A Tool proving Well-definedness of Streams using Termination Tools

Hans Zantema

Department of Computer Science, TU Eindhoven, P.O. Box 513,
5600 MB Eindhoven, The Netherlands, H.Zantema@tue.nl
and
Institute for Computing and Information Sciences, Radboud University
Nijmegen, P.O. Box 9010, 6500 GL Nijmegen, The Netherlands

Abstract. A stream specification is a set of equations intended to define a stream, that is, an infinite sequence over a given data type. In [5] a transformation from such a stream specification to a TRS is defined in such a way that termination of the resulting TRS implies that the stream specification admits a unique solution. In this tool description we present how proving such well-definedness of several interesting boolean stream specifications can be done fully automatically using present powerful tools for proving TRS termination.

1 Introduction

Streams are among the simplest data types in which the objects are infinite. We consider streams to be maps from the natural numbers to some data type D. The basic constructor for streams is the operator ':' mapping a data element d and a stream s to a new stream $d : s$ by putting d in front of s. Using this operator we can define streams by equations. For instance, the stream zeros only consisting of 0's can be defined by the single equation $\mathsf{zeros} = 0 : \mathsf{zeros}$.

More complicated streams are defined using stream functions. For instance, a boolean steam c together with stream functions f and g is uniquely defined by

$$\begin{aligned} c &= 1 : f(c) & \qquad g(0, xs) &= 0 : 1 : f(xs) \\ f(x : xs) &= g(x, xs) & \qquad g(1, xs) &= 0 : f(xs), \end{aligned}$$

in fact, c is the tail of the well-known Fibonacci stream. However, the following modification of this stream specification

$$\begin{aligned} c &= 1 : f(c) & \qquad g(0, xs) &= 0 : 1 : f(xs) \\ f(x : xs) &= g(x, xs) & \qquad g(1, xs) &= f(xs) \end{aligned}$$

does **not** uniquely define a stream c.

The main issue we address is the following: given a stream specification like the two examples above, can we automatically check that it uniquely defines the streams and stream functions involved? If so, we call the stream specification

well-defined. For this question in [5] a technique has been proposed in which the stream specification S is transformed to its *observational variant* $\mathsf{Obs}(S)$, being a term rewrite system (TRS). The main result of [5] states that if the TRS $\mathsf{Obs}(S)$ is terminating, then the stream specification S admits a unique solution. As for proving termination of TRSs automatically during the last ten years strong tools have been developed like AProVE [2] and TTT2 [3], our approach to prove well-definedness of a stream specification S simply consists of proving termination of $\mathsf{Obs}(S)$ by a tool like AProVE or TTT2. It is not even required to download such a termination tool: both tools admit a web interface in which any TRS is easily entered. Our tool transforms any boolean stream specification S in the format as given here to $\mathsf{Obs}(S)$ in the TRS format as is accepted by the termination tools. A full version with graphical user interface runs under Windows; a command line version runs under Linux. Both versions are downloaded from

`www.win.tue.nl/~hzantema/str.zip`

together with a number of examples.

This paper is structured as follows. In Section 2 we present the basic definitions and theorems. However, playing around with the tool and trying termination tools on the resulting systems can be done without mastering all details. In Section 3 we describe the features of the tool. In Section 4 we show a number of examples. We conclude in Section 5.

2 Definitions and Theory

In stream specifications we have two sorts: s (stream) and d (data). We write D for the set of data elements; here we restrict to the boolean case where $D = \{0, 1\}$. We assume a particular symbol : having type $d \times s \to s$. For giving the actual stream specification we need a set Σ_s of stream symbols, each being of type $d^n \times s^m \to s$ for $n, m \geq 0$. Terms of sort s are defined in the obvious way. As a notational convention we will use

- x, y, z for variables of sort d,
- u, u_i for terms of sort d,
- xs, ys, zs for variables of sort s,
- t, t_i for terms of sort s.

