Constraint Programming on Infinite Data Streams

A. Lallouet
Université de Caen, GREYC
Campus Côte de Nacre
Boulevard du Maréchal Juin
B.P. 5186, 14032 Caen Cedex, France
arnaud.lallouet@info.unicaen.fr

Y.C. Law, J.H.M. Lee, and **C.F.K. Siu**
Department of Computer Science and Engineering
The Chinese University of Hong Kong
Shatin, N.T., Hong Kong
{yclaw,jlee,fksiu}@cse.cuhk.edu.hk

Abstract

Classical constraint satisfaction problems (CSPs) are commonly defined on finite domains. In real life, constrained variables can evolve over time. A variable can actually take an infinite sequence of values over discrete time points. In this paper, we propose constraint programming on infinite data streams, which provides a natural way to model constrained time-varying problems. In our framework, variable domains are specified by ω-regular languages. We introduce special stream operators as basis to form stream expressions and constraints. Stream CSPs have infinite search space. We propose a search procedure that can recognize and avoid infinite search over duplicate search space. The solution set of a stream CSP can be represented by a Büchi automaton allowing stream values to be non-periodic. Consistency notions are defined to reduce the search space early. We illustrate the feasibility of the framework by examples and experiments.

1 Introduction

The standard domains of classical Constraint Satisfaction Problems (CSPs) [Dechter, 2003] are of simple types, such as integers, reals, and sets, which are inadequate in describing problems with values that change over time. Discrete simulation of a person juggling indefinitely is an example of constrained time-varying problems with discrete time points. Changing continuously, the loci of the balls are governed by the juggler's throws, juggling rules, and laws of physics. In addition, an experienced juggler should be able to exhibit non-repetitive patterns so long as all the rules and laws are obeyed. Modeling such a problem as a CSP would require an infinite number of variables and constraints.

We propose Constraint Programming on infinite data streams, which are difficult to manipulate due to the lack of finite representation, especially for non-periodic ones. The domains of stream variables are represented compactly using ω-regular languages which are recognizable by Büchi automata [Büchi, 1962]. Different from model checking, the automata are modified during search and synthesized into different stream values. We define stream operators (*a la* Lucid [Wadge and Ashcroft, 1985]) and constraints. The searching approach used in classical CSP is no longer practical due to infinite domain size. We propose a search scheme which limits our attention to a window of time points. Consistency enforcement is integrated to the search procedure to eliminate infeasible search space. We have implemented a prototype solver, which is used to model and solve the simulation of juggling and jazzy harmonization of music as proof of concept.

2 Infinite Data Streams

Streams are infinite sequences of data items, called datons, over natural number time points. A *stream* α is an ordered sequence $\langle\alpha(0), \alpha(1), \alpha(2), \ldots\rangle$, where $\alpha(i)$ is a *daton* of α at time point $i \geq 0$. We use $\alpha(i, j)$, $0 \leq i \leq j$, to denote the finite string $\langle\alpha(i), \alpha(i+1), \ldots, \alpha(j)\rangle$. In particular, $\alpha(i, \infty)$ is the stream $\langle\alpha(i), \alpha(i+1), \ldots\rangle$. We overload these notations to apply on a set of stream values similarly. Given a set of streams S, $S(i) = \{\alpha(i) \mid \alpha \in S\}$ and $S(i, j) = \{\alpha(i, j) \mid \alpha \in S\}$.

Without loss of generality, we assume that datons are of the same type. In particular, we focus on integer ($\mathbb{Z}$) streams in this paper. For example $\alpha = \langle 1, 2, 3, 2, 4, 5, \ldots\rangle$ is an integer stream, in which $\alpha(2) = 3$, $\alpha(1, 3) = \langle 2, 3, 2\rangle$, and $\alpha(3, \infty) = \langle 2, 4, 5, \ldots\rangle$.

An *ω-regular language* generalizes a regular language to a set of infinite strings (*a la* streams), and can be expressed as a finite union $\cup_{i=0}^{n} U_i V_i^{\omega}$ where U_i and V_i are regular languages and the empty string $\epsilon \notin V_i$. The ω operator in V_i^{ω} indicates the infinite concatenation of the regular language V_i. In this paper, we are interested only in problems whose solution sets are ω-regular languages.

A *Büchi automaton* over an alphabet Σ is a 4-tuple $\mathcal{A} = (Q, q_0, \triangle, F)$ where Q is a finite set of states, $q_0 \in Q$ is an initial state, $\triangle \subseteq Q \times \Sigma \times Q$ is a transition relation, and $F \subseteq Q$ is the set of final states. The automaton $\mathcal{A}$ accepts an infinite string if and only if there exists a run of the automaton which visits at least one of the final states infinitely often. An ω-regular language is recognizable by a Büchi automaton. We use $L(\mathcal{A})$ to denote the ω-regular language recognized by $\mathcal{A}$.

Temporal operators are defined over streams. The unary `first` operator gives the stream formed by *repeating* the first daton of the stream. Formally, `first` $\alpha = \beta$, where

$\forall i \geq 0, \beta(i) = \alpha(0)$. The unary `next` operator gives the stream formed by *removing* the first daton of the stream. Formally, $\texttt{next}\ \alpha = \alpha(1, \infty)$. The binary `fby` operator takes two streams and gives the resulting stream by concatenating the first daton in the first stream and the second stream. Formally, $\alpha\ \texttt{fby}\ \beta = \gamma$ where $\gamma(0) = \alpha(0)$ and $\forall i \geq 1, \gamma(i) = \beta(i-1)$.

In addition to temporal operators, *pointwise operators* are extensions to functions defined over integers. Given an n-ary function $f : \mathbb{Z}^n \mapsto \mathbb{Z}$, an extension of f to a corresponding pointwise operator f is defined by $\mathsf{f}(\alpha_1, \alpha_2, \ldots, \alpha_n) = \beta$ where $\forall i \geq 0, \beta(i) = f(\alpha_1(i), \alpha_2(i), \ldots, \alpha_n(i))$. In particular, we highlight some useful pointwise operators, which will be used in infix notation as usual. Arithmetic operators, including $+$, $-$, $\times$, and $/$ (integer division) on numbers are the extension of the usual operators over integers. When the streams are pseudo-Boolean streams, containing only datons 0 (false) and 1 (true), there are logical operators `and`, `or`, and `not`. Relational operators determine the truth of relation on the stream values pointwisely and return a pseudo-Boolean stream. The operators include $==$, $<>$, $<=$, and $>=$ on numbers.

A stream expression can involve different operators as in "$\alpha + \beta - (\gamma\ \texttt{fby}\ (\texttt{next}\ \alpha))$".

3 Stream Constraint Satisfaction

A *Constraint Satisfaction Problem* (CSP) $\mathcal{P}$ is a tuple $(\mathcal{X}, \mathcal{D}, \mathcal{C})$ where $\mathcal{X}$ is a finite set of *variables*, $\mathcal{D}$ is a finite set of *(variable) domains*, and $\mathcal{C}$ is a finite set of *constraints*. A variable $X \in \mathcal{X}$ can only take a value from its corresponding variable domain $D(X) \in \mathcal{D}$. Each constraint $C \in \mathcal{C}$ has $scope(C)$ corresponding to the list of variables involved in C. A constraint C limits the combinations of values that can be taken by the variables in $scope(C)$. A *solution* to a CSP is an assignment of values from the variable domains to all variables such that all constraints in $\mathcal{P}$ are satisfied simultaneously. We denote the set of solutions to CSP $\mathcal{P}$ as $sol(\mathcal{P})$.

