Программные системы и вычислительные методы
Правильная ссылка на статью:
Дагаев Д.В. — Ограничительная семантика языка в системе МультиОберон // Программные системы и вычислительные методы. – 2023. – № 1. DOI: 10.7256/2454-0714.2023.1.36217 EDN: IWIODR URL: https://nbpublish.com/library_read_article.php?id=36217

Ограничительная семантика языка в системе МультиОберон

Дагаев Дмитрий Викторович

Главный Эксперт, АО "РАСУ"

115230, Россия, г. Москва, шоссе Каширское, 3, корп. 2, стр.16

dvdagaev@oberon.org

Статья из рубрики "Языки программирования"

DOI:

10.7256/2454-0714.2023.1.36217

EDN:

IWIODR

Дата направления статьи в редакцию:

03-08-2021

Аннотация: Язык и системы на базе Оберон в реализации демонстрируют минималистский подход для достижения надежности, существенного отличающийся от большинства программных систем, стремящихся к максимизации числа поддерживаемых функций. Требования к критически важным системам для АЭС категории А запрещают использование еще большего числа практик программирования. Для соответствия требованию категории А стабильного числа итераций устанавливается запрет использования операторов цикла по условию. Для обеспечения эргодичности используется запрет использования динамической памяти и рекурсии. Уязвимость типа переполнения буфера закрывается запрещением модуля системных операций SYSTEM. Ограничения могут быть установлены на выявление проблемы хрупкого базового класса, операции смены типа и использование вложенных процедур. Отмечено, что переход к диалекту Оберон-07 в основном коснулся дополнительных ограничений и хорошо встраивается в рамки ограничительной семантики. Вместо языков и диалектов под каждый набор требований автором предложен подход ограничительной семантики, при котором используется один язык с системой ограничений. В язык введен один оператор RESTRICT как декларация ограничений на данный модуль. Реализован компилятор МультиОберон с одним фронтендом, включающем систему ограничений, и несколькими сменными бэкендами. Продемонстрирован синтаксический анализ

компилятора на примерах. Показана стратегия масштабирования компилятора в зависимости от системы требований. Новизной подхода ограничительной семантики является достижение набора минимально необходимых свойств, соответствующих требованиям к системе. Использование разработчиками систем подхода «от ограничений» является преимуществом, т.к. декларирует действительно необходимые свойства системы, увязанные с требованиями.

Ключевые слова:

Оберон, Компонентный Паскаль, эргодичность, надежность ПО, ограничение, компилятор, синтаксическое дерево, семантический анализ, модульность, хрупкий базовый класс

Введение

Язык программирования и система Оберон [1] реализовали крайне минималистский подход, при котором из ПО исключались все функции, не являющиеся жизненно необходимыми. Этот вектор движения сохранился во всех последующих реализациях Оберонов, что является их существенным отличием от общепринятого подхода к построению программных систем, при котором объем заявленных функций является необходимой коммерческой особенностью каждого нового продукта.

Не только проект Оберон развивался в сторону уменьшения функциональности. Аналогичные подходы применяются в формировании требований к системам для АЭС, выполняющим функции категории А [2]. Это – наивысшая категория в части критически важного ПО, применяемая для систем защит реакторов. ПО категории А должно гарантировать не наличие, а отсутствие многих стандартных решений, применяемых программистами. В данном ПО необходимо отсутствие стандартной ОС и библиотек сторонних разработчиков, за исключением сертифицированных для категории А. Требование неизменности времени прогона вне зависимости от входных данных приводит к исключению циклов с переменным числом итераций и выходом по условию. Требование защиты содержимого памяти приводит к запрету выделения динамической памяти во время работы и контроля. Известные реализации ПО категории А используют специализированные под данные требования языки и компиляторы, например, язык ΨС [3].

Описанные подходы демонстрируют, что требования влияют существенным образом на программную реализацию систем. При ужесточении требований может возникнуть проблема структурных изменений в ПО. Более того, при очень жестких требованиях может измениться и язык программирования реализации, и компилятор. Это, естественно, приведет не к доработке существующей, а к разработке новой системы.

Автором предложен подход **ограничительной семантики**, который состоит в следующем: **вместо нескольких языков и компиляторов использовать один язык и фронтенд компилятора с системой семантических ограничений**.

Ниже данный подход будет детально описан, равно как и программная реализация на языке Оберон в системе МультиОберон.

Все программные примеры будут представлены в Паскале-подобном синтаксисе, данная разработка продолжает традиции школы Никлауса Вирта в части разработки языков

Паскаль/Модула/Оберон.

Оператор RESTRICT как декларирование ограничений

Механизм ограничительной семантики очень прост – используется только оператор RESTRICT, декларирующий систему ограничений для данного программного модуля.

