Система автоматизации поиска логических ошибок в исходных текстах с поддержкой объектных конструкций языка PHP

Автор: Храмченко В.О.

классификация методов анализа кода

1. Анализ кода вручную.

2. Автоматический анализ:

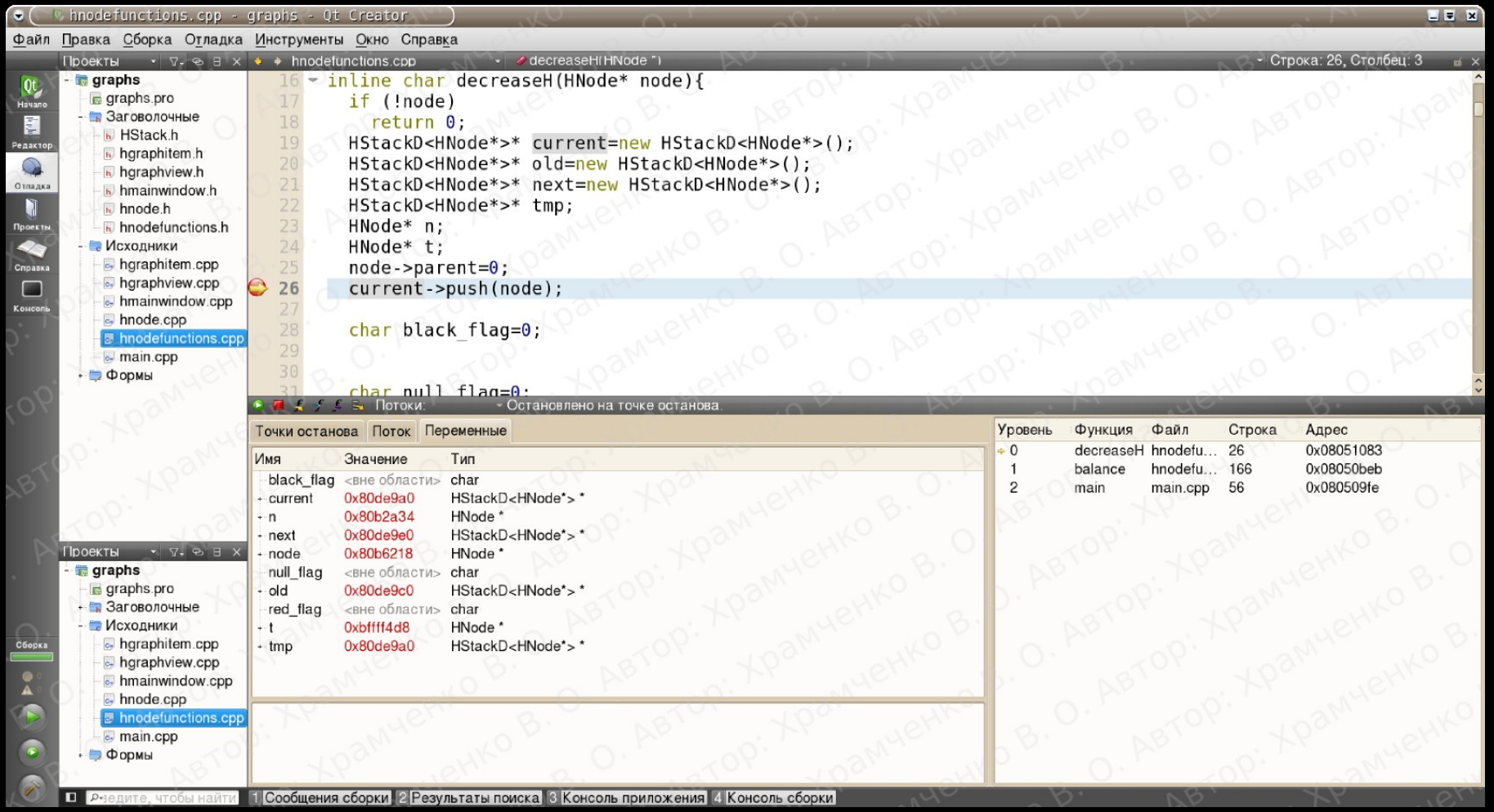
   Статический анализ:

   1. Поиск уязвимостей по шаблону.
   2. Глубокое исследование кода с использованием графов потоков состояний.

   Динамический анализ:

   1. Фаззинг.
   2. Анализ кода, посредством его интерпретации.

просмотр исходного кода



+ Более качественный анализ.

+ Возможность поиска нетипичных ошибок.

- Большая продолжительность по времени.

поиск уязвимостей по шаблону

```
...
if (strstr(code, «eval»)){
   printf («Warning!»);
}
...
```

+ Простота реализации.

- Возможность поиска только простейших уязвимостей.

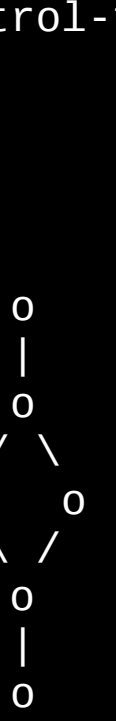
Пример: RATS.

исследование с помощью графов потоков данных

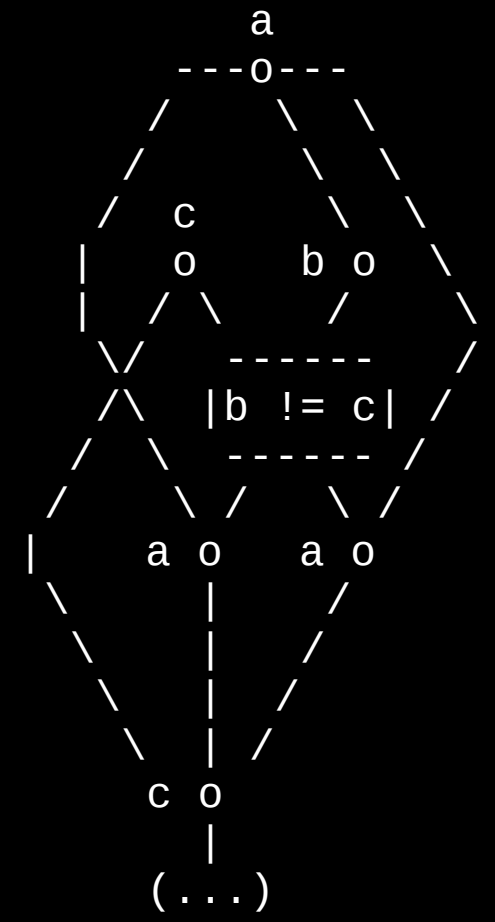
Code

```
c = 21;
b = a;
a = 42;
if (b != c)
  a++;
else
  a--;
c += a;
-------
```

