



НАКОПЛЕНИЕ ЛЕГКОВЕСНОЙ СЕМАНТИЧЕСКОЙ ИНФОРМАЦИИ ПО СИНТАКСИЧЕСКОМУ ДЕРЕВУ

Волошин Б.И.
студент 4 курса 9 группы

Постановка задачи

- Выяснить, какую семантическую информацию на первом этапе компиляции считать **легковесной**
- Выяснить, какие задачи анализа и преобразования синтаксических деревьев может решить компилятор, не проводя полного семантического анализа
- Предоставить для манипулирования легковесной семантикой и решения наиболее распространённых задач на этапе преобразования синтаксического дерева в семантическое

Легковесная семантика




Для чего нужна легковесная семантика

- Поиск некоторых ошибок компиляции
- Реализация синтаксически сахарных конструкций
- Intellisense

Вопросы, на которые отвечает легковесная семантика

- **Какие поля содержит класс**
- **Какие процедуры и функции
вызываются в программе**
- **Содержит ли пространство имён
переменную с указанным именем**
- **Какой тип имеет переменная
(частично)**
- **...**

Этапы решения



Построение
математической
модели

Реализация
модели

Реализация API

Используемая математическая модель

- Для работы с легковесной семантикой нами использована математическая модель, описанная в статье **Representing Concerns in Source Code** Мартином Робиллардом и опубликованная в журнале **ACM Transactions on Software Engineering and Methodology (TOSEM)**
- В статье модель использовалась для другой цели
- Модель основана на реляционной алгебре

Пример: программа и её модель

Mapping Function J1

$E = \{x \mid \text{IsAClass}(x)\}$

$\text{names}(N) = \{\text{Declares}, \text{Extends}, \text{SuperclassOf}, \text{SubclassOf}\}$

$a(\text{Declares}, P) = \{(x, y) \mid \text{Declares}(x, y)\}$

$a(\text{Extends}, P) = \{(x, y) \mid \text{Extends}(x, y)\}$

$a(\text{SubclassOf}, P) = a(\text{Extends}, P)^+$

$a(\text{SuperclassOf}, P) = a(\text{SubclassOf}, P)^\top$

```
public class A
{
    int aField;
    class B {}
}
```

```
class C extends A
{
    void aMethod() {};
    class D extends A {}
    class E extends D {}
}
```

Модель
конкретной
программы

Model $P1_{J1}$

$E_{P1} = \{A, B, C, D, E\}$

$\text{Declares}_{P1} = \{(A, B), (C, D), (C, E)\}$

$\text{Extends}_{P1} = \{(C, A), (D, A), (E, D)\}$

$\text{SubclassOf}_{P1} = \{(C, A), (D, A), (E, D), (E, A)\}$

$\text{SuperclassOf}_{P1} = \{(A, C), (A, D), (D, E), (A, E)\}$

Математическая модель

$E = \{x \mid \text{IsAProcedure}(x) \wedge \text{IsAFunction}(x) \wedge \text{IsAVaribale}(x) \wedge \text{IsACustomType}(x) \wedge \text{IsAStructure}(x) \wedge \text{IsAClass}(x) \wedge \text{IsAnInterface}(x) \wedge \text{IsAField}(x) \wedge \text{IsAMethod}(x) \wedge \text{IsANamespace}(x)\}$

$\text{names}(N) = \{\text{Accesses}, \text{AccessedBy}, \text{Calls}, \text{CalledBy}, \text{Contains}, \text{ContainedBy}, \text{Creates}, \text{Declares}, \text{HasParameterType}, \text{HasReturnType}, \text{I}, \text{Implements}, \text{ImplementedBy}, \text{OfType}, \text{Overrides}, \text{OverridenBy}, \text{PartialName}, \text{FullName}\}$

