Подходы к переводу и компиляции в многоязыковой системе

В. А. Ковыршина[1], А. Г. Леонов[2], М. В. Райко[3]

[1] НИЦ «Курчатовский институт» - НИИСИ, Москва, Россия, potchtovy_jashik@mail.ru

[2] НИЦ «Курчатовский институт» - НИИСИ, МПГУ, МГУ, Москва, Россия, dr.l@vip.niisi.ru

[3] НИЦ «Курчатовский институт» - НИИСИ, МПГУ, Москва, Россия, rayko@niisi.ru

Аннотация. Цифровая трансформация образования актуализирует задачу снижения порога входа в программирование для самой юной аудитории. В качестве решения предлагается использование блочных сред, таких как «ПиктоМир-К», которые позволяют концентрироваться на алгоритмической составляющей, минуя сложности профессиональных инструментов. В статье детально описывается ядро многоязыковой учебной среды программирования, построенное на разделении универсального синтаксического дерева SyntaxTree), хранящего семантику программы, и визуального дерева (VisualTree), ответственного за отрисовку представлений программы на различных языках программирования. Такой подход реализует функции многоязыковости, позволяя мгновенно переключать представление кода между различными синтаксисами (КуМир, Python, C++). Кроме того, синтаксическое дерево используется для компиляции программы в набор инструкций для виртуальной стековой машины. Показано, что предложенная архитектура является гибкой и расширяемой, открывая возможности для поддержки новых языков и трансляции в различные исполняемые форматы.

Ключевые слова: компиляторы, стековая машина, синтаксическое дерево

1. Введение

Цифровая трансформация общества обуславливает растущий интерес к преподаванию азов программирования детям самых ранних возрастов. Ключевой задачей становится снижение порога входа в деятельности, которые предлагаются детям для ознакомления с основными понятиями программирования. Среди таких деятельностей наиболее важной и трудной является самостоятельное составление программ в какой-то среде программирования. Традиционные среды, как правило, требуют от учащихся непривычных или логически непростых действий: клавиатурного ввода текста; чтения и ввода служебных слов в латинской транскрипции и, что труднее всего, поиска и исправления синтаксических ошибок в программе. Требуются специальные подходы, для того, чтобы не погасить интерес учащихся из-за необходимости выполнять подобные действия.

Одним из успешно работающих подходов служат блочные среды программирования, применяемые во вводных курсах программирования. В таких средах используется традиционное текстовое изображение кода программы, однако ввод кода проводится не клавиатурно, а пиктограммно, путем манипуляций блоками, представляющими синтаксически правильные конструкции используемого языка программирования.

Примером отечественной блочной среды программирования является среда ПиктоМир-К [1], которая предоставляет учащемуся предоставляет ограниченный, но педагогически достаточный набор школьного алгоритмического языка. Главное преимущество среды ПиктоМир-К — обеспечение корректности синтаксиса на всех этапах составления программы, что делает ненужными этапы лексического и синтаксического анализа.

Традиционный подход, при котором начинающие сразу работают со сложными учебными или даже промышленными языками программирования (школьный алгоритмический язык в среде КуМир, C++, Python), вынуждает новичка при решении учебных задач одновременно преодолевать и трудности придумывания алгоритма и трудности освоения интерфейса среды разработки и синтаксиса языка программирования. Инструменты блочного программирования, будучи специализированным педагогическим решением, демонстрируют большую эффективность на начальном этапе, уменьшая затраты на освоение интерфейса среды программирования и синтаксиса используемого языка. Это позволяет учащемуся

концентрироваться на алгоритмических аспектах решаемых учебных задач.

Особенности блочной организации процесса составления программы позволяют разработчикам среды программирования ПиктоМир-К избежать использования тяжеловесных библиотек для компиляции и выполнения программ и создать собственные упрощенные алгоритмы компиляции и интерпретации кода программы, ориентированные на упрощенный синтаксис и использование всего лишь двух базовых типов данных.