Control-flow



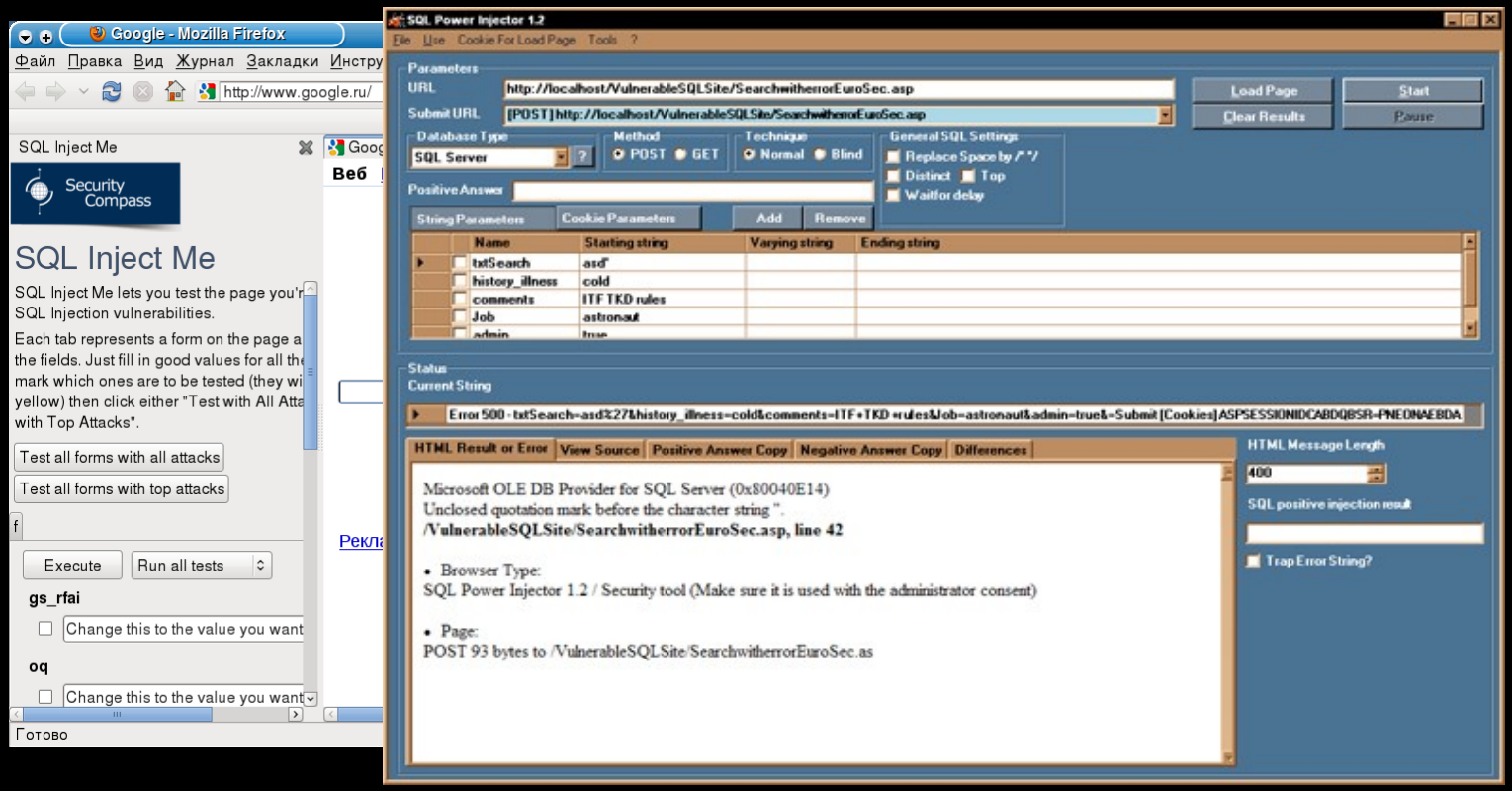
Data-flow with predicates



Участок кода (code), и соответствующие ему, графы потоков состояний(Control-flow) и данных(Data-flow).

Пример: Pixy.