3.1 Stream Constraints

Stream constraints are relations on stream expressions, which are composed of stream variables, stream constants, and stream operators. Stream constants have the same daton over all the time points which are denoted with the daton in bold, such as $\mathbf{2} = \langle 2, 2, 2, \ldots \rangle$. The relations can be $=$, $\neq$, $\geq$, $\leq$, and $\rightarrow$ (implication), which are all enforced *pointwisely*. An example constraint is "$X + \mathbf{3} = Y\ \texttt{fby}\ Z$". When the constraints involve $\geq$ or $\leq$, the set of datons, such as $\mathbb{Z}$ in this paper, is assumed to have some ordering.

A stream constraint $C \in \mathcal{C}$ with $scope(C) = (X_1, X_2, \ldots, X_n)$ is a subset of $(\mathbb{Z}^\omega)^n$, i.e. $C \subseteq (\mathbb{Z}^\omega)^n$. Relational operators are different from stream relations. The former are functions returning pseudo-Boolean streams, while the latter are constraints to be enforced.

In a *Stream CSP* (*St-CSP*), variables take on stream values so that domains are possibly infinite sets of streams. Expressions and constraints involving streams are those defined earlier. We require stream variable domains to be ω-regular languages. For simplicity, initial domains are specified in the form of Σ^ω where Σ is the set of possible datons at each time point.

The following shows an example St-CSP having variables X, Y, and Z, with domains $D(X) = D(Y) = D(Z) = (0|1|2)^\omega$, where "$|$" denotes choice. The two constraints are:

$$X = \texttt{next}\ Y + \mathbf{1} \quad \text{and} \quad Y = X\ \texttt{fby}\ Z$$

This problem has infinitely many solutions. Three such solutions are: (a) $\{X = (1)^\omega, Y = 1(0)^\omega, Z = (0)^\omega\}$, (b) $\{X = 121(2)^\omega, Y = 1010(1)^\omega, Z = 010(1)^\omega\}$, and (c) $\{X = 211(212)^\omega, Y = 2100(101)^\omega, Z = 100(101)^\omega\}$. For example, solution (b) satisfies all constraints since "$\texttt{next}\ Y$" gives $010(1)^\omega$ and "$010(1)^\omega + \mathbf{1}$" is $121(2)^\omega$ which is equal to X's value. Furthermore, "$X\ \texttt{fby}\ Z$" takes the first daton of X, i.e., 1, followed by the stream $Z = 010(1)^\omega$, which gives $1010(1)^\omega$ and is equal to Y's value.

An St-CSP can be viewed as a classical CSP with an infinite number of variables and constraints. A stream variable X corresponds to an infinite sequence of *daton variables* $\langle X(0), X(1), \ldots \rangle$ in which $D(X(i)) = D(X)(i)$. Similarly, a stream constraint C corresponds to an infinite sequence of *daton constraints* $C(0), C(1), \ldots$. Each daton constraint $C(i)$ of C can be obtained by applying translation operation T_i with the rules listed in Table 1 such that $T_i(C)$ gives $C(i)$. Thus, each stream constraint C is translated to the set $\{T_i(C) \,|\, i \geq 0\}$. For example, the stream constraint "$X = Y\ \texttt{fby}\ Z$" at time point 0 is translated by Rule 3 from "$T_0(X = Y \texttt{fby}\ Z)$" to "$T_0(X) = T_0(Y \texttt{fby}\ Z)$", and then by Rules 6 and 1 to "$X(0) = Y(0)$". For time point $i > 0$, "$T_i(X) = T_i(Y \texttt{fby}\ Z)$" is translated by Rule 7 to "$T_i(X) = T_{i-1}(Z)$", and then by Rule 1 to "$X(i) = Z(i-1)$". Therefore, the stream constraint is equivalent to the conjunction of daton constraints: $X(0) = Y(0), X(1) = Z(0), X(2) = Z(1), \ldots$. An St-CSP $\mathcal{P} = (\mathcal{X}, \mathcal{D}, \mathcal{C})$ can be viewed as a classical CSP $\mathcal{P}' = (\mathcal{X}', \mathcal{D}', \mathcal{C}')$ where $X' = \{X(i) \,|\, i \geq 0 \wedge X \in \mathcal{X}\}$, $\forall X(i) \in \mathcal{X}', D'(X(i)) = D(X)(i)$, and $\mathcal{C}' = \{T_i(C) \,|\, i \geq 0 \wedge C \in \mathcal{C}\}$.

3.2 Characterizing the Solution Space

We consider a variable assignment as a tuple. The solution set of an St-CSP contains possibly infinite number of tuples of streams. We introduce the $\otimes$ operator on streams such that $\alpha_1 \otimes \alpha_2 \otimes \ldots \otimes \alpha_n = \langle (\alpha_1(0), \alpha_2(0), \ldots, \alpha_n(0)), (\alpha_1(1), \alpha_2(1), \ldots, \alpha_n(1)), \ldots \rangle$. The operator turns a sequence of streams into a stream of tuples of corresponding datons. Then, given a set of tuples of streams S, we define $\mathcal{L}(S) = \{\alpha_1 \otimes \alpha_2 \otimes \ldots \otimes \alpha_n \,|\, (\alpha_1, \alpha_2, \ldots, \alpha_n) \in S\}$.

Lemma 1. *$\mathcal{L}(sol(\mathcal{P}))$ is isomorphic to $sol(\mathcal{P})$.*

Lemma 2. *Given a stream constraint C, $\mathcal{L}(C)$ is an ω-regular language.*

The solution set of an St-CSP is, mathematically, the conjunction of constraints and the Cartesian product of variable domains. Since stream domains and stream constraints are ω-regular languages, by the closure of operations for ω-regular languages [Thomas, 1990], we have the following theorem.

