Programming with Sequence Variables: the *Sequentica* Package

Mircea Marin, Johann Radon Institute for Computational and Applied Mathematics, Austria
Dorin Tepeneu, University ofTsukuba
University of Tsukuba
mircea.marin@oeaw.ac.at dorinte@score.is.tsukuba.ac.jp

1 Abstract

Sequence variables are an advanced feature of modern programming languages. They enhance the support for writing programs in a declarative and easily understood way. To our knowledge, *Mathematica* provides the best support for programming with sequence variables, but it requires a good understanding of how the interpreter chooses the matcher. This is so because matching against patterns with sequence variables is in general not unitary. We claim that there is room to improve the programming style with sequence variables. We propose a number of new programming constructs which impose certain strategies on the pattern matching process. Our constructs enable to control the selection of a matcher by annotating sequence variables with binding priorities and ranges for their lengths, and to compute optimal values characterized by a score function to be optimized. To this end we have developed the package *Sequentica*. With *Sequentica* the *Mathematica* programmers and users get additional support for defining functions and transformation rules in an easy and convenient way. We outline the algorithmic difficulties to support these extensions and describe how they are implemented in *Sequentica*. The usefulness of these extensions is illustrated with various examples. We regard these extensions as a first step towards identifying a new programming style: programming with sequence variables. Such a programming style is useful to solve problems based on sequence analysis such as bio-informatics, cryptography or data mining.

2 Motivation

The concept of sequence variable has proved useful in many areas, such as symbolic computation, theorem proving [2], term rewriting [1], unification [5] and computational

logic [4]. We feel that the expressive power of this concept is not yet completely understood and that a better formalization is still desirable. It is obvious that sequence variables, as defined in *Mathematica*, are very powerful and useful for proving, solving and computing. Still, the expressive power of this concept is superior to what *Mathematica* provides. To emphasize this fact, we mention:

1. Matching against patterns with sequence variables is in general not unique. There is no way to specify which matcher to choose but to accept the matcher chosen by the underlying interpreter. Often, the programmer wants to control this choice.
Example 1. Assume we want to define a function which returns the longest run of successive integers in a sequence. This may become possible if we could write:

```
GetMaxSublist[___, x___Integer, ___] := {x}
```

and specify a strategy to choose a matcher which yields the longest sequence binding for the sequence variable **x**. *Mathematica* has no programming support for such definitions.

2. Another desirable feature is to select the matcher which yields an optimum value, where optimum is defined for a matcher which minimizes or maximizes a certain expression.
Example 2.

Suppose we have a list of lists $L=\{\{T_{11}, T_{12}, \ldots\},\ldots,\{T_{12,1}, T_{12,2}, \ldots\}\}$ of temperature measurements during one year, where each sublist contains the temperatures measured in one month. We want to find the sublist of temperatures of the month with largest variation. It would be convenient to be able to write something like

```
GetMaxVariation[___,{Ts___Real},___]:={Ts}
                where Max[Ts]-Min[Ts] is maximum
```

to search among all matchers and keep the value of `{Ts}` for which the difference `Max[Ts]-Min[Ts]` is maximum. There is no direct support for such specifications.

3. There is no means to control the lengths of sequences in *Mathematica*. Interval bounds for sequences are supported in the specification of regular expressions and by many tools and languages based on their specification.
Example 3. Assume `L` is a list of daily temperature measurements during a long period of

time, and we want to find the week with highest average temperature by calling `HotWeek[L]`. In this case, we want to write something like

```
HotWeek[x___,y___,___]/;
    And[Length[{y}]==7,Mod[Length[{x}],7]==0]:=
Quotient[Length[{x},7],1]
    for which Plus[y]/7 is maximum
```

There is no adequate programming support to specify such a function in *Mathematica*.

To overcome these limitations, we propose a couple of new *Mathematica* syntactic constructs for programming with sequence variables and give an account to their *Mathematica* implementation. To this end we have developed the package *Sequentica* which supports the extensions reported in this paper.

3 Pattern Matching With Sequences in *Mathematica*

Pattern matching is the core of the definitional mechanisms of modern functional languages [7] and has wide applications in areas such as cryptography, computational biology, combinatorics, and parallel algorithms [6]. Pattern matching with sequences extends conventional pattern matching with a new syntactic category: sequence variables. In *Mathematica,* a sequence variable is a placeholder of a sequence of expressions which are spliced automatically in the body of the function being defined.
We have already mentioned that matching against patterns which contain sequence variables is in general not unique. For example, matching `f[1,2,3]` against `f[x___,y__]` yields 3 possible matchers, as witnessed by the call:

```
In[6]:= ReplaceList[f[1, 2, 3], f[x___, y__] :> {{x}, {y}}]

Out[6]= {{{}, {1, 2, 3}}, {{1}, {2, 3}}, {{1, 2}, {3}}}
```