RESTRICT -, -*, +, ..;

где - представляет собой некоторую лексему, которую компилятор интерпретирует как дополнительное ограничение для данного модуля (знак '-' означает исключить лексему). В свою очередь, + представляет собой включение имеющейся функциональности, которая исключена по умолчанию. Область действия ограничения – программный модуль. Оберон является модульным языком программирования, где модуль находится в одном файле и является единицей компиляции, загрузки и выполнения. Следовательно, ограничения распространяются только на данный программный модуль. Если необходимо систему ограничений распространять на несколько программных модулей, то создается файл ограничений с внешней областью видимости (знак '*'), который необходимо импортировать в каждый программный модуль. Например, оператором IMPORT RestrictIEC880 импортируются все ограничения, записанные в модуле RestrictIEC880 как внешние. Например, если в нем записаны RESTRICT –WHILE*, -LOOP*, это запрети использование операторов WHILE, LOOP.

При несоответствии кода модуля системе ограничений не должен проходить семантический анализ, и, как следствие, выдаваться ошибка компиляции.

Ограничение числа итераций цикла

Допустим, к разрабатываемой прикладной системе реального времени предъявлено требование ограничения числа итераций цикла. В стандарте [2] это сформулировано как «время прогона не должно существенно изменяться в результате изменения входных данных». Это означает, что в тексте разрабатываемой программы должны быть только циклы FOR с постоянным числом итераций. Следовательно, откомпилированный бинарный код данного модуля также будет с фиксированным числом итераций, вне зависимости от конфигурации или условий запуска данного кода. Ограничительная семантика в части постоянных итераций цикла будет означать запрет остальных типов циклов.

RESTRICT –WHILE, -REPEAT, -UNTIL, -LOOP;

В результате останется только цикл FOR j := 1 TO N-1 DO .. END, который имеет постоянное число итераций N, вне зависимости от внешних условий. Но даже для цикла FOR есть ряд случаев, когда от внешних условий может зависеть параметр N. Это может привести к варьированию времени прогона. В таком случае следует ограничить использование пределов цикла FOR константами, т.е. запретить переменные для пределов.

RESTRICT –FOR(VAR);

Обработка данного ограничения компилятором требует уровня семантического анализа. В отличие от запрещения ключевого слова (например, WHILE) целиком, исключение пределов цикла-переменных приводит к необходимости анализа синтаксического дерева [4] с учетом семантической информации о типах. Рисунок 1 иллюстрирует дерево для проведения подобного анализа.

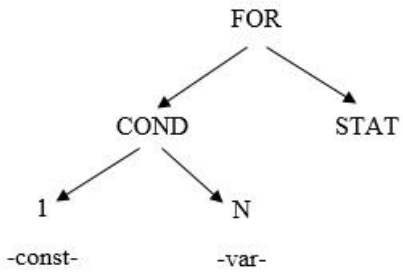

Рис. 1. Абстрактное синтаксическое дерево цикла FOR

Fig.1 Abstract Syntax Tree of FOR cycle

При установленном ограничении –FOR(VAR) компилятор должен активировать проверки токена предела FOR->COND->N данным ограничением. Поскольку токен предела на рисунке является переменной, то проверка завершается неудачей, и компилятор выдает сообщение об ошибке.

Время прогона модуля может зависеть также от времени прогона процедур других, импортируемых модулей. Например, может быть такой вызов процедуры OtherModule.LongCalculations(). Это означает, что в модуле OtherModule требования ограничения числа итераций циклов могут не выполняться. Следовательно, аналогичным образом необходимо рассматривать и модуль OtherModule. Если семантические ограничения на модуль не установлены, то отсутствует подтвержденная декларация о выполнении требований.

Ограничение динамической памяти

В работе [5] показано, как свойства компьютерной системы в части памяти могут со временем деградировать, и что важным критерием является эргодичность, т.е. стабильность этих свойств, что требует доказательств, в том числе и архитектурного характера. Там же показано, что нужно исходить из презумпции неэргодичности, т.е. предполагать, что характеристики разрабатываемого ПО могут деградировать со временем. Система семантических ограничений служит хорошим инструментом такого доказательства эргодичности.

Для многих систем реального времени нужен запрет выделения динамической памяти. Внутри модуля это осуществляется ограничением встроенной функции NEW.

```
RESTRICT –NEW;
```

Это означает, что в данном модуле непосредственно функция NEW быть вызвана не может. При этом аллокирование динамической памяти может быть вызвано функцией из другого, импортируемого модуля, например, фабричного Factory.New(). Помимо рассмотрения ограничений импортируемого модуля, можно запретить использование указателей.

```
RESTRICT –NEW, -POINTER;
```

Тогда проверка не будет пройдена и завершится синтаксической ошибкой.

```
VAR ptr: POINTER TO FactoryElem;

ptr := Factory.NewElem;
```

Следующим способом обойти ограничение будет использование библиотечных функций malloc, calloc, strdup и т.д. Данные функции возвращают безтэговый (untagged) указатель на память, который может потом преобразовываться в используемый в Обероне указатель с помощью модуля SYSTEM. Для исключения такого рода указателей требует запрета использования модуля SYSTEM.

RESTRICT –NEW, -POINTER, -SYSTEM;

Итоговое условие ограничения динамической памяти модуля будет иметь все три составляющие.