$a(\text{Accesses}, P) = \{(x, y) \mid \text{Accesses}(x, y)\}$

$a(\text{AccessedBy}, P) = a(\text{Accesses}, P)^T$

$a(\text{Calls}, P) = \{(x, y) \mid \text{Calls}(x, y)\}$

$a(\text{CalledBy}, P) = a(\text{Calls}, P)^T$

$a(\text{Contains}, P) = \{(x, y) \mid \text{Contains}(x, y)\}$

$a(\text{ContainedBy}, P) = a(\text{Contains}, P)^T$

$a(\text{Creates}, P) = \{(x, y) \mid \text{Creates}(x, y)\}$

$a(\text{Declares}, P) = \{(x, y) \mid \text{Declares}(x, y)\}$

$a(\text{HasParameterType}, P) = \{(x, y) \mid \text{HasParamterType}(x, y)\}$

$a(\text{HasReturnType}, P) = \{(x, y) \mid \text{HasReturnType}(x, y)\}$

$a(\text{I}, P) = \{(x, y) \mid x = y\}$

$a(\text{Implements}, P) = \{(x, y) \mid \text{Implements}(x, y)\}$

$a(\text{ImplementedBy}, P) = a(\text{Implements}, P)^T$

$a(\text{OfType}, P) = \{(x, y) \mid \text{OfType}(x, y)\}$

$a(\text{Overrides}, P) = \{(x, y) \mid \text{Overrides}(x, y)\}$

$a(\text{OverridenBy}, P) = a(\text{Overrides}, P)^T$

$a(\text{PartialName}, P) = \{(x, y) \mid \text{PartialName}(x, y)\}$

$a(\text{FullName}, P) = a(\text{PartialName}, P)^T$

Типов элементов программы - 10
Отношений - 18

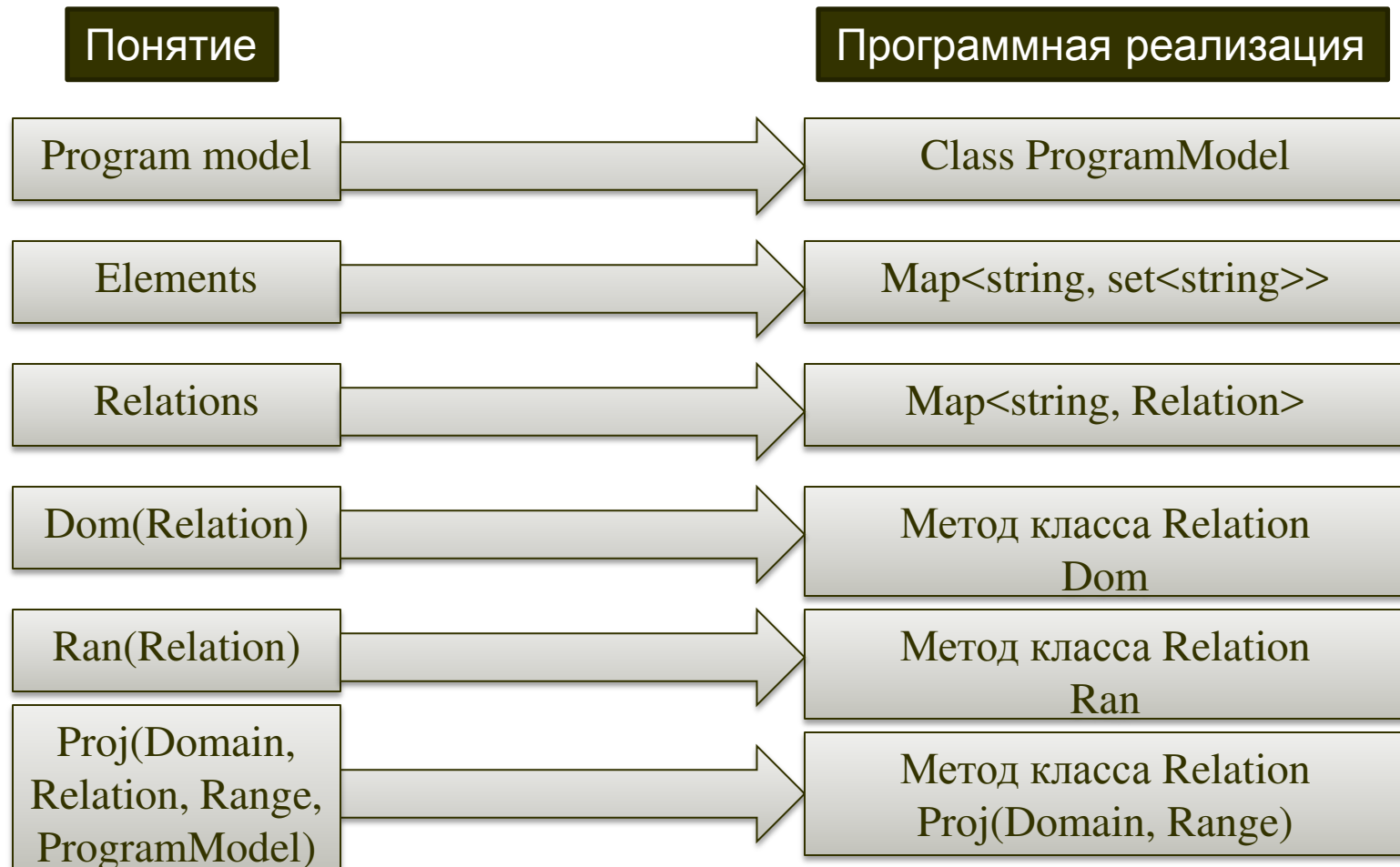
Преимущества

- **Однообразное хранение и доступ ко всей семантической информации, избавляемся от сложных таблиц символов и иерархий классов**
- **Можно построить составной запрос (например, вызываемые методами А процедуры)**
- **Модель конкретизирует и классифицирует задачи анализа синтаксического дерева**

