Towards Practical Infinite Stream Constraint Programming: Applications and Implementation

Jasper C.H. Lee$^1$ and Jimmy H.M. Lee$^2$

$^1$ Churchill College and Computer Laboratory, University of Cambridge, UK  
chjl2@cam.ac.uk  

$^2$ Department of Computer Science and Engineering, 
The Chinese University of Hong Kong, Shatin, N.T., Hong Kong  
jlee@cse.cuhk.edu.hk

Abstract. Siu et al. propose stream CSPs (St-CSPs) as a generalisation of finite domain CSPs to cater for constraints on infinite streams, and a solving algorithm that produces a deterministic Büchi automaton recognising the solution language. As a novel application, we demonstrate how St-CSPs can model mathematically and generate a PID controller for driving a self-balancing tray and an inverted pendulum in real-time. We propose and give formally the correctness of an improvement to the implementation that eliminates numerous unnecessary states in the solution automaton for St-CSPs involving the first temporal operator, thereby reducing solving time. We give two St-CSP examples that can benefit from our new implementation techniques. Our approach always generates a solution automaton not bigger than, but potentially exponentially smaller than, that produced by the original implementation. Experimental results show substantial improvements.

1 Introduction

Streams of data are ubiquitous. They can either be discrete sequences on their own (e.g. stock market data), or discrete samples of continuous signals (e.g. positional data with respect to time). The evolution of such sequences is typically governed by some physical laws or mathematical equations. However, standard finite domain constraint satisfaction problems (CSPs) do not model such problems very well, because they can only model a finite segment of an otherwise infinite problem. To model such discrete time constraint problems more naturally, Siu et al. [8,11] introduce stream constraint satisfaction problems (St-CSPs) by adapting temporal operators in Wadge and Ashcroft’s [12] Lucid programming language. They give the definition of St-CSPs, and a solving algorithm that produces a deterministic Büchi automaton recognising the solutions of an St-CSP. The termination, soundness and completeness of their algorithm are proven. They also suggest practical applications for St-CSPs, such as generating harmonic accompaniment to a melody, and the game engine for the once popular Digi Invaders game in early Casio calculators in the 1970s.

$^1$ http://www.youtube.com/watch?v=1YafgAcmov4
This paper is about practical stream constraint programming. The goal is to push the limit of this relatively new member of the CP family, and take the first step towards putting the theoretical framework into practice. Since there are currently no common modelling idioms, and we know little about implementation technology and applications, we approach this idea from two angles. First, we demonstrate that St-CSPs can be used for solving interesting real-life problems. Continuing the work on game engine generation [8,11], we model real-time hardware controllers as St-CSPs. Even though discretisation and approximations have to be applied, we find that the approach produces stable control on our hardware. Second, we propose an improvement for the search algorithm to reduce solving time and the size of the solution automaton. Our improvement is restricted to a certain class of St-CSPs, and we give practical usages of this class on two applications. We also state formally the correctness of our technique. To demonstrate the efficiency of our proposal, we give experimental results to compare our new search algorithm against the original, showing orders of magnitude improvement in terms of runtime and solution automaton size.

2 Background

This section introduces the background for stream constraint solving. We first state the definition of St-CSPs and related notions, followed by the constraint specification language. The solving algorithm of Siu et al. [8,11] is summarised.

2.1 Infinite Strings and Stream Constraint Satisfaction Problems

An infinite string $\alpha$ over an alphabet $\Sigma$ is a function $\mathbb{N}_0 \to \Sigma$. Given $i$, $\alpha(i)$ is an individual daton of $\alpha$ at time point $i$. The set of all such strings with alphabet $\Sigma$ is denoted $\Sigma^\omega$. Infinite strings are also referred to as streams.

The notation $\alpha' = \alpha(i, \infty)$ is used for the string suffix $\alpha'(j) = \alpha(j + i)$. For a language $L$, $L(i, \infty) = \{\alpha(i, \infty) \mid \alpha \in L\}$. As for a finite prefix of a string, infinite or not, the notation $\alpha' = \alpha[0 : i]$ is used to denote the string $\alpha'(j) = \alpha(j)$ if $0 \leq j \leq i$ and undefined otherwise. When $i < 0$, $\alpha[0 : i]$ is the empty string.

A stream constraint satisfaction problem (St-CSP) is a tuple $P = (X, D, C)$ [8,11], where $X = \{x_1, \ldots, x_n\}$ is a finite set of variables, $D(x) = (\Sigma(x))^\omega$ is a function that maps a variable to its domain which is the set of all infinite strings with alphabet $\Sigma(x)$, $C$ is a finite set of constraints. A constraint $c \in C$ is a relation $R$ defined on an ordered subset $\text{Scope}(c)$ of variables. The relation gives all the valid simultaneous assignments of values to variables in $\text{Scope}(c)$. Every constraint $c \in C$ must also be a deterministic $\omega$-regular language [2].

An assignment $A(x_i) \in D(x_i)$ is a function mapping a variable to an element in its domain. A satisfies a constraint $c$ if and only if $(A(x_{i_1}), A(x_{i_2}), \ldots, A(x_{i_k})) \in c$, where $\text{Scope}(c) = \{x_{i_1}, \ldots, x_{i_k}\}$. The notion can be generalised to say that the string $\beta$ of tuples $\beta(i) = (A(x_1)(i), \ldots, A(x_n)(i))$ satisfies the constraint $c$ where $X = \{x_1, \ldots, x_n\}$. An St-CSP is satisfied by a variable assignment $A$ or a string of tuples $\beta$ if and only if all constraints are satisfied.