фаззинг



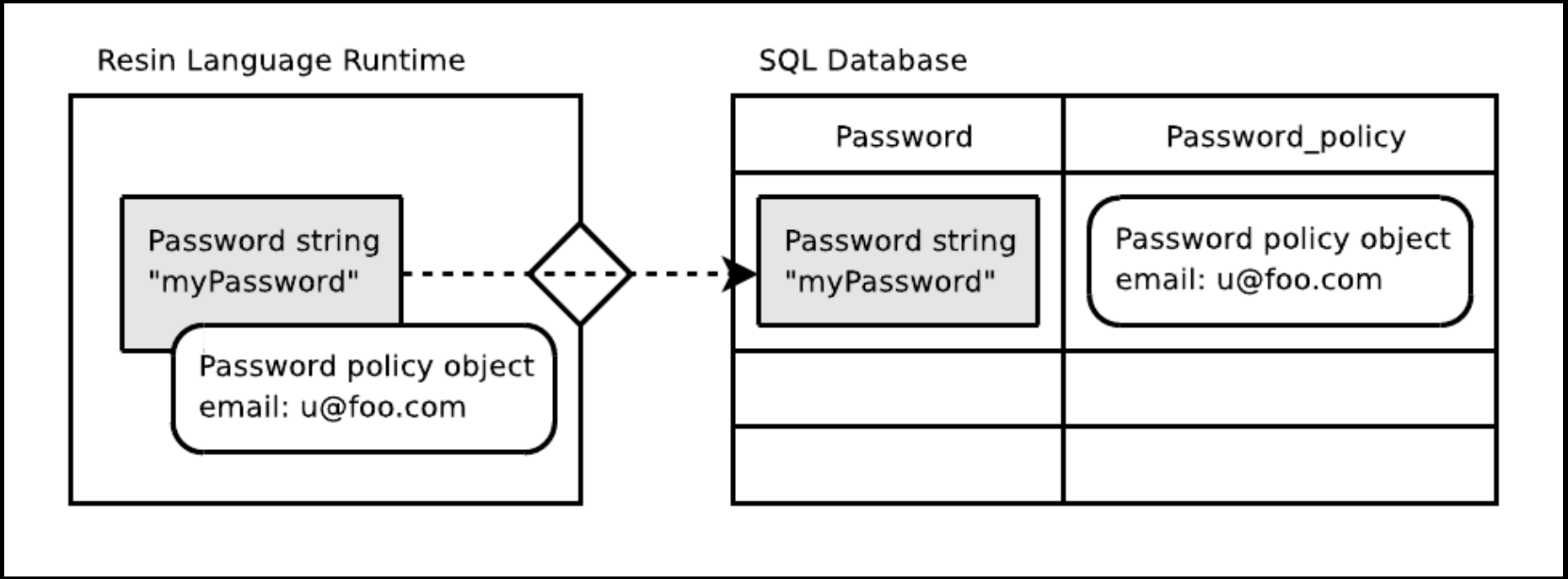
+ Относительная простота.

+ Имитирует действия злоумышленника.

- Способен найти лишь уязвимости, не скрытые внутри циклов и условных переходов.

Примеры: SQL Inject Me, XSS Inject Me, SQL Power Injector.

специальные интерпретаторы



+ Производят глубокое исследование программного кода.

- Высокая сложность создания таких систем.

- Необходимость создания новых политик, если в приложении используются сторонние библиотеки.

Пример: RESIN.

постановка задачи

Автоматические методы:

+ Высокая скорость работы.

- Обнаруживают лишь часть уязвимостей.

Анализ кода вручную:

+ Позволяет найти ошибки, которые невозможно обнаружить при автоматическом анализе.

- Большая продолжительность по времени.

Цель: Создать интегрированную среду для поиска уязвимостей веб-приложений, предоставляющую дополнительную информацию о работе программы, в том числе промежуточные результаты сканирования, с возможностью корректировки процесса сканирования экспертом.

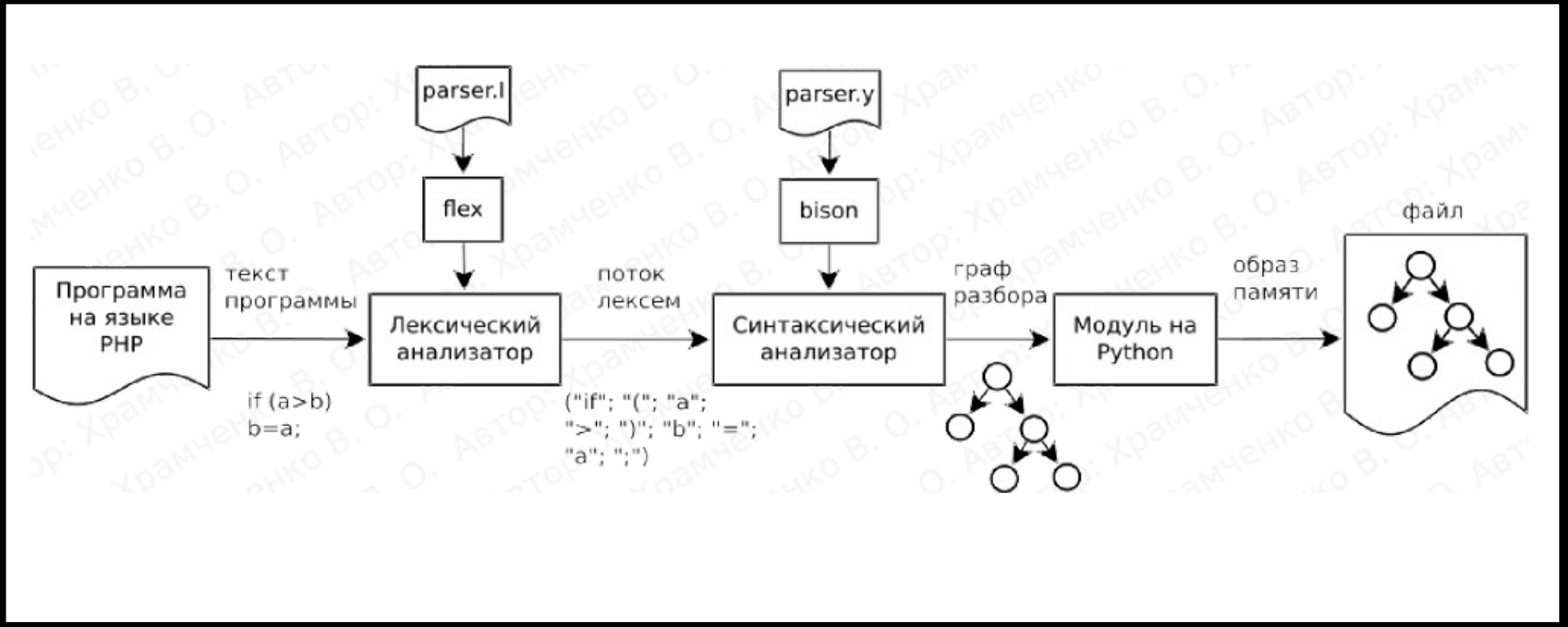
АРХИТЕКТУРА СИСТЕМЫ

архитектура системы

Интегрированная среда анализа программного кода должна состоять из следующих модулей:

1. Модуль разбора кода.
2. Модуль предварительного упрощения кода и определения значений переменных.
3. Модуль определения структуры баз данных.
4. Модуль определения критически важных участков кода.
5. Модуль отображения результатов анализа, и предоставления интерфейса пользователя.

