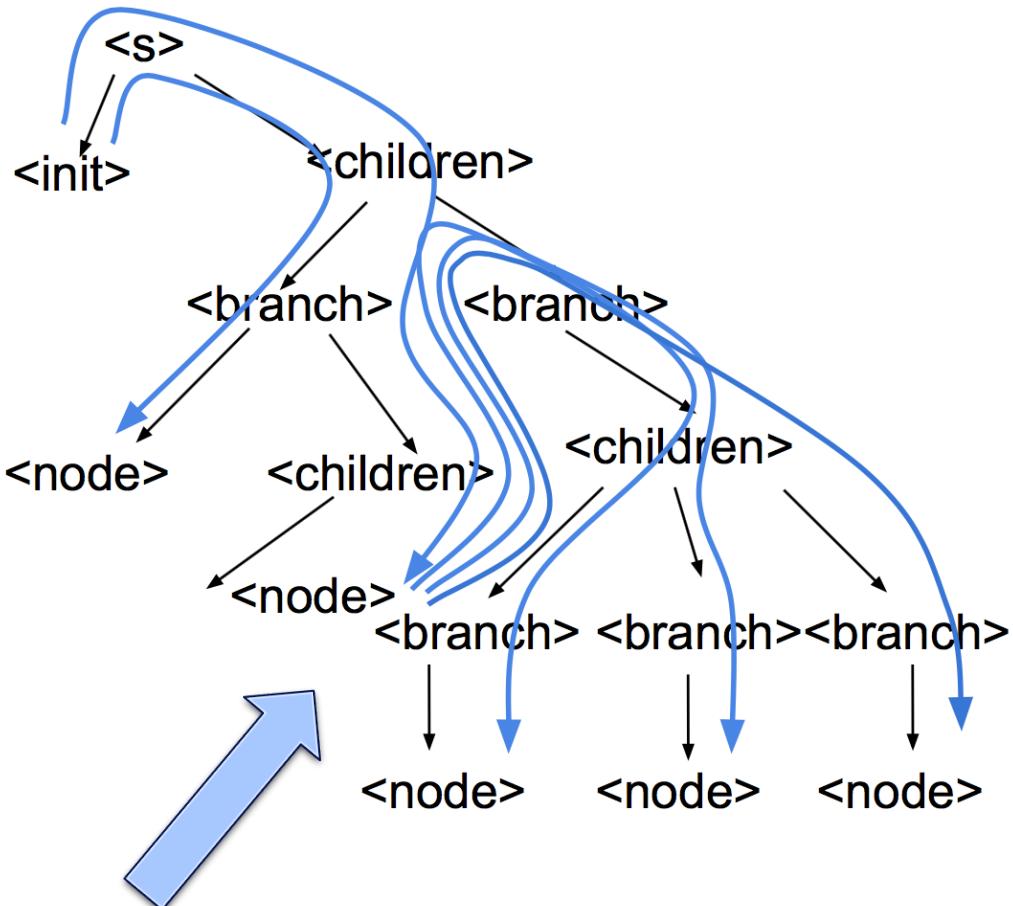
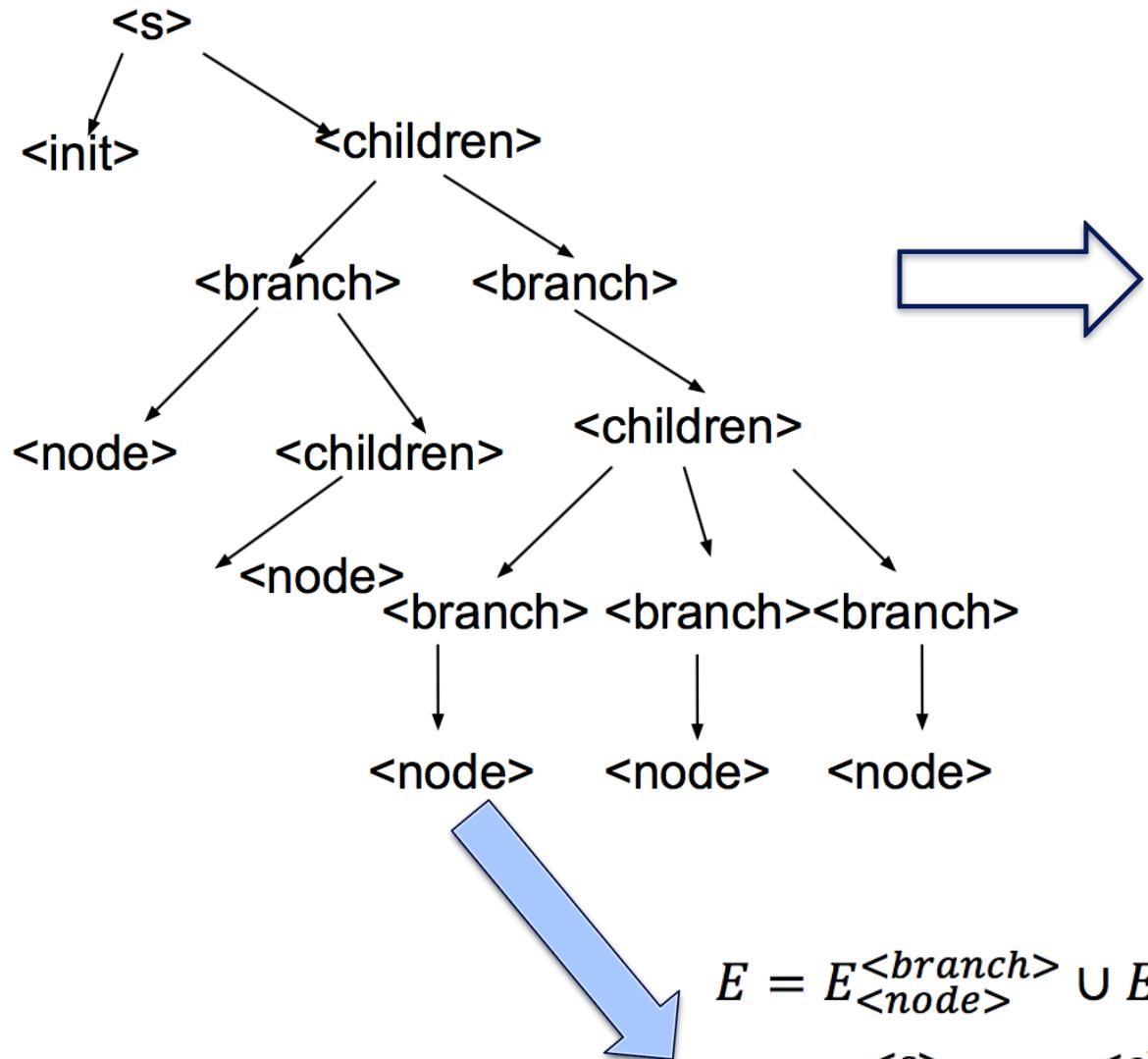
Такая грамматика задаёт язык деревьев с выделенной корневой вершиной – то что нужно !

