Российская академия наук

Сибирское отделение

Институт систем информатики

имени А. П. Ершова

В. Н. Касьянов, Е. В. Касьянова

ЯЗЫК ПРОГРАММИРОВАНИЯ CLOUD SISAL

Препринт

181

Новосибирск 2018

Описывается язык программирования Cloud Sisal, который является начальной версией входного языка системы параллельного программирования, создаваемой в ИСИ СО РАН при поддержке Российского научного фонда.

Siberian Division of the Russian Academy of Sciences

A. P. Ershov Institute of Informatics Systems

V. N. Kasyanov, E. V. Kasyanova

PROGRAMMING LANGUAGE CLOUD SISAL

Preprint

181

Novosibirsk 2018

Abstract. The programming language Cloud Sisal is described. Cloud Sisal is the first version of an input language for the parallel programming system currently developed at the A. P. Ershov Institute of Informatics Systems with support from the Russian Science Foundation.

Введение

Язык Cloud Sisal, описанный в данном материале, является начальной версией входного языка системы параллельного программирования CPPS, создаваемой в ИСИ СО РАН за счет гранта Российского научного фонда (код проекта 18-11-00118).

Цель проекта – дать возможность широкому кругу лиц, находящихся в удаленных населенных пунктах или в местах с недостаточными вычислительными средствами, но имеющих выход в Интернет, дистанционно, без установки дополнительного программного обеспечения на своих недорогих вычислительных устройствах в визуальном стиле создавать и отлаживать переносимые параллельные программы на языке Cloud Sisal, а затем дистанционно (в облаке) осуществлять эффективное решение своих задач, исполняя созданные и отлаженные переносимые Cloud-Sisal-программы на некоторых доступных удалённых супервычислителях, предварительно адаптировав их под используемые супервычислители с помощью облачного оптимизирующего кросс-компилятора, предоставляемого системой параллельного программирования.

Язык Cloud Sisal основан на языке Sisal 3.2 [1] и является его значительно упрощенной версией, из которой убраны языковые абстракции, трудные для описания и излишне усложняющие транслятор. Упрощённая версия унифицирована и расширена возможностями, ориентированными на облачное программирование.

Данный материал следует соглашениям, используемым при описании языка Sisal 3.2 [1], в частности, имена метапонятий в нем обозначаются подчеркиванием.

Авторы благодарны к.ф.-м.н. А. П. Стасенко и всем коллегам, участвующим в выполнении проекта.

1. Общая структура программы

Программой является множество модулей и определение главной функции. Модули бывают родными, смешанными или инородными.

В любом месте программы, которое допускает пробельный символ, допустимы строковые комментарии, начинающиеся символами «//», и не вложенные друг в друга блочные (многострочные) комментарии, начинающиеся символами «/*» и оканчивающиеся символами «*/» (комментарий без завершения продолжается до конца файла).

1.1. Родной модуль

Родной модуль (или просто модуль) определяется на языке Cloud Sisal. Родной модуль состоит из интерфейса модуля и тела модуля, каждый из которых является отдельной единицей трансляции. Интерфейс модуля состоит из

- имени интерфейса модуля: *interface* имя;
- директив импорта типов интерфейсов других модулей: *import* имя модуля, …, имя модуля;
- объявлений функций;
- определений типов, используемых в объявлениях функций.

Тело модуля состоит из

- имени тела модуля: *module* имя;
- директив импорта интерфейсов других модулей: *import* имя модуля, …, имя модуля;

- определений типов и функций.

Интерфейс и тело модуля с одинаковым именем относятся к одному модулю. Тело модуля, если оно присутствует, обязано определять все функции, объявленные в его интерфейсе. Интерфейс модуля может быть пустым (отсутствовать).

1.2. Смешанный модуль

Смешанный модуль определяется на языке реализации транслятора языка Cloud Sisal. Смешанный модуль состоит только из своего интерфейса, который состоит из

- имени интерфейса смешанного модуля: *interface* имя;
- директив импорта типов интерфейсов других модулей: *import* имя модуля, …, имя модуля;
- объявлений смешанных функций;
- определений типов, которые используются в объявлениях смешанных функций.

1.3. Инородный модуль

Инородный модуль определяется на языке Си. Инородный модуль состоит только из своего интерфейса, который состоит из

- имени интерфейса инородного модуля: *interface* имя;
- директив импорта типов интерфейсов других инородных или смешанных модулей: *import* имя модуля, … , имя модуля;
- объявлений инородных функций;
- определений инородных типов, которые используются в объявлениях инородных функций.

1.4. Правила импорта

При импортировании интерфейса модуля *X* в интерфейс или тело модуля *Y* все типы интерфейса модуля *X* становятся частью интерфейса или тела модуля *Y*, включая все типы, которые модуль *X* импортировал в свой интерфейс. Дополнительно при импортировании интерфейса модуля *X* в тело модуля *Y* все объявления функций модуля *X* становятся доступны для использования в определениях функций модуля *Y*.

1.5. Определение типов

Определение типа выглядит как утверждение *type* имя типа = тип и определяет имя типа как эквивалентное обозначение типа.

1.6. Объявление функций

Объявление функции выглядит как «*function* имя функции (список формальных параметров *returns* список типов возвращаемых значений)», где необязательные имена формальных параметров «имена формальных параметров : тип формальных параметров» игнорируются компилятором. Формой функции называется список типов её формальных параметров. Формой возвращаемых значений функции называется список типов её возвращаемых значений. Две формы функции равны, если число их типов совпадает, а соответствующие типы эквивалентны между собой. Функциями, неоднозначными по форме возвращаемых значений, называются функции, у которых совпадают имена и формы формальных параметров и различны формы возвращаемых значений.

1.7. Определение функций

Определение функции выглядит как объявление функции (называемое заголовком определения), в котором должны быть указаны имена формальных параметров, которые

должны быть различны и за которым следует список выражений. Размерность списка выражений равна количеству возвращаемых значений, а типы значений можно неявно преобразовать в соответствующие типы возвращаемых значений. Определение функции завершается ключевыми словами «*end function*». Определение функции также является соответствующим объявлением, если его не было сделано ранее.