Theorem 1. *Given an St-CSP $\mathcal{P}$, $\mathcal{L}(sol(\mathcal{P}))$ is an ω-regular language.*

Table 1: Translation rules for stream constraints and stream expressions at time point i

Rule	Expression	Translation
1	$T_i(\alpha)$	$\alpha(i)$
2	$T_i(X)$	$X(i)$
3	$T_i(expr_1\ rel\ expr_2)$	$T_i(expr_1)\ rel\ T_i(expr_2)$ where rel is a stream relation
4	$T_i(\texttt{first}\ expr)$	$T_0(expr)$
5	$T_i(\texttt{next}\ expr)$	$T_{i+1}(expr)$
6	$T_0(expr_1\ \texttt{fby}\ expr_2)$	$T_0(expr_1)$
7	$T_i(expr_1\ \texttt{fby}\ expr_2)$	$T_{i-1}(expr_2)$ where $i > 0$
8	$T_i(\mathsf{f}(expr_1, expr_2, \ldots, expr_n))$	$f(T_i(expr_1), T_i(expr_2), \ldots, T_i(expr_n))$ for n-ary function f

Proof. We prove it by induction. When there is one stream variable in the problem $\mathcal{P}$, the set of solutions $\mathcal{L}(sol(\mathcal{P}))$ is the conjunction of the initial domain and the set of constraints. Since domains and constraints are ω-regular languages, by the closure of ω-regular languages over conjunction, the resulting set is also an ω-regular language. Given two ω-regular languages L_1 and L_2, we let $S = \{(\alpha_1, \alpha_2) \mid \alpha_1 \in L_1, \alpha_2 \in L_2\}$. Since S is an ω-regular language, the induction applies. □

Thus we can solve an St-CSP by constructing a Büchi automaton for $\mathcal{L}(sol(\mathcal{P}))$. In addition, by the nature of ω-regular languages, solution streams of an St-CSP can be *non-periodic*.

4 Solving Stream CSPs

An St-CSP has infinite domains. The tree search method [Dechter, 2003] widely applied in solving traditional finite domain CSPs is not applicable in this case as stream variable domains can never be enumerated exhaustively. We propose a depth-first search approach which determines the datons in the stream variables in the order of time points. We define a dominance relation among the search states or nodes so that when a search state is dominated by an ancestor node in the search tree, the search down that branch can terminate.

4.1 Search Tree

A *search state* is an St-CSP $\mathcal{P}$. Given $\mathcal{P}_1 = (\mathcal{X}_1, \mathcal{D}_1, \mathcal{C}_1)$ and $\mathcal{P}_2 = (\mathcal{X}_2, \mathcal{D}_2, \mathcal{C}_2)$, $\mathcal{P}_1$ is a *sub-problem* of $\mathcal{P}_2$, denoted $\mathcal{P}_1 \subseteq \mathcal{P}_2$, when $\mathcal{X}_1 = \mathcal{X}_2$, $\mathcal{D}_1 \subseteq \mathcal{D}_2$, and $\mathcal{C}_1 = \mathcal{C}_2$. In a search tree, a parent search state $\mathcal{P}$ has a finite set of child states $\mathcal{P}'_i$ such that $\forall i, \mathcal{P}'_i \subseteq \mathcal{P} \wedge \cup_i sol(\mathcal{P}'_i) = sol(\mathcal{P})$.

Figure 1 shows the first 7 nodes of the search tree for an St-CSP $\mathcal{P}$, having variables X and Y with the initial domains $D(X) = D(Y) = (1\,|\,2)^\omega$ and a constraint $X = \texttt{first}\ Y$.

The search procedure attempts to determine the daton assignment in the order of increasing time points. We define the *current time point* of a variable X as $t(X)$ which is the maximum time point before which all the daton variables of X can be fixed according to $D(X)$. Formally, $t(X) = \min\{i \mid i \geq 0$ s.t. $|D(X)(i)| > 1\}$. The *current time point* of an St-CSP $\mathcal{P}$, $t(\mathcal{P})$ is the minimum current time point among all the variables in $\mathcal{P}$, i.e. $t(\mathcal{P}) = \min\{t(X) \mid X \in \mathcal{X}\}$. Thus, there exists at least one variable whose daton variable at time point $t(\mathcal{P})$ is unbound in a given $\mathcal{P}$. When $t(\mathcal{P}) = \infty$, all the daton variables are bound. For example, in $\mathcal{P}_2$ of the search tree in Figure 1, the datons in both domains $D_2(X)$ and $D_2(Y)$ are fixed up to time point 0; therefore $t(\mathcal{P}_2)$ is 1.

$\mathcal{P} = (\{X, Y\}, \{D(X) = D(Y) = (1\,|\,2)^\omega\}, \{C : X = \texttt{first}\ Y\})$

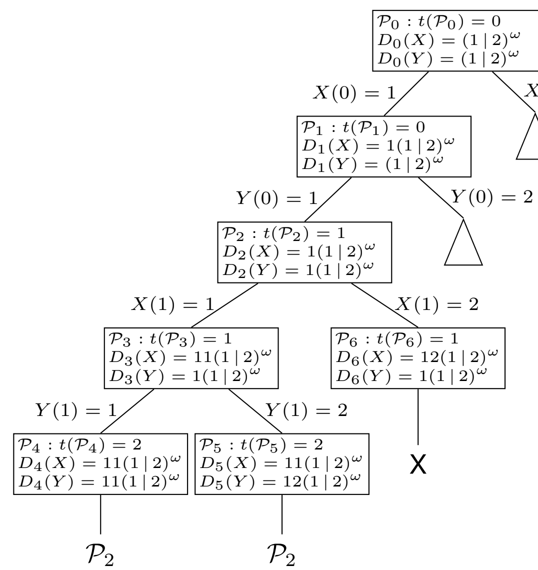

Figure 1: A search tree for an St-CSP.

In each search state $\mathcal{P}$, a variable X with $t(X) = t(\mathcal{P})$ is selected. With each $d \in D(X)(t(\mathcal{P}))$, $\mathcal{P}$ is branched with the assignment $X(t(\mathcal{P})) = d$. In Figure 1, upon completion of assignment to datons in time point 0 at $\mathcal{P}_2$, the search selects a variable with current time point as $t(\mathcal{P}_2)$ for daton assignment. The search tree here first selects $X(1)$ and branches for $X(1) = 1$ and $X(1) = 2$ respectively.

Note that $t(\mathcal{P}_0) = t(\mathcal{P}_1) = 0$, but $t(\mathcal{P}_2) = t(\mathcal{P}_1) + 1$. We say that there is an *advancement* of current time point from $\mathcal{P}_1$ to $\mathcal{P}_2$ but not from $\mathcal{P}_0$ to $\mathcal{P}_1$. We define the set of search states with advancement of current time point from their parent search states plus $\mathcal{P}_0$ to be $\Phi = \{\mathcal{P}_i \mid t(parent(\mathcal{P}_i)) < t(\mathcal{P}_i)\} \cup \{\mathcal{P}_0\}$, where $parent(\mathcal{P})$ gives the parent search state of $\mathcal{P}$ and $\mathcal{P}_0$ is the root node of the search tree. Each search state in Φ corresponds to a complete assignment to all daton variables at and before a time point. In the search tree in Figure 1, Φ includes $\mathcal{P}_0$, $\mathcal{P}_2$, $\mathcal{P}_4$, and $\mathcal{P}_5$ among the first 7 nodes.