- Хранение параметров процессов в терминалах
- Извлечение параметров для анализа

Дерево разбора → Дерево процессов



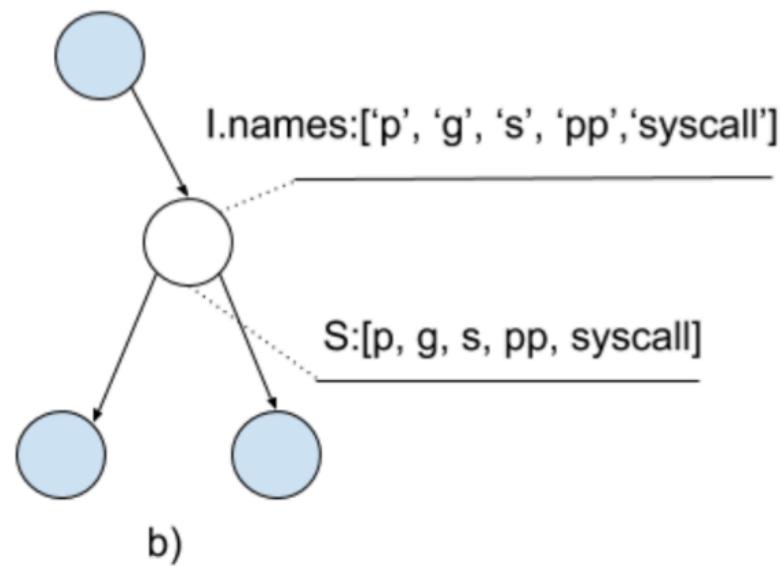
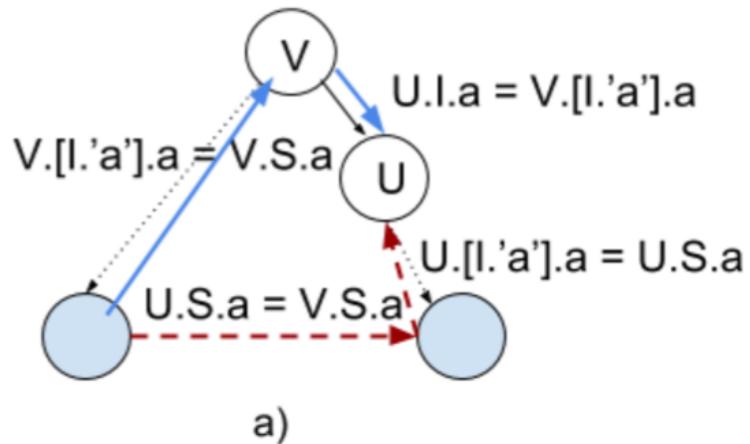
Дерево разбора → Дерево процессов



$$E = E_{<\text{branch}>}^{<\text{node}>} \cup E_{<\text{branch}>}^{<\text{children}>} \cup E_{<\text{children}>}^{<\text{branch}>} \cup E_{<\text{branch}>}^{<\text{node}>},$$
$$E = E_{<\text{init}>}^{<\text{s}>} \cup E_{<\text{s}>}^{<\text{children}>} \cup E_{<\text{children}>}^{<\text{branch}>} \cup E_{<\text{branch}>}^{<\text{node}>}$$

Атрибутная грамматика деревьев процессов: $\{A, N, P, \langle s \rangle, AT, SA\}$

- **AT** – Набор атрибутов:
 - $.I$ – наследуемые – ‘переменные’: $p, pp, g, s,$ ‘syscall’.
 - $.S$ – синтезируемые – значения переменных.
- **SA** – Семантические действия:
 - Переписывание атрибутов
 - Добавление вершин
 - Добавление и удаление рёбер



Attributed Grammar Tree Transformation

Method: анализатор

- Последовательные трансформации дерева на основании проверок атрибутов
- В результате: неявное построение графа зависимостей текущей вершины обходом дерева
- → Трансформации
- $O(n^2)$ -Time в худшем случае (доказательство см. в [4]).

Attributed Grammar Tree Transformation

Method: анализатор

- Последовательные трансформации дерева на основании проверок атрибутов
- **В результате: неявное построение графа зависимостей текущей вершины обходом дерева**
- → Трансформации
- **O(n^2)-Time** в худшем случае (доказательство см. в [2]).

Algorithm 1:

(Input: IR , D , $expr$; Output: T)

begin:

```
    for any Node in dfs(T.root):
        cond = f(expr(Node.S.I[1..m])) # m - attributes
        if cond not in IR: # context check
            cond = context_checking()
            ...
    D = TR(D, Ex, IR, k(cond))
```

return D

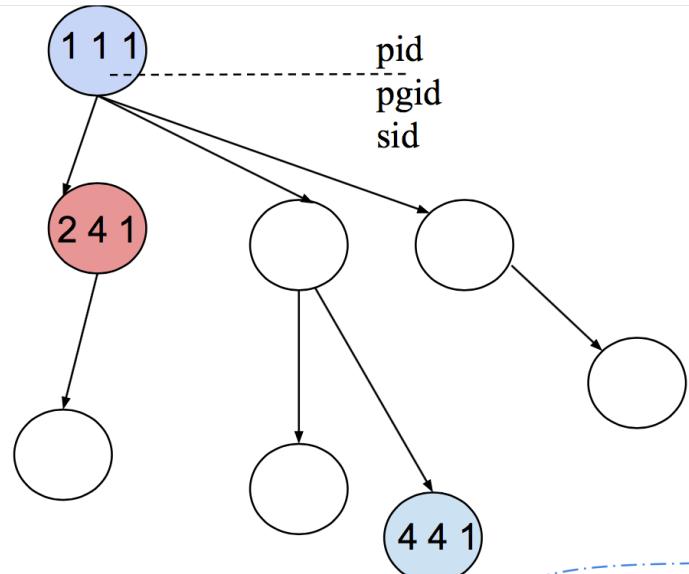
To Prove this theorem, it should be demonstrated that for each node the state reconstruction requires j extra *upbranch* and *dfs* traverses on the tree, $j \ll n$, then the algorithm reconstructs the whole tree in $O(jn^2) = O(n^2)$. Constructing the subroutine of context checking, where all of the input expressions $expr$ can be evaluated together in a single *upbranch* or *dfs* routine otherwise, for handling overlying and cross-branch dependencies respectively, it is gotten that $j = 2$ by design:

Subprogram 1:

```
context_checking():
    cond = upbranch({expr(Node, Current)})
    if not cond in IR:
        cond = dfs ({expr(Node, Current)})
    return cond
```

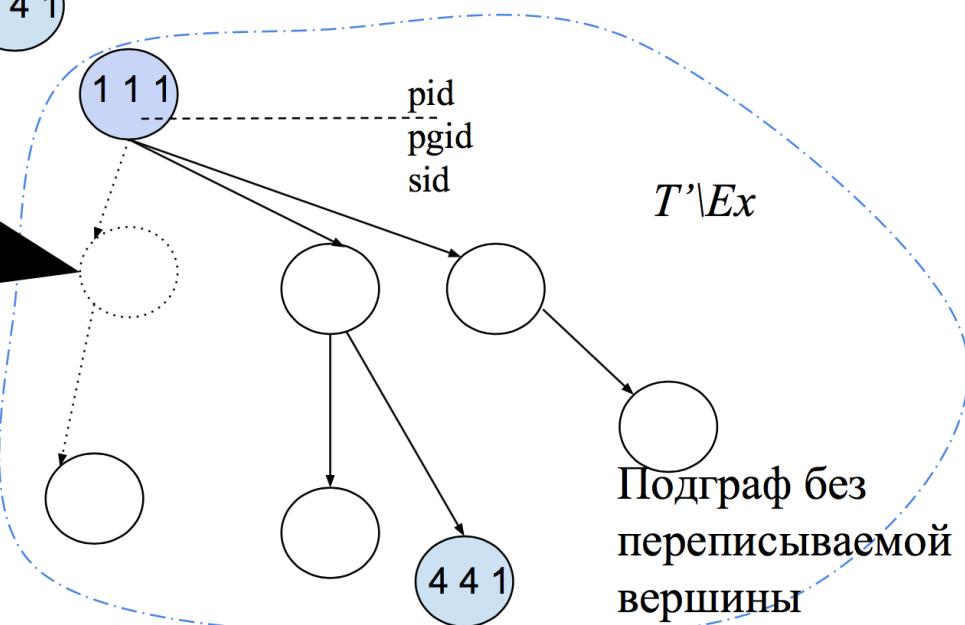
Q.E.D.