Ограничение стековой памяти

В Оберон-системах стековая память фиксируется для каждого процесса и не может быть увеличена впоследствии. Выход за пределы стековой памяти может происходить при глубокой рекурсии в используемом ПО. Следствием выхода за пределы является фатальная ошибка памяти. Этого нужно избежать. Запрет использования рекурсии выглядит следующим образом.

RESTRICT –PROCEDURE(PROCEDURE);

При установленном ограничении -PROCEDURE(PROCEDURE) производится анали синтаксического дерева на предмет выявления рекурсий. Выявленные случаи приводят к синтаксической ошибке.

Способом обойти эти ограничения может стать принудительная установка указателя стека за счет регистровых операций, например, SYSTEM.PUTREG(SP, address). Во избежание этого нужно запретить функцию PUTREG, работающую непосредственно с регистрами.

Другим способом обойти ограничение будет использование библиотечной функции alloca или аналогичных. Способом борьбы также будет запрещение SYSTEM.

RESTRICT –PROCEDURE(PROCEDURE), -SYSTEM;

Использование гарантий ограничений стековой памяти имеет практическую пользу. Автор сталкивался с ситуацией, когда нужно было организовать быструю сортировку динамически поступающих сигналов в системе мягкого реального времени. Была обнаружена редкая ситуация, что при пиковом потоке сигналов программа быстрой сортировки аварийно завершилась из-за выхода за границы стековой памяти. В результате был найден и реализован алгоритм не рекурсивной быстрой сортировки [6], который в последующем показал свою стабильную работу. Если бы имелись разработанные 2 модуля, один – без ограничений, а второй – с запретом рекурсии, то выбор был бы однозначен в пользу варианта 2.

Атаки на переполнение буфера

Атака на переполнение буфера является атакой на самую распространенную уязвимость в области безопасности ПО [7]. Авторы [7] рекомендуют «обратить внимание на языки, обеспечивающих проверку и сохранение типов, как в Java, исключающие переполнение буфера. Однако, не следует забывать, что виртуальная машина Java написана на C и, таким образом, может иметь уязвимости».

Оберон представляет собой статически жестко типизированную систему. Исполняющая

среда Оберона написана на Обероне, который является статически типизированным языком. Поэтому на чистом Обероне атака на переполнение буфера невозможна, поскольку операции копирования производятся между жестко типизированными объектами с фиксированной длиной.

Однако есть модуль SYSTEM,который декларирует использование низкоуровневых системных операций. Например, при его использовании можно копировать, используя адреса и длины.

SYSTEM.MOVE(srcAddr, dstAddr, length);

Кроме этого, процедурами SYSTEM.PUT(addr, value) можно записывать информацию по произвольному адресу, например, изменять адрес возврата или указатель на функцию. Есть еще способ – получить адрес стека и работать с ним как с массивом указателей.

rawAddr := SYSTEM.ADR(localVar); rawAdr[16 (* offset *)] := x;

Способом борьбы с уязвимостями в виде переполнения буфера будет, как минимум, запрещение указанных функций.

RESTRICT -MOVE, -PUT, -ADR;

Более точным решением будет запрещение модуля SYSTEM.

RESTRICT -SYSTEM;

Следует заметить, что использование команд из SYSTEM в стандартном Обероне требует явной декларации импортирования этого модуля.

IMPORT SYSTEM;

Такая декларация явно указывает на необходимость использования в разрабатываемом программном модуле потенциально небезопасных функций (в смысле безопасности типов). Если разработчик указал импорт SYSTEM, то он декларирует потенциально уязвимые места. Если при этом имеется ограничение использования SYSTEM (например, путем импорта модуля с RESTRICT -SYSTEM), то это приводит к синтаксической ошибке при компиляции.

Разумеется, к данной уязвимости приведет использование функций стандартной библиотеки memcpy, strcpy, работающие с безтэговыми (untagged) указателями. Однако, работа с данными указателями возможна только при использовании модуля SYSTEM.

Проблема хрупкого базового класса

Проблема хрупкого базового класса [8] заключается в скрытых зависимостях при наследовании неабстрактного класса сторонними разработчиками. Рассмотрим такой базовый класс, имеющий реализацию метода inc2. Эта реализация наследуется, что определено ключевым словом EXTENSIBLE.

TYPE Base = EXTENSIBLE RECORD counter: INTEGER END;

PROCEDURE (VAR b: Base) inc2(), NEW, EXTENSIBLE;

BEGIN INC(b.counter) END inc2;

PROCEDURE (VAR b: Base) inc1(), NEW; BEGIN b.inc2 END inc1;

Производный класс, зная только интерфейс базового, может предложить свою реализацию inc2.

```
TYPE Ext = RECORD (B.Base) END;

PROCEDURE (VAR e: Ext) inc2(); BEGIN e.inc1 END inc2;
```

Такая внешне правильная конструкция приведет к появлению бесконечной рекурсии, inc1->inc2->inc1.

