APRICOT USER GUIDER

APRICOT USER GUIDER

FANG HUI-XING

APRICOTRESEARCH.COM

Published by FANG Hui-Xing & APRICOTRESEARCH.COM, Shanghai, China.

Apricot User Guider, etc

To my parents and friends

CONTENTS IN BRIEF

CONTENTS

LIST OF FIGURES

LIST OF TABLES

FOREWORD

INTRODUCTION

PART I

THE MODELING LANGUAGE

CHAPTER 1

THE SYNTAX OF APRICOT

1.1 Identifiers

An identifier is an unlimited-length (but the length is greater than one) sequence of letters and digits, but not a Keyword:

$$
\begin{aligned}
Letter &::= \texttt{a..z} \mid \texttt{A..Z};\\
Digit &::= \texttt{0..9};\\
ValidChar &::= Letter \mid Digit;\\
Identifier &::= \texttt{\^{}?(}\ Letter \mid \texttt{\_}\ \texttt{)(}\ ValidChar \mid \texttt{\_}\ \texttt{)*;}.
\end{aligned}
$$

In which the letter is defied as the character in the set { a, b, c, d, e, f, g, h, i, j, k, l, m, n, o, p, q, r, s, t, u, v, w, x, y, z, A, B, C, D, E, F, G, H, I, J, K, L, M, N, O, P, Q, R, S, T, U, V, W, X, Y, Z }.

1.2 Types, Values, and Variables

The types of the Apricot language are divided into three categories: primitive types, mathematic types and reference types.

$$Type ::= PrimitiveType \mid MathematicType \mid ReferenceType;$$

1.2.1 Primitive Types

The primitive type is defined by:

$$PrimitiveType \quad ::= \quad \texttt{int} \mid \texttt{real} \mid \texttt{boolean} \mid \texttt{String};$$

The Boolean type represents a logical quantity in the literals set $\{\texttt{true}, \texttt{false}\}$.

1.2.2 Mathematic Types and Values

Primitive type variable can not be shared and has the feature of "call-by-value" during method calls. Call-by-value requires the evaluation of the arguments before passing them to the definition of the method. Another style is call-by-name which passing the arguments directly to the definition. For mathematic and reference types we take the call-by-name style argument passing for method invocation. In addition, there is a difference between mathematic type and reference type. Reference type variables can refer to another object with the same type by the assignment statement. But, the assignment can only change the mathematical value of the object for mathematic type variables. It means that, when a mathematic type variable refers to a methematic type object for the first time, the variable will hold this object all the time and only the mathematical value of this object can be updated.

$$MathematicType \quad ::= \quad \texttt{Int} \mid \texttt{Real};$$

1. Mathematic types : Numeric types. The numeric types are the integer type Int, and the real number type Real;

2. Reference types : class types, interface types, and array types.

An object is a dynamically created instance of a class type or a dynamically created array. The values of a reference type are references to objects.

1.2.3 Reference Types and Values

There are four kinds of reference types: class types, interface types, type variables.

$$
\begin{aligned}
ReferenceType ::= & ClassType \mid InterfaceType \\
& \mid ArrayType \mid IntervalType; \\
ClassType ::= & Identifier; \\
InterfaceType ::= & Identifier \mid \texttt{System} \mid \texttt{Plant} \\
& \mid \texttt{Controller} \\
& \mid \texttt{Dynamic} \mid \texttt{Assignment} \\
& \mid \texttt{ParallelAssignment} \\
& \mid \texttt{SequentialAssignment}
\end{aligned}
$$

1.2.4 Variables

A variable is a physical quantity name in physical world or a storage location in the memory of computer, and has an associated type that is either a mathematic type or a reference type.

The value of a variable is changed by an assignment or according to the differential equations defined in *Dynamic* classes.

For all types, the default value of any type variable is the special value `null`.

1.2.4.1 Variables of Mathematic Type Mathematic type variables are always hold a mathematic value of that exact mathematic type.