AGTTM - анализатор: пример



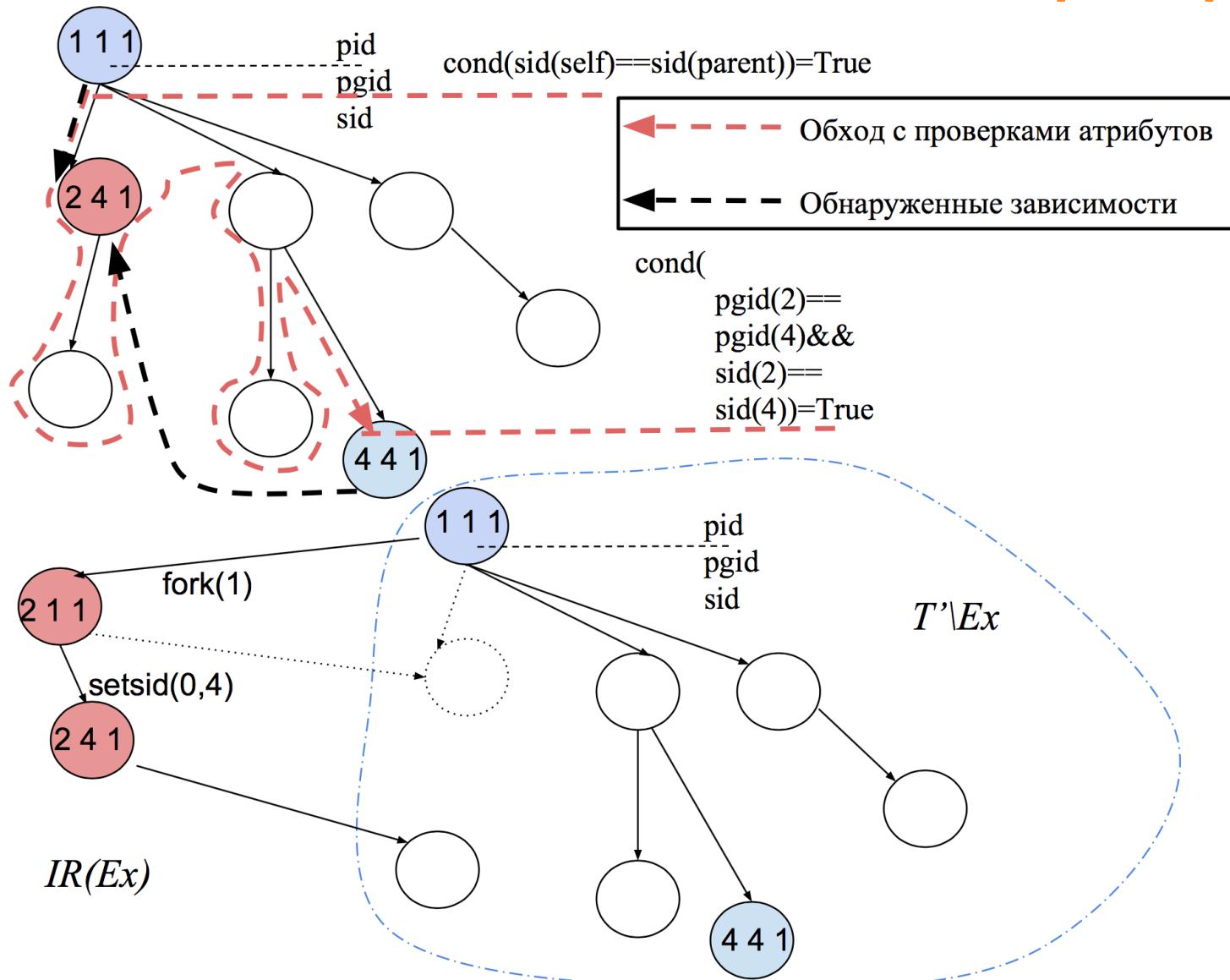
Задача: найти атрибуты в вершинах, от которых зависят атрибуты в текущей. “Безопасно” исключить вершину или подграф Ex . Сформировать или выбрать переписываемый подграф $IR(Ex)$.

Место
подстановки
подграфа
(дерева)