Проблема данного примера и хрупкого базового класса заключается в наследовании реализации. В Обероне (здесь и далее используется диалект Oberon-CP, иногда упоминающийся под коммерческим именем Компонентный Паскаль, англ. Component Pascal, https://blackboxframework.org/index.php?cID=home-ru,ru,) наследование реализации явно отмечается ключевым словом EXTENSIBLE. Если процедура EXTENSIBLE, то это указывает на наследование реализации. Использование EXTENSIBLE, равно как и SYSTEM – это декларация последующих нарушений нормального порядка вещей в программном коде. В других языках, например, Java, для этих целей было не совсем удачно введено ключевое слово final, отсутствие которого возникает при ненормальном проектировании системы классов.

Декларация запрещения наследования реализации выглядит как

```
RESTRICT -EXTENSIBLE;
```

Это обеспечивает отсутствие проблемы хрупкого базового класса для типов всего модуля. Такой удачный, по сравнению с Java, синтаксис Оберона позволяет просто запретить ключевое слово, а не выстраивать сложные семантические схемы проверок.

Проблема вложенных процедур

Вложенные процедуры располагаются внутри других процедур и имеют доступ к их локальным переменным. Проблема вложенных процедур [9] возникает при выходе из области действия родительского фрейма, что иллюстрирует следующий код.

```
VAR vProc = PROCEDURE(val: INTEGER);

PROCEDURE Parent();

VAR loc_val: INTEGER;

PROCEDURE Child(val: INTEGER);

BEGIN loc_val := val END Child;

BEGIN Child(12);

vProc := Child;

vProc(13) (* --- CRASH --- *)

END Parent;
```

Присваивание глобальной переменной vProc адреса процедуры Child не запрещено. Но активация этой процедуры vProc(13) приводит к ошибке выполнения из-за отсутствия корректного доступа к стеку процедуры Parent.

Способом решения данной проблемы является запрет использования локальных процедур, определяемый как

RESTRICT –BEGIN(PROCEDURE);

При таком запрете процедура примера выше переместится в глобальную область с прототипом PROCEDURE Child(val: INTEGER; VAR loc_val: INTEGER), в котором буд добавлен выходной параметр.

Проблема оператора WITH

Проблема оператора WITH заключается в том, что он одновременно реализует функциональность переключателя и смены типа. Это приводит к неоднозначности кода и неясности в механизмах проверки при компиляции. Рассмотрим следующий пример.

```
TYPE Base = POINTER TO ABSTRACT RECORD END;

Ext1 = POINTER TO RECORD (Base) END;

Ext2 = POINTER TO RECORD (Base) END;

VAR vBase: Base;

vExt2: Ext2;

WITH vBase: Ext1 DO (* vBase IS Ext1 *)

vBase := vExt2 (* vBase IS Ext2 ??? *)

END;

vBase := vExt2; (* vBase IS Ext2 – Possible? *)
```

В данном примере переменная vBase до входа в область WITH рассматривается как указатель на тип Base, а после входа рассматривается как указатель на тип Ext1. Поэтому присваивание вне области WITH допустимо, а внутри – нет.

Способом решения является запрет WITH.

RESTRICT –WITH;

При этом можно использовать оператор проверок типа IS и вводить локальные переменные типов Ext1, Ext2 с явным преобразованием типов ext1 := vBase(Ext1).

Расширяемость синтаксиса языка

Приводимые выше лексемы оператора RESTRICT работали в сторону ограничения функциональности. Однако данный оператор позволяет реактивировать отключенную по умолчанию функциональность.

Например, при переходе с 32 на 64 бита потребовалось вводить целочисленный тип размером в адрес, 4 байта для 32-битных сред и 8 байт для 64-битных. Компилятор был расширен типом ADRINT, но, поскольку данный тип не входил в сообщение о языке, он стал деактивирован по умолчанию. Реактивировать его можно операцией RESTRICT.

```
RESTRICT +ADRINT;

VAR ai: ADRINT;
```

ai := SYSTEM.VAL(ADRINT, Api.GetAddr());

Данное число сохраняет целочисленный адрес, полученный из указателя, возвращенного Api.GetAddr().

Языки программирования тоже эволюционируют со временем. Однако при переходе на новые версии происходит часто прекращение поддержки старых. Например, Питон 2.7 прекратили поддерживать с 2020 года (https://legacy.python.org/dev/peps/pep-0373/).

Вместо расширения и изменения версий синтаксиса языка предлагается с помощью механизма ограничительной семантики использовать поэтапное явно декларируемое расширение добавлением новых ключевых слов. Декларация RESTRICT +ADRINT добавляет новое ключевое слово и тип ADRINT. Расширение языка программирования за счет новых ключевых слов и понятий не может происходить бесконечно. Такой путь является тупиковым. Поэтому допускается расширение за счет очень ограниченного количества ключевых слов.

Динамика развития диалектов, Оберон-07/13/16

Оберон от автора развивался в динамике, были созданы диалекты Оберон от 2007, 2013 года и уточнения в версии 2016. Развитие шло главным образом в сторону исключения не являющейся необходимой функциональности [10].