Because of this situation, when we define functions with sequence variables we must specify the matcher which is being considered. The interpreter of *Mathematica* chooses the matcher which assigns the shortest sequences of arguments to the first sequence variables that show up when traversing the pattern in a leftmost-innermost manner, e.g.

```
In[9]:= f[x___, y__] := {{x}, {y}}; f[1, 2, 3]

Out[10]= {{}, {1, 2, 3}}
```

4 Extensions

Sequentica [8] is a *Mathematica* package which extends the capabilities of *Mathematica* with constructs for programming with sequence variables. After loading the package with `Get["Sequentica.m"]`, the user can define new functions f_i by calling

```
Sequentica[...; f_i[arg1,...,argn]:=<body_i>;...;]
```

or sets R of transformation rules by calling

```
R=Sequentica[{f[arg1,...,argn]:→<body>,...}]
```

The novelty is that the user can control the pattern matching mechanism and the computation with sequence variables by annotating the arguments and body of definitions and transformation rules. The following two sections explain how these new programming features.

Extension 1

This extension addresses the possibilities to (1) control the choice of a particular matcher instead of relying on some built-in pattern matching strategy; (2) confine the lengths of sequence variable bindings to certain intervals; and (3) impose equality constraints between the lengths of bindings of sequence variables. This can be achieved by annotating sequence variables with binding priorities which specify the order in which the sequences are assigned bindings upon pattern matching, and bounds for the lengths of their bindings. We propose to extend sequence annotations with the construct `patt←{p,m,M}` to denote a sequence with priority p and length varying from m to M. The bindings assigned to the sequence variables (such as `patt`) are computed in the increasing order of their priorities, by looking for the first matcher that is obtained by varying the sequence length from `m` to `M`. For example, the call

```
Sequentica[GetSubsequence[x___←{2,0,∞},y__←{1,∞
,1},y__,z___←{3,0,∞}]:={y}]
```

defines the function `GetSubsequence[]` which computes the list of the longest nonempty subsequence in the input, which occurs repeatedly at least twice. The assigned priorities are: 1 for **y**, 2 for **x** and 3 for **z**, and thus we first try to bind the sequence variable **y**, next **x**, and finally **z**. The bindings for **y** are looked up starting from the largest possible length (denoted here by ∞) down to length 1.

Example 5. It is convenient to be able to constrain different sequences to have the same length. We achieve this by imposing the condition that sequence variables annotated with same priority denote sequences of same length. The function

```
Sequentica[
 GetPalindrome[x___ ← {2, 0, ∞}, y__ ← {1, ∞, 1},
    u_, z__ ← {1, ∞, 1}, t__ ← {3, 0, ∞}] /;
   {y} === Reverse[{z}] := {y, u, z}]
```

computes the longest palindrome of odd length contained in the input sequence.

To avoid excessive annotations, we will assume that (1) it is sufficient to annotate one occurrence of a sequence variable; all other occurrences of that variable are assumed to have the same annotation, (2) anonymous sequence variables have assigned distinct names, and (3) a sequence variable y__ (resp. y___) which is not explicitly annotated has an implicit sequence annotation of the form **←{p,1,∞}** (resp. **←{p,0,∞}**). The default priorities **p** are assigned in a leftmost-innermost manner, and are assumed to be larger than the priorities given explicitly. This means that the non-annotated sequence variables are the last ones which get assigned bindings. With these assumptions in mind, the function **`GetMaxSublist[]`** from Example 1 can be defined by

```
Sequentica[GetMaxSublist[___, x___Integer, ___] := {x}]
```

Extension 2

Our second extension addresses the possibility to select matching substitutions which render an expression optimal. To illustrate, let us reconsider Example 2: we want to select the sublist of a list which has a maximum variation of values. In particular, we suggest the following syntax:

```
Sequentica[GetMaxVariation[___, {x___Real}, ___] :=
  BestFit[{x}, Max[x] - Min[x], Greater]]
```

to find the list `{x}` computed by a matching against `{___,{x___Real},___}` which produces the greatest value of the expression `Max[x]-Min[x]`.

We suggest the following new definitional mechanisms inside `Sequentica[]` calls:

```
f[patts]:=BestFit[expr,optim,test]
f[patts]:=BestFits[n,expr,optim,test]
```

The first call defines a partial function `f` which associates a call `f[a1,…,am]` to the output Θ`(expr)` produced by the matcher Θ between `f[a1,…,an]` and `f[patts]` which satisfies `test(`Θ`(optim),`γ`(optim))=True` for all other matchers γ between `f[a1,…,am]` and `f[patts]`.
The second call assumes `n>0` and defines a function `f` which associates to a call `f[a1,…,am]` the list of instances `{`Θ`i(expr)` | `1≤i≤k}` produced by the longest list of distinct matchers `{`Θ`1,…,`Θ`k}` between such that k≤n and
1. `test[` Θ_j`(optim),`Θ_{j+1}`(optim)]=True` if 1≤j<k,
2. `test[` Θ_k`(optim),`Θ`(optim)]=True` for all matchers $\Theta \notin \{\Theta_1, \ldots, \Theta_k\}$.