Определение функции определяет область видимости имён функций N_1, объявленных ранее, а также область видимости имён формальных параметров N_2. Выражения функции могут определять последующие области видимости имён. Имена из областей видимости с бо́льшим номером перекрывают имена из областей видимости с меньшим номером.

1.8. Объявления смешанных функций

Объявление смешанных функции выглядит как «*function* имя функции (список формальных параметров *returns* список типов возвращаемых значений)», где типы формальных параметров и возвращаемых значений могут быть инородными.

1.9. Объявления инородных функций

Объявление инородной функции выглядит как «*function* имя функции (список инородных формальных параметров *returns* список инородных типов возвращаемых значений)», где именам или типам инородных параметров и типам возвращаемых значений могут предшествовать «[целое число]». Целое число обозначает номер параметра, начиная с единицы, которому соответствует тип формального параметра и тип возвращаемых значений. Если номер параметра не указан, то для формальных параметров он предполагается равным номеру параметра в списке, а отсутствующий номер типа возвращаемого

значения обозначает результат инородной функции. Только один тип возвращаемых значений может иметь отсутствующий номер.

Номера формальных параметров (включая неявно заданные номера) не должны пересекаться. Номера типов возвращаемых значений не должны пересекаться. Типы формальных параметров и типы возвращаемых значений с одинаковыми номерами должны совпадать. Тип возвращаемого значения с номером должен задаваться инородной записью, которая в этом случае передаётся по указателю, или инородным типом со строковым представлением, который является указателем.

1.10. Уникальность и область видимость имен

Имя задаётся цепочкой букв верхнего регистра и букв нижнего регистра (отличных от букв верхнего регистра), десятичных цифр и знака подчеркивания. Имя не может начинаться с десятичной цифры и состоять из единственного знака подчеркивания. Имена не должны совпадать с ключевыми словами (обозначаемые в данном документе курсивом).

Имена модулей программы должны быть уникальны. Имена типов интерфейса и тела модуля должны быть уникальны. Допускается лишь эквивалентное переопределение типа. Имена функций интерфейса и тела модуля должны быть уникальны.

Имена типов и функций нельзя использовать до их объявления или определения.

1.11. Точка начала исполнения программы

Программа указывает главный модуль и определение главной функции, с которой начинается выполнение программы. Аргументы главной функции являются

входными данными программы, а результаты главной функции – результатами работы программы.

2. Типы

Типы языка подразделяются на определённые, встроенные, составные встроенные типы и инородный тип. К встроенным типам относятся типы: логический (boolean), символьный (character), целочисленный (integer), вещественный (real) и комплексный (complex). К составным встроенным типам относятся типы записи (record), потока (stream), массива (array) и функции (function).

Для каждого типа, кроме инородного, существует выделенное ошибочное значение. Если не оговорено обратное и аргумент операций над встроенными типами или аргумент предопределённых функций ошибочен, то результаты тоже будут ошибочными значениями. Узнать, ошибочно ли значение выражения, можно с помощью операции «*error* (выражение)».

2.1. Логический тип (boolean)

Булевский тип состоит из значений истина (*true*), ложь (*false*) и ошибочного значения *error*[boolean].

Для булевского типа определены следующие операции:

- бинарная операция равенства «=»;
- бинарная операция неравенства «!=»;
- бинарная операция конъюнкции «&»;
- бинарная операция дизъюнкции «|»;
- бинарная операция исключающей дизъюнкции «^»;
- унарная операция отрицания «!»;
- операция явного преобразования к целому типу: «*false* : integer» = 0, «*true* : integer» = 1.

2.2. Символьный тип (character)

Символьный тип содержит как минимум, символы стандарта ASCII [2] и ошибочное значение *error*[character]. Литералы символьного типа имеют вид знака символа в одинарных кавычках. Символ одинарной кавычки задаётся как '\''. Символ с десятичным кодом ddd задаётся как '\ddd', где ddd содержит 3 цифры. Символ с шестнадцатеричным кодом hh задаётся как '\xhh', где hh – запись шестнадцатеричного числа цифрами и прописными буквами. Можно указывать символ перевода строки '\n', возврата каретки – '\r', табуляции – '\t', двойной кавычки – '\"'.

Для символьного типа определены следующие операции:

- бинарная операция равенства «=»;
- бинарная операция неравенства «!=»;
- операция явного преобразования к целому типу «символ : integer», которая возвращает код символа в таблице ASCII.

2.3. Целочисленный тип (integer)

Целый тип содержит по крайней мере все числа, достаточные для представления кодов символов символьного типа, их отрицания и ошибочное значение *error*[integer]. Количество байт для представления целого типа определяется во время компиляции и по умолчанию равно 8. Значения типа задаются цепочкой десятичных чисел, для повышения читаемости которой везде, кроме ее начала, могут использоваться символы подчеркивания. Знак числа, как и для вещественных чисел, считается унарной операцией, и поэтому между ним и числом допускается произвольное число пробельных символов. Целые литералы могут задаваться в произвольной системе счисления: «основание#число», где ее основание является числом от 2 до 36.

Для числовых типов, к которым относится целый тип, определены следующие операции:

- унарная операция идентичности «+»;
- унарная операция смены знака «-»;
- бинарная операция сложения «+»;
- бинарная операция вычитания «-»;
- бинарная операция умножения «*»;
- бинарная операция деления «/»;
- бинарная операция возведения в степень «**»;
- бинарная операция равенства «=»;
- бинарная операция неравенства «!=»;
- бинарная операция больше «>»;
- бинарная операция меньше «<»;
- бинарная операция больше или равно «>=»;
- бинарная операция меньше или равно «<=».

Для целого типа определены следующие операции:

- операция неявного преобразования к вещественному типу;
- операция явного преобразования к вещественному типу «целое : real»;
- бинарная операция взятия остатка от деления или модуля «%»;
- встроенная функция «abs (целое)», возвращающая модуль целого числа;
- встроенная функция «min (целое, целое)», возвращающая минимум двух целых чисел;
- встроенная функция «max (целое, целое)», возвращающая максимум двух целых чисел.