Основные изменения в языке сведены автором в таблицу на рисунке 2.

1	Изменены синтаксис/семантика WHILE, который поднят до цикла Дейкстры добавлением ELSE	+ELSE(WHILE)
2	Цикл LOOP исключен	-LOOP, -EXIT
3	Исключены типы SHORTINT, LONGINT, SHORTREAL	-SHORTINT, -LONGINT, -SHORTREAL
4	Исключен переключатель WITH	-WITH
5	Изменен синтаксис переключателя CASE	Изменение CASE
6	Оператор RETURN может быть только в конце функции, возвращающей значение	-RETURN(PROCEDURE)
7	Запрет передачи структурированных параметров по значению	- «Структура по значению»
8	Исключение возможности импортировать экспортируемые переменные по записи	- «Импорт по записи»
9	Исключено понятие указателей на массивы	-POINTER(ARRAY)
10	Отсутствие методов, относящихся к объектам, введенным в диалекте Оберон-КП	-ABSTRACT, -EXTENSIBLE
11	Добавлен процедурный тип для обработки прерываний	Изменение PROCEDURE
12	Исключены вложенные процедуры	-BEGIN(PROCEDURE)

Рис. 2. Изменения Н.Вирта в языке Оберон

Fig 2. N.Wirths changes in Oberon language

Анализ приведенных изменений показывает, что 9 из 12 изменений представляют собой запреты использования в языке некоторых синтаксических и семантических конструкций. Например, для выполнения п.6 (оператор RETURN может быть только в конце функции, возвращающей значение) необходимо добавить семантическое ограничение.

RESTRICT –RETURN(PROCEDURE);

Это позволит сделать соответствующую проверку на узлах синтаксического дерева.

В то же время 3 из 12 изменений несут за собой изменение синтаксиса операторов ELSE(WHILE), CASE, PROCEDURE. Изменение оператора CASE связанно с тем, ч операция проверки типа переходит от оператора WITH к оператору CASE, в то же время преобразование типов исключается вместе с оператором WITH. Оставшиеся изменения вносят дополнительную функциональность. Для данных изменений следует определить лексемы и предусмотреть их активацию в компиляторе, возможный вариант приведен.

RESTRICT +ELSE(WHILE), +CASE(POINTER), +PROCEDURE(SYSTEM);

Если есть поддержка новых измененных синтаксических конструкций, то можно выстраивать компилятор для диалекта 07/13/16.

По результатам оценки внесенных в Оберон 07/13/16 изменений можно сделать вывод об исключении потенциально проблемных мест. Среди них – WITH с преобразованием типа переменной, цикл LOOP с неявно определенным условием выхода, исключение неоднозначности при приведении целочисленных и действительных типов, возврат значения в конце процедуры и т.д. Исключение проблемных мест минимизирует возможность написания проблемного кода, что, в свою очередь, даст возможность утверждать о повышении надежности ПО.

Особенности реализации компилятора

Заложенные в систему свойства расширяемости и ограничиваемости диктуют структуру реализации компилятора, удовлетворяющего указанным требованиям. В условиях такой вариативности наилучшим представляется решение с разделением фронтенда и бэкенда[11]. Фронтенд занимается построением синтаксического дерева, бэкенд выполняет кодогенерацию на целевую платформу.

Предложенный во введении подход с ограничительной семантикой предлагает использовать один язык и компилятор вместо нескольких. При этом необходимо отметить, что для различных сред могут быть различные, и отличающиеся в существенной части бэкенды. Бинарные коды для встроенного ПО на ARM будут существенно отличаться от ПО для Windows, а байт-код LLVM будет отличаться от байт-кода Java Virtual Machine.

В части реализации предложенного подхода был разработан компилятор МультиОберон с набором сменных бэкендов.