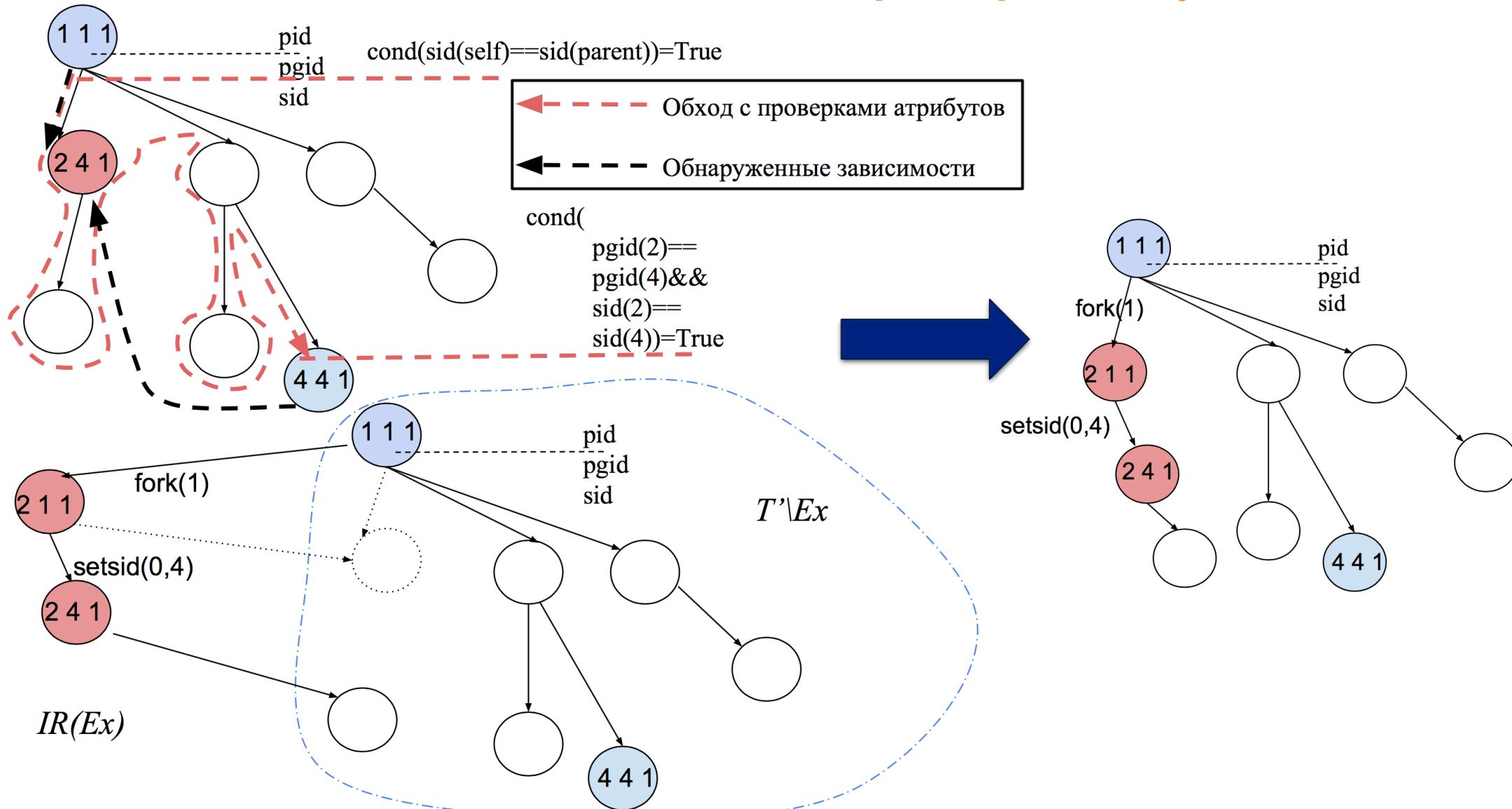


Подграф без
переписываемой
вершины

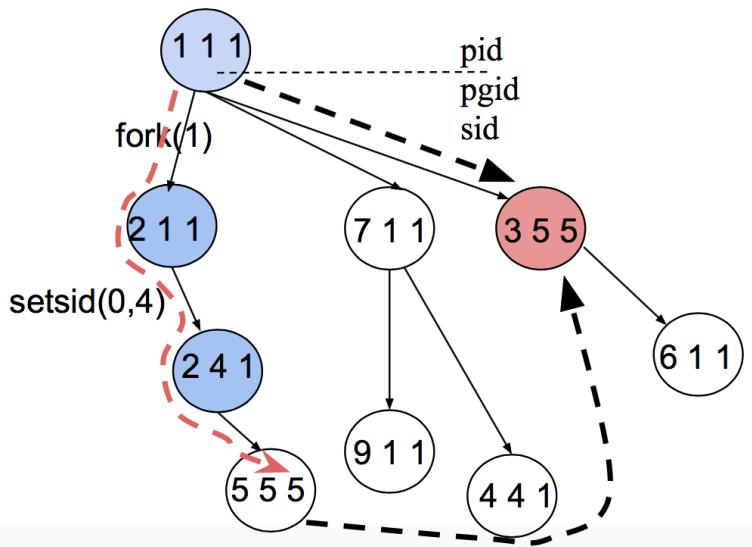
AGTTM - анализатор: пример



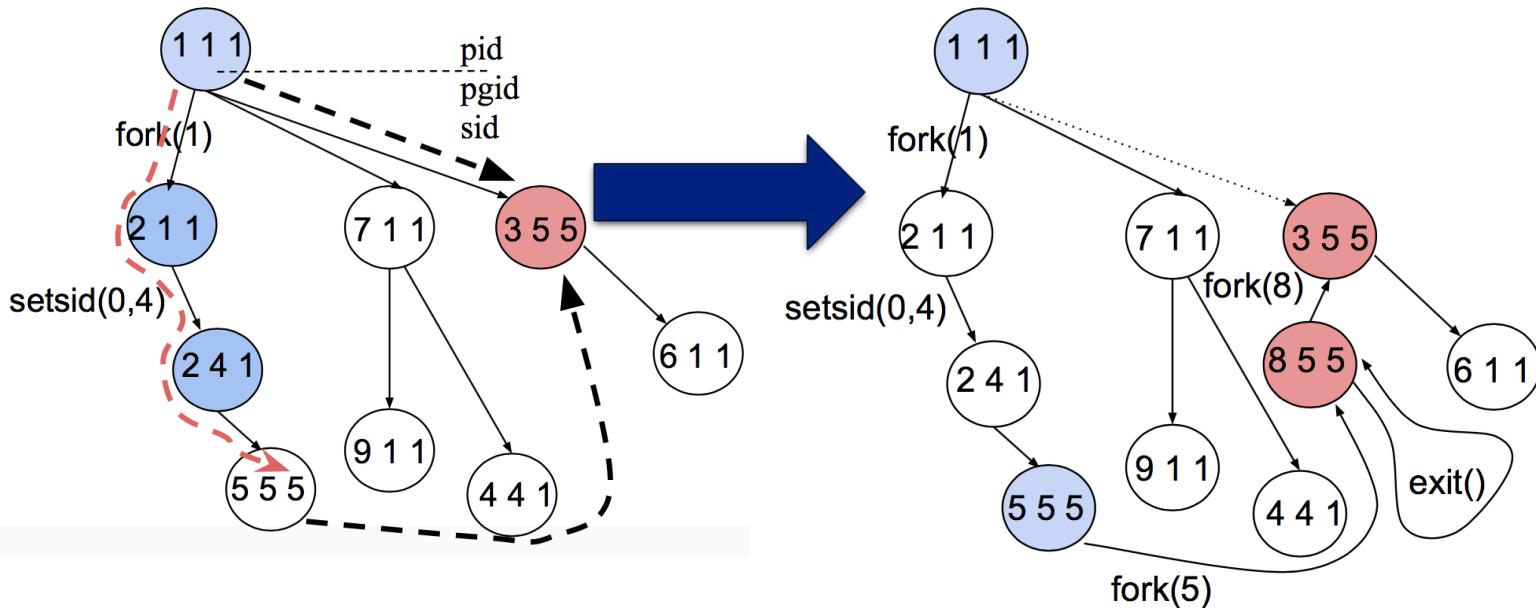
AGTTM - анализатор: пример



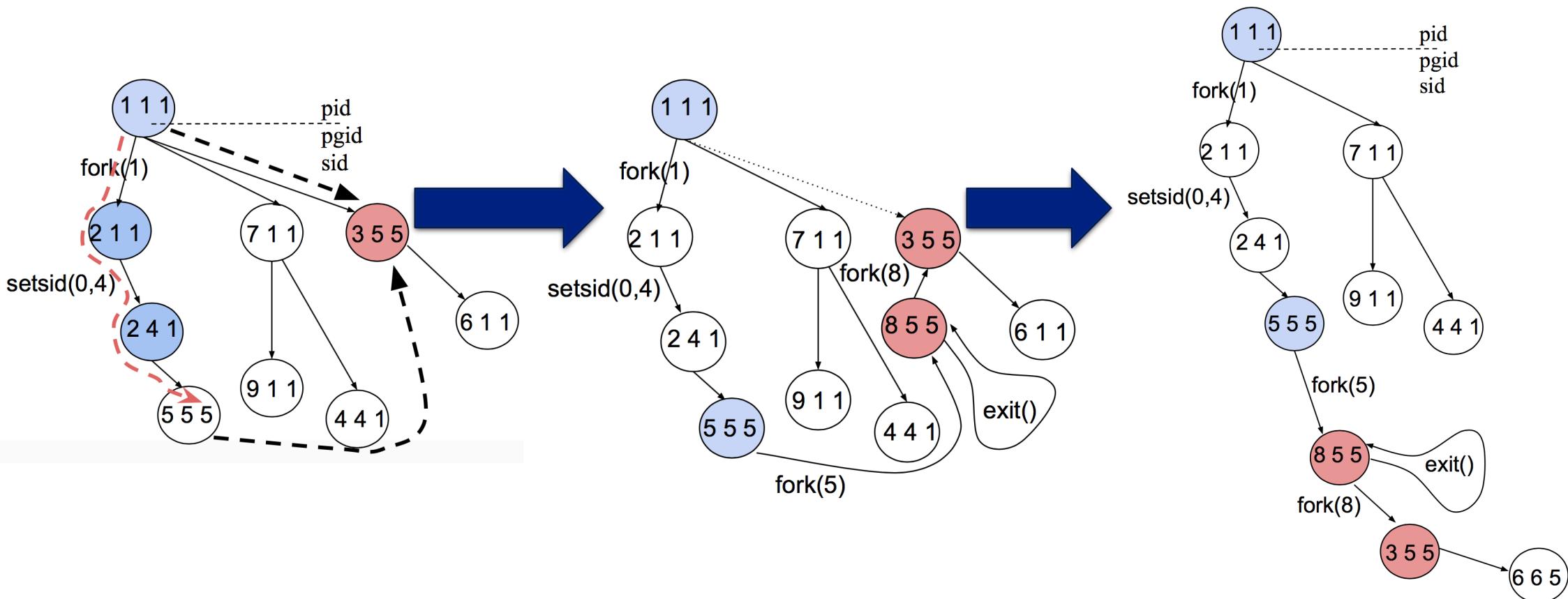
AGTTM - анализатор: пример с `exit()`



AGTTM - анализатор: пример с `exit()`