2. Описание языка ЦОС ПиктоМир-К

Программирование в ПиктоМир-К начинается с визуального конструирования: ученик собирает код из пиктограмм, которые система автоматически преобразует в текст, объединенный в логические блоки. В качестве основы используется подмножество школьного алгоритмического языка, реализованного в отечественной учебной среде программирования КуМир. Функционал выбранного подмножества языка сознательно ограничен. Для лучшего представления структуры кода задействованы синтаксические диаграммы Вирта — метод визуализации, разработанный для языка Паскаль.

Центральная концепция — "исполнитель". Доступно 9 различных исполнителей (Вертун, Черепаха, Робот и др.), каждый со своим уникальным набором команд, представленных в виде пиктограмм. Все действия происходят в виртуальной "обстановке", где робот, соответствующий выбранному исполнителю, выполняет составленную учащимся программу.

Программа структурно состоит из директивы подключения исполнителя и основного алгоритма без имени. Для создания функций (без параметров или с параметрами) по умолчанию предусмотрены пять стандартных имен (первые буквы алфавита). Отдельные команды исполнителей (например, у Черепахи) также используют параметры. Таким образом, ученик, манипулируя командами, может управлять роботом, рисуя и перемещая его по полю.

Основу языка Пикто-К составляют следующие элементы:

Управляющие конструкции: несколько видов циклов и условные операторы.

Работа с данными: объявление переменных, присваивание значений и математические операции.

Система типов: язык строго типизирован и включает только два типа — целый и логический.

Условные операторы представлены формами «если-то» и «если-то-иначе». Условия в них задаются с помощью логических выражений, переменных или констант «да» и «нет».

Грамматика алгоритмического языка Пикто-К подробно изложена в статье «Пиктограммный язык программирования «Пикто» авторов Бесшапошников Н.О., Леонов А.Г. [2]

3. Многоязыковость и архитектура визуализации

Представление программы в системе ПиктоМир-К основано на параллельном существовании двух синхронизированных структур данных:

Универсальное синтаксическое дерево (*SyntaxTree*): содержит чистую семантику программы, независимую от способа её отображения.

Универсальное визуальное дерево (*VisualTree*): отвечает за отрисовку программы, используя модули визуального представления (рендереры).

Синтаксическое дерево *SyntaxTree* состоит из узлов класса *Node*, а визуальное дерево *VisualTree* состоит из узлов класса *NodeLayer* или дочерних от него классов. Узлы этих двух деревьев находятся во взаимно однозначном соответствии: каждому узлу *Node* в *SyntaxTree* соответствует узел *NodeLayer* в *VisualTree* и наоборот.

По завершении сеанса работы пользователя с какой-либо учебной программой, в архиве пользователя сохраняется только *SyntaxTree*, и по нему при новом открытии данной учебной программы в среде ПиктоМир-К заново строится *VisualTree*. После этого пара деревьев (*SyntaxTree*, *VisualTree*), соответствующих данной программе, в ходе сеанса редактирования учебной программы пользователем изменяются одновременно и взаимосогласованно.

Пользователь работает с программой посредством взаимодействия с *VisualTree*. Визуальное дерево, будучи *UI*-объектом отлавливает и обрабатывает действия пользователя (перетаскивания, клики), в частности, добавление/удаление/изменение узлов, а также смену формы графического представления программы. Создание, изменение или удаление визуального (*NodeLayer*) и синтаксического (*Node*) узлов происходит одновременно и согласованно, за счёт этого и обеспечивается полное соответствие синтаксического и визуального деревьев.

В то время как класс *Node* хранит всю

логическую и смысловую информацию (тип узла *type*, значение *value*, идентификатор *id* и связи с родителем *parent* и дочерними узлами *children*[]), он не содержит никакой информации о том, как соответствующий узел будет изображен графически. Его графическая реализация *NodeLayer* содержит шаблон для визуального представления *Node* и знает, как «перевести» этот шаблон на конкретный язык программирования. Данный шаблон состоит из компонент-рендереров, которые неизменны и не зависят от синтаксиса, они перерисовываются при его смене. То, в каком синтаксисе мы работаем, учитывается непосредственно при рендеринге *NodeLayer*.