Definition 1. *A* stream specification (Σ_s, S), *or shortly* S, *consists of* Σ_s *as given before, and a set* S *of equations of the shape* $f(u_1, \ldots, u_n, t_1, \ldots, t_m) = t$, *where*

- $f \in \Sigma_s$ *is of type* $d^n \times s^m \to s$,
- $u_1, \ldots, u_n$ *are terms of sort* d,
- *for every* $i = 1, \ldots, m$ *the term* t_i *is either a variable of sort* s, *or* $t_i = x : xs$ *where* x *is a variable of sort* d *and* xs *is a variable of sort* s,
- t *is any term of sort* s,
- *no variable occurs more than once in* $f(u_1, \ldots, u_n, t_1, \ldots, t_m)$,

– *for every term of the shape* $f(u_1, \ldots, u_n, u_{n+1} : t_1, \ldots, u_{n+m} : t_m)$ *where* $u_i \in D$ *for* $i = 1, \ldots, m + n$, *and* $t_1, \ldots, t_m$ *are variables, exactly one rule from* S *is applicable.*

Example 1. The Thue-Morse sequence morse can be specified by the following stream specification:

$$\begin{aligned} \mathsf{morse} &= 0 : \mathsf{zip}(\mathsf{inv}(\mathsf{morse}), t(\mathsf{morse})) & g(1, xs) &= 0 : \mathsf{inv}(xs) \\ \mathsf{inv}(x : xs) &= g(x, xs) & t(x : xs) &= xs \\ g(0, xs) &= 1 : \mathsf{inv}(xs) & \mathsf{zip}(x : xs, ys) &= x : \mathsf{zip}(ys, xs). \end{aligned}$$

Here g is introduced to meet our format: $\mathsf{inv}(0 : xs)$ is not allowed as a left hand side. The operation t represents the tail function; the symbol tail is reserved for later use.

Stream specifications are intended to specify streams for the constants in Σ_s, and stream functions for the other elements of Σ_s. The combination of these streams and stream functions is what we will call a *stream model*. As streams over D are maps from the natural numbers to D we write D^ω for the set of all streams over D.

Definition 2. *A* stream model *is defined to consist of a set* $M \subseteq D^\omega$ *and a set of functions* $[f]$ *for every* $f \in \Sigma_s$, *where* $[f] : D^n \times M^m \to M$ *if the type of* f *is* $d^n \times s^m \to s$.

We write $\mathcal{T}_s$ for the set of ground terms of sort s. For $t \in \mathcal{T}_s$ the interpretation $[t]$ in a stream model is defined inductively by:

$$\begin{aligned} [f(u_1, \ldots, u_n, t_1, \ldots, t_m)] &= [f]([u_1], \ldots, [u_n], [t_1], \ldots, [t_m]) && \text{for } f \in \Sigma_s \\ [u : t](0) &= [u] \\ [u : t](i) &= [t](i - 1) && \text{for } i > 0 \end{aligned}$$

for $u, u_i \in D$ and $t, t_i \in \mathcal{T}_s$

So in a stream model a user defined operator f is interpreted by the given function $[f]$, and the operator : applied on a data element d and a stream s is interpreted by putting d on the first position and shifting every stream element of s to its next position.

A stream model $(M, ([f])_{f \in \Sigma_s})$ is said to *satisfy* a stream specification (Σ_s, S) if $[\ell\rho] = [r\rho]$ for every equation $\ell = r$ in S and every ground substitution ρ. We also say that the specification *admits* the model.

Now we define a transformation Obs transforming the original stream specification S to its *observational variant* $\mathsf{Obs}(S)$. The basic idea is that the streams are observed by two auxiliary operator head and tail, of which head picks the first element of the stream and tail removes the first element from the stream, and that for every $t \in \mathcal{T}_s$ of type stream both $\mathsf{head}(t)$ and $\mathsf{tail}(t)$ can be rewritten by $\mathsf{Obs}(S)$.