Если в результате выполнения операции деления или модуля от целых операндов происходит деление на ноль, то возвращается значение «*error*[integer]».

2.4. Вещественный тип (real)

Вещественный тип содержит значения, определяемые стандартом IEEE-754 [3], и ошибочное значение *error*[real]. Количество байт для представления вещественного типа определяется во время компиляции, и по умолчанию равно 8. Значения вещественных типов задаются десятичными литералами, отличающимися от целых литералов наличием точки и/или знаком экспоненты. Кроме того, для вещественных чисел символы подчеркивания не могут идти после десятичной точки. Вещественное число задается литералом простой или экспоненциальной формы. Простая форма является десятичным числом, разделенным знаком точки на две части, одна из которых может быть пустой. Экспоненциальная форма состоит из двух частей, разделенных буквой «e». Левая часть – это десятичное число, возможно, разделяемое на две части точкой, которая может стоять в начале. Правая часть – это необязательный знак минуса или плюса и десятичное число.

Для вещественного типа определены все операции числовых типов и следующие операции:

- операция явного преобразования вещественного значения в значение целого типа «число : integer», возвращающая значение floor (0.5 + число). В случае, если получается число, не представимое целым типом, то возвращается значение «*error*[integer]»;
- встроенная функция «floor (число)», возвращающая наибольшее целое число, не большее, чем вещественный аргумент;
- встроенная функция «trunc (число)», возвращающая целое число, равное вещественному аргументу без дробной части;
- встроенная функция «abs (число)», возвращающая модуль вещественного числа;

- встроенная функция «min (число, число)», возвращающая минимум двух вещественных чисел;
- встроенная функция «max (число, число)», возвращающая максимум двух вещественных чисел.

2.5. Комплексный тип (complex)

Комплексный тип содержит два значения, определяемые стандартом IEEE-754, и ошибочное значение *error*[complex]. Литерал комплексного типа задаётся как «complex (вещественное значение, мнимое значение)» или «complex (вещественное значение)», если мнимая часть равна 0.

Для комплексного типа определены все операции числовых типов, а также следующие операции:

- унарная операция взятия вещественной части «комплексное число . r»;
- унарная операция взятия мнимой части «комплексное число . i».

2.6. Тип записи (record)

Тип записи «*record* [объявление полей записи]» задает декартово произведение типов своих полей, к значениям которых имеется прямой доступ по их имени, уникальному в пределах одного типа записи. Объявление полей записи содержит разделяемые точкой с запятой объявления полей с одинаковым типом «список имён полей : тип полей». Тип записи также содержит ошибочное значение. Если запись ошибочна, то значения всех её полей тоже ошибочны. Типы записей эквивалентны, если они имеют одинаковое количество полей и типы соответствующих полей эквивалентны. Допускается рекурсивное определение типа записи в конструкции определения типа.

Запись конструируется как «*record* тип записи [определение полей записи]» или «*record* [определение полей записи]». Определение полей записи содержит разделяемые точкой с запятой определения одного или нескольких полей «список имён полей := список выражений», указанных с помощью их имен. Список выражений должен определять все указанные поля записи. В выражении конструктора записи все её поля должны быть определены единственным образом. Тип записи определяется конструктором записи без типа, используя указанные имена полей записи и типы списка выражений.

Для записи определены следующие операции:

- Получение значения поля записи «запись . имя поля этой записи»;
- Операция замещения полей записи «запись . [определение полей записи]».

2.7. Тип потока (stream)

Тип потока описывается как «*stream of* тип элемента потока» и содержит возможно бесконечные цепочки последовательно доступных элементов одного типа и ошибочное значение «*error* [*stream of* тип элемента потока]». Типы потоков эквивалентны, если эквивалентны типы их элементов.

Поток можно сконструировать в циклическом выражении или с помощью выражения конструктора потока «*stream of* тип элемента потока [список значений элементов потока]» или «*stream* тип потока [список значений элементов потока]». Для пустого потока список значений элементов потока может отсутствовать. Для непустого списка значений элементов потока тип потока может отсутствовать и неявно задаваться типом первого элемента потока. Каждый элемент списка значений элемента потока последовательно задаёт одно или несколько значений

элементов потока, начиная с элемента с индексом 1, и может быть выражением или триплетом.

2.7.1. Определение триплета

> Триплетом является структура вида «нижняя граница .. верхняя граница .. шаг», где все три элемента должны быть унарными выражениями целого типа и часто могут быть опущены независимо друг от друга (шаг опускается вместе с точками «..», идущими впереди). Триплет задаёт арифметическую прогрессию значений, начинающихся с нижней границы, которые меньше или равны указанной верхней границы, если шаг больше нуля.
> Если шаг меньше нуля, то задаются значения, которые меньше или равны указанной верней границы. Шаг не может быть равен нулю. Если опущена нижняя граница, то она предполагается равной единице. Если опущена верхняя граница, то она предполагается равной бесконечности (положительной для положительного шага и отрицательной для отрицательного шага). Шаг по умолчанию равен единице, если верхняя граница больше нижней, и минус единице иначе.

Верхняя граница триплета конструктора потока может быть опущена только в последнем элементе списка значений элементов конструктора потока.

Для потока определены следующие операции:

- выражение выбора элемента потока «поток [унарное целое выражение]» (нумерация элементов потока начинается с единицы);
- выражение выбора элементов потока «поток [выбирающее выражение]», образующее новый поток из выбираемых элементов, где выбирающее

выражение может триплетом или унарным выражением типа целого потока или одномерного массива (индексным вектором).

- бинарная операция конкатенации потоков «||» с эквивалентными типами;
- все операции (и встроенные функции) над потоками, допустимые для типа элемента потока, порождающие поток со значениями, полученными при поэлементном выполнении операции;
- Встроенная функция «empty (поток)», возвращающая булевское значение *true*, если поток пуст, и *false* – иначе.

2.8. Тип массива (array)

Тип массива описывается как «*array* форма массива *of* тип элемента массива» и содержит конечные цепочки элементов одного типа с прямым доступом по их многомерному индексу и ошибочное значение «*error* [*array* форма массива *of* тип элемента массива]». Форма имеет вид «[список двойных точек]». Форма может быть опущена и по умолчанию полагается равной «[..]». Количество размерностей массива задаётся размерностью формы массива, равной количеству элементов её списка двойных точек.