Такой подход позволяет реализовать в системе ПиктоМир-К возможность программирования на разных языках без перестройки самой программы. Для одной и той же пары (*SyntaxTree*, *VisualTree*) достаточно выбрать модуль визуального представления, и все узлы *NodeLayer* будут автоматически нарисованы с новыми терминалами. При переключении языка представления программы каждая вершина *VisualTree* перерисовывается независимо и автоматически. Опишем подробнее, как это происходит.

3.1. Принцип работы механизма визуализации

Класс *NodeLayer* (и, соответственно, всякий дочерний от него класс) хранит следующую информацию:

parent (**NodeLayer*) – родительский узел в *VisualTree*,

node (**Node*) – соответствующий узел в *SyntaxTree,*

syntax (*String*) – текущий модуль визуального представления (например, "kumir", "python", "cpp"), то есть синтаксис, в котором представлена программа.

Рассмотрим, по какому принципу строится и изменяется *NodeLayer*, на примере языков Python и Кумир, реализованных в ПиктоМир-К. *NodeLayer*'ы всех типов имеют универсальные конструкции, «перевод» на конкретный язык программирования происходит посредством замены терминалов его конструкции. Ниже приведена таблица 1, описывающая эти замены.

Таблица 1. Описание замены терминалов

Тип Node	**Конструкция NodeLayer**	**Python**		**КуМир**	
		компонента	**терминал**	**компонента**	**терминал**
Prog	использовать <robot name> eol	использовать eol	*from* *import **	использовать eol	*использовать* -
MainAlg	алг нач <body> кон	алг нач кон	- - -	алг нач кон	*алг* *нц* *кц*
Func	алг <identifier> [<args>] eol нач <body> кон	алг eol нач кон	*def* *:* - -	алг eol нач кон	*алг* - *нц* *кц*
Body	<Empty>	<tab> <Empty>		■ <Empty>	
Statement	<identifier> := <expression> ;	:= ;	= -	:= ;	:= ;

IfStatement	если <condition> eol то <body> [ElseBlock] все	если eol то все	*if* : - -	если eol то все	*если* - *то* *всё*
ElseBlock	[иначе eol <body>]	иначе eol	*else* :	иначе eol	*иначе* -
Loop (nTimes)	нц <expression> раз <body> кц	нц раз кц	for _ in range( ) : -	нц раз кц	*нц* *раз* *кц*
Loop (for)	нц для <identifier> “от” <expression> до <expression> eol <body> кц	нц для от до eol кц	for in range(,) : -	нц для от до eol кц	*нц для* *от* *до* *eol* *кц*
Loop (while)	нц пока <condition> eol <body> кц	нц пока eol кц	*while* : -	нц пока eol кц	*нц пока* - *кц*
Type	тип	тип	-	тип	цел | лог
Identifier	“a1, a2, …”, “л1, л2, ..»	“a1, a2, …”, “л1, л2, ..»		“a1, a2, …”, “л1, л2, ..»	
Number	0 - 999	0 - 999		0 - 999	
Bool	value	value	True | False	value	да | нет
Expression	“add” | “sub” | “mult” | “divide”	+ | - | * | /		+ | - | * | /	
LogicExpression	“llr”, “rll”, “equal”, ”not-equal”, ‘llr-equal’, ‘rll-equal’	< | > | == | != | <= | >=		< | > | == | != | <= | >=	
Alloc	тип <identifier>	тип	-	тип	цел | лог
Empty	-	-		-	

Условные обозначения в таблице 1:

- `eol` — конец строки (учитывается в форматировании)
- `<tab>` — табуляция
- [ ] — необязательные элементы
- <Empty> – не отображаемая

графически компонента, позволяющая вставлять в указанное место новые узлы.