модуль разбора кода

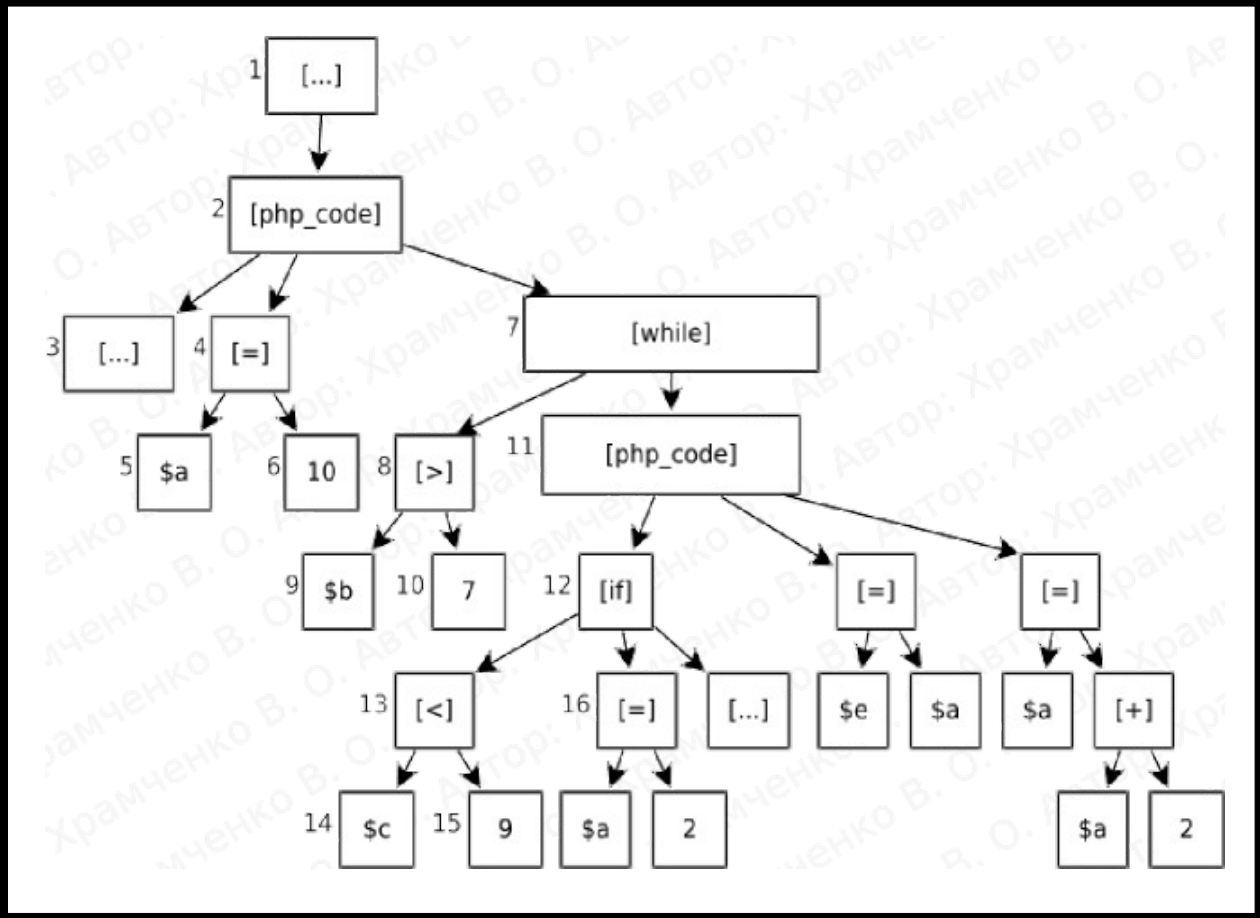


Процесс построения графа разбора кода.

модуль разбора кода

```
  ...
 1 $a=10;
 2 while ($b > 7){
 3 ...
 4 if ($c < 9){
 5  $a=2;
 6  ...
 7 }
 8 $e=$a;
 9 $a+=2;
10 }
  ...
```