Два типа массива эквивалентны, если эквивалентны типы их элементов и массивы имеют одинаковую форму. Формы массивов совпадают, если они имеют одинаковую размерность.

Строковые литералы с встроенным типом string рассматриваются как одномерные массивы символов «*array of* character» и задаются цепочкой символов, заключенной в двойные кавычки. Для обозначения символов строки допустимо использовать все обозначения, приведённые для литералов символьного типа. Если непосредственно перед начальной кавычкой находится

символ «@», то все специальные обозначения символов, кроме цепочки «\"», воспринимаются буквально. Последовательные строковые литералы, возможно расположенные на разных строках, склеиваются в один.

Массив можно сконструировать в циклическом выражении или с помощью выражения конструктора массива. Выражение конструктора массива задаётся одним из следующих образов:

- «*array* тип массива [размерности формы массива] [список значений элементов массива]»;
- «*array* [размерности формы массива] *of* тип элемента массива [список значений элементов массива]»;
- «*array of* тип элемента массива [список значений элементов массива]»;
- «*array* [список значений элементов массива]»;
- «[список значений элементов массива]».

Размерности формы массива представляют собой список целочисленных выражений, задающий количество элементов в каждой размерности массива, которая должна совпадать с размерностью формы типа массива, если он указан. Размерность должна быть больше или равна единице. Если тип элемента и тип массива отсутствуют, то тип элемента массива неявно задаётся типом первого элемента массива. Если размерности формы элемента массива отсутствуют, то задаётся одномерный массив с количеством элементов, определяемым списком значений его элементов.

Если размерности формы массива отсутствуют, то список значений элементов массива определяется размерностью списка выражений. Если размерности формы массива присутствуют, то список значений элементов массива определяется одном из следующих образов:

- «:= список выражений», задающее элементы массива в row-major порядке, при котором элементы перечисляются, начиная с внутренней размерности массива;
- Список элементов «диапазон := значение элемента массива», разделяемых точкой с запятой, за которыми может следовать утверждение «*else* := значение элемента массива».

2.8.1 Конструкция диапазона

Диапазон задаётся списком, элементами которого могут быть

- целочисленное выражение;
- ряд значений, задаваемый триплетом;
- ряд значений, задаваемый массивом или потоком с необязательной частью «*at* [список имен размерностей]», где список имён размерностей должен содержать количество элементов равных размерности формы массива (у потока подразумевается единичная размерность), а имя размерности может быть заменено на двоеточие;
- именованный ряд значений «имя *in* ряд значений».

Размерность диапазона определяется как сумма размерностей целочисленных выражений, триплетов и размерности рядов значений. Размерность ряда значений определяется как размерность массива или потока за вычетом количества двоеточий в необязательной части «*at*». Размерность ряда значений не может быть равной нулю. Имена ряда значений или имен размерностей части «*at*» имеют область видимости в следующих элементах текущего диапазона.

Мощность диапазона определяется как сумма триплетов и размерности рядов значений.

Для диапазона массива массив или поток ряда значений должен иметь целочисленный тип и необязательная часть «*at*» не должна содержать двоеточий. Размерность диапазона должна совпадать с размерностью формы массива. Неуказанные верхние границы триплетов обозначают верхнюю границу текущей размерности массива.

Все незаданные элементы массива равны ошибочным значениям. Если присутствует утверждение «*else*», то все незаданные элементы массива равняются указанному значению. Если в выражении конструктора массива диапазоны элементов пересекается, то возвращается ошибочный массив.

Для массива определены следующие операции:

- выбор элемента массива «массив [диапазон]», где мощность диапазона равна нулю;
- построение многомерного массива «массив [диапазон]» с размерностью формы, равной мощности диапазона, которая больше или равна 1, где ряды значений диапазона не зависят от имен элементов диапазона (порождается прямоугольный массив);
- построение массива массивов «массив [диапазон]», где мощность диапазона больше или равна 1, а ряды значений диапазона зависят от имен элементов диапазона (порождается массив массивов, каждый из которых соответствует последовательности размерностей формы, задающей прямоугольный массив);
- выражение замещения элементов массива «массив [список расположенных элементов]», где список расположенных элементов состоит из элементов

«диапазон := значение элемента массива», разделяемых точкой с запятой.

- бинарная операция конкатенации массивов с эквивалентными типами элемента («||»), возвращающая одномерный массив, склеенный из двух указанных массивов, элементы которых располагаются в row-major порядке.
- все операции (и встроенные функции) над массивами с одинаковой формой, допустимые для типа элемента массива, порождающие массив со значениями, полученные при поэлементном выполнении операции;
- для массива определяется встроенная функция одного аргумента «size», которая берёт на входе массив и возвращает количество элементов во всех его размерностях;
- встроенная функция «size» определена для двух элементов и возвращает количество элементов его размерности, указанной вторым аргументом целого типа;
- для массива определяется встроенная функция одного аргумента «transpose», которая берёт на входе массив и возвращает транспонированный массив.

2.9. Инородный тип

Инородный тип задаётся одним из следующих образов:

- строкой "строковое представление инородного типа", где строковое представление инородного типа должно задаваться на языке C;
- нерекурсивной записью *record* "имя инородного языка" [объявление полей записи], где имя инородного языка должно равняться "C", а типы объявлений полей записи должны быть инородными типами.

Инородные типы, заданные строкой, эквивалентны, только если совпадает их строковое представление, что является более строгим критерием эквивалентности, чем это необходимо, и в обязанности программиста входит поддержание его правильности, если это необходимо.

Никаких встроенных операций для инородного типа, заданного строкой, не определено. Для инородной записи определена операция получение значения поля «инородная запись . имя поля этой записи».

Для инородного типа, заданного строкой, в смешанном модуле может определяться функция «*function* delete (формальный параметр инородного типа *returns* "")», которая будет использована для освобождения памяти более не нужного значения инородного типа, если это необходимо.