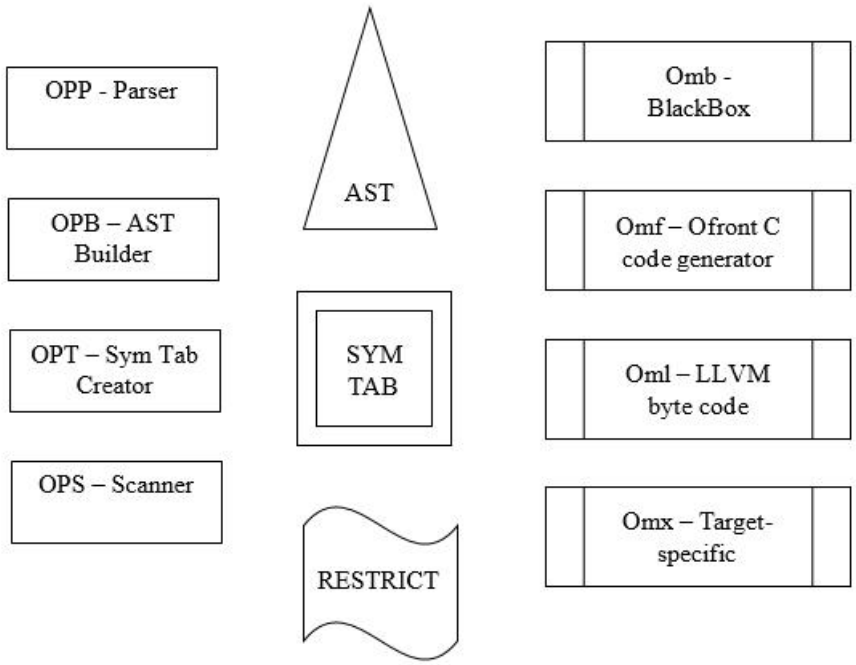

Рис. 3. Мульти-компилятор с набором бэкендов

Fig.3. Multi-compiler with a set of backends

Концепция универсального фронтенда для разработки портируемого компилятора OP2 была предложена в [12] и получила несколько программных реализаций.

Реализация диалекта Oberon-CP в виде системы, построенной на основе компонентного ПО была осуществлена в системе BlackBox Component Builder[13]. Компилятор указанной системы был переработан и подключен в МультиОберон в качестве сменного бэкенда Omb. Построенный на основе OP2 фронтенд системы BlackBox был переработан и лег в основу фронтенда МультиОберона.

Другой компилятор на основе OP2 Ofront (https://github.com/jtempl/ofront) создает на выходе код на языке C. Он уточняет систему проверок корректности типов указателей по сравнению с базовой в OP2 [14]. Ofront был существенно доработан и подключен в систему в качестве бэкенда Omf.

Бэкенд Oml, имеющий на выходе байт-код LLVM, был полностью разработан автором для системы МультиОберон.

Предполагается возможность расширения бэкендами для целевых платформ.

На выходе фронтенда создается синтаксическое дерево AST, таблица символов Sym Tab и набор использованных ограничений RESTRICT. Набор использованных ограничений должен соответствовать возможностям бэкенда. Если такого соответствия нет, то выдается ошибка при компиляции.

Вся система МультиОберон со сменными бэкендами доступна по https://github.com/dvdagaev/Mob . Текущая версия 1.2 поддерживает архитектуры X86, X64, ARM32, Aarch64 для Linux, Windows, Raspberry Pi OS.

Пример работы компилятора

Пример анализа компилятором с установленными ограничениями из графической системы BlackBox приведен на рисунке 4.

```
RESTRICT -PROCEDURE(PROCEDURE), -RETURN(PROCEDURE), -BEGIN(PROCEDURE);

PROCEDURE Fact (n: INTEGER): INTEGER;
   VAR x1234: INTEGER; nested procedure is not allowed
   PROCEDURE One(): INTEGER;
   BEGIN
      RETURN 1
   END One;
BEGIN
   IF n <= 1 THEN RETURN One() END; unexpected RETURN position
   x1234 := n * Fact(n-1 recursion is not allowed );
   RETURN x1234
END Fact;
```

Рис. 4. Листинг с нарушениями ограничений

Fig.4. Listing with restrictions violations

В данном листинге видно, что, несмотря на установленное ограничение на использование вложенных процедур –BEGIN(PROCEDURE), процедура One находится внутри родительской Fact. Следствием этого является синтаксическая ошибка «nested procedure is not allowed». Кроме этого, несмотря на установленное ограничение –RETURN(PROCEDURE), в функции Fact реализован возврат из середины кода. Следствием этого является синтаксическая ошибка «unexpected RETURN position». И, наконец, несмотря на ограничение использования рекурсии –PROCEDURE(PROCEDURE), функция Fact разработана рекурсивной. Следствием этого является синтаксическая ошибка «recursion is not allowed».

Стратегия масштабирования/демасштабирования МультиОберона

Предложенная реализация системы МультиОберон на данном этапе представляет собой эталонный уровень в части функциональности. При этом возможно как и уменьшение функциональности за счет дополнительных ограничений, так и лимитированное увеличение функциональности. Рисунок 5 иллюстрирует направления движения развития.