1.2.4.2 Variables of Reference Type A variable of a reference type `R` can hold a null reference, a reference to an instance of class *C*, any class that is a subclass of *C*, any class that is a implementation of interface *C* or any array type.

1.3 Mathematical Operations

1.3.1 Arithmetic Operators

For $x, y \in \mathcal{R}$, the following arithmetic operators are defined on Real numbers ($\mathcal{R}$):

1. $x + y$, binary plus, addition;
2. $x - y$, binary minus,subtraction;
3. $x * y$, binary multiple, multiplication;
4. x/y, binary divide, division;
5. $+x$, unary plus, it denotes the identity operation on x, thus, $x == +x$ with respect to the evaluation;
6. $-x$, unary minus, inverse operation on x, thus, $(-x) + x == 0$.

1.3.2 Boolean Operators

Standard boolean operators are defined for all *Boolean* type values x, y:

1. $==$, equality;
2. $!=$, inequality;
3. !, logical complement;
4. `in`, belong to interval, the result value of (x `in` $[a, b]$) is `true` iff $a \leq x \leq b$;
5. `and`, the result value of (x `and` y) is `true` if both operand values are `true`;
6. `xor`, the result value of (x `xor` y) is `true` if the operand values are different;
7. `or`, the result value of (x `or` y) is `true` if one of the operand values is `true`.

1.3.3 Numeric Comparisons

Standard comparison operations are defined for all Real numbers ($\mathcal{R}$), which result in a value of type *Boolean*:

1. $==$, equality;
2. $!=$, inequality;
3. $<$, less than;
4. $<=$, less than or equal to;
5. $>$, greater than;
6. $>=$, greater than or equal to.

Special Symbol numbers:

1. Inf is stands for ∞, which is equal to itself and greater than any other number;
2. $-Inf$ is stands for $-\infty$, which is equal to itself and less then any other number;

1.3.4 Mathematical Functions

We provides a comprehensive collection of mathematical functions and operators. These mathematical operations are defined on Real numbers ($\mathcal{R}$).

1. $dot(x, n)$, n-th order derivative of x over time (t), i.e. $dot(x, n) = \frac{d^n x}{dt^n}$.
2. $dot(x, y, n)$,n-th order derivative of x over y, i.e. $dot(x, y, n) = \frac{d^n x}{dy^n}$.
3. Standard trigonometric functions: sin, cos, tan, cot, sec and csc.

4. $round(x)$, round x to the nearest integer, omitting decimal fractions smaller than 0.5, e.g. $round(2.5) = 3$, $round(0.4) = 0$.
5. $floor(x)$, round x towards $-Inf$, e.g. $round(2.5) = 2$.
6. $ceil(x)$, round x towards $+Inf$, e.g. $ceil(2.5) = 3$.
7. $div(x, y)$, truncated division, and quotient rounded towards zero.
8. $fld(x, y)$, floored division, quotient rounded towards $-Inf$.
9. $rem(x, y)$, remainder, satisfies $x = div(x, y) * y + rem(x, y)$, implying that sign of $rem(x, y)$ matches x.
10. $mod(x, y)$, modulus; satisfies $x = fld(x, y) * y + mod(x, y)$, implying that sign of $mod(x, y)$ matches y.
11. $gcd(x_1, x_2, ..., x_n)$, greatest common divisor of $x_1, x_2, ..., x_n$ with sign matching x_1.
12. $lcm(x_1, x_2, ..., x_n)$, least common multiple of $x_1, x_2, ..., x_n$ with sign matching x_1.
13. $abs(x)$, a positive value with the magnitude of x.
14. $sign(x)$, indicates the sign of x, returning -1, 0, or $+1$.
15. $sqrt(x)$, the square root of x, i.e. $\sqrt[2]{x}$.
16. $root(x, b)$, the b-th root of x, i.e. $\sqrt[b]{x}$.
17. $hypot(x, y)$, accurate $sqrt(x^2 + y^2)$ for all values of x and y.
18. $pow(x, y)$, x raised to the exponent y, i.e. x^y.
19. $exp(x)$, the natural exponential function at x, i.e. e^x.
20. $log(x)$, the natural logarithm of x, i.e. $\log(x)$ or $\ln(x)$.
21. $log(b, x)$, the base b logarithm of x, i.e. $\log_b(x)$.
22. $erf(x)$, the error function (Gauss error function) at x, i.e. $erf(x) = \frac{2}{\sqrt{\pi}} \int_0^x e^{t^2} dt$.
23. $gamma(x)$, the gamma function at x.
24. $max(x_1, ..., x_n)$, the maximum value in the set $\{x_i \mid 1 \leq i \leq n\}$.
25. $min(x_1, ..., x_n)$, the minimum value in the set $\{x_i \mid 1 \leq i \leq n\}$.
26. $pause(x)$, make a pause for x seconds.
27. $size(x)$, the size of x (x is an array).