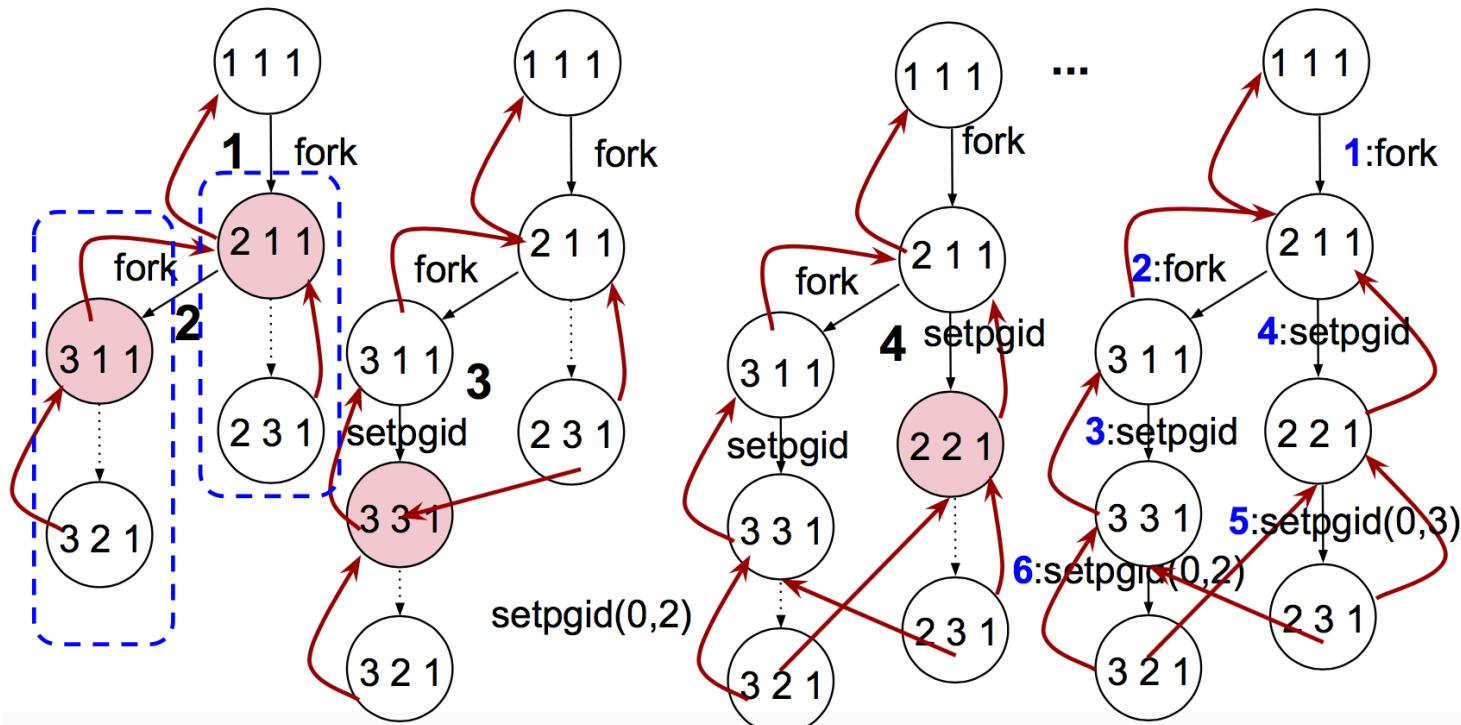
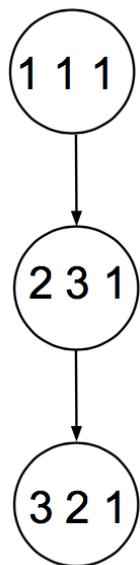


AGTTM - анализатор: пример с `exit()`

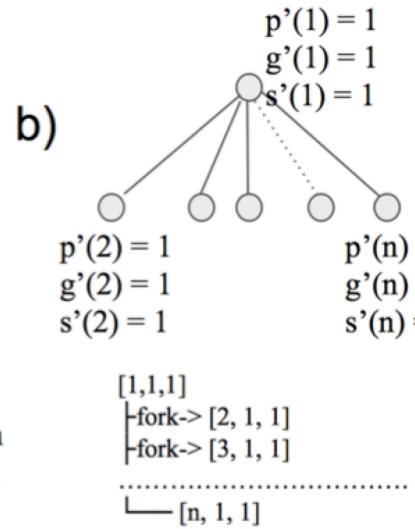
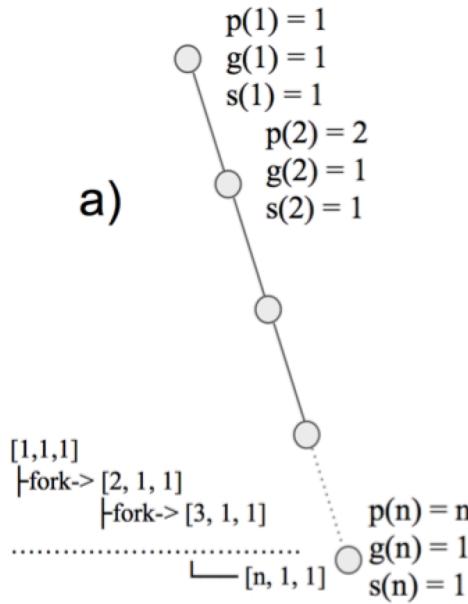


Решение известных проблем C/R:

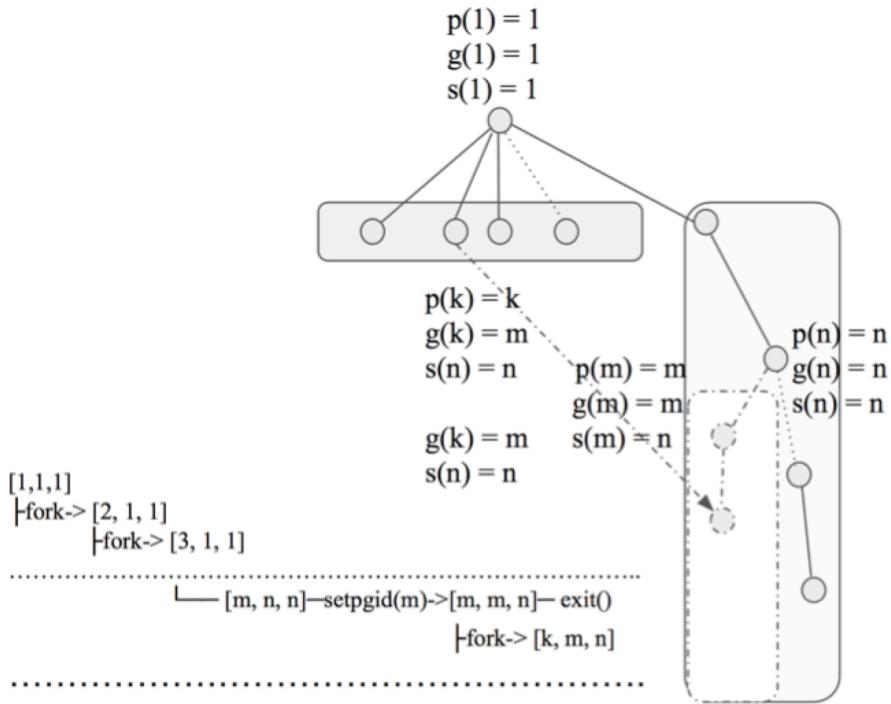
- CRIU fails on ...:
 - Восстановление по предложенной модели:



Тесты и метрика затрат:



“Simple-load” profile of tests



Reverse reparent-highloaded “heuristic” test

$$C_{overall} = C_s + C_r$$

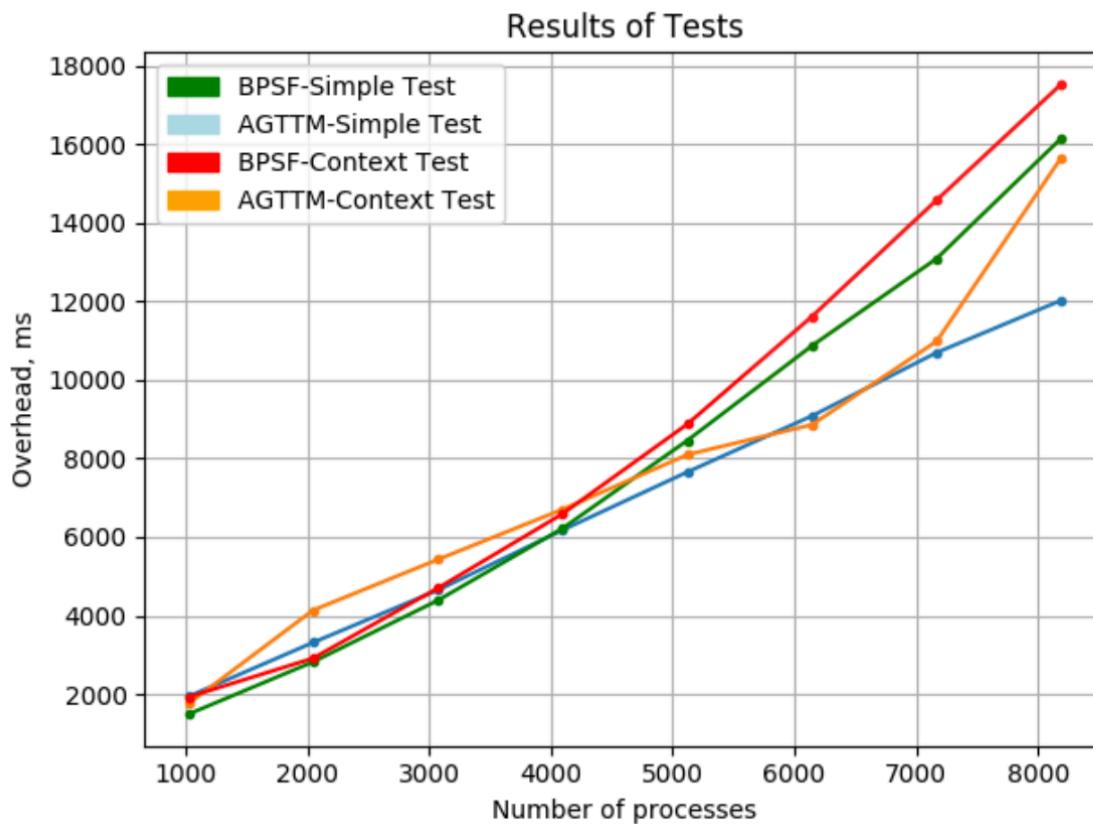
Результаты экспериментов

Table 1: The “Simple-load” test results

Number of Processes	AGTTM	BPSF
1024	1938ms	1488ms
2048	3314	2821
3072	4645	4385
4096	6176	6218
5120	7651	8453
6144	9080	10865
7168	10692	13085
8192	12023	16149

Table 2: The “Heuristic-load” test results

Number of Processes	AGTTM	BPSF
1024	1755ms	1925ms
2048	4135	2913
3072	5425	4696
4096	6701	6588
5120	8092	8868
6144	8856	11610
7168	10981	14573
8192	15659	17527



Выводы и дальнейшие действия

1. Метод на основе атрибутных грамматик применим к поставленной задаче
2. Метод эффективнее, чем строчный, при числе процессов $> 4000\text{-}5000$
3. Правила трансформации ограничены только применяемым методом
→ негативная черта строчного метода устранена!

- Дальнейшие действия:

- Поддержка новых системных вызовов/функций/аспектов на практике
- Апробация работы интеграцией с CRIU («CRIUGen»).
- Разработка других методов: «семантика как синтаксис» – грамматики с контекстами, мультикомпонентные грамматики надстройки деревьев
- Статистический анализ деревьев процессов, выработка метрик, оптимизация последовательностей системных вызовов.

Публикации и другая литература:

Опубликованы:

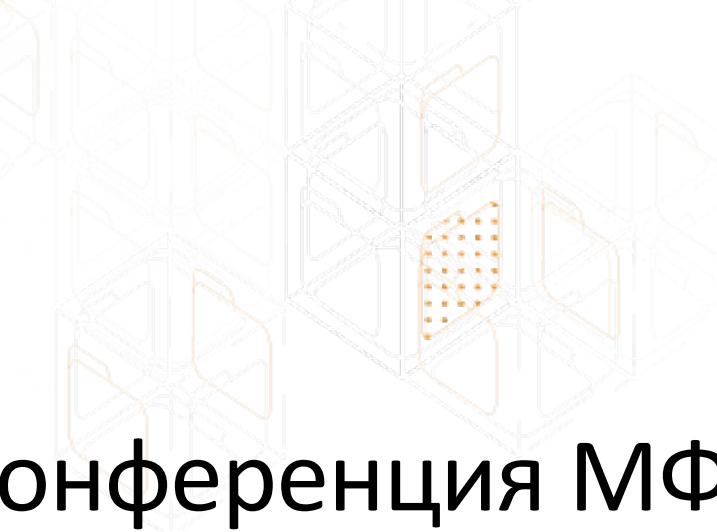
- 1) Ефанов Н. Н., Емельянов П.В. Построение формальной грамматики системных вызовов // М. Информационное обеспечение математических моделей, кафедра МОУ МФТИ, 2017 – 8 С.
- 2) Nikolay Efanov and Pavel Emelyanov. 2017. Constructing the formal grammar of system calls. In Proceedings of the 13th Central & Eastern European Software Engineering Conference in Russia (CEE-SECR '17). ACM, New York, NY, USA, Article 12.
- 3) Ефанов Н.Н. Комбинаторные и групповые свойства деревьев процессов Linux. Сборник трудов XV международной конференции «Алгебра, теория чисел и дискретная геометрия: современные проблемы и приложения», Тула, 28–31 мая 2018 г., С.180-183.
- 4) Ефанов Н.Н. Классификация правил проверки атрибутов в деревьях процессов Linux // "Естественные и технические науки", №6 2018 г., С.216-221.
- 5) N.N. Efanov, E.S. Shtypa. 2018. Optimization of syscall sequences using minimal spanning trees search // Труды IX Московской международной конференции по исследованию операций (ORM2018-GERMEYER100), Москва, С. 12-18.

