УДК 004.05

ПОСТРОЕНИЕ МОДЕЛИ ПРОГРАММЫ НА ЯЗЫКЕ GO

О. С. Крюков

Тульский государственный университет, пр. Ленина, 92, 300012, г. Тула, Россия, ol_kryukov97@mail.ru

Рассматривается проблема верификации и построения модели исходного кода программы на языке Go. Приводится описание представления программы в формате абстрактного синтаксического дерева (АСД). Предложен обобщённый алгоритм построения на основе АСД расширенной сети Петри с семантическими связями, пригодной для последующего анализа.

Ключевые слова: параллелизм, статическая верификация, модель, абстрактное синтаксическое дерево, сеть Петри, язык Go.

BUILDING PROGRAM MODEL IN GO LANGUAGE

O. S. Kryukov

Tula State University, ave. Lenin, 92, 300012, Tula, Russia, ol_kryukov97@mail.ru

The problem of verification and construction of a model of the source code of a program in the Go language is considered. A description of the program representation in the abstract syntax tree (AST) format is given. A generalized algorithm for constructing an extended Petri net with semantic connections based on ASD, suitable for subsequent analysis, is proposed.

Keywords: parallelism, static verification, model, abstract syntax tree, Petri net, Go language.

Язык программирования Go нашел широкое применение в сфере высокопроизводительных вычислений, в особенности там, где возможно и необходимо применение параллельного исполнения. При этом проблема надежности программного обеспечения становится все более важной. Однако, для её решения зачастую недостаточно имеющихся средств верификации, так анализ параллельных программ представляет довольно сложную и плохо верифицируемую задачу. Потенциальным решением является применение статической верификации, выполняющей анализ программного обеспечения путем проверки соответствия её модели набору определенных правил [1]. И очевидной проблемой данного решения является необходимость выбора достаточно точной модели и формирование ее на основе исходного кода программы.

Любой вычислительный процесс подчиняется определенным правилам и условиям, направленным на достижение заданной алгоритмом цели. Алгоритм определяет последовательность типовых операций, в большинстве случаев, связанных друг с другом, поэтому при формировании модели параллельного алгоритма в первую очередь необходимо составить схемы связей операций. Алгоритм реализуется в программном коде путём ввода множества операторов в определенной последовательности.

Выделяется два вида зависимости операторов: по управлению и по семантическим связям. Связь по управлению отображает последовательный процесс перехода от одного оператора к другому. Семантические связи определяют использование результатов одного или нескольких операторов в ходе исполнения другого.

Наиболее простым способом разбора исходного кода для его последующего анализа является построение абстрактного синтаксического дерева (*далее* – АСД). Абстрактное синтаксическое дерево – конечное помеченное ориентированное дерево, в котором внутренние вершины сопоставлены с операторами языка программирования, а листья – с соответствующими операндами. Таким образом, листья являются пустыми операторами и представляют только переменные и константы.

В языке Go АСД является базовым инструментом для работы с исходным кодом. Используя данное представления, возможно построение различных статических анализаторов и инструментов, позволяющих в автоматическом режиме вносить некие изменения в код. Структурно любая программа на языке Go в представлении АСД имеет следующий вид:

1) информация о программном пакете, содержащем исходный код, и описывают расположение ключевого слова package в файле, расположение названия файла и непосредственно название;
2) различные объявления внутри файла, например, функции;
3) дополнительная информация о файле и его содержимом.

Для анализа наибольший интерес представляет именно вторая часть, так как содержит в себе непосредственно модель исходного кода. И так как функция представляет собой основной элемент программы на языке Go, опишем именно её представление в формате АСД.

В формате АСД любая функция языка Go состоит из трёх блоков:

1) Name – описывает идентификатор, представляющий собой имя функции;
2) Type – описывает прототип функции, в том числе и параметры функции;
3) Body – описывает тело функции, сохраняя последовательность операторов.

Основными элементами Type и Body в АСД, являются объекты ast.Ident, описывающие идентификаторы, объекты ast.Stmt, описывающие операторы, и объекты ast.Expr, описывающие выражения и их операнды [2]. Типы ast.Stmt и ast.Expr представляют собой интерфейсы, описывающие основные требования к оператору и выражению, и, в зависимости от реализации, могут принимать различный вид, например ast.ReturnStmt – оператор return; и ast.BinaryExpr.

Несмотря на очевидные достоинства АСД, такая модель не в полной мере отвечает заявленным целям, поскольку хотя операции и соблюдают порядок вызова, то есть при последовательном выполнении возможно учесть зависимости по управлению, однако при параллельном исполнении данные зависимости уже не просматриваются, а также практически полностью отсутствует информация о семантических зависимостях, существующая в АСД языка Go только в формате ссылки на место объявления идентификатора операнда. Тем не менее данное представление может быть использовано как промежуточное представление при построении необходимой модели, так как предоставляет исчерпывающую информацию об исходном коде, что нашло своё отражение в некоторых работах [3, 4].

С нашей точки зрения более подходящим вариантом представления исходного кода программы в виде модели является расширенная сеть Петри с семантическими связями (РСПСС) [5], способная одновременно отображать как зависимости по управлению, так и семантические связи между операциями. Преобразование АСД в РСПСС в наиболее общем виде можно описать следующим алгоритмом.

1. Для каждой объявляемой как ast.Ident переменной создаём позицию сети.

2. Для каждого ast.Stmt и ast.Expr выполняем анализ и построение сети.

2.1. Если оператор является ast.GoStmt, то в данном месте формируется переход по управлению типа Fork, на входе которого формируется позиция, представляющая оператор go.

2.2. Тип перехода по управлению Join создается либо перед оператором типа ast.ReturnStmt, если сейчас рассматривается параллельное исполнение, на выходе которого формируется позиция, представляющая оператор return.

2.3. Тип перехода по управлению Synchro создается перед вызовом метода Wait() для идентификатора типа sync.WaitGroup, и, в зависимости от числа потоков, вызывающих Wait(), создаётся равнозначное количество позиций на выходе перехода.

2.4. Для прочих операторов и выражений при формировании позиций и зависимостей по управлению соблюдается порядок аналогичный объявлению.

2.5. Для семантических связей на входе устанавливаются операнды выражения, а на выходе – сформированная позиция, представляющая выражение.

Получаемая в результате модель позволяет учесть больше различных аспектов выполнения параллельной программы. Дальнейшая работа направлена на разработку методов обнаружения взаимоблокировок и гонок данных в параллельных программах.

Библиографический список

1. Гурин Р. Е., Рудаков И. В., Ребриков А. В. Методы верификации программного обеспечения // Машиностроение и компьютерные технологии. 2015. № 10. С. 235–250.
2. Пакет языка Go «go/ast». URL: https://pkg.go.dev/go/ast (дата обращения: 07.10.24).
3. Wang C., Sun H., Xu Y. [et al.]. Go-Sanitizer: Bug-Oriented Assertion Generation for Golang // 2019 IEEE International Symposium on Software Reliability Engineering Workshops (ISSREW). Берлин, Германия, 2019. С. 36–41.
4. Fu S., Liao Y. Golang Defect Detection based on Value Flow Analysis // 2024 9th International Conference on Electronic Technology and Information Science (ICETIS). Ханчжоу, Китай, 2024. С. 358–363.
5. Крюков О. С. Статический метод обнаружения гонок данных на основе анализа модели расширенной сети Петри // Оптико-электронные приборы и устройства в системах распознавания образов и обработки изображений. Распознавание-2023: сб. материалов XVII Междунар. науч.-техн. конф. 2023. С. 131–133.