2.10. Тип функции (function)

Тип функции задается как «*function* [типы аргументов *returns* типы результатов]», содержит все функции с указанными типами аргументов и результатов и ошибочное значение. Типы аргументов могут отсутствовать и тогда функция определяет константу. Значение типа функции можно сконструировать с помощью:

- имени объявленной функции «имя функции», если имя функции не перекрыто локальным именем и функция задаётся однозначно;
- имени функции «*function* имя функции [..]», позволяющего указать однозначно заданную функцию, даже если она и имя её модуля перекрыто локальным именем;
- конструкции «*function* имя функции [список типов формальных параметров]» для указания конкретной функции, однозначной по типам возвращаемых значений;

- конструкции «*function* имя функции [список типов формальных параметров *returns* список типов возвращаемых значений]» для указания конкретной функции;
- конструкции «*function* имя функции (список формальных параметров *returns* типы результатов) список выражений *end function*» для определения λ-функции, в которой имя функции не обязательно и используется в списке выражений функции только для задания рекурсивной зависимости (в списке выражений функции разрешается использовать имена значений, определённых вне тела функции).

Тип функции имеет следующие операции:

- вызов функции «функция (значения аргументов)», возвращающая результаты функции, вычисленные после подстановки значений аргументов на место имён формальных параметров;
- сужение функции «функция (значения аргументов)», где пропущен один или несколько значений аргументов функции, а результатом операции будет функция от пропущенных аргументов в порядке их следования и с результатами исходной функции.

Операция вызова функции автоматически разрешает неоднозначность функции, заданной её именем, на основании типов ее аргументов (если функция однозначна по возвращаемым значениям). Если не существует функции с формой, состоящей из типов, эквивалентных типам аргументов, то выбирается функция с минимальным количеством неявных преобразований, которое необходимо осуществить для преобразования типов значений аргументов в типы формальных параметров.

Операция сужения функции также разрешает неоднозначность функции на основании типов её аргументов, указываемых как «: тип» для пропускаемых аргументов.

Операции вызова или сужения инородной функции автоматически применяет следующие функции преобразования «*function* convert (: A *returns* B)»:

- родных типов A языка в инородные типы B формальных параметров инородной функции;
- инородных типов A возвращаемых значений инородной функции в родные типы B языка.

3. Выражения

Выражения могут быть *n*-арными. Любое унарное (*n*=1) выражение языка рассматривается как арифметическое выражение. Список выражений задаётся перечисленными через запятую выражениями. Размерность списка выражений равна сумме размерностей выражений в него входящих.

3.1. Арифметическое выражение

Арифметическое выражение содержит операнды и операции. Операндами являются унарные выражения. Операции могут быть постфиксными, префиксными и инфиксными.

Постфиксные операции имеют вид «операнд операция». Цепочка постфиксных операций вычисляется слева направо до начала вычисления префиксных операций. К постфиксным операциям относятся операции вызова и сужения функции, выбора и замены элементов массива, выбора элементов потока, доступа к полю записи, замены элементов записи и операция явного приведения типов.

Префиксные (унарные) операции имеют вид «знак операции операнд». Цепочка префиксных операций вычисляется справа налево до начала вычисления инфиксных операций. К префиксным операциям относятся операции смены знака («-»), идентичности («+») и логического отрицания («!»).

Инфиксные (бинарные) операции имеют вид «операнд знак операнд». Среди нескольких инфиксных операций раньше выполняются операции на более глубоком уровне вложенности арифметических скобок. Среди инфиксных операций одного уровня вложенности сначала выполняются операции с большим приоритетом,

указанным в таблице 1. Лево-связываемые операции одного приоритета выполняются слева направо, а право-связываемые операции – справа налево. Цепочка операций сравнения объединяется операциями конкатенации, например, «*A* < *B* <= *C*» рассматривается как «*A* < *B* & *B* <= *C*».

Таблица 1

Свойства инфиксных операций

Приоритет	1	2	3	4	5	6	7	8	9
Знак	||	|	^	&	= !=	< > <= >=	- +	* / %	**
Связывание	«Левое»								«Правое»

Если для операндов инфиксной операции не объявлена операция для типов, эквивалентных типам операндов, то делаются попытки неявных преобразований и применений операции над получившимися типами.

3.2. Выражение «let»

Выражение «*let*» определяет новую область и множество ее имен, используя их для вычисления списка выражений своих результатов: «*let* определения имен *in* список выражений результатов *end let*». Определения имен содержат разделенные точкой с запятой определения, содержащие левую и правую части, разделенные символами «:=».

Левая часть – это разделенные запятыми определяемые имена. Правая часть состоит из списка выражений, сумма размерностей которых равна числу имен левой части

определения. Выражения правой части определения не могут зависеть от имен его левой части. Область действия определённых имён состоит из правых частей последующих определений и списка выражений результатов.

3.3. Выражение «if»

Выражение «*if*» выглядит как «*if* булевское условие *then* список выражений результата ветви *elseif* ветвь *else end if*», где каждая из необязательных ветвей «*elseif*» задаётся как «*elseif* булевское условие *then* список выражений результата», а необязательная ветвь «*else*» задаётся как «*else* список выражений результата».

Все списки выражений результата должны иметь одинаковые размерности. Типы возвращаемых значений выражения «*if*» определяются типами размерностей в первом списке выражений результата (после ключевого «*then*»). Типы размерностей в других списках выражений результата должны быть эквивалентны типам соответствующих результатов выражения «*if*» или неявно к ним приводиться.

Выражения булевских условий вычисляются последовательно, пока не получится истинное значение. Список выражений результата, идущий за первым истинным булевским условием, определяет результаты выражения «*if*». Если все булевские условия ложны, то ветвь «*else*» определяет результат конструкции. Если ветвь «*else*» отсутствует, или встретилось ошибочное значение булевского условия, то выражение «*if*» возвращает ошибочные значения.

3.4. Выражение «case»

Выражение «*case*» выглядит как «*case* управляющее выражение условные ветви ветвь *else end case*», где должна

присутствовать хотя бы одна условная ветвь вида «*of* список значений тестов *then* список выражений результата», а необязательная ветвь «*else*» задаётся как «*else* список выражений результата». Управляющее выражение должно быть унарным.