Аналогичным образом происходит перевод языка Кумир с русского на английский: терминалы на русском заменяются в соответствии с таблицей терминалами на английском.

3.2. Преимущества архитектуры с двумя параллельно поддерживаемыми согласованными деревьями

Данный механизм имеет четыре ключевых преимущества:

Универсальность синтаксического дерева: SyntaxTree служит единым источником истины для компилятора, отладчика и визуализатора, обеспечивая согласованность всех компонентов системы.

Удобство технической реализации всей системы: для компиляции программы достаточно SyntaxTree, а для визуализации – VisualTree. Разделение этих двух задач делает процессы компиляции и визуализации независимыми, что ускоряет работу системы, также упрощает архитектуру узлов Node и NodeLayer.

Модульность визуализации: Система визуализации построена как набор независимых модулей-рендереров. Добавление поддержки нового языка программирования (например, C, JavaScript) сводится к созданию нового модуля, который "знает" соответствующие терминалы для универсальных шаблонов NodeLayer.

Производительность: При смене языка представления не требуется перестраивать структуру деревьев — достаточно обновить терминалы в существующих узлах NodeLayer, что обеспечивает мгновенное переключение между языками.

Такой подход не только реализует многоязыковость, но и создает фундамент для будущих расширений, таких как поддержка новых парадигм программирования или специализированных предметно-ориентированных языков (DSL).

В таблице 2 приведен пример того, что должны были бы начать делать разработчики среды ПиктоМир-К, если бы потребовалось добавить представление кода программы в синтаксисе языка C++:

Таблица 2. Перевод для добавления языка C++

Тип Node	**Конструкция NodeLayer**	**C++**	
		компонента	**терминал**
Prog	использовать <robot name> eol	использовать eol	#include “ >.h”

MainAlg	алг
	нач
	<body>
	кон
		алг
		нач
		кон
			void main()
			{
			}

Func	алг <identifier> [<args>] eol
	нач
	<body>
	кон
		алг
		eol
		нач
		кон
			void
			-
			{
			}

Body	<Empty>	<tab> <Empty>	
Statement	<identifier> := <expression> ;	:= ;	= ;

IfStatement	если <condition> eol
	то
	<body>
	[ElseBlock]
	все
		если
		eol
		то
		все
			if (
			)
			{
			}

ElseBlock (частный случай Body)	[иначе eol <body>]	иначе eol	} *else* {

Loop (nTimes)	нц <expression> раз
	<body>
	кц
		нц
		раз
		кц
			for(j = 0; j <=
			; j ++) {
			}

Loop (for)	нц для <identifier> “от” <expression>
	до <expression> eol
	<body>
	кц
		нц для
		от
		до
		eol
		кц
			for(
			=
			; <identifier> <=
			; <identifier> ++) {
			}

Loop (while)	нц пока <condition> eol
	<body>
	кц
		нц пока
		eol
		кц
			while (
			){
			}

Type	тип	тип	int | bool
Identifier	“a1, a2, …”, “л1, л2, ..»	“a1, a2, …”, “л1, л2, ..»	
Number	0 - 999	0 - 999	
Bool	value	value	true | false
Expression	“add” | “sub” | “mult” | “divide”	+ | - | * | /	
Condition	Команды вопросы конкретного робота	Команда вопрос	
LogicExpression	“llr”, “rll”, “equal”, ”not-equal”, ‘llr-equal’, ‘rll-equal’	< | > | == | != | <= | >=	
Action	Методы конкретного робота или название вызывамой функции	Метод или функция	
Alloc	тип <identifier> eol	тип eol	int | bool ;

Empty	-	-

Как видно, нам бы потребовалось добавить компоненту `eol` в конец AllocNode (NodeLayer для Node типа Alloc), однако для этого достаточно было бы доопределить замены добавленной компоненты `eol` на пустой терминал - в Python и в Кумире. Таким образом, функционал ПиктоМир-К может быть легко адаптирован в случае добавления нового языка, даже если синтаксические конструкции каких-то его компонент не укладываются в имеющийся шаблон.