Since the streams are defined on infinite time points, the search procedure will advance the time point forever. To avoid infinite search, we define the notion of dominance of one search state over another. A search state $\mathcal{P}_i = (\mathcal{X}, \mathcal{D}_i, \mathcal{C})$ is *dominated* by $\mathcal{P}_j = (\mathcal{X}, \mathcal{D}_j, \mathcal{C})$, denoted as $\mathcal{P}_i \prec \mathcal{P}_j$, if and only if $\mathcal{P}_i, \mathcal{P}_j \in \Phi$, $\mathcal{P}_j$ is an ancestor of

$\mathcal{P}_i$, and $t_i = t(\mathcal{P}_i), t_j = t(\mathcal{P}_j), \forall X \in \mathcal{X}, D_i(X)(t_i, \infty) = D_j(X)(t_j, \infty) \wedge \forall C \in \mathcal{C}, \prod_{X \in scope(C)} D_i(X)(t_i, \infty) \cap C = \prod_{X \in scope(C)} D_j(X)(t_j, \infty) \cap C$. The conditions for dominance ensure the solution space of both $\mathcal{P}_i$ and $\mathcal{P}_j$, when only considering the time points after $t(\mathcal{P}_i)$ and $t(\mathcal{P}_j)$ respectively, is the same, since the domains are the same and each constraint represents the same set of tuples of streams.

The search states $\mathcal{P}_4$ and $\mathcal{P}_5$ in Figure 1 are dominated by $\mathcal{P}_2$. Since $D_4(X)(2, \infty) = D_2(X)(1, \infty)$, $D_4(Y)(2, \infty) = D_2(Y)(1, \infty)$ and $((D_4(X) \times D_4(Y)) \cap C)(2, \infty) = ((D_2(X) \times D_2(Y)) \cap C)(1, \infty) = ((1, 1) \,|\, (1, 2))^\omega$. This is similar for $\mathcal{P}_5$.

Suppose $\mathcal{P}_i$ is dominated by $\mathcal{P}_j$, where $\mathcal{P}_i, \mathcal{P}_j \in \Phi$. There is a path from $\mathcal{P}_j$ to $\mathcal{P}_i$ in the search tree. The path corresponds to a sequence of daton assignments, denoted as s, between time points $t(\mathcal{P}_j)$ and $t(\mathcal{P}_i) - 1$. Therefore, for all $\alpha \in \mathcal{L}(sol(\mathcal{P}_i)), s\alpha \in \mathcal{L}(sol(\mathcal{P}_j))$, where $s\alpha$ is s appended with α. As $\mathcal{P}_i$ and $\mathcal{P}_j$ share the same solution space after a certain time point, such operation can be done infinitely many times and s^ω is one of the solutions to $\mathcal{P}_i$ and $\mathcal{P}_j$. For example, in Figure 1, the path from $\mathcal{P}_2$ to $\mathcal{P}_4$ corresponds to the assignment $\{X(1) = 1, Y(1) = 1\}$. Since $\mathcal{P}_4$ is dominated by $\mathcal{P}_2$, the solution space of $\mathcal{P}_2$ is the same as that of $\mathcal{P}_4$ after 1 time point. Therefore, the assignment $\{X(i) = 1, Y(i) = 1\}$ can always be satisfied and $\{X = (1)^\omega, Y = (1)^\omega\}$ is one of the solutions.

As the domains can be infinite, the computation of conjunction of constraints and domains is infeasible. We propose simple and sufficient conditions for dominance detection. Given an St-CSP $\mathcal{P} = (\mathcal{X}, \mathcal{D}, \mathcal{C})$. As all the datons are fixed before time point $t(\mathcal{P})$, we limit our attention to time point $t(\mathcal{P})$ and onwards. We define a *limited view* of $\mathcal{P}$ to be $\hat{\mathcal{P}} = (\mathcal{X}, \hat{\mathcal{D}}, \hat{\mathcal{C}})$, which can be obtained from $\mathcal{P}$ by removing the time points from 0 to $t(\mathcal{P}) - 1$ such that $\forall X \in \mathcal{X}, \hat{D}(X) = D(X)(t(\mathcal{P}), \infty)$ and $\hat{\mathcal{C}} = \{\hat{C}(i) \,|\, \forall i \geq 0, C \in \mathcal{C}\}$ where $\hat{C}(i)$ is $C(i)$ with all the occurrences of $X(i), i < t(\mathcal{P})$, replaced by the assigned values to $X(i)$'s.

Theorem 2. *Given $\mathcal{P}_i, \mathcal{P}_j \in \Phi$. If $\hat{\mathcal{P}}_i = \hat{\mathcal{P}}_j$ and $\mathcal{P}_j$ is an ancestor of $\mathcal{P}_i$, then $\mathcal{P}_i \prec \mathcal{P}_j$.*

Proof. Suppose $\mathcal{P}_i = (\mathcal{X}, \mathcal{D}_i, \mathcal{C})$, $\mathcal{P}_j = (\mathcal{X}, \mathcal{D}_j, \mathcal{C})$, $t_i = t(\mathcal{P}_i)$, and $t_j = t(\mathcal{P}_j)$. Since $\mathcal{P}_i, \mathcal{P}_j \in \Phi$ and, $\mathcal{P}_j$ is an ancestor of $\mathcal{P}_i$, we only have to show that when $\hat{\mathcal{P}}_i = \hat{\mathcal{P}}_j$, (1) $\forall X \in \mathcal{X}, D_i(X)(t_i, \infty) = D_j(X)(t_j, \infty)$ and (2) $\forall C \in \mathcal{C}$, $\prod_{X \in scope(C)} D_i(X)(t_i, \infty) \cap C = \prod_{X \in scope(C)} D_j(X)(t_j, \infty) \cap C$ are true.

When $\hat{\mathcal{P}}_i = \hat{\mathcal{P}}_j$, condition (1) is satisfied by the definition of limited view, as $\forall X \in \mathcal{X}, \hat{D}_i(X) = \hat{D}_j(X)$. Since $\forall C \in \mathcal{C}, \hat{C}_i = \hat{C}_j$ and by condition (1), condition (2) is also satisfied. $\square$

Next, we analyze the termination and complexity of this search approach. From Table 1, we observe that different translation rules are applied to the `fby` operator depending on the time point i. A stream constraint C, in which the maximum nested applications of `fby` is n, for all time points $i \geq n$, $T_i(C)$ is translated with the same set of rules. Therefore, we have the following property.

Part	(1)	(2)	(3)		
Time	0	1	2	3	...
$D(X) =$	1	1	$(1 \,\|\, 2)$	$(1 \,\|\, 2)$	...
$D(Y) =$	1	$(1 \,\|\, 2)$	$(1 \,\|\, 2)$	$(1 \,\|\, 2)$	...

Figure 2: The division of time line into three parts for search state $\mathcal{P}_3$ in Figure 1.

Property 1. *Given a stream constraint C with n nested applications of `fby`. $\forall i > n$, $T_i(C)$ share the same structure as $T_n(C)$.*

Two daton constraints $C(i)$ and $C(j)$ *share the same structure* when $C(i)$ becomes $C(j)$ after replacing i by j. Take the above stream constraint "$X = Y$ `fby` Z" as an example, since there is only one `fby` operator, for all time points $i \geq 1$, the daton constraint is $X(i) = Z(i-1)$.

When $\hat{\mathcal{P}}_i = \hat{\mathcal{P}}_j$, $\mathcal{P}_i$ and $\mathcal{P}_j$ share the same search space after $t(\mathcal{P}_i)$ and $t(\mathcal{P}_j)$ respectively. The order of variable assignment in the search strategy divides the time line into three parts: (1) *all* daton variables are fixed, (2) *some* daton variables are fixed, and (3) *no* daton variables can be fixed. For example, the search state P_3 in Figure 1, the window of part (1) is $[0, 0]$, that of part (2) is $[1, 1]$, and that of part (3) is $[2, \infty]$ which is depicted in Figure 2.