У всех списков выражений результата размерности должны быть равны. Типы возвращаемых значений выражения «*case*» определяются типами размерностей в первом списке выражений результата. Типы размерностей в других списках выражений результата должны быть эквивалентны типам соответствующих результатов выражения «*case*» или неявно к ним приводиться.

Значение теста может быть унарным выражением и, если тип управляющего выражения имеет операции со знаками «=» и «<», то дуплетом «нижняя граница .. верхняя граница», в котором опущенные унарные выражения нижней и верхней границы по умолчанию равняются минус и плюс бесконечности. Если значение теста является унарным выражением, то тестом является сравнение его значения со значением управляющего выражения. Если значением теста является дуплет, то тестом является булевское выражение «нижняя граница <= управляющее выражение <= верхняя граница».

Может быть истинным только один тест. Список выражений результата, идущий за истинным тестом, определяет результаты выражения «*case*». Если все тесты ложны, то ветвь «*else*» определяет результат конструкции. Если ветвь «*else*» отсутствует или значение теста ошибочно, то выражение «*case*» возвращает ошибочные значения.

3.5. Циклические выражения

Язык поддерживает циклические выражения, управляемые диапазоном и тестом.

3.5.1. Управление диапазоном

Циклические выражения, управляемые диапазоном, имеют один вид: «*for* список диапазонов *let* определения имён начальных значений *do* определения имён циклических значений *returns* список редукций *end for*», где «*let* определения имён начальных значений» и «*do* определения имён циклических значений» здесь и далее могут быть опущены.

Циклическое выражение, управляемое диапазоном, выполняется до тех пор, пока не будут перебраны все элементы диапазона. Элементы списка диапазонов разделяются ключевым словом «*cross*». Мощность диапазона должна совпадать с его размерностью (в диапазон могут входить только ряды значений). Также в диапазоне не допускаются ряды значений без имени.

Циклическое выражение, управляемое диапазоном, асинхронно параллельно [4], если его список диапазонов не перечисляет потоки, оно не содержит редукций «*stream*» и не содержит «*old*» имён. Если используются пользовательские редукции, то, чтобы цикл оставался параллельным, необходимо, чтобы все функции начальных значений пользовательских редукций были помечены прагмой «identity», а все собирающие функции пользовательских редукций помечены прагмой «associative». Для сохранения свойства асинхронной параллельности все собирающие функции пользовательских редукций дополнительно помечены прагмой «commutative».

3.5.2. Управление тестом

Циклические выражения, управляемые тестом, имеют один из следующих видов:

- «*for let* определения имён начальных значений *while* булевское условие *do* определения имён

циклических значений *returns* список редукций *end for*»;

- «*for let* определения имён начальных значений *do* определения имён циклических значений *while* булевское условие *returns* список редукций *end for*».

Циклическое выражение, управляемое тестом, выполняется, пока «*while*» булевское условие истинно. Тест проверяется до или после выполнения итерации в зависимости от своего положения относительно ключевого слова «*do*».

Определения имён начальных значений семантически полностью эквивалентны их заданию в окружающем выражении «*let*». Имена начальных значений и имена значений, определённых извне цикла, являются константами цикла. Имена, определяемые в левых частях определений имён циклических значений, вычисляются на каждой итерации цикла и могут присутствовать в правых частях всех определений имён циклических значений с префиксом «*old* имя», если данное имя также является константой цикла. Имя «*old* имя» обозначает значение циклического имени с предыдущей итерации цикла или константу цикла на первой итерации.

3.5.3. Редукция циклических значений

Итерации цикла определяют редуцируемую последовательность значений для каждого имени, задаваемого генератором диапазона и телом цикла. Редукции формируют из редуцируемых последовательностей одно или несколько возвращаемых значений циклического выражения. Редукция имеет один из следующих видов:

- «*stream of* выражение», которая формирует поток из своей редуцируемой последовательности с пустым потоком по умолчанию;
- «*array of* выражение», которая формирует одномерный массив из своей редуцируемой последовательности с пустым массивом по умолчанию;
- «*array* форма массива *of* выражение», которая может использоваться только вместе с диапазоном цикла и должна задавать форму с размерностью, равной мощности диапазона цикла, который используется для определения размерностей конструируемого многомерного массива с ошибочным массивом по умолчанию;
- «value *of* выражение», возвращающая последнее значение редуцируемой последовательности с ошибочным значением по умолчанию;
- «sum *of* выражение», возвращающая сумму значений редуцируемой последовательности с нулевым значением по умолчанию;
- «product *of* выражение», возвращающая произведение значений редуцируемой последовательности с единичным значением по умолчанию;
- «greatest *of* выражение», возвращающая наибольшее значение редуцируемой последовательности с наименьшим значением типа по умолчанию;
- «least *of* выражение», возвращающая наименьшее значение редуцируемой последовательности с наибольшим значением типа по умолчанию;
- «catenate *of* выражение массива или потока», возвращающая поток или массив, склеенный из потоков или массивов редуцируемой последовательности с пустым массивом или потоком по умолчанию;

- Пользовательская редукция с начальными значениями «пользовательская редукция *let* список значений начальных параметров редукции *of* список значений циклических параметров редукции», где значения начальных параметров должны задаваться константами цикла;
- Пользовательская редукция без начальных значений «пользовательская редукция *of* список значений циклических параметров редукции».

Выражение фильтра «*if* булевское условие» может присутствовать после редукции (кроме редукции массива с указанной формой массива) и включает циклические значение в редуцируемую последовательность только в случае, если булевское условие истинно.

Если редуцируемая последовательность значений для редукции пуста (генератор цикла задаёт пустой диапазон значений, тест перед циклом не был удовлетворён или фильтр не пропустил ни одного значения), то редукция возвращает значения по умолчанию. Для пользовательских редукций по умолчанию возвращается начальное значение «ret(ini(…))».