Приняты к публикации (конец 2018-начало 2019):

- 1) Ефанов Н.Н. «О некоторых комбинаторных свойствах деревьев процессов Linux», Чебышевский сборник
- 2) N. Efanov, P. Emelyanov. 2018. Linux Process Tree Reconstruction Using The Attributed Grammar-Based Tree Transformation Model. In Proceedings of CEE-SEC(R)'2018, Moscow, Russian Federation, 7 pages.

Прочие источники и полезные ссылки:

- 1) CRIU: https://criu.org/Main_Page
- 2) On Theory of Graph Transformation Schemes: <http://www.informatik.uni-trier.de/~ley/db/conf/gg/handbook1997.html>



61-я научно-практическая конференция МФТИ

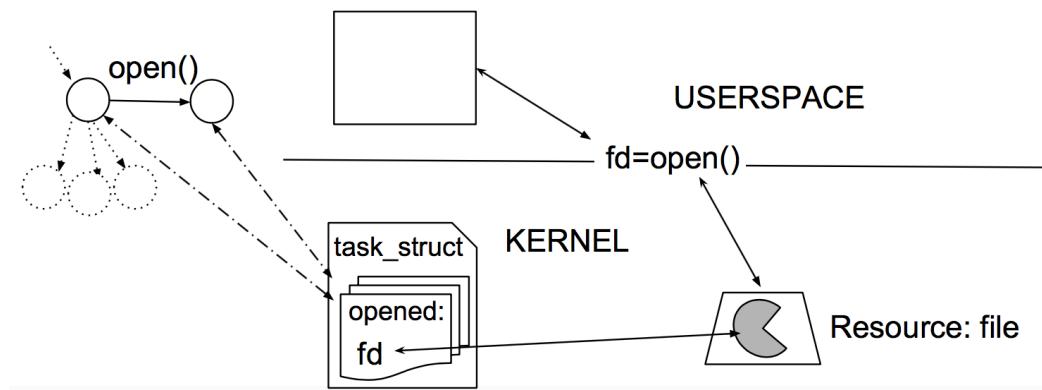
November 22
Moscow/Dolgoprudnyi

Об NP-полноте некоторых задач, связанных с
восстановлением дерева процессов

Николай Ефанов

Конфигурация системы

- Под конфигурацией системы подразумеваем дерево процессов + набор ресурсов системы R , ссылки на которые содержатся в дереве процессов как ссылки через значения атрибутов (описание). Ссылки также доступны через /procfs в момент снятия снапшота.



- Вопрос, является ли произвольное дерево деревом процессов существенно зависит от:
 - 1) вида дерева: вопрос решается частичным порядком, заданным на дереве процессов
 - 2) состоянием ресурсов и взаимосвязями между ними
- Задача: можно ли определить корректность входного дерева, не восстанавливая структуры промежуточных состояний процессов?**
 - Под структурой понимается не только состояния, но и порядок их достижений

CurrentConf:

- Вход: 1) дерево процессов $T: \{V, E\}$, каждая вершина v содержит $H=\{h\}$ – атрибутное описание ресурсов, используемых некоторым процессом; начальное состояние I .
 - T соответствует состоянию системы во время снятия снапшота.
 - Атрибутному описанию соответствуют выделенные системные ресурсы, то есть неявно реализуется схема $\{h, r\}$, r – ресурс из множества ресурсов R , отношение многие к одному.
 - $action$ – пара $\{pid, \text{системный вызов}\}$ из некоторого множества A .
 - $flag=0$, если ресурс наследован или присвоен, 1 – если создан текущим процессом.
- 2) Список действий, для которых проверяется получение дерева T из I .
 - булева формула F над множеством пар $\{flag, action\}$, соответствующая выполнению действий, приводящих из I к дереву процессов T . Считается предварительно построенной.
- Выход: набор пар $X' = \{ \{flag, action\} \}$, выполняющий восстановление T из I , то есть таких что, $F(X')=1$.
- Соответственно, если набор X' не найден, дерево не является корректным (см. Т.1)

Задача выполнимости булевых формул

- Существует ли такой набор значений переменных $X=\{\{0,1\}\times\dots\times\{0,1\}\}$, что булева формула F над ними обращается в истину на данном наборе?
- Не теряя общности (полный базис $\vee, \&, \neg$), формула представима в КНФ.
- Пример:

$$X = \{x_i, i = 1..4\} \mid F(X) = (\neg x_1 \vee \neg x_2 \vee \neg x_3) \& (x_2) \& (\neg x_2 \vee x_3) \& (\neg x_2 \vee \neg x_3 \vee \neg x_4)$$

Решение: набор $X'=\{0,1,1,0\}$: $F(X') = 1$

- 3-SAT: КНФ, в каждом конъюнкте не более 3 дизъюнктов (см. пример выше)
- SAT сводится к 3-SAT (доказывается заменами переменных).
- SAT является NP-полной (теорема Кука-Левина(1971)).
- → 3-SAT КНФ – NP-полна.

Полиномиальная сводимость (по Карпу)

- Пусть A, B - суть два языка. Тогда A сводится к B , если:

существует всюду определённая функция $f: \{0, 1\}^* \rightarrow \{0, 1\}^*$, вычислимая за полиномиальное время, такая что $x \in A \Leftrightarrow f(x) \in B$. Обозначение: $A \leqslant_p B$. (Индекс p означает полиномиальность).

- В нашем случае: CurrentConf \rightarrow язык наборов корректных пар {flag, action}
 - 3-SAT \rightarrow язык выполняющих наборов {0,1}
- Требуется свести 3-SAT к CurrentConf, тем самым доказывается NP-трудность.
- CurrentConf принадлежит классу NP – следует из AGGTM, Теорема 1: подали корректное дерево, порождённое некоторой корректной цепочкой действий
- Если CurrentConf принадлежит классу NP и NP-трудна, то она NP-полна.

Лемма о совместности операций

- Множество операций, необходимых для восстановления дерева процессов, описывается разрешающим набором для задачи CurrentConf без ограничений на виды конъюнктов.
- Доказательство:** Представим некоторые действия из CurrentConf, допускающие альтернативы при работе с ресурсами, в виде дизъюнкций.
- Рассмотрим возможные значения флагов над действиями (+симметрия):
 $(action_i \vee action_j \vee action_k)$ – хотя бы 1 из ресурсов создан процессом
 $(action_i \vee action_j \vee !action_k)$ – хотя бы 1 из 2-х ресурсов создан, либо наследован 3-й.
 $(action_i \vee !action_j \vee !action_k)$ – 1-й создан, либо хотя бы 1 из 2,3 наследован
 $(!action_i \vee !action_j \vee !action_k)$ – хотя бы 1 из ресурсов наследован
- Возможная ситуация:**
 $(action_i \vee action_j \vee action_k) : mmap(addr1), mmap(addr2), mmap(addr3)$
 $(action_i \vee action_j \vee !action_k) : open(file1), open(file2),$ наследованы при *fork*
 $(action_i \vee !action_j \vee !action_k) : pipe(fd1, fd2),$ наследование *fd1* или *fd2*
 $(!action_i \vee !action_j \vee !action_k) : setpgid(0, pgroup), pgroup = pgid_i | i = (1|2|3)! = pid$

Лемма о совместности операций

- (Продолжение доказательства)
- Возможная ситуация (*):

$(action_i \vee action_j \vee action_k) : mmap(addr1), mmap(addr2), mmap(addr3)$

$(action_i \vee action_j \vee !action_k) : open(file1), open(file2)$, наследованы при *fork*

$(action_i \vee !action_j \vee !action_k) : pipe(fd1, fd2)$, наследование *fd1* или *fd2*

$(!action_i \vee !action_j \vee !action_k) : setpgid(0, pgroup), pgroup = pgid_i | i = (1|2|3) != pid$

- Может ли одно действие в (*) входить в разные дизъюнкты с разными флагами?