3.3. Компиляция программы

На этапе компиляции осуществляется обход синтаксического дерева и генерации соответствующих инструкций. В результате обхода дерева появляется набор инструкций для виртуальной стековой машины. Эта виртуальная машина выполняет определенные действия в зависимости от типа инструкции и ее данных. Для выполнения любой программы на языке Пикто-К. требуется всего лишь 14 инструкций, подробно описанных в таблице 3.

Таблица 3. Описание инструкций для выполнения программы на языке Пикто-К

№	Тип	Описание	Аргументы
0	Execute	Выполняет метод робота либо подпрограмму. При выполнении подпрограммы с вершины стека достаются параметры, в стек заносится текущая позиция в списке инструкций, а в регистр вызовов функций заносится текущая позиция в стеке для определения области локальных данных этого метода.	isNative– является ли вызовом подпрограммы methodID – уникальный идентификатор метода робота paramCount– количество параметров, которые нужно взять из регистра.
1	Check Condition	Выполняет проверку условия для робота и кладет результат на вершину стека.	robot – ссылка на исполнителя condition – уникальный идентификатор условия для проверки роботом
2	Start loop	Является индикатором начала цикла, задает начальное значение итератору цикла. Достает с вершины стека текущее значение итератора и конечное, если текущее значение> конечного, то переходит на конец блока цикла. В противном случае кладет на вершину стека начальное и конечное значение итератора.	blockEndLabel – ссылка на конец блока цикла repType – тип цикла
3	End loop	Достает с вершины стека текущее и конечное значения итератора. Если итератор <= конечному значению, то выполняет переход в начало цикла, увеличивает итератор на единицу и кладет в стек текущее значение итератора и конечное.	jumpLabel – ссылка для перехода для выполнение следующей итерации цикла robot – ссылка на исполнителя
4	Return	Выполняет выход из подпрограммы, если есть выходные параметры, то достает их из вершины стека и кладет в регистр параметров, далее получает из стека вызовов функций позицию, начиная с которой в нем расположены локальные данные этой функции. Удаляет все локальные данные и последнее значение из регистра вызовов. Если регистр вызовов пуст – программа	paramQuantity – количество параметров, которые нужно взять из регистра

		завершилась.	
5	Jump_if	Переход по ссылке, если условие на вершине стека верно.	jumpLabel – ссылка для перехода
6	Jump_nif	Переход по ссылке, если условие на вершине стека неверно.	jumpLabel – ссылка для перехода
7	Jump	Безусловный переход по ссылке.	jumpLabel – ссылка для перехода
8	Alloc	Выделяет место для переменной в памяти, если вызов осуществляется вне функции или кладет переменную в стек, если в регистре вызовов функции уже есть вызов.	varType – тип переменной varName – имя переменной value – значение переменной
9	Fetch	Кладет на вершину стека значение переменной, ищет значение этой переменной в стеке в области видимости этой функции.	varName – имя переменной
10	Push	Кладет в стек значение	value - значение
11	Store	Сохраняет значение с вершины стека или из регистра параметров в переменной.	varName – имя переменной fromReg – взять значение из регистра параметров или из стека
12	Pop	Удалить последнее значение из стека	-
13	Calc	Выполняет операцию над последними двумя значениями с вершины стека.	operation - операция

Таким образом, программа, описываемая синтаксическим деревом, переводится в набор ассемблер-подобных инструкций для виртуальной исполняющей машины. Виртуальная исполняющая машина состоит из стека, регистров и памяти. Стек хранит локальные данные функций и операций. Регистр вызовов функций сохраняет в себе позицию в стеке, начиная с которой расположены локальные данные вызванной функции. Регистр параметров сохраняет входные и выходные параметры функций. Память – это ассоциативный массив глобальных переменных в смысле работы [3].