1.4 Expressions

1.4.1 Primary Expressions

$$
\begin{aligned}
Primary &::= Literal \mid ParExpression \mid NewObjectExpression \mid NewArrayExpression \\
&\mid \ 'this' \ ('.' Identifier \ (DimExpr)^*)^* \ (IdentifierSuffix)? \\
&\mid \ Identifier \ (DimExpr)^* \ ('.' \ Identifier \ (DimExpr)^*)^* \ (IdentifierSuffix)? \\
IdentifierSuffix &::= ('[]')^+ \ '.class' \mid Arguments \mid '.class' \mid '.this' \mid '.super' \ Arguments \\
&\mid '.at' Arguments \mid '.Composition()' \mid '.Start()' \mid '(!)' \mid '(?)' \\
Arguments &::= '(' \ ExpressionList? \ ')' \\
ExpressionList &::= Expression \ (',' \ Expression)^* \\
DimExpr &::= '[' \ Expression \ ']' \\
Literal &::= 'Inf' \mid IntegerLiteral \mid FLOATING\_POINT\_LITERAL \mid CHARACTER\_LITERAL \\
&\mid STRING\_LITERAL \mid BooleanLiteral \mid 'null'
\end{aligned}
$$

1.4.2 Boolean Expressions

1.4.3 Conditional Expressions

1.4.4 Loop Expressions

1.4.5 Continuous Flow Expressions

CHAPTER 2

OPERATIONAL SEMANTICS FOR APRICOT

CHAPTER 3

THE APPLICATION ON SUBWAY CONTROL SYSTEMS

3.1 Case Study: Subway Control Systems

In this section, we illustrate the application of **Apricot** by a case study on subway control systems. The architecture of the model is depicted in Fig. 3.1. In this system, we have one subway line (Fig. 3.2), and four stations on the line. The train starts running from platform-A of Station-1, and goes through Station-2, 3 and 4. At each platform, the train stops for some time and picks up passengers. When it leaves the platform-A of Station-4, the direction of the trains has to change from A to B, and stops at the platform-B of Station-4 at last. Then, the train goes thought Station-3 and 2, stops at the platform-B of Station-1. When it leaves the platform, the train shall change the direction from B to A, and stops at the platform-A of Station-1, and so on.

In Fig. 3.3, where train A and B have the same running direction from left to right, and the distance between A and B is 300 meters. This distance is a urgent distance that we have to stop A with deceleration of $-1.5\ m/s^2$ when this situation occurs.

The subway control system consists of several trains, stations, one distance controller for urgent distance control between trains, and a dispatcher for controlling the running order of trains. For each train, it consists of a subsystem for the train doors, a controller for the train and some *Dynamic* objects. The train-door system contains several trains and a controller for the doors. Similarly, for the stations, each has two platforms, A and B, the reader can review the detailed structure in Fig. 3.1.