Theorem 3. *The time complexity for dominance detection on a pair of search states is $O(w(d|\mathcal{X}| + a|\mathcal{C}|))$, where w is the width of part (2), d is the maximum number of possible daton at any time point, and a is the maximum arity of stream constraints.*

Proof. (Sketch) The starting point of part (2) for an St-CSP $\mathcal{P}$ is $t(\mathcal{P})$. To check domain equivalence, we can consider only part (2) of the time line since there is no difference for part (3). This takes time $O(wd|\mathcal{X}|)$. We then check constraint equivalence. Every constraint can involve only a finite number of `fby` operators. By Property 1, after a finite number of time points, all the daton constraints share the same structure. As the constraint may involve a finite number of daton variables before time point $t(\mathcal{P}_j)$ or $t(\mathcal{P}_i)$, we have to check the equivalence of the values which are assigned to those daton variables. This checking takes $O(wa|\mathcal{C}|)$. $\square$

The sufficient condition (Theorem 2) depends on the number of datons, width of part (2), and the number of variables. As all are finite, there must be two search states in each branch matching the condition for dominance detection.

Lemma 3. *Each branch in a search tree is finite and must either (a) end in failure or (b) contain search states $\mathcal{P}_i$ and $\mathcal{P}_j$ such that $\mathcal{P}_i \prec \mathcal{P}_j$ and the branch terminates at $\mathcal{P}_i$.*

Proof. The search procedure branches for each possible daton for a selected variable at a time point. Since the daton domain is finite, there is a finite number of branches. The branch ends in failure once there is no consistent daton to be assigned; otherwise, the branch continues. At every advancement of time point, the search performs dominance detection.

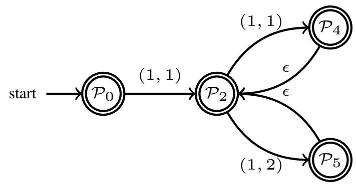

Figure 3: A Büchi automaton representing a subset of all solutions of the St-CSP in Figure 1.

As there are finite possible datons and finite number of stream constraints in the problems, there must be two search states along a branch of the search tree that satisfy the dominance relation. □

Theorem 4. *The search procedure terminates.*

Proof. The theorem follows directly from Lemma 3. □

Among the first seven search states shown in Figure 1, the search states $\mathcal{P}_4$ and $\mathcal{P}_5$ are dominated by $\mathcal{P}_2$. In search state $\mathcal{P}_6$, the assignment $X(1) = 2$ cannot satisfy the constraint $X = \texttt{first}\ Y$ and the search fails.

4.2 Construction of Solution Set

When solving solutions of St-CSP $\mathcal{P}$ through the search procedure, we are actually building the corresponding Büchi automaton $\mathcal{A}$, which can recognize and thus also generate the solution set. We want $L(\mathcal{A}) = \mathcal{L}(sol(\mathcal{P}))$.

The automaton $\mathcal{A} = (Q, q_0, \triangle, F)$ is built according to the search tree. For each search state $\mathcal{P}_i \in \Phi$, $\mathcal{P}_i$ is associated to a state $state(\mathcal{P}_i)$ in $\mathcal{A}$, thus $Q = \{state(\mathcal{P}_i) \mid \mathcal{P}_i \in \Phi\}$. The root node of the search tree, $\mathcal{P}_0$, is associated with the starting state of $\mathcal{A}$ where $q_0 = state(\mathcal{P}_0)$. For every non-root search state $\mathcal{P}_i \in \Phi \setminus \{\mathcal{P}_0\}$, there is an edge pointing from $state(\mathcal{P}_j)$ to $state(\mathcal{P}_i)$ where $\mathcal{P}_j$ is the nearest ancestor of $\mathcal{P}_i$ in Φ. The edge is labelled with the assignment tuple made from the search state $\mathcal{P}_j$ to $\mathcal{P}_i$. For each leaf node $\mathcal{P}_i$ associated with state $state(\mathcal{P}_i)$, if $\mathcal{P}_i \prec \mathcal{P}_j$, there is an edge pointing from $state(\mathcal{P}_i)$ to $state(\mathcal{P}_j)$ labelled with an empty string ϵ. Since the automaton is generated from the search tree, all the possible runs correspond to solutions. The set of final states contains all the states in the automaton, thus $F = Q$. The final automaton can be simplified. When a path in search tree leads to failure, there are some states in $\mathcal{A}$ cannot be included in any accepting runs. These states can be removed. When $\mathcal{P}_i \prec \mathcal{P}_j$, $state(\mathcal{P}_i)$ can be merged with $state(\mathcal{P}_j)$ such that the edge labelled with ϵ can be eliminated.

Figure 3 shows the subset of solutions corresponding to the first seven search states in Figure 1. The associated search states are labelled on the states in the automaton. From the automaton, the subset of solutions is $(1,1)((1,1) \mid (1,2))^\omega$.

The solution automaton $\mathcal{A}$ corresponds to the structure of search tree, where every search state $P_i \in \Phi$ is a state and every complete daton assignment is an edge.

Theorem 5. *The solution automaton takes $O(wa|\mathcal{C}| + d^{|\mathcal{X}|})$ space, where w is the width of part (2), a is the maximum arity of constraint, and d is the maximum number of possible datons at any time point.*

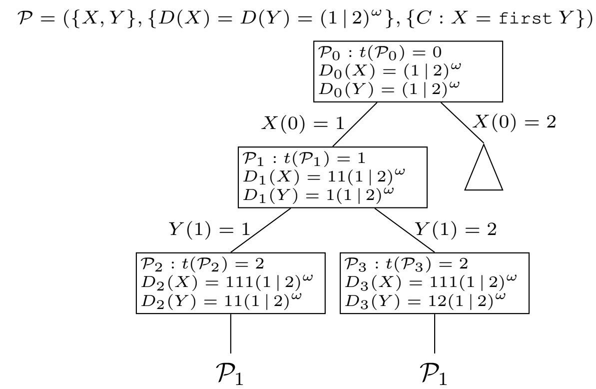

Figure 4: A search tree for an St-CSP enforced with prefix-1 consistency.

Proof. Each advancement of current time point in the search corresponds to a state in the automata. The number of nodes $\mathcal{P}_i$, where $\mathcal{P}_i \in \Phi$ along the search path, is $O(wa|\mathcal{C}|)$, which is the number of different patterns in part (2). Each state contributes at most $d^{|\mathcal{X}|}$ edges for every possible daton assignment. □

The following theorem shows that the constructed automaton recognizes *all* solutions and *only* solutions of $\mathcal{P}$.