First we define $\mathsf{P}(S)$ obtained from S by modifying the equations as follows. By definition every equation of S is of the shape $f(u_1, \ldots, u_n, t_1, \ldots, t_m) = t$

where for every $i = 1, \ldots, m$ the term t_i is either a variable of sort s, or $t_i = x : xs$ where x is a variable of sort d and xs is a variable of sort s. In case $t_i = x : xs$ then in the left hand side of the rule the subterm t_i is replaced by xs, while in the right hand side of the rule every occurrence of x is replaced by $\mathsf{head}(xs)$ and every occurrence of xs is replaced by $\mathsf{tail}(xs)$.

For example, the zip rule in Example 1 will be replaced by

$$\mathsf{zip}(xs, ys) = \mathsf{head}(xs) : \mathsf{zip}(ys, \mathsf{tail}(xs)).$$

Now we are ready to define Obs.

Definition 3. *Let (Σ_s, S) be a stream specification. Let $\mathsf{P}(S)$ be defined as above. Then $\mathsf{Obs}(S)$ is the TRS consisting of*

- *the two rules* $\mathsf{head}(x : xs) \to x$, $\mathsf{tail}(x : xs) \to xs$,
- *for every equation in* $\mathsf{P}(S)$ *of the shape* $\ell = u : t$ *the two rules*

$$\mathsf{head}(\ell) \to u, \quad \mathsf{tail}(\ell) \to t,$$

- *for every equation in* $\mathsf{P}(S)$ *of the shape* $\ell = r$ *with* $\mathsf{root}(r) \neq :$ *the two rules*

$$\mathsf{head}(\ell) \to \mathsf{head}(r), \quad \mathsf{tail}(\ell) \to \mathsf{tail}(r).$$

From [5] we recall the two variants of the main theorem.

Theorem 1. *Let (Σ_s, S) be a stream specification for which the TRS $\mathsf{Obs}(S)$ is terminating. Then the stream specification admits a unique model $(M, ([f])_{f \in \Sigma_s})$ satisfying $M = \{[t] \mid t \in \mathcal{T}_s\}$.*

Theorem 2. *Let (Σ_s, S) be a stream specification for which the TRS $\mathsf{Obs}(S)$ is terminating and the only subterms of left hand sides of S of sort d are variables. Then the stream specification admits a unique model $(M, ([f])_{f \in \Sigma_s})$ satisfying $M = D^\omega$.*

In [5] it is shown by an example that this distinction is essential: uniqueness of stream functions does not hold if the stream specification is data dependent, that is, left hand sides contain symbols 0 and 1. Moreover, in [5] it has been shown that the technique is not complete: some fix-point specifications are well-defined while the observational variant is non-terminating.

3 The Tool

The full version of the tool with graphical user interface under Windows provides the following features:

- Boolean stream specifications can be entered, loaded, edited and stored. The format is the same as given here, with the only difference that for the operator ':' a prefix notation is chosen, in order to be consistent with the user defined symbols.

– By clicking a button the observational variant of the current stream specification is tried to be created. In doing so, all requirements of the definition of stream specification are checked. If they are not fulfilled, an appropriate error message is shown.
– If all requirements hold, then the resulting observational variant is shown on the screen by which it can be entered by cut and paste in a termination tool. Alternatively, it can be stored.
– An initial part of a stream can be computed by applying outermost rewriting based on the stream specification starting in a given term.
– Several stream specifications, including Fibonacci stream, Thue-Morse stream and Kolakoski stream are predefined. For all of these examples termination of the observational variant can be proved fully automatically both by AProVE [2] and TTT2 [3], proving well-definedness of the given stream specification.

The main facility, that is, generating the observational variant in case the syntax is correct, and generating an appropriate error message otherwise, is also provided by the command line verison under Linux.

None of the actions require substantial computation: for all features the result shows up instantaneously. On the other hand, proving termination of a resulting observational variant by a tool like AProVE or TTT2 may take some computation time, although never more than a few seconds for the given examples. This was one of the objectives of the project: the transformation itself should be simple and direct, while the real work to be done makes use of the power of current termination provers.