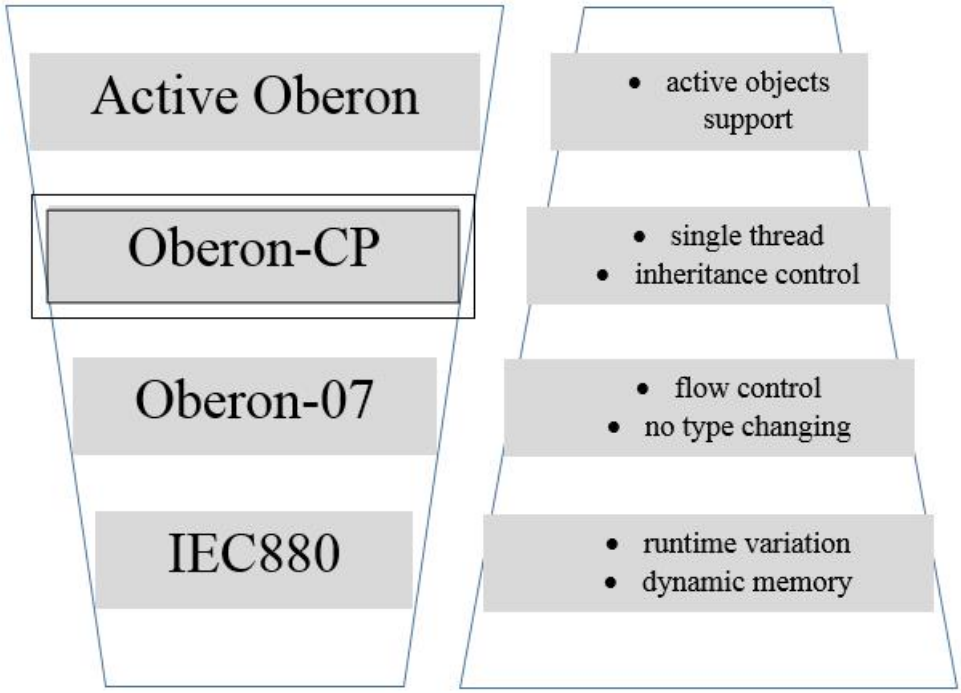

Рис. 5. Масштабирование/демасштабирование МультиОберона

Fig.5. MultiOberon scaling up and down

Oberon-CP представляет собой эталонный уровень, не требующий установки ограничений. При переходе к функциональности Оберона-07 необходимо использовать систему ограничений, при этом получается более определенная картина в управлении типами и потоком программы (flow control, no type changing). Введение дополнительных ограничений к Оберону 07 для соответствия стандарту IEC880 потребует запрета динамической памяти и вариативности выполнения циклов (runtime variation, dynamic memory). Это будет означать демасштабирование. При демасштабировании (на рисунке – движение вниз) число ограничений растет, а число функций уменьшается. При этом уменьшается число уязвимостей и увеличивается надежность.

Масштабирование от эталонного уровня означает явное введение дополнительных функций, например, введения активных объектов языка Active Oberon [15]. Используемый бэкенд должен поддерживать данные дополнительно вводимые функции.

Система ограничений языка ADA

При проектировании системы ограничений учитывался опыт компилятора GNAT языка Ada в создании ограничений (https://docs.adacore.com/gnat_rm-docs/html/gnat_rm/gnat_rm/standard_and_implementation_defined_restrictions.html). Подход Ada акцентирован на использование ряда семантических проверок (рекурсия, выделение динамической памяти) и большого числа настроек исполняющей системы. Высокая надежность Ada достигается за счет избыточности программных решений.

Подход МультиОберона нацелен на расширение и демасштабирование возможностей самого языка Оберон. Высокая надежность Оберона достигается за счет минимализма используемых программных решений.

Заключение

Предложенный в данной статье подход ограничительной семантики позволяет выбирать набор минимально необходимых средств для решения задач с разной шкалой требований. Для критически-важных систем и требований необходимо разрабатывать код с максимальной системой ограничений. Для задач реального времени необходимо отказаться от решений, потенциально блокирующих поток выполнения.

Гарантии выполнения требований ограничительной семантики предоставляются на основе синтаксического и семантического анализа программ при разработке, а не последующими экспертными средствами анализа кода на уязвимость. Соответствия модуля системе ограничений является необходимым условием успешной компиляции программы.

Подход «от ограничений» предполагает декларирование разработчикам соответствия необходимым требованиям проектирования. И наоборот, отказ от ограничений подразумевает несоответствие в полном объеме системе требований.

При построении эргодичных систем отказ от ограничений означает отказ от доказательства эргодичности. Такая практика не является общепринятой. Однако, учитывая презумпцию неэргодичности, постулируется, что характеристики разрабатываемого ПО деградируют со временем. И если в ПО нет ограничений за использованием динамической памяти, то она будет использоваться по худшему сценарию из тех, что позволит ядро ОС и система своппинга.

Общее правило построения: чем больше ограничений на использование ПО, тем данное ПО будет более надежно и предсказуемо.