Theorem 6. *(Soundness and Completeness) Given a Büchi automaton $\mathcal{A}$ constructed from the search procedure for an St-CSP $\mathcal{P}$, $(\alpha_1, \alpha_2, \ldots, \alpha_n) \in sol(\mathcal{P}) \leftrightarrow \alpha_1 \otimes \alpha_2 \otimes \ldots \otimes \alpha_n \in L(\mathcal{A})$.*

5 Consistency Algorithm

Enforcing consistency helps reduce search space, by identifying and avoiding infeasible search branches. In St-CSP, due to the infinite domain size, it is expensive to enforce generalized arc consistency (GAC) [Bessière and Régin, 1997]. According to the search strategy introduced in the previous section, we define a weaker notion of consistency, namely *prefix-k* consistency, which enforces GAC on the daton variables in a size k window of time points.

In the search tree, the current time point $t(\mathcal{P})$ of a search state $\mathcal{P}$ contains the first unbound daton variable. We limit our attention to the width k window of time points starting from $t(\mathcal{P})$, which is $R = [t(\mathcal{P}), t(\mathcal{P})+k-1]$. Among the daton variables in this window of time points, we enforce GAC.

By the definition of GAC [Bessière and Régin, 1997], a daton variable $X_v(i)$ in an St-CSP $\mathcal{P}$ is *GAC* with respect to daton constraint $C(j)$ if and only if $D(X_v(i)) = (\prod_{X_u(m) \in scope(C(j))} D(X_u(m)) \cap C(j)) \downarrow_{X_v(i)}$ where $\downarrow_{X_v(i)}$ projects the tuples to $X_v(i)$.

A stream variable X is *prefix-k consistent* with respect to a stream constraint C if and only if $\forall i \in R$, $X(i)$ is GAC with respect to all the daton constraint $C(j) \in C$ such that $X(i) \in scope(C(j))$. An St-CSP $\mathcal{P}$ is *prefix-k consistent* if and only if all the stream variables in $\mathcal{P}$ are prefix-k consistent with respect to all $C \in \mathcal{C}$.

For example, in Figure 1, the search state $\mathcal{P}_1$ is not prefix-1 consistent with respect to the constraint because there are no datons $d \in D(X(0))$ such that $X(0) = Y(0)$ when $Y(0) = 2$. Figure 4 shows the search tree of the problem with prefix-1 consistency enforced. After the assignment $X(0) = 1$ from search state $\mathcal{P}_0$, prefix-1 consistency is enforced at time point 0 and removes 2 from $D(Y(0))$. As both $X(0)$ and $Y(0)$ are bound, the search advances the current time point and enforces prefix-1 consistency at time point 1, which gives the search state $\mathcal{P}_1$. We can see that the search tree becomes smaller and some nodes leading to failure, such as $\mathcal{P}_6$ in Figure 1, are pruned earlier.

The notion of prefix-k consistency is enforced on the daton variables and daton constraints. The enforcement algorithm in Algorithm 1 is based on the classical GAC enforcement, but we are only interested in the daton variables $X(i)$ whose time point i falls in R. In the procedure `PrefixK`, only daton variables with time points in the width R will be considered. The `Revise` procedure checks if each of the values in the daton variable domain can be extended to a tuple which is consistent to the daton constraint. When there are changes made to the domain, all the constraints with variables inside the range of time points will be enqueued.

1 **Procedure** `Revise`($\mathcal{P}, x_i, c$)
2 //x_i and c are daton variable and daton constraint respectively.
3 $change := false$;
4 **for** $d_j \in D(x_i)$ **do**
5     $support := false$;
6   **for** $(d_0, d_1, \ldots, d_j, \ldots, d_n) \in D(x_0) \times D(x_1) \times \ldots \times \{d_j\} \times \ldots \times D(x_n)$ **do**
7   **if** $(d_0, d_1, \ldots, d_j, \ldots, d_n) \in c$ **then**
8             $support := true$;
9   **if** $support = false$ **then**
10         $D(x_i) := D(x_i) \setminus \{d_j\}$;
11         $change := true$;
12 **return** $change$;
13 **Procedure** `PrefixK`($\mathcal{P}, k$)
14 $R := [t(\mathcal{P}), t(\mathcal{P}) + k - 1]$;
15 $Q := \{(X_m(i), C_n(j)) \mid X_m(i) \in scope(C_n(j)) \wedge i \in R\}$;
16 **while** $Q \neq \emptyset$ **do**
17   take and remove $(X_m(i), C_n(j))$ from Q;
18   **if** `Revise`($\mathcal{P}, X_m(i), C_n(j)$) **then**
19   **for** $C_{n'}(j') \in \mathcal{C}$ *s.t.* $X_m(i) \in scope(C_{n'}(j'))$ **do**
20   **for** $X_{m'}(i') \in Cn'(j')$ **do**
21   **if** $X_{m'}(i') \neq X_m(i) \wedge i' \in R$ **then**
22                     $Q := Q \cup (X_{m'}(i'), C_{n'}(j'))$;

Algorithm 1: Enforcing prefix-k consistency.

Theorem 7. *(Correctness) If St-CSP $\mathcal{P}'$ is obtained from $\mathcal{P}$ by applying Algorithm 1, then $\mathcal{P}'$ is equivalent to $\mathcal{P}$ and $\mathcal{P}'$ is prefix-k consistent.*

Proof. Given $i \in R$. Suppose $\exists d \in D(X(i))$ such that $\exists C(j), X(i) \in scope(C(j)), d \notin C(j) \downarrow_{X(i)}$ and d remains in $D(X(i))$ after executing Algorithm 1. In line 18 of the algorithm, $C(j)$ will be selected. In procedure `Revise`, the condition in line 7 will never be satisfied. Thus d is removed from the domain of $D(X(i))$ and this contradicts the assumption. ☐

Theorem 8. *The algorithm to enforce prefix-k consistency takes $O(ad^a k|\mathcal{C}|)$ time, where a is the maximum arity of daton constraints, and d is the maximum possible datons at any time point.*

Proof. The complexity of `Revise` is $O(d^a)$ to check for support for each of the possible daton in the daton domain. The procedure `PrefixK` enforces prefix-k consistency. There are $O(ak|\mathcal{C}|)$ tuples in the queue Q, each of them will be put into queue again for at most $O(d^a)$ time. ☐

6 Examples and Experiments

To verify the feasibility of our framework, we have modelled the periodic still life problem, traffic light scheduling, 15-puzzle, simulation of juggling, and jazzy harmony generation as St-CSPs. The periodic still life problem looks for initial patterns that lead to oscillating patterns after a finite number of steps. The traffic light scheduling problem arranges traffic light signals in a road junction such that the vehicles will never crash. Though optimal solutions to a valid 15-puzzle always involves finite number of moves, the problem looks for all possible solutions so that the number of moves is not known in advance. Due to space limitation, we describe only the juggling problem and harmony generation in details. These problems have non-UP-stream solutions. We implement a prototype St-CSP solver enforcing prefix-k consistency. Comparison among different k values is conducted. The solution automata are constructed automatically on-the-fly during search and translation time is included in our results. Experiments are conducted on a Sun Blade 2500 machine with 2GB memory.