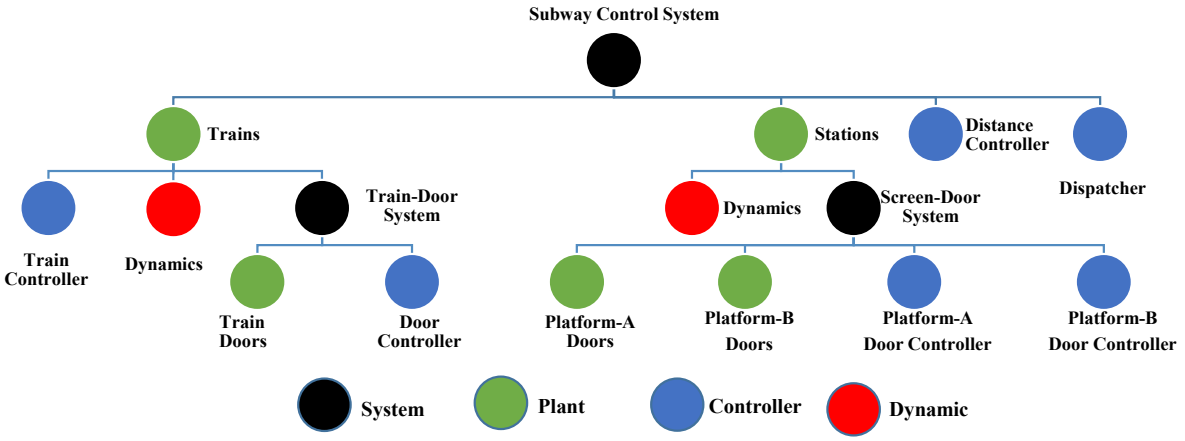

Figure 3.1 The Architecture of Subway Control System. We use four key elements (i.e., System, Plant, Controller, and Dynamic) of **Apricot** to illustrate the architecture of the model.

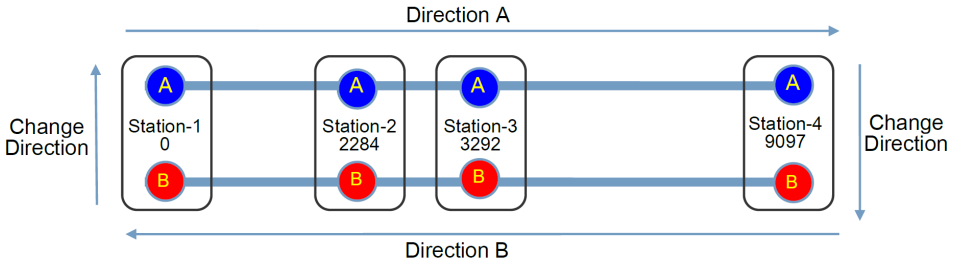

Figure 3.2 The Topology of Subway Line. We have four stations: Station-1, 2, 3 and 4. Station-1 is a reference point, the position of the station in the subway line is 0. Station-2 is positioned at 2284, which means the distance between Station-1 and 2 is 2284 meters.

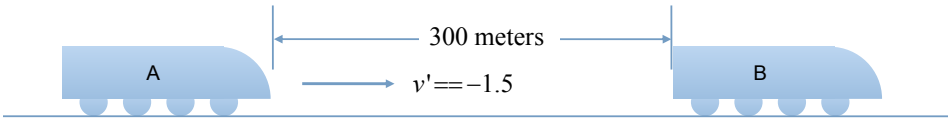

Figure 3.3 Urgent Distance Control.

PART II

FORMAL VERIFICATION AND CONTROLLER SYNTHESIS

CHAPTER 4

FORMAL VERIFICATION

PART III

SIMULATION AND ANALYSIS

PART IV

ANIMATION AND VISUALIZATION

PART V

FEEDBACK-ADVANCEMENT FRAMEWORK

PART VI

CONCLUSION