4 Examples

In this section we give a few examples to give an impression of how the input and output of our tool looks like. The first example is the Fibonacci stream as presented in the introduction. The input is

```
f(:(x,xs)) = g(x,xs)
g(0,xs) = :(0,:(1,f(xs)))
g(1,xs) = :(0,f(xs))
c = :(1,f(c))
fib = :(0,c)
```

Within a fraction of a second our tool transforms this to the observational variant

```
(VAR xs ys x)
(RULES
hd(f(xs)) -> hd(xs)
tl(f(xs)) -> zip(f(odd(tl(xs))),f(even(tl(xs))))
hd(zip(xs,ys)) -> hd(xs)
tl(zip(xs,ys)) -> zip(ys,tl(xs))
hd(odd(xs)) -> hd(xs)
tl(odd(xs)) -> even(tl(xs))
```

```
hd(even(xs)) -> hd(odd(tl(xs)))
tl(even(xs)) -> tl(odd(tl(xs)))
hd(c) -> 0
tl(c) -> :(1,c)
hd(:(x,xs)) -> x
tl(:(x,xs)) -> xs
)
```

As a second example we consider the Thue-Morse sequence as specified in Example 1:

```
morse = :(0,zip(inv(morse),t(morse)))
inv(:(x,xs)) = g(x,xs)
g(0,xs) = :(1,inv(xs))
g(1,xs) = :(0,inv(xs))
t(:(x,xs)) = xs
zip(:(x,xs),ys) = :(x,zip(ys,xs))
```

yielding as output:

```
(VAR xs ys x)
(RULES
hd(morse) -> 0
tl(morse) -> zip(inv(morse),t(morse))
hd(inv(xs)) -> hd(g(hd(xs),tl(xs)))
tl(inv(xs)) -> tl(g(hd(xs),tl(xs)))
hd(g(0,xs)) -> 1
tl(g(0,xs)) -> inv(xs)
hd(g(1,xs)) -> 0
tl(g(1,xs)) -> inv(xs)
hd(t(xs)) -> hd(tl(xs))
tl(t(xs)) -> tl(tl(xs))
hd(zip(xs,ys)) -> hd(xs)
tl(zip(xs,ys)) -> zip(ys,tl(xs))
hd(:(x,xs)) -> x
tl(:(x,xs)) -> xs
)
```

Finally we consider the stream specification

```
f(:(x,xs)) = g(x,xs)
g(0,xs) = :(1,f(xs))
g(1,xs) = :(0,f(f(xs)))
c = :(1,c)
conc = f(c)
```

yielding as output:

```
(VAR xs x)
(RULES
hd(f(xs)) -> hd(g(hd(xs),tl(xs)))
tl(f(xs)) -> tl(g(hd(xs),tl(xs)))
hd(g(0,xs)) -> 1
tl(g(0,xs)) -> f(xs)
hd(g(1,xs)) -> 0
tl(g(1,xs)) -> f(f(xs))
hd(c) -> 1
tl(c) -> c
hd(conc) -> hd(f(c))
tl(conc) -> tl(f(c))
hd(:(x,xs)) -> x
tl(:(x,xs)) -> xs
)
```

Both AProVE [2] and TTT2 [3] are able to prove termination each of the resulting TRSs within a few seconds, in this way proving well-definedness of all three original specifications. Unfortunately the generated proofs of both tools are quite long and not easily readable.

5 Conclusions, Related Work

We presented a technique and described a tool by which well-definedness of stream specifications can be proved fully automatically. This is done by proving termination of a transformed term rewrite system $\mathsf{Obs}(S)$ using any termination provers. In this way the power of present termination provers is exploited.

We mention two other approaches to this problem. The first is based on the well-known property that productivity implies well-definedness. An approach to prove productivity is given in [1], together with a corresponding tool. The second s by proving equality of two copies of the same specification by the tool Circ [4]. In the latter it is essential that the specification is deterministic.

However, for proving well-definedness of $f(c)$ in

$$\begin{aligned} f(0:xs) &= 1 : f(xs) \qquad & c &= 1 : c \\ f(1