In both cases, `test[]` is assumed to define a total order on the set of values Θ`(optim)` generated by the matchers Θ of arbitrary expressions against `f[patts]`.

Example 3 revisited. We can now define `HotWeek[]` as follows:

```
Sequentica[HotWeek[x___, y___] ← {1, 7, 7}, ___] /;
    Mod[Length[{x}], 7] == 0 := BestFit[
    Quotient[Length[{x}], 7] + 1, Plus[y] / 7, Greater]];
```

5 Algorithmic Aspects and Implementation Issues

Matchers are computed by traversing the sequence pattern outside-in and computing the bindings of the sequence variables in the order given by their priorities and by varying the bindings of their lengths between their specified limits. For example

```
In[29]:= Sequentica[f[(x__Integer) ← {1, ∞, 2}, z___,
             x___, y__ ← {2, ∞, 1}] := {{x}, {y}, {z}}];
         f[1, 2, 1, 1, 2, 1, 2, 2]
```

yields `{{1,2,1},{2,2},{}}` because `{x:→Sequence[1,2,1],y:→ Sequence[]}` is the first matcher encountered by varying the length of the binding of x from ∞ down to 2 and next the length of y from ∞ down to 1. .

```
In[32]:= Sequentica[f[(x__Integer) ← {1, ∞, 2}, z___,
             x___, y__ ← {2, ∞, 1}] := {{x}, {y}, {z}}];
```

If we redefine `f` and call

```
In[34]:= Sequentica[f[(x__Integer) ← {1, 1, ∞}, z___,
             x___, y__ ← {2, ∞, 1}] := {{x}, {y}, {z}}];
         f[1, 2, 1, 1, 2, 1, 2, 2]
```

then we obtain `{{1},{1,2,1,2,2},{2}}` because we have changed the order to look up for a matcher.

The implementation of such a pattern matching algorithm enumerates the possible lengths of bindings in the order specified by sequence annotations, detects the first combination of lengths for which a matcher exists, and evaluates the body of the corresponding function (or transformation rule) for the computed matcher. Note that the algorithm must be recursive on the structure of matchers, because sequence variables can occur at different depths in the pattern specified by the user.

Of crucial importance is the identification of an efficient algorithm which enumerates the possible lengths of sequence variables. For this purpose, *Sequentica* relies heavily on an iterative solver for systems of linear diophantine constraints over finite domains [3]. This means that the solver does not compute all solutions at once, but enumerates them in the order imposed by the sequence annotations. Actually, it would be unreasonable to rely on

a solver for linear diophantine constraints which computes all solutions at once because (1) the space of solutions can be very large, and (2) we are not interested in all solutions but only in the first one which realizes a matching (where *first* is defined w.r.t. the order inferred from the priorities attached to sequence variables).
The second extension is of a slightly different nature: here the sequence annotations are irrelevant since the evaluation must check all possible matchers and resume the computation for the matchers which fulfill the requirements of the score function specified by the user. In this case, *Sequentica* generates a *Mathematica* definition which accumulates the optimum value during an exhaustive search of all possible matchers.

6 Conclusion and Further Work

We believe that the support for programming with sequence variables can be further enhanced. We will continue to look for such useful programming idioms and will integrate them in the *Sequentica* package which can be downloaded from http://www.score.is.tsukuba.ac.jp/~mmarin/Sequentica.
Further implementation details of *Sequentica* are available from http://www.score.is.tsukuba.ac.jp/~mmarin/Sequentica/sequentica.ps.

(Mircea Marin is supported by the Austrian Academy of Sciences)

References

[1] B. Buchberger. *Mathematica* as a Rewrite Language. In T. Ida, A. Ohori, and M. Takeichi, editors, *Proceedings of the 2nd Fuji International Workshop on Functional and Logic Programming*, Shonnan Village Center, 1996.

[2] B. Buchberger, T. Jebelean, F. Kriftner, M. Marin, and D. Vasaru. An Overview of the *Theorema* Project. In *Proc. of ISSAC'97*, pp. 384-391, Maui, Hawaii, 1997. ACM Press.

[3] E. Contejean. Solving Linear Diophantine Constraints Incrementally. In D.S. Warren, editor, *Proc. of the 10th Intl. Conf. on Logic Programming*, pp. 532-549, Budapest, Hungary 1993. MIT Press.

[4] The Logic Group. Knowledge Interchange Format. Technical report, Stanford University, http://logic.stanford.edu/kif/kif.html, 2002.

[5] T. Kutsia. *Solving and Proving in Equational Theories with Sequence Variables and Flexible Arity Symbols*. PhD thesis, Institute RISC-Linz, Johannes Kepler University, Hagenberg, Austria, June 2002.

[6] Pattern Matching Pointers. http://www.dei.unipd.it/~stelo/pattern.html.

[7] S. Thompson. *Haskell: The craft of functional programming*. Addison-Wesley, second edition, 1999.

[8] The *Sequentica* Package. http://www.score.is.tsukuba.ac.jp/~mmarin/Sequentica.