6.1 Simulation of Juggling

The task is to simulate basic juggling [Apt and Brand, 2006] involving n balls. For simplicity, the patterns ensure that there is at most one ball in hand at any time, and every ball is thrown for maximum m time points after which the ball is caught. Each problem is characterized by (n, m). We aim to find all possible sequences of juggling patterns which may change over time.

The n variables $X_1, X_2, X_3, \ldots, X_n$ represent the time interval after which the ball is caught. For example, if X_1 has daton 5 at time point 3, ball 1 will be in the air for 5 time points and be caught at time point 7. The variable A indicates the force to throw the ball, which reflects the time interval for the ball in the air. A ball thrown with odd (even) units of force will be caught by different (same) hand. The domain of the variables are: $\forall 1 \leq i \leq n, D(X_i) = (1|2|3|\ldots|m)^\omega$ and $D(A) = (0|1|2|\ldots|m)^\omega$. The variable A has daton 0 at the time when no ball is at hand.

A ball falls down by 1 unit at a time, unless it is being thrown with force A again. The constraints are: $\forall 1 \leq i \leq n$, $(X_i == \mathbf{1}) \rightarrow (\texttt{next}\ X_i == A)$ and $(X_i \neq \mathbf{1}) \rightarrow (\texttt{next}\ X_i = X_i - \mathbf{1})$. Also, no two balls are being caught simultaneously: $\forall 1 \leq i < j \leq n, X_i \neq X_j$.

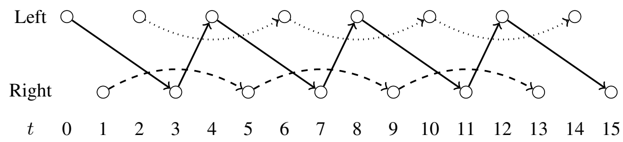

Figure 5: Space-time diagram of a juggling pattern. The three balls are represented by a solid, dashed, and dotted lines respectively.

Table 2: Run time and number of fails for simulating juggling for instance (n, m). The '-' marks 6000 seconds timeout.

	No Consistency		Prefix-1		Prefix-2		Prefix-3	
(n,m)	Time	Fails	Time	Fails	Time	Fails	Time	Fails
(3,3)	0.15	332	0.01	18	0.00	8	0.00	3
(3,4)	-	-	387.65	570049	259.40	235680	741.52	141277
(4,4)	13.88	1058	0.16	28	0.05	11	0.05	5
(5,5)	2644.42	2686	23.12	40	5.19	15	5.76	6
(6,6)	-	-	5452.18	55	1011.82	17	1112.05	7

One solution of instance $(3, 5)$: $X_1 = (3, 2, 1, 4)^\omega, X_2 = (2, 1, 4, 3)^\omega, X_3 = (1, 3, 2, 1)^\omega, A = (3, 4, 4, 1)^\omega$ which is a UP solution, is shown in Figure 5 by a space-time diagram. The automaton in Figure 6 recognizes a subset of solutions to the problem. The solution can be obtained in a run starts at state 0 and followed by sequence of states 1, 2, 3, 4 repeatedly. Other solutions can also be obtained by transversing different edges, including non-UP solutions, such as: $X_1 = \langle 3, 2, 1, 3, 2, 1, 4, 3 \ldots \rangle, X_2 = \langle 2, 1, 3, 1, 4, 3, 2 \ldots \rangle, X_3 = \langle 1, 3, 2, 3, 2, 1, 1 \ldots \rangle, A = \langle 3, 3, 3, 4, 4, 1, 4 \ldots \rangle$ which is the run of states $0, 1, 5, 0, 1, 2, 3, 4, \ldots$.

We conduct experiments on instances of (n, m) with prefix-k consistency where $k \in \{1, 2, 3\}$ and the results are listed in Table 2. When $n = m$, there are only repetitive juggling patterns as solutions. After enforcing consistency, the solutions can be easily obtained and thus the number of fails is small in those cases. When k is larger, the consistencies become stronger and thus more infeasible search space is pruned. As the time complexity of prefix-k consistency increases with k, the overall runtime cannot be compensated by the extra pruning when k is large.

In the problem, all constraints relate daton variables across only two time points, e.g. $X_i(t)$ and $X_i(t + 1)$ in the constraint $(X_i == 1) \rightarrow (\texttt{next}\ X_i == A)$. We conjecture that the optimal solving performance is obtained when k is chosen as the maximum difference of time points of all constraints involving the "`next`" and "`fby`" operators. The long solving time for instances $(3, 4)$ and $(6, 6)$ is due to the enumeration of many solutions and large problem size respectively.

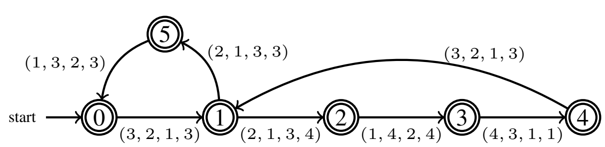

Figure 6: A Büchi automaton representing a subset of all solutions for (X_1, X_2, X_3, A) of instance $(3, 5)$.

6.2 Towards Generating Jazzy Harmonization

This problem is to generate the harmonization of four-part choral music. A choral music contains soprano, alto, tenor, and bass. Given the soprano notes which are repeated indefinitely, we have to determine the notes for alto, tenor, and bass so that the music is pleasant to listen for human beings.

We use variables X_1, X_2, X_3, X_4 to represent the sequences of notes for soprano, alto, tenor, and bass respectively. A music note is encoded as a number. For example, 60 is middle C (C4). We limit the range of notes to two octaves from 48 (C3) to 72 (C5). The domains are $D(X_1) = D(X_2) = D(X_3) = D(X_4) = (48 \,|\, \ldots \,|\, 72)^\omega$. Auxiliary variables help in modeling. For example, we have a set of pseudo-Boolean variables indicating the notes of each part, such as $\mathit{CInX_1}$, $\mathit{CsharpInX_1}$, and $\mathit{DInX_1}$ which represent whether X_1 takes the note C, C♯, and D respectively.

In this problem, we use the first four bars of the melody from "Twinkle Twinkle Little Star" (CCGGAAG, FFEEDDC) as a sentence and repeat it indefinitely: $\mathit{melody} = (60, 60, 67, 67, 69, 69, 67, 65, 65, 64, 64, 62, 62, 60)^\omega$. The end of the sentence is indicated by a pseudo-Boolean stream: $\mathit{end} = (0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1)^\omega$.

We implement a number of rules for harmonization [Tsang and Aitken, 1991]. For example, the parallel fifth rule is specified as $\forall i < j, X_i - X_j == \mathbf{7} \rightarrow \texttt{next}(X_i - X_j) <> \mathbf{7}$. The rule requiring that voices should never cross each other is expressed by $\forall \mathbf{2} \leq i \leq \mathbf{4}, X_i > X_{i-1}$.