Библиография

1. Н.Вирт, Ю.Гуткнехт. Разработка операционной системы и компилятора. Проект Оберон. ДМК-Пресс, 2015.
2. ГОСТ Р МЭК 60880, Программное обеспечение компьютерных систем, выполняющих функции категории А, 2009 / GOST R IEC 60880, Software for computer systems performing category A functions, 2009 (in Russian).
3. S. Louise, M. Lemerre, C. Aussagues and V. David. The OASIS Kernel: A Framework for High Dependability Real-Time Systems. In Proc. of the IEEE 13th International Symposium on High-Assurance Systems Engineering, 2011, pp. 95-103.
4. А.Ахо, М.Лам, Р.Сети, Д.Ульман. Компиляторы: принципы, технологии и инструментарий, второе издание, 2008, с. 137-144.
5. Дагаев Д.В. О разработке Оберон-системы с заданными свойствами эргодичности. Труды ИСП РАН, том 32, вып. 6, 2020 г., стр. 67-78. DOI: 10.15514/ISPRAS–2020–32(6)–5
6. Вирт Н., Алгоритмы и структуры данных. ДМК-Пресс, 2016.
7. В. Н. Гугнин, Д. В. Сотник. Атака с использованием переполнения буфера. Вестник Нац. техн. ун-та "ХПИ" : сб. науч. тр. Темат. вып. : Информатика и моделирование. – Харьков : НТУ "ХПИ". – 2004. – № 34. – С. 52-57.
8. Ермаков И.Е. Объектно-ориентированное программирование: прояснение принципов? //Объектные системы — 2010: Материалы I Международной научно-практической конференции — г. Ростов-на-Дону, Южно-Российский ГТУ — 2010. С. 130-135
9. Keller R. Improved Stackmanagement in Active Oberon Kernel / Master Thesis, ETH, march 2006, pp. 40-41.

10. Wirth N. The Programming Language Oberon. Revision 1.10.2013 / 3.5.2016. – pp. 1-17.
11. Вирт Н., Построение компиляторов, ДМК-Пресс, 2016.
12. Crelier R. OP2: A portable Oberon Compiler / ETH Zurich, Department Informatik, 1990, pp. 4-19.
13. Szyperski C. Component Software: Beyond Object-Oriented Programming / 01/2002, 2nd edition, Addison Wesley, ISBN: 0-201-745572-0
14. Templ J., Metaprogramming in Oberon, diss. ETH No 10655, 1994, pp. 120-121
15. Pieter J. Muller, The Active Object System Design and Multiprocessor Implementation. Diss. ETH No.14755, for the degree of Doctor of Technical Sciences, ETH Zurich 2002, 197 p.

Результаты процедуры рецензирования статьи

В связи с политикой двойного слепого рецензирования личность рецензента не раскрывается.
Со списком рецензентов издательства можно ознакомиться здесь.

Рецензируемый материал посвящен совершенствованию конструкции языка программирования путем реализации минималистского подхода, при котором из программного обеспечения исключаются функции, не являющиеся жизненно необходимыми. Это может быть актуальным для критически важного программного обеспечения, например, при защите реакторов атомных электростанций.
Методология исследования базируется на обобщении современных научных публикаций зарубежных о российских авторов по разрабатываемой теме с учетом государственных стандартов, регламентирующих требования к программному обеспечению компьютерных систем, выполняющих функции категории А.
Научная новизна результатов представленного исследования, по мнению рецензента, состоит в предложенном подходе к ограничительной семантике, в котором вместо нескольких языков и компиляторов предлагается использовать один язык и фронтенд компилятора с системой семантических ограничений.
В статье выделены следующие разделы: Введение; Оператор RESTRICT как декларирование ограничений: Ограничение числа итераций цикла; Ограничение динамической памяти; Ограничение стековой памяти; Атаки на переполнение буфера; Проблема хрупкого базового класса; Проблема вложенных процедур; Проблема оператора WITH; Расширяемость синтаксиса языка; Динамика развития диалектов, Оберон-07/13/16; Особенности реализации компилятора; Пример работы компилятора; Стратегия масштабирования/демасштабирования МультиОберона; Система ограничений языка ADA; Заключение; Библиография. Во введении кратко изложена необходимость и суть проводимого исследования. В последующих разделах, наименования которых ёмко отражают их содержание, излагаются способы осуществления конкретных ограничений с целью сужения функционала для обеспечения надежности и безопасности программного обеспечения в соответствии с требованиями к компьютерным системам, выполняющим функции категории А. Повествование сопровождается иллюстрациями программных реализаций на языке Оберон в системе МультиОберон в Паскале-подобном синтаксисе.
В Заключении представлены преимущества предложенного подхода, состоящие, по мнению авторов, в возможности выбирать набор минимально необходимых средств для решения задач с разной шкалой требований: для критически важных систем и требований разрабатывать код с максимальной системой ограничений, для задач реального времени -отказываться от решений, потенциально блокирующих поток

выполнения.

Библиографический список статьи включает 15 наименований, на приведенные источники в тексте имеются адресные ссылки, наличие которых свидетельствует об апелляции к оппонентам.

По рецензируемой статье следует высказать некоторые замечания.

Во-первых, вряд ли стоит в статье, подготовленной на русском языке, дублировать наименование каждого рисунка на английском.

Во-вторых, в разделе «Динамика развития диалектов, Оберон-07/13/16» при оформлении ссылки на рисунок 2 «Изменения Н. Вирта в языке Оберон» фраза «Основные изменения в языке сведены автором в таблицу на рисунке 2» нуждается в корректировке, поскольку иллюстрации могут быть представлены либо таблицей, либо рисунком.

Представленный материал соответствует тематике журнала «Кибернетика и программирование», содержит оригинальные разработки, ориентированные на повышение надёжности программного обеспечения компьютерных систем, выполняющих функции категории А, и рекомендуется к опубликованию.