Этапы решения



Программная реализация модели



Пример: запросы к программной модели

**Какие поля содержит класс
Program.Declares.Proj(className, Program.Fields)**

**Какие процедуры вызываются в программе
Program.Calls.Proj(Global, Program.Procedures)**

**Содержит ли пространство имён переменную с
указанным именем**

**Program.Declares.Proj(
Program.PartialName(variableName, Program.Variables).Ran(),
nameSpace)**

Этапы решения



Структура библиотеки

- Визитор по синтаксическому дереву отвечает за сбор семантической информации (в используемой теории соответствует понятию Mapping Function)
- InformationContainer хранит минимальный контекст и по запросам визитора изменяет конкретную программную модель
- Программная модель хранит легковесную семантику

Пример: Intellisense по первым символам

**Задача: Получить все имена, доступные
в пространстве имён nameSpace**

// Сбор информации

```
SpecificProgramModel program;
```

```
VisitorCollector visitor = new VisitorCollector(); visitor.visit(tree);
```

```
program = visitor.program;
```

// Сбор пространств имён

```
SortedSet<string> nameSpaces = new SortedSet<string>();
```

```
nameSpaces.Add(nameSpace);
```

```
SortedSet<string> nstmp = new SortedSet<string>(); nstmp.Add(nameSpace);
```

```
do {
```

```
    nstmp = program.Contains.ProjRan(nstmp);
```

```
    nameSpaces.AddRange(nstmp);
```

```
} while (nstmp.Count != 0);
```

// Получение всех имён

```
return program.PartialNames.ProjRan(program.Declares.ProjDom(nameSpaces).Ran()).Dom();
```


Пример: Intellisense по первым символам

● Накапливание информации в визиторе

```
public override void visit(var_def_statement _var_def_statement) {  
    base.visit(_var_def_statement);  
    foreach (var variable in _var_def_statement.vars.idents)  
        information.nameController.AddVariable(variable.name);  
}  
public override void visit(procedure_definition _procedure_definition) {  
    procedure_header _procedure_header = _procedure_definition.proc_header;  
    function_header _function_header = _procedure_header as function_header;  
    if (_function_header != null)  
        information.nameController.AddFunction(_procedure_header.name.meth_name.name);  
    else  
        information.nameController.AddProcedure(_procedure_header.name.meth_name.name);  
  
    information.nameController.PushNameSpace(_procedure_header.name.meth_name.name);  
    if (_function_header != null)  
        information.nameController.AddVariable("Result");  
    foreach (var param in _procedure_header.parameters.params_list)  
        foreach (var variable in param.idents.idents)  
            information.nameController.AddVariable(variable.name);  
    base.visit(_procedure_definition);  
    information.nameController.PopNameSpace();  
}
```

Пример: Intellisense по первым символам

● Непосредственная работа с МОДЕЛЬЮ

```
public void PushNameSpace(string nameSpace) {  
    string fullName = curNS + "." + nameSpace;  
    program.NameSpacesElems.Add(fullName);  
    program.NameSpacesElems.Add(nameSpace);  
    program.PartialNamesRel.AddRelation(nameSpace, fullName);  
    program.ContainsRel.AddRelation(curNS, fullName);  
    curNS = fullName;  
}  
public void PopNameSpace() {  
    curNS = program.ContainsRel.ProjRange(curNS).Dom().First();  
}  
public void AddVariable(string name) {  
    string fullName = curNS + "." + name;  
    program.VariablesElems.Add(fullName);  
    program.VariablesElems.Add(name);  
    program.PartialNamesRel.AddRelation(name, fullName);  
    program.DeclaresRel.AddRelation(curNS, fullName);  
}
```

Результаты работы

- **Выделены основные семантические элементы и отношения**
- **Найдена оптимальная модель хранения и взаимодействия с легковесной семантической информации**
- **Реализовано API для манипулирования легковесной семантикой и решения наиболее распространённых задач на этапе преобразования синтаксического дерева в семантическое**

Ссылка на материалы работы

- <https://github.com/voloshinbogdan/LightSemantic>

Список литературы

- **Alfred V. Aho, Monica S. Lam, Ravi Sethi, and Jeffrey D. Ullman. Compilers: Principles, Techniques, and Tools, Second Edition. 2006.**
- **Martin P. Robillard. Representing Concerns in Source Code. 2003.**