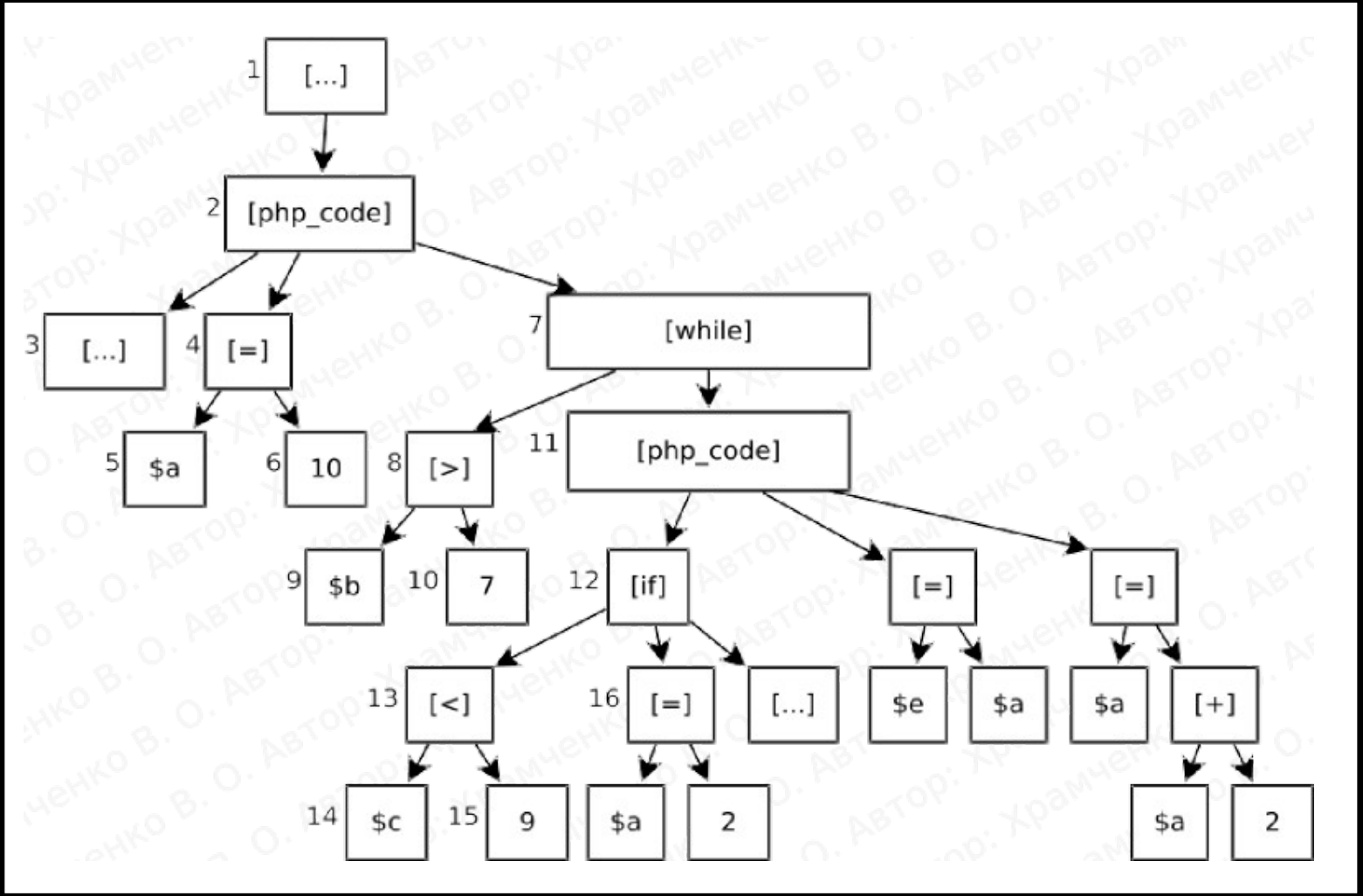
(a)



(b)

Участок кода (a) и соответствующее ему дерево разбора (b).

начальный этап преобразования графа



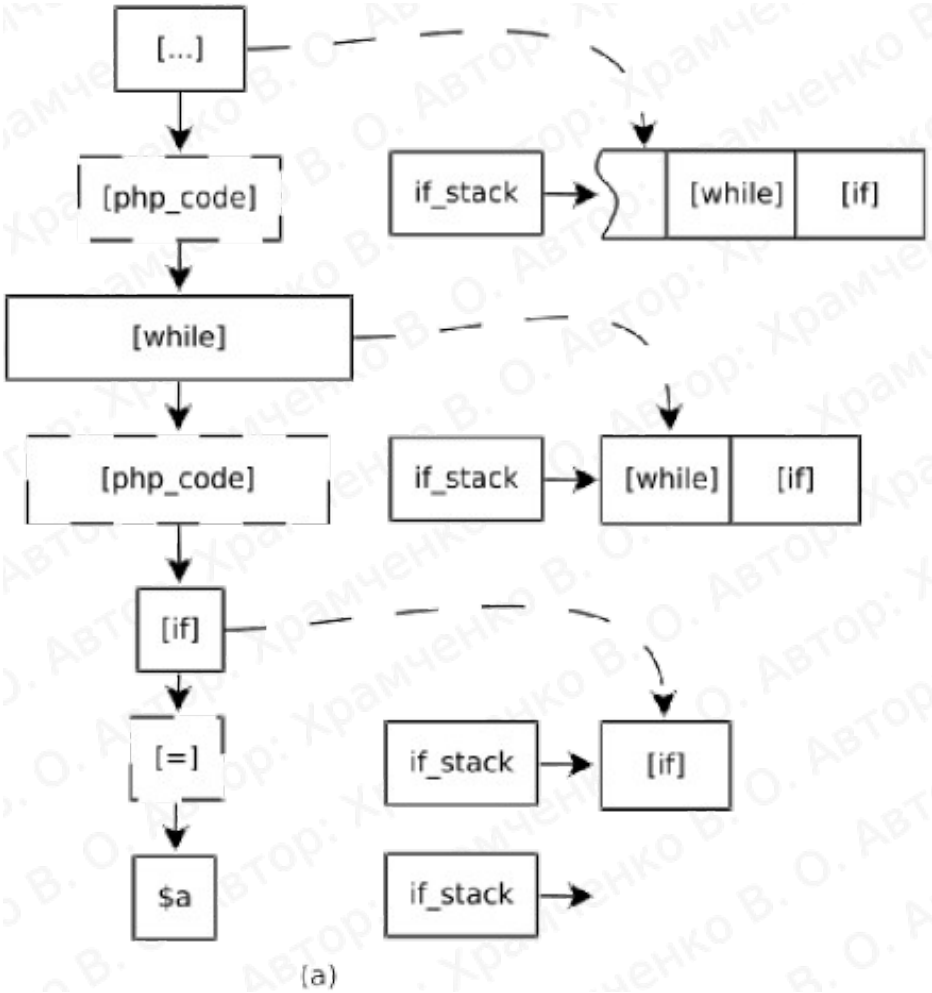
Дерево разбора (порядок обхода указан слева).

