Управляемый запросами статический анализ для языка Ruby

Квалификационная работа на соискание степени специалиста

Выполнил: Калугин М.Б., ММФ, гр. 4113

Руководитель: доцент, с.н.с. к.ф.-м.н. И.Н.Скопин

Ruby

- Популярный динамический язык
  - Императивный, функциональный, объектно-ориентированный, рефлективный
- Ruby on Rails
- Проблемен для разработчиков IDE (до сих пор не существует решения)

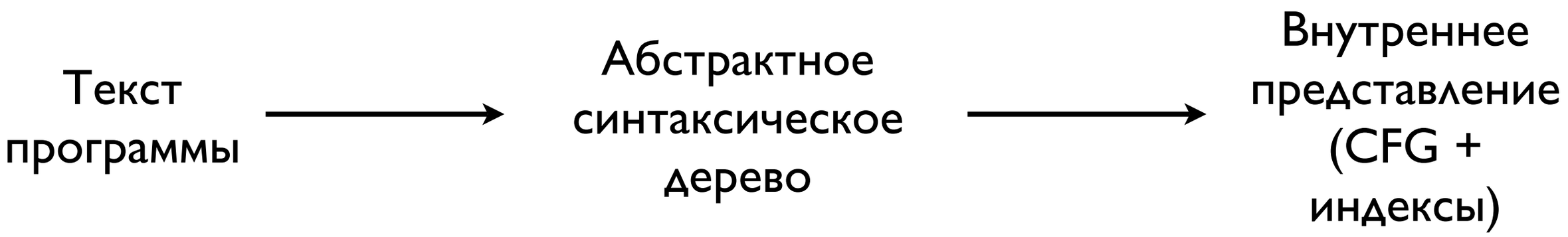
Задача

- Вход: программа на языке Ruby с указанным в ней выражением или переменной
- Выход: описание типа данного выражения или переменной

Задача

- Способность анализатора отвечать на “вопросы”
- Время 1-2 сек на вопрос (в идеале, время не зависит от размера программы)
- Иметь возможность контролировать время и точность работы анализатора
- Анализатор должен эффективно работать в условиях постоянно изменяющейся программы

