Toward a New Constraint Imperative Programming Language for Interactive Graphics

(Position Paper)

Hiroshi Hosobe
Hosei University
hosobe@acm.org

Abstract

To construct interactive graphics such as graphical user interfaces and interactive webpages is an important matter in computer programming. For this purpose, imperative programming usually has been used. On the other hand, researchers have been attempting to apply constraint programming to interactive graphics. Furthermore, the paradigm of constraint imperative programming has been proposed. This position paper reports our ongoing work on P5CP, a new constraint imperative programming language for interactive graphics. To integrate imperative and constraint programming, we adopt the notion of events in imperative programming and the notion of guards in concurrent constraint programming. We show a simple example program in this language.

Categories and Subject Descriptors D.3.2 [*Programming Languages*]: Language Classifications—Constraint and logic languages

Keywords Constraint imperative programming, Programming languages, Interactive graphics

1. Introduction

To construct interactive graphics such as graphical user interfaces and interactive webpages is an important matter in computer programming. For this purpose, imperative programming usually has been used, and imperative programming languages such as C++, Java, and JavaScript are usually employed.

On the other hand, researchers have been attempting to apply constraint programming to interactive graphics (Borning and Duisberg 1986). *Constraints* are used to denote, for example, the maintenance of consistency between internal data and their graphical representations, the specification of graphical layouts on screens, and the specification of dynamic behaviors for animations. Furthermore, the paradigm of constraint imperative programming (Felgentreff et al. 2014; Freeman-Benson and Borning 1992), which integrates imperative and constraint programming, has been proposed, although it is not yet popular.

This position paper reports our ongoing work on P5CP, a new constraint imperative programming language for interactive graphics. To integrate imperative and constraint programming, we adopt the notions of events and guards. *Events* are widely used in imperative programming to trigger and handle dynamic behaviors in interactive graphics. *Guards* are used in concurrent constraint programming (Gupta et al. 1998; Ueda et al. 2012) to describe conditions for adopting constraints. We also show a simple example program in this language.

2. Our Language

The P5CP programming language that we propose adds constraint programming to the combination of the JavaScript language and the p5.js (McCarthy 2014) library. Since JavaScript is an imperative programming language, P5CP can be regarded as a constraint imperative programming language.

In P5CP, one program mixes constraint program (CP) parts and imperative program (IP) parts, but these parts exist in such a way that they can be clearly distinguished. The execution of a program is done by alternately performing CP phases for processing CP parts and IP phases for processing IP parts. The basic computation mechanisms are guards in CP phases and events in IP phases.

States of CP and IP parts are held by CP and IP variables respectively. CP and IP parts can read values of both CP and IP variables. However, CP and IP phases can change values of CP and IP variables respectively. CP and IP variables are distinguished by their names; names that begin with a dollar sign such as `$x` and `$foo` are CP variables, and others are IP variables.

In P5CP, IP parts form the main body of a program, in which CP parts are embedded. When CP parts appear during the execution of IP phases, the creation and initialization of CP variables or the creation of constraints (possibly with guards) are performed.

CP parts follow the paradigm of hybrid concurrent constraint programming, which was developed for hybrid systems and is known for languages such as Hybrid cc (Gupta et al. 1998) and HydLa (Ueda et al. 2012). In P5CP, CP parts consist of CP variable declaration sentences and `always` sentences. A CP variable declaration sentence creates and initializes a CP variable. Syntactically, it is the same as a property declaration sentence using an equality sign in standard JavaScript, except that the variable name begins with a dollar sign. Specifically, CP variables can be declared by describing "`$x = 1;`" for a new variable `$x` in a global environment or by describing "`this.$foo = 100;`" for a new variable `$foo` in the constructor of an object.

An `always` sentence creates a constraint that should hold after the initialization of a CP variable. In this sentence, a constraint with a guard, or a guarded constraint, can be described; the constraint is adopted only when the guard holds. A guarded constraint uses the same syntax as an `if` sentence in JavaScript, describing a guard in the conditional part and one or more constraints (that also can

be guarded) in the then clause. In addition, this sentence can have its else clause. A guard is described as a JavaScript conditional expression without side effects. A constraint can be described as a one-way or an ordinary differential equation (ODE) constraint.[1] For example, a guarded ODE constraint "`if ($x > 0) { $x' == -1; }`" means that `$x` is decreased as long as `$x` is positive.[2]

IP parts are similar to JavaScript programs, except that they can include CP variables and read their values. They can include event handlers that are called when events occur.

3. Example

One of application areas of P5CP is physical simulation as with Hybrid cc and HydLa because it supports ODE constraints. Especially, P5CP has the advantage of making a simulation intuitive because it is able to visualize the simulation by using the functionality of p5.js.