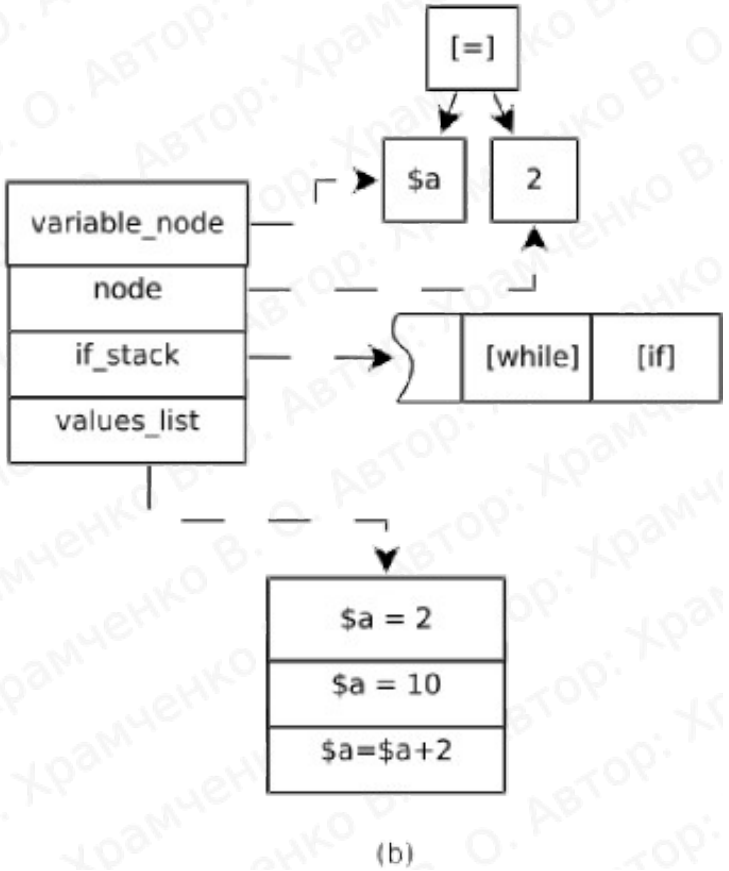
определение значений переменных

```
  ...
 1 $a=10;                  // Безусловное присваивание.
 2 while ($b > 7){
 3 ...
 4 if ($c < 9){
 5  $a=2;                  // Присваивание при условии $c < 9.
 6  ...
 7 }
 8 $e=$a;                  // $a == ?
 9 $a+=2;                  // Увеличение на каждой итерации.
10 }
  ...
```

Проблема определения значения переменной.

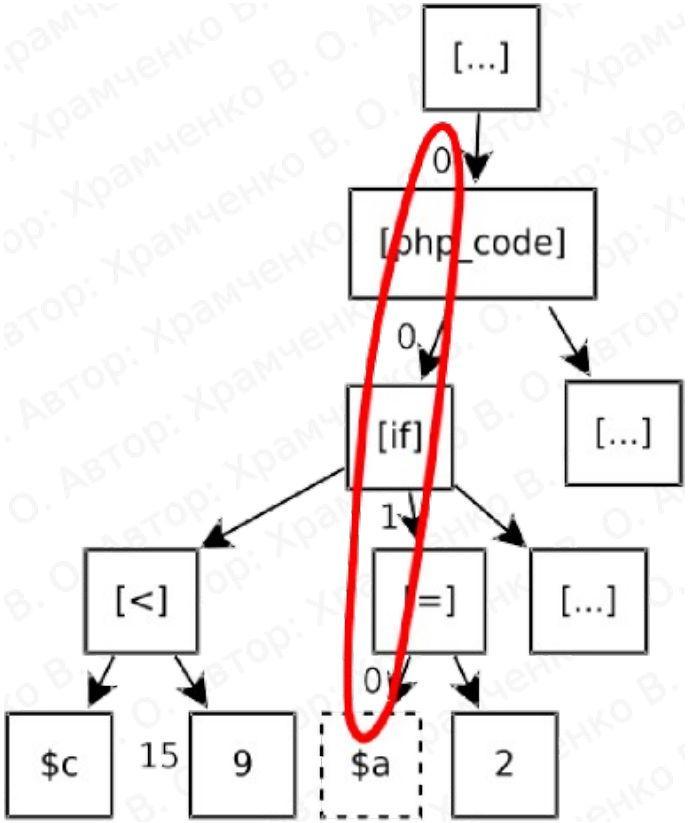
определение значений переменных





Создание стека условий (a) и структура переменной (b).

определение значений переменных



Введём операцию кодирования: $code(A): A \to (a_n, \dots, a_0)$, где $a_0, \dots, a_n \in \mathbb{Z}$, $A \in V$ и V - множество всех вершин дерева, вектор $(a_n, \dots, a_0)$ строится по следующему правилу:

1.изначально он пуст,

2.двигаясь по дереву начиная от вершины, код которой хотим узнать, к корню дерева, будем дописывать позицию текущего элемента в начало.