3.5.4 Пользовательская редукция

Пользовательская редукция задаётся записью с именами полей ini, rep, ret и возможно join. Типы полей записи должны иметь функциональный тип со следующими требованиями к типам аргументов и результатов, где тип T является произвольным типом, подходящим для хранения промежуточных значений редукции:

- функция ini, формирующая начальное значение редукции, должна иметь тип «*function* [типы значений начальных параметров редукции *returns* T]» или «*function* [*returns* T]», если у редукции нет начальных значений;

- функция rep, вычисляющая частичное циклическое значение редукции, должна иметь тип «*function* [T, типы значений циклических параметров редукции *returns* T]»;
- функция ret, возвращающая результаты редукции, должна иметь тип «*function* [T *returns* типы результатов пользовательской редукции]»;
- необязательная собирающая функция join, объединяющая частичные значения редукции, должна иметь тип «*function* [T, T *returns* T]».

В данном примере определяется пользовательская редукция my_sum_red, эквивалентная встроенной редукции sum:

```
type my_sum_red = record [
    ini:  function [ returns real ]
    rep:  function [ real, real returns real ]
    ret:  function [ real returns real ]
    join: function [ real, real returns real ]
]

function zero ( returns real) 0 end function

//$ associative
//$ commutative
function sum (a, b: real returns real) a + b end function

function my_sum_red ( returns my_sum_red)
    record my_sum_red [
        /*$ identity */ ini := zero;
        rep, join := sum, sum;
        ret := function (a: real returns real) a end function
    ]
end function

function my_sum_all (A: array of real returns real)
    for a in A returns my_sum_red ( ) of a end for
```

```
end function

function sum_all (A: array of real returns real)
    for a in A returns sum of a end for
end function
```

4. Оптимизирующие указания компилятору (прагмы)

Язык допускает использование утверждений (прагм), описывающих свойства программы, известные программисту, которые могут использоваться системой при обработке программы. Комментарий, который начинается с символа доллара «$», называется прагмой и задаёт свойства конструкции, идущей следом (одной конструкции можно сопоставлять несколько прагм). Прагма может иметь вид «имя» или «имя = список выражений», где в выражениях списка могут принимать участие имена, видимые в месте расположения прагмы.

4.1. Прагма assert

Перед каждым выражением могут располагаться прагмы «assert = булевское условие», которые должны быть истинными непосредственно сразу после вычисления выражения. Результат унарного выражения в булевском выражении обозначается через одиночный символ подчеркивания «_». Арности n-арного (n>1) выражения обозначаются как «_[1]», …, «_[n]».

Прагмы данного вида могут располагаться в объявлениях функций как перед ключевым словом «*returns*» и накладывать условия на возвращаемые значения, так и перед первым формальным параметром и задавать условия на указанные в них имена формальных параметров, которые должны быть справедливы при вызове функции непосредственно перед выполнением тела функции.

На месте булевского условия в прагме разрешается использовать и так называемое расширенное булевское условие, которое имеет вид «(all имя : булевское условие : расширенное булевское условие)» или вид «(is имя :

булевское условие : расширенное булевское условие)» и определяет область действия для заданного в нем имени.

Например, расширенное булевское условие (all i: i>2: A[i]=0) истинно, если равны нулю в массиве или потоке A все те его элементы, у которых индекс больше двух, а условие (is i: i>2: A[i]=0) истинно, если в A существует хотя бы один нулевой элемент с индексом больше двух.

4.2. Прагма non_used

Перед каждым выражением могут располагаться прагмы «non_used = список значений», которые сразу после вычисления выражения становятся ненужными (в дальнейшем при исполнении программы никак не используются) и могут быть удалены из программы.

Как и в прагме «assert», значения обозначаются своими именами, результат унарного выражения обозначается через одиночный символ подчеркивания «_», а арности n-арного (n>1) выражения обозначаются как «_[1]», …, «_[n]».

Если ненужной является некоторая часть составного значения, то она задается с помощью заключенного в квадратные скобки списка полей, положения или выбирающего выражения в зависимости от того, является ли составное значение записью, массивом или потоком.

Прагмы «non_used = список результатов» могут располагаться перед ключевым словом «*returns*» в объявлениях функции и указывать на ненужность возвращаемых значений функции или их частей.

4.3. Прагма no_error

Прагма «no_error» может располагаться перед выражениями и обозначает, что результат выражения не может быть ошибочным значением.

4.4. Прагма trip_count

Прагма «trip_count = целочисленное выражение» может располагаться перед выражениями цикла и обозначает типичное количество итераций цикла, которое должно быть известно во время компиляции.

4.5. Прагма cloud_compute

Прагма «cloud_compute» может располагаться перед выражениями и обозначает, что значение выражения нужно вычислить удаленно.

4.6. Прагма weight

Прагма «weight = целочисленное выражение» может располагаться перед объявлениями инородных функций и обозначает приблизительное количество абстрактного времени, необходимого в среднем для исполнения данной функции, которое должно быть известно во время компиляции.

4.7. Прагмы пользовательских редукций

Определение поля записи, задающее пользовательскую редукцию, с именем «ini» может быть помечена прагмой «identity», если определена функция «join» и возвращаемое значение является единичным значением типа **T** относительно операции «join», описываемой ниже: «join(a, ini()) = join(ini(), a) = a».

Если начальное редукционное значение было построено функцией, не помеченной прагмой «identity», то определять отдельную функцию «join» нет смысла. Объявление или определение функции «join» можно помечать прагмой «associative», если функция ассоциативна: «join(join(a, b), c) = join(a, join(b, c))». Объявление или определение функции «join» можно помечать прагмой «commutative», если функция

коммутативна: «join(a, b) = join(b, a))». Если функция «join» ассоциативна, то она будет использована для параллельного вычисления значений редукции, тем самым определять отдельную не ассоциативную функцию «join» смысла нет. Если функция «join» ассоциативна и коммутативна, то она будет использована для (более эффективного) асинхронного параллельного вычисления значений редукции.