Решение

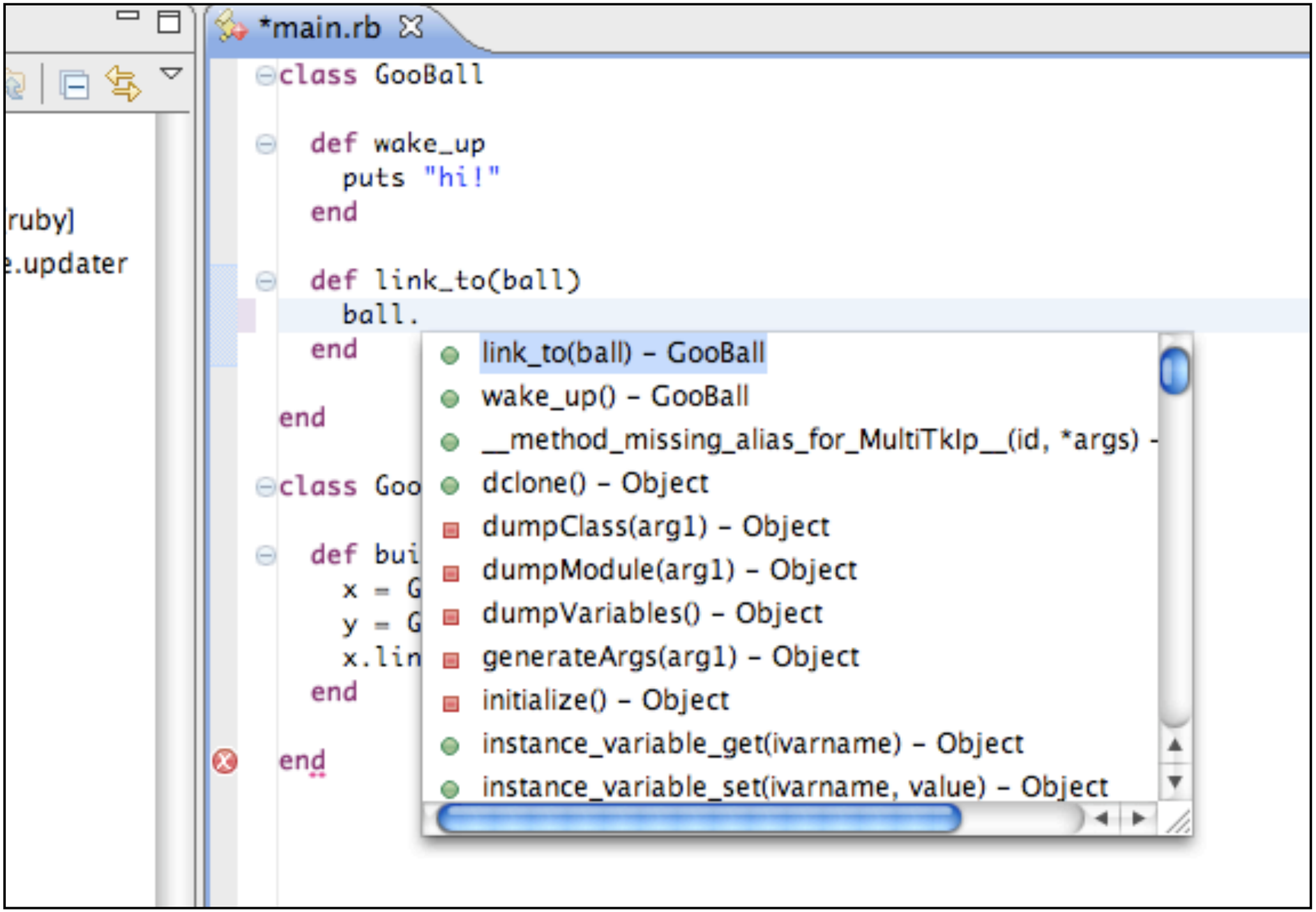


- Управляемый запросами анализ с отсечениями (DDP)
- + инкрементальное обновление закешированных запросов

Практика

Программа	LoC + ~5k	Процент точных ответов	Время работы (всего)
address-tool	600	43%	~0
levenshtein	200	36.6%	~0
ujudge	5k	~20-30%	1 sec/query

Проблемы: информация о встроенных функциях и коллекции


*main.rb
class GooBall
def wake_up
puts "hi!"
end
def link_to(ball)
ball.
end
end
class Goo
def bui
x = G
y = G
x.lin
end
end
ruby]
.updater
link_to(ball) - GooBall
wake_up() - GooBall
__method_missing_alias_for_MultiTkIp__(id, *args) -
dclone() - Object
dumpClass(arg1) - Object
dumpModule(arg1) - Object
dumpVariables() - Object
generateArgs(arg1) - Object
initialize() - Object
instance_variable_get(ivarname) - Object
instance_variable_set(ivarname, value) - Object

Результаты

- исследован, доработан и реализован универсальный алгоритм DDP в качестве отдельной системы
- разработан и реализован анализ для языка Ruby на основе DDP (две версии: DLTK Ruby + сейчас)
- полученный анализ контекстно-зависим (аргументы) и потокозависим внутри процедур, но при этом масштабируем
- рассмотрены проблемы получившихся реализаций и понята дальнейшая работа

Q/A

Время работы

- Потенциально очень долго — поставленная задача сложная, точная оценка будет бессмысленна
- На самом деле быстро — большинство вещей в программе вычисляется моментально
- Если не быстро, то отсекаем

Что анализ не может?

```
arr = [proc ..., proc ...] # A = Array[Proc]
...
foo(a)
    a[0].call() # oops
```

Что анализ не может?

```
d = [0 => "string", 1 => 52]
...
bar(hash) {
    return hash[0] # oops
}
```

Что анализ не может?

- Невычисленные eval() и define_method()
- Нетривиальные встроенные методы и нативные библиотеки
- Исключения и продолжения(continuations)

Корректность и полнота?

- Анализатор выдает результат в сложном виде: Тип1 или Тип2 или ... — решение остается за пользователем
- Корректный или полный результат в реальной жизни не возможен

Корректность и полнота?

все нормально

eval string

здесь мы никому не можем верить

Корректность и полнота?

- как правило ничего страшного не случается
- в подавляющем числе случаев переменные в программе принимают значения одного единственного типа.
- пользователь не дурак и знает про свои трюки