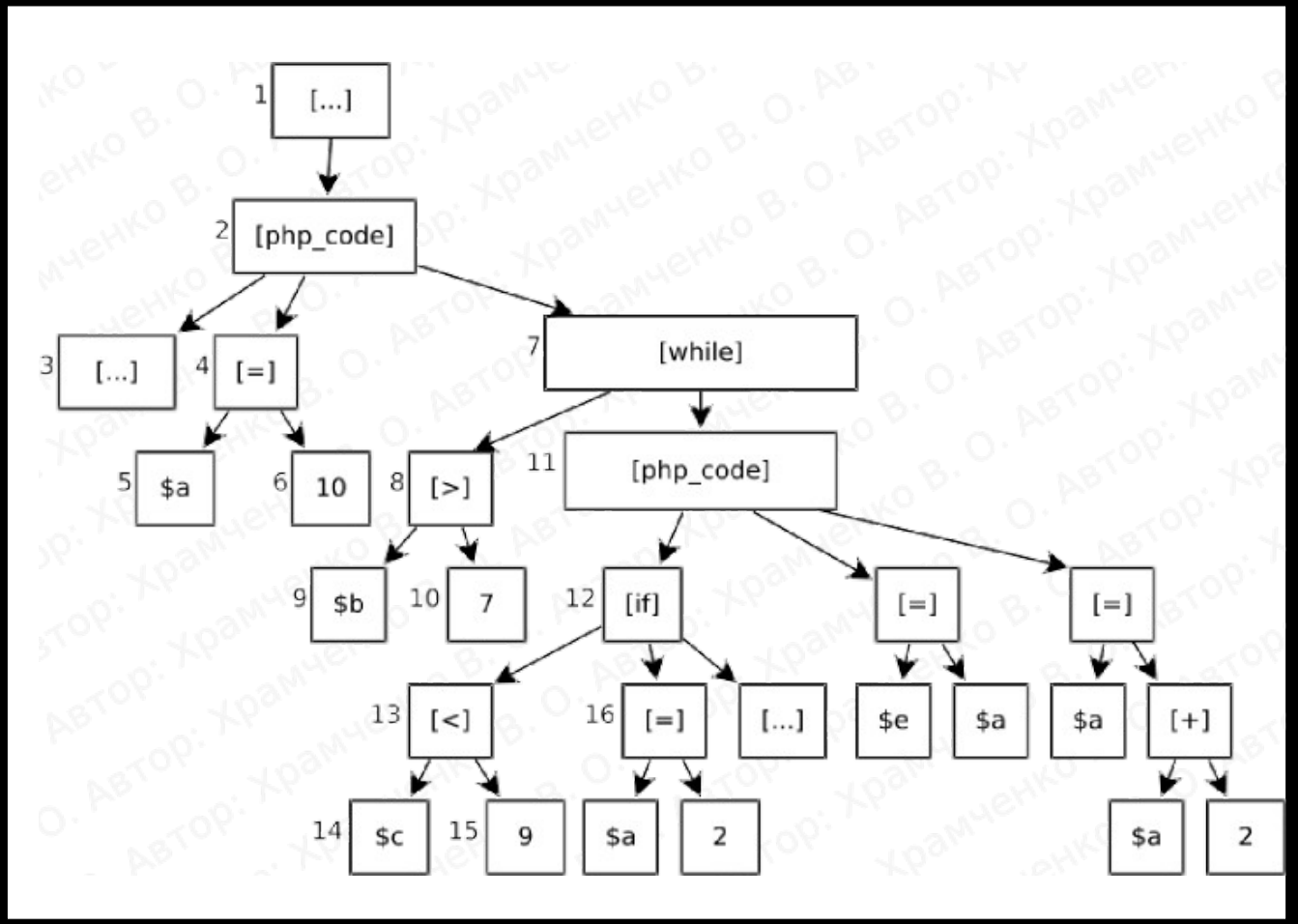
Пример: .

$$code(\$a) = (0, 1, 0, 0)$$

Алгоритм кодирования позиции узла.

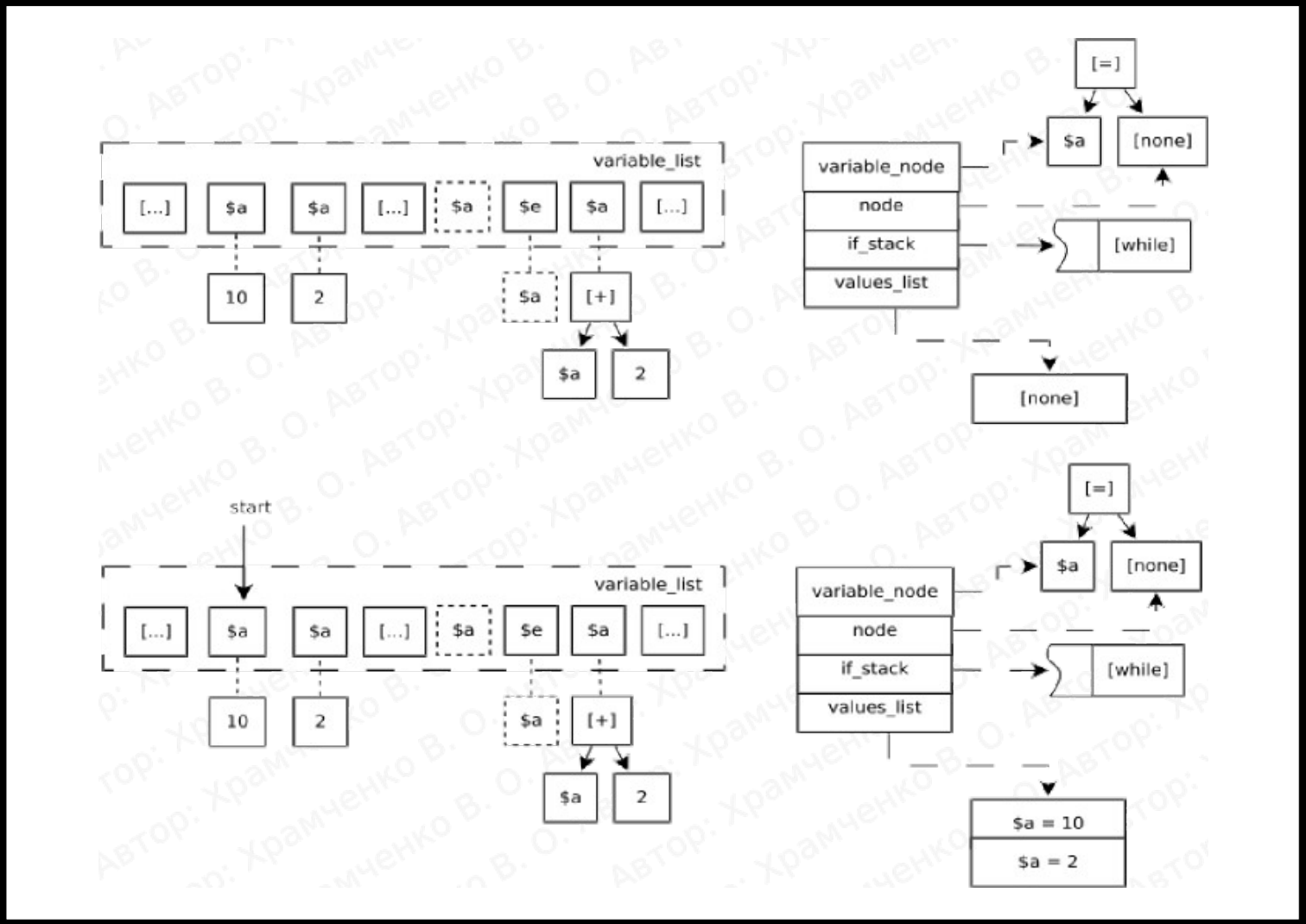
определение значений переменных

```
  ...
 1 $a=10;
 2 while ($b > 7){
 3 ...
 4 if ($c < 9){
 5  $a=2;
 6  ...
 7 }
 8 $e=$a;
 9 $a+=2;
10 }
  ...
```