$!mmap(addr)$, : да – запрет на *mmap* в данную область памяти

$!open(file1)$ – *Permission Denied*

$!pipe(fd1, fd2)$ – *EMFILE*, *ENFILE* – лимит открытых дескрипторов

$!setpgid(0, pgroup)$ – лидер сессии

- Таким образом, продемонстрировано покрытие всех возможных вариантов флагов в формуле *F*. Q.E.D.

Сведение 3-SAT к CurrentConf

- (\Rightarrow) Дано произвольная 3-КНФ от переменных X , получим $f(X)$ за $O(P(n))$:
 - Считаем, что сведение производится к $CurrentConf(I, T', A')$, где A' содержит действия из Леммы 1 или аналогичные им, и хотя бы некоторое подмножество вершин из T' соответствует выполнению этих действий.
 - **Алгоритм для $f(\dots)$:**
 - Просмотрим последовательно каждый конъюнкт в исходной 3-КНФ.
 - Для каждой переменной x из X : x входит без отрицания ? ($flag(x==1:0)$) $\rightarrow return(x, flag)$. По Лемме 1, флаги могут быть любыми $\rightarrow f(\dots)$ всюду определена.
 - Поиск действия для x : просмотрим список возможных действий из A , которые реализуют условия внутри конъюнкта (перебором за $O(|A|^3)$), получим метки действий \rightarrow описание.
 - Сложность: $O(C^*|A|^3) - Time$, $|A|$ - число действий, C – число конъюнктов. $=> O(P(n))-Time$
- (\Leftarrow) Образим метки действий, перепишем формулу, так что $\forall x: x=(flag==1?1:0)$
 - По Лемме 1, описание $CurrentConf$ -формулой может содержать все значения флагов над необходимыми действиями из A .

Q.E.D.

Основные результаты

- Деревья процессов в общем случае эффективно не распознаются без анализа и восстановлению промежуточных состояний, а главное, их порядка воспроизведения.
 - Иногда такой порядок не следует явно из самого дерева, нужно строить граф зависимостей, добавлять и объединять состояния процессов, топологически сортировать такие домены, и только потом запускать восстановление. Причём новое состояние может быть вписано в уже, казалось бы, восстановленный домен на топологически отсортированном графе (см. Пример ‘CRIU fails on...’)
- Нельзя эффективно (за полином) сгенерировать все возможные деревья процессов без применения вывода системными вызовами (за $\exp [2]$).
- CurrentConf → В класс NP-полных задач
- На практике, NP-полнота не так страшна:
 - Заведомо корректные деревья процессов восстанавливаются за полином
 - Процедура восстановления структурно отличается от CurrentConf внутренним анализом
 - Никто не отменял эвристики!

Благодарю за внимание!

Контактная информация:

- Email: nefanov90@gmail.com
- Phone:+7(916)0911108

Hidden2: The AGTTM analyzer: DFS routine with extra DFS / upbranch traversal checks

Algorithm 1:

(Input: $IR, D, expr$; Output: T)

begin:

```
for any Node in dfs(T.root):
    cond = f(expr(Node.S,I [1..m])) # m - attr
    if cond not in IR: # context check
        cond=context_checking()
        ...
D= TR( $D, Ex, IR, k(cond)$ )
for child in Node.children:
    dfs(child)
return D
```

Theorem 2 (on time complexity). The worst-case time complexity of Algorithm 1 is $O(n^2)$ for n nodes in the input tree, if all of traverse type-dependent checks can be performed via single *upbranch* and optionally single *dfs* routines.

Subprogram 1:

context_checking():

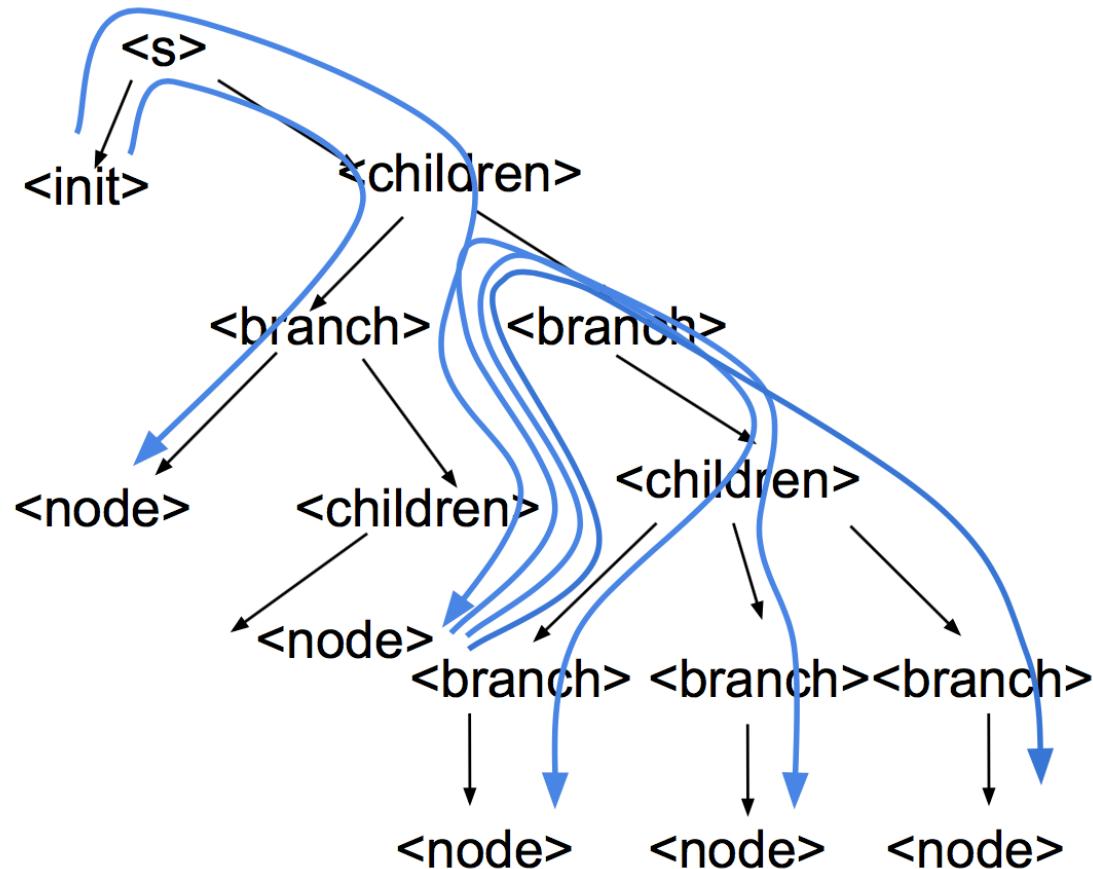
```
    cond = upbranch({expr(Node, Current)})
    if not cond in IR:
        cond = dfs ({expr(Node, Current)})
    return cond
```

Q.E.D.

The main recap:

Total time complexity is quasi-square of nodes number n

Дерево разбора $\{A, N, P, \langle s \rangle\}$



$$E = E_{<node>}^{<branch>} \cup E_{<branch>}^{<children>} \cup E_{<children>}^{<branch>} \cup E_{<node>}^{<node>},$$
$$E = E_{<init>}^{<s>} \cup E_{<s>}^{<children>} \cup E_{<children>}^{<branch>} \cup E_{<branch>}^{<node>}$$

Дерево разбора {A, N, P,<s>}