Figure 1 shows an example program in P5CP and its screen. This is based on an example of HydLa (Ueda et al. 2012) that performs a simulation of a ball bouncing on a ground; it is rewritten to fit to P5CP and also to perform the drawing of the ball and the ground on a screen.[3]

In the program of Figure 1(a), lines 1–2 and 12–19 are IP, and lines 3–11 are CP. It includes only one CP variable `$y` for expressing the height of the ball, which is created and is initialized to 10 by the CP variable declaration sentence in line 3. In lines 4–11 is an `always` sentence. This includes an `if` sentence (lines 5–10), and creates an ODE constraint `$y'' == -G` (line 6) with a guard `$y > 0` (line 5) and two one-way constraints `$y = 0` (line 8) and `$y' = -E * pre $y'` (line 9) with the negation of `$y > 0` as a guard. Intuitively, when `$y > 0` holds, `$y'' == -G` is adopted, and hence the ball is accelerated by gravity. On the other hand, when `$y > 0` does not hold, `$y = 0` and `$y' = -E * pre $y'` are adopted, then the height of the ball is set to 0, and its velocity is changed from the downward to the upward.[4]

Lines 12–14 define a `setup` function, and lines 15–19 defines a `draw` function. As event handlers in p5.js, `setup` is called once to perform initialization right after the invocation of the program, and `draw` is repeatedly called to draw the screen every 1/60 seconds by default. In this program, `setup` determines the size of the screen (line 13), and `draw` clears the screen with a background color (line 16), draws the ground (line 17), and draws the ball (line 18). To draw the ball, it reads the value of the CP variable `$y`.

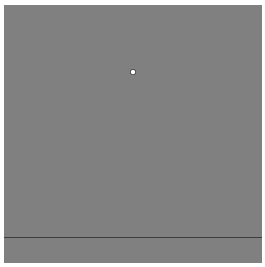
4. Status and Future Work

The implementation of the P5CP language is ongoing. It is a translator of a P5CP program to a JavaScript program. It uses a constraint solver for executing CP phases that is newly developed in JavaScript. It also uses p5.js to execute IP phases.

Our future work includes the extension of the language to enable various interactive graphical applications. Especially, it is necessary to support the dynamic creation of constraints, the explicit deletion of constraints, and the declaration of constraints for a dynamically determined number of CP variables by using a collection.

```
 1  const G = 9.8;  // gravity acceleration
 2  const E = 0.5;  // restitution coefficient
 3  $y = 10;        // height of a ball (CP variable)
 4  always {
 5    if ($y > 0) { // guard
 6      $y'' == -G; // ODE constraint
 7    } else {
 8      $y = 0;             // one-way constraint
 9      $y' = -E * pre $y'; // one-way constraint
10    }
11  }
12  function setup() {
13    createCanvas(500, 500);
14  }
15  function draw() {
16    background(128);
17    line(0, 450, 499, 450);              // ground
18    ellipse(250, 450 - 40 * $y, 10, 10); // ball
19  }
```

(a)



(b)

Figure 1. A ball bouncing on a ground: (a) its program and (b) screen.

Also, it is necessary to show more useful examples especially in the area of hybrid systems including the ones that were treated by different approaches (e.g. (Bourke and Pouzet 2013)) other than hybrid concurrent constraint programming. Another future direction is to compare our approach with functional reactive programming for ODEs and events (Elliott and Hudak 1997).

Acknowledgments

This work is supported by JSPS KAKENHI Grant Number 25540029.

References

A. Borning and R. Duisberg. Constraint-based tools for building user interfaces. *ACM Trans. Gr.*, 5(4):345–374, 1986.

T. Bourke and M. Pouzet. Zélus: A synchronous language with ODEs. In *Proc. HSCC*, pages 113–118, ACM, 2013.

C. Elliott and P. Hudak. Functional reactive animation. In *Proc. ACM ICFP*, pages 263–273, 1997.

T. Felgentreff, A. Borning, and R. Hirschfeld. Babelsberg: Specifying and solving constraints on object behavior. *J. Object Tech.*, 13(4):1:1–38, 2014.

B. Freeman-Benson and A. Borning. The design and implementation of Kaleidoscope'90, a constraint imperative programming language. In *Proc. ACM ICCL*, pages 174–180, 1992.

V. Gupta, R. Jagadeesan, and V. Saraswat. Computing with continuous change. *Sci. Comput. Program.*, 30(1–2):3–49, 1998.

L. McCarthy. P5.js overview, 2014. https://github.com/processing/p5.js/wiki/p5.js-overview .

K. Ueda, S. Matsumoto, A. Takeguchi, H. Hosobe, and D. Ishii. HydLa: A high-level language for hybrid systems. In *Proc. Workshop on Logics for System Analysis (LfSA)*, pages 3–17, 2012.

[1] In theory, constraints in `always` sentences could be multi-way or arithmetic constraints although such constraints would require a more powerful constraint solver.

[2] The ODE constraint `$x' == -1` implies the differentiability of `$x` for the associated condition; conversely, `$x` is nondifferentiable when `$x` is 0.

[3] The implementation of P5CP is not yet sufficient for executing this program.

[4] Semantically, the `pre $y'` indicates the limiting value of `$y'` when `$y` is infinitesimally greater than 0. However, since our solver uses an approximate finite-step algorithm, it is not able to compute such a limiting value.