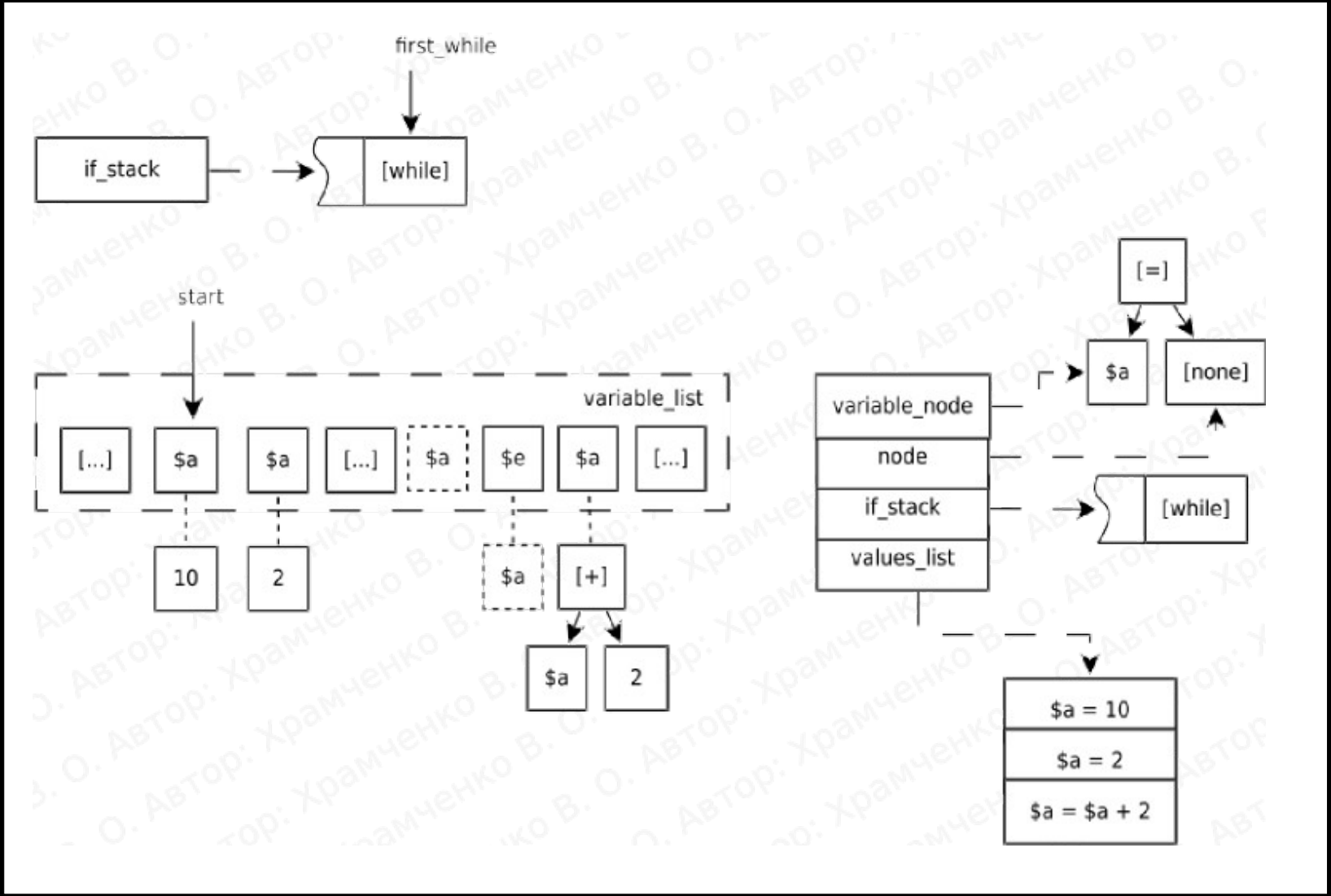
Практическое значение введённых операций.

определение значений переменных



Вставка переменной в список и поиск первого безусловного присваивания.

определение значений переменных



Дополнительная обработка переменных, находящихся внутри циклов.

дополнительные зависимости между переменными

```
$a = new Test();
$b = $a;
$b->x = 4; // $a->x = 4;
$b=new Text();
$b->x = 1; // $a->x=4;
```

```
function f($b, &$c){
   $b->x = 10;
   $c = new Test();
   $b = new Test();
   $b->x = 15;
   return $c->k;
}
$b = new Test();
$c = new Test();
$c->h = 11;
$k=f($b, $c); /* $c->h=0;
$b->x=10; */
$k->l=11; // $c->k->l=11;
```

Проблемы, возникающие при анализе функций, обращений к методам и свойствам объектов.

ВЫВОДЫ

В ходе данной работы была предложена система, обладающая следующими преимуществами:

1. Определение зависимостей значений переменных от истинности условий.

2. Контроль изменения значений переменных внутри функций, в том числе, с поддержкой вложенных вызовов.

3. Поддержка объектных конструкций языка PHP.

4. Установление зависимостей переменных, хранящими ссылки на один и тот же объект, и автоматическое применение модификации поля одной из них, ко всем остальным.

5. Простота адаптации для других языков программирования с динамической типизацией.

6. Упрощение математических и строковых выражений.

ВЫВОДЫ

Направления дальнейшего развития системы автоматизации поиска логических ошибок в исходных текстах:

1. Добавление модуля работы с базами данных, позволяющего определять их структуру, выявлять критически важные поля, и осуществлять контроль за их использованием.
2. Интеграция интерфейса к отладчику.
3. Предоставление функций удобного управления проектами.
4. Создание модуля составления отчётов о результатах анализа

СПАСИБО ЗА ВНИМАНИЕ