The auxiliary variables can be constrained by $\mathit{CInX_i} = (X_i/12 == 0)$, $\mathit{CsharpInX_i} = (X_i/12 == 1)$, $\mathit{DInX_i} = (X_i/12 == 2)$, etc. The existence of a note in a chord can be defined in terms of these auxiliary variables: $\mathit{existC} = \mathit{CInX_1}$ `or` $\mathit{CInX_2}$ `or` $\mathit{CInX_3}$ `or` $\mathit{CInX_4}$. Then, each of the seven chord types can be given by constraints, e.g., $\mathit{chord_I} = (\mathit{existC}$ `and` existE `and` $\mathit{existG})$ for Chord I. Now, we can require that each chord must be one of the seven standard types: $\mathit{chord_I} + \ldots + \mathit{chord_{VII}} = 1$

By changing pitch, tempo, and delay of harmony, we can introduce jazzy feeling to the music.

When we decide to change the pitch of the song up to five intervals, we have to change it for every note in a sentence. Therefore, $D(\mathit{offset}) = (-5 \,|\, \ldots \,|\, 5)^\omega$ and `not` $\mathit{end} \rightarrow (\texttt{next}\ \mathit{offset} = \mathit{offset})$ and thus $X_1 = \mathit{melody} + \mathit{offset}$.

The change of tempo is also applied to a sentence for up to three times slower. $D(\mathit{tempo}) = (1 \,|\, 2 \,|\, 3)^\omega$ which represents the multiples of tempo of the original note: `not` $\mathit{end} \rightarrow (\texttt{next}\ \mathit{tempo} = \mathit{tempo})$.

The last feature is delay of harmony. When this style is applied to a chord, the harmony will be silent in the first half of the time. However, this style cannot be applied frequently to maintain pleasant feeling. Among any three consecutive chords, at most one chord can apply this style. Moreover, by the convention of music composition, the last note of each melody should keep long, and thus the style cannot be applied to the last note. We use a pseudo-Boolean variable delay to indicate the application of this style with initial domain $(0 \,|\, 1)^\omega$. The style is implied by imposing the following constraints: $\mathit{delay} + \texttt{next}\ \mathit{delay} + \texttt{next}\ \texttt{next}\ \mathit{delay} \leq 1$ and $\mathit{end} \rightarrow \texttt{not}\ \mathit{delay}$.

With the remaining constraints, we generate harmony for a given soprano which contains a repeated melody. The harmony can vary as the soprano repeats over time based on the solution automaton, which can serve as a basis for musical improvisation. Sample MIDI files generated from our solver can be downloaded online[1].

7 Concluding Remarks

Streams are related to coinduction [Rutten, 2005]. Fages and Rizk [2009] specify the problem using a formula in LTL which is the first approach to softness and optimization by quantifying the satisfaction degree of the formula. Pralet and Verfaillie [2008] use different techniques to solve problems in which variables have temporal dimension. Work on classical temporal constraints are too numerous to be mentioned [Dechter, 2003]. Our work also has some loose connections with online constraint solving [Verfaillie and Jussien, 2005]. The work by Gavanelli *et al.* [2005] is related but different from ours. It is the variable domains that are changing with possible values coming in incrementally, but variables still take just a scalar value from the evolving but always finite variable domains. In our case, each variable takes an infinite data stream as value from a possibly infinite variable domain of streams. Planning problems have been solved by constraint programming [van Beek and Chen, 1999]. While the number of steps is not known prior to solving, the problem is modelled for a fixed number of steps. The problem is re-modelled with increased number of steps until there is a solution found.

We consider data streams as a new domain for constrained variables. The constraint language allows us to use any classical constraint interpreted pointwisely and temporal operators inspired by the data-flow language Lucid [Wadge and Ashcroft, 1985]. The modelling examples show that the St-CSP framework makes it possible to give a declarative statement, such as the juggling specification, of the problem, which separates problem formulation and solution methods. This brings us one step towards the Holy Grail of programming [Freuder, 1997]: the user states the problem, the computer solves it. We have implemented a prototype solver for the framework to find all solutions. By using Büchi automata, the solver can give solutions including non-UP ones.

Optimization in St-CSP is an important future direction. For example, in musical generation, some rules can be more preferable to others. The framework opens a new direction of research. We have described the application to simulate the juggling and generate music harmonization in this paper. Other real life applications, such as controller synthesis, are worth for exploration. Interaction with live data streams is another possible venue for future work. Studying the effect of variable and value orderings is also worthwhile. Enhancement on the search strategies, such as applying more accurate heuristics for dominance detection, and introducing new consistency notions to the St-CSP, can improve the search performance. Improvement to the prototype solver in terms of implementation techniques and the use of data structures is also imminent.

[1]`http://www.cse.cuhk.edu.hk/~jlee/stcsp.mid`

Acknowledgments

We thank the anonymous referees for constructive comments and Jasper Lee for the advice on music harmonization. The work described in this paper was substantially supported by grants (CUHK413808 and CUHK413710) from the Research Grants Council of Hong Kong SAR.

References

[Apt and Brand, 2006] K. R. Apt and S. Brand. Infinite qualitative simulations by means of constraint programming. In *CP'06*, pages 29–43, 2006.

[Bessière and Régin, 1997] C. Bessière and J. C. Régin. Arc consistency for general constraint networks: Preliminary results. In *IJCAI'97*, pages 398–404, 1997.

[Büchi, 1962] J. R. Büchi. On a decision method in restricted second order arithmetic. In *International Congress on Logic, Method and Philosophy of Science*, pages 1–11, 1962.

[Dechter, 2003] R. Dechter. *Constraint Processing*. Elsevier Morgan Kaufmann, 2003.

[Fages and Rizk, 2009] F. Fages and A. Rizk. From model-checking to temporal logic constraint solving. In *CP'09*, pages 319–334, 2009.

[Freuder, 1997] E. C. Freuder. In pursuit of the holy grail. *Constraints*, 2(1):57–61, 1997.

[Gavanelli *et al.*, 2005] M. Gavanelli, E. Lamma, P. Mello, and M. Milano. Dealing with incomplete knowledge on CLP(FD) variable domains. *ACM Transactions on Programming Languages and Systems*, 27(2):236–263, 2005.

[Pralet and Verfaillie, 2008] C. Pralet and G. Verfaillie. Using constraint networks on timelines to model and solve planning and scheduling problems. In *ICAPS'08*, pages 272–279, 2008.

[Rutten, 2005] J. J. M. M. Rutten. A coinductive calculus of streams. *Mathematical Structures in Computer Science*, 15(1):93–147, 2005.

[Thomas, 1990] W. Thomas. Automata on infinite objects. In *Handbook of Theoretical Computer Science, Volume B: Formal Models and Semantics*, pages 133–192. Elsevier and MIT Press, 1990.

[Tsang and Aitken, 1991] C. P. Tsang and M. Aitken. Harmonizing music as a discipline of constraint logic programming. In *ICMC'91*, pages 61–64, 1991.

[van Beek and Chen, 1999] P. van Beek and X.G. Chen. Cplan: a constraint programming approach to planning. In *AAAI'99*, pages 585–590, 1999.

[Verfaillie and Jussien, 2005] G. Verfaillie and N. Jussien. Constraint solving in uncertain and dynamic environments: A survey. *Constraints*, 10(3):253–281, 2005.

[Wadge and Ashcroft, 1985] W. W. Wadge and E. A. Ashcroft. *Lucid, the Dataflow Programming Language*. Academic Press Professional, Inc., 1985.