4. Универсальность компилятора и целевые платформы

Универсальное синтаксическое дерево служит не только для мультиязыковой визуализации, но и как идеальное промежуточное представление (Intermediate Representation, IR) для компиляции. Набор инструкций для стековой машины, генерируемый из дерева, является лишь одной из возможных целевых платформ.

Алгоритмы обхода дерева и генерации кода являются универсальным. Это открывает возможности для трансляции программ ПиктоМир-К в другие формы исполнения:

Трансляция в «нативный код»: при наличии соответствующего бэкенда, обходчик синтаксического дерева может генерировать не инструкции для виртуальной машины, а, например, исходный код на C или JavaScript, который затем можно выполнить с максимальной производительностью.

Исполнение на микроконтроллерах: Тот же самое дерево можно использовать для генерации машинного кода или упрощенного байт-кода для образовательных робототехнических платформ (например, на базе Arduino). Это позволит программам, написанным детьми в ПиктоМире, управлять реальными физическими устройствами.

Интеграция с другими образовательными средами: Синтаксическое дерево можно сериализовать в универсальный формат (например, JSON или XML) и импортировать в другие системы, что делает ПиктоМир-К открытой и интегрируемой платформой.

5. Заключение

Описанный метод компиляции и выполнения программы реализован на языке JavaScript [4] в обучающей среде программирования ПиктоМир-К. Данная система позволяет выполнять программы, созданные в среде ПиктоМир-К для целей обучения азам алгоритмики.

Предложенная архитектура с универсальным синтаксическим деревом в качестве ядра и независимыми модулями визуализации и компиляции демонстрирует высокую степень гибкости и расширяемости. Она не только решает задачу обучения основам алгоритмизации, но и создает прочный фундамент для будущего развития системы. За счет описанных принципов обеспечивается:

Низкий порог для добавления поддержки новых языков программирования.

Легкость модификации существующих языковых представлений.

Потенциал для трансляции в различные исполняемые форматы и платформы.

Это делает ПиктоМир-К не просто еще одной средой блочного программирования, а универсальным инструментом для построения моста между визуальным, блочным и текстовым программированием.

Работа выполнена в рамках темы государственного задания НИЦ «Курчатовский институт» - НИИСИ по теме № FNEF-2024-0001, этап 2025 года (1023032100070-3-1.2.1).

Approaches to Translation and Compilation in a Multilingual System

V. A. Kovyrshina, A. G. Leonov., M. V. Rayko

Abstract. Digital transformation in education brings to the forefront the task of lowering the entry barrier into programming for the youngest audience. As a solution, the use of block-based environments, such as "PiktoMir-K," is proposed. These environments allow users to focus on the algorithmic component, bypassing the complexities of professional tools. The article details the system's core, which is built on the separation of a universal SyntaxTree, storing the program's semantics, and a VisualTree, responsible for its rendering. This approach implements a multilingualism function, allowing for instant switching of the code representation between different syntaxes (KuMir, Python, C++). Furthermore, the syntax tree is used to compile the program into a set of instructions for a virtual stack machine. It is shown that the proposed architecture is flexible and extensible, opening up possibilities for supporting new languages and translation into various executable formats.

Keywords: compilers, stack machine, syntax tree

Литература

1. Стартовая страница проекта «ПиктоМир - К» на сайте НИЦ «Курчатовский институт» - НИИСИ . URL: https://www.niisi.ru/piktomir/ (дата обращения 01.10.2025)
2. Бесшапошников Н.О., Леонов А.Г. Пиктограммный язык программирования «Пикто» // Вестник кибернетики. 2017. № 4 (28). С. 173–180
3. Райко М.В. Построение компилятора-интерпретатора для гибридной текстово-пиктограммной цифровой образовательной среды ПиктоМир-К / М.В. Райко, Д.Б. Аглямутдинова, А.Г. Леонов // Труды НИИСИ РАН. — 2020. — Т. 10, № 5-6. — С. 148–160
4. ISO/IEC 22275:2018. Information technology — Programming languages, their environments, and system software interfaces — ECMAScript Specification Suite.