Приложение 1. Интерфейс стандартного смешанного модуля языка C

Далее приведен текст интерфейса инородного модуля, определяющего стандартные инородные типы и функции преобразования между ними и типами языка Cloud Sisal.

```
interface C

type c_char = "char"
function convert (:character returns c_char)
function convert (:c_char returns character)

type c_int8  = "__int8"
type c_int16 = "__int16"
type c_int32 = "__int32"
type c_int64 = "__int64"
function convert (:integer returns c_int8)
function convert (:integer returns c_int16)
function convert (:integer returns c_int32)
function convert (:integer returns c_int64)
function convert (:c_int8  returns integer)
function convert (:c_int16 returns integer)
function convert (:c_int32 returns integer)
function convert (:c_int64 returns integer)

type c_real32 = "float"
type c_real64 = "double"
type c_real80 = "long double"
function convert (:real returns c_real32)
function convert (:real returns c_real64)
function convert (:real returns c_real80)
function convert (:c_real32 returns real)
function convert (:c_real64 returns real)
function convert (:c_real80 returns real)

type c_pint8  = "__int8*"
```

```
type c_pint16 = "__int16*"
type c_pint32 = "__int32*"
type c_pint64 = "__int64*"
function convert (:integer  returns c_pint8)
function convert (:integer  returns c_pint16)
function convert (:integer  returns c_pint32)
function convert (:integer  returns c_pint64)
function convert (:c_pint8  returns integer)
function convert (:c_pint16 returns integer)
function convert (:c_pint32 returns integer)
function convert (:c_pint64 returns integer)
function delete  (:c_pint8  returns "")
function delete  (:c_pint16 returns "")
function delete  (:c_pint32 returns "")
function delete  (:c_pint64 returns "")

type c_preal32 = "float*"
type c_preal64 = "double*"
type c_preal80 = "long double*"
function convert (:real returns c_preal32)
function convert (:real returns c_preal64)
function convert (:real returns c_preal80)
function convert (:c_preal32 returns real)
function convert (:c_preal64 returns real)
function convert (:c_preal80 returns real)
function delete  (:c_preal32 returns "")
function delete  (:c_preal64 returns "")
function delete  (:c_preal80 returns "")

type c_psz = "char*"
function convert (:string returns c_psz)
function convert (:c_psz returns string)
function delete (:c_psz returns "")

type c_psz_static = "char*"
function convert (:c_psz_static returns string)

type c_pvoid = "void*"
function convert (:array of integer returns c_pvoid)
```

```
function convert (:array of real returns c_pvoid)
function make_array (:c_pvoid, len: integer, sizeof: integer
                                returns array of integer)
function make_array (:c_pvoid, len: integer, sizeof: integer
                                returns array of real)
function delete (:c_pvoid returns "")

type c_pvoid_static = "void*"
function make_array (:c_pvoid_static, len: integer, sizeof:
integer
                                returns array of integer)
function make_array (:c_pvoid_static, len: integer, sizeof:
integer
                                returns array of real)
```

Приложение 2. Список ключевых слов

В данном разделе приведен список 27 ключевых слов языка Cloud Sisal в алфавитном порядке.

array	at	case	cross
do	else	elseif	end
error	false	for	function
if	import	in	interface
let	module	of	old
record	returns	stream	then
true	type	while	

Приложение 3. Упрощения языка Sisal 3.2

В данном разделе приведен список упрощений языка Sisal 3.2.

Упрощения системы типов:

- нет пользовательских операций ввиду усложнения синтаксиса во многих местах;
- нет пользовательских типов ввиду отсутствия пользовательских операций;
- нет обобщенных процедур ввиду значительного усложнения семантики;
- нет контрактов ввиду отсутствия обобщенных процедур;
- нет типа union ввиду его сомнительных преимуществ перед потоками для задания рекурсивных типов;
- нет типа null ввиду его сомнительной пользы без типа union;
- нет типа массива с фиксированной формой ввиду неоправданного усложнения многих синтаксических конструкций языка (при необходимости может быть задана средствами прагм);
- нижняя граница размерностей массивов не может быть изменена ввиду усложнения вида алгоритмов;
- есть встроенный комплексный тип ввиду отсутствия пользовательских операций для его задания.

Упрощение выражений:

- нет выражения case tag ввиду устранения типа union;

- нет операции dot для устранения синтаксического сахара (замещается использованием имен рядов значений);
- может быть истинным только один тест выражения *case* для увеличения роли конструкции *case* для описания параллельного выбора;
- нет теста until в циклическом выражении для избавления от синтаксического сахара;
- нет фильтра unless для редуцируемой последовательности цикла для избавления от синтаксического сахара.

Упрощение объявлений инородных функций, которое использует нумерацию параметров вместо модификаторов in, out.

Количество ключевых слов уменьшено с 42 до 27.

Список литературы

1. **Касьянов В. Н.**, **Стасенко А. П.** Язык программирования Sisal 3.2 // Методы и инструменты конструирования программ. — Новосибирск: ИСИ СО РАН, 2007. — С. 56–134.
2. **ANSI X3.4:1986.** Information systems: coded character sets: 7-Bit American national Standard Code for Information Interchange (7-Bit ASCII). — NY: American National Standards Institute (ANSI), 1986.
3. **ANSI/IEEE 754-1985.** IEEE standard for binary floating-point arithmetic. — NY: Institute of Electrical and Electronics Engineers, 1985 (Reprinted in SIGPLAN Notices, 22(2):9-25, 1987).
4. **Легалов А. И.** Использование асинхронных вычислений в функциональных языках параллельного программирования. // Распределенные и кластерные вычисления. Избранные материалы четвертой школы-семинара. — Красноярск: Институт вычислительного моделирования СО РАН, 2005. — С. 172–183.

Оглавление

В.Н. Касьянов , Е.В. Касьянова

ЯЗЫК ПРОГРАММИРОВАНИЯ CLOUD SISAL

Препринт
181

Рукопись поступила в редакцию 10.05.2018

Редактор Т. М. Бульонкова

Рецензент В.И. Шелехов

Подписано в печать 04.06.2018

Формат бумаги 60 × 84 1/16 Объем 1.8 уч.-изд.л., 2.0 п.л.

Тираж 50 экз.

Центр оперативной печати «Оригинал 2»
г.Бердск, ул. О. Кошевого, 6, оф. 2, тел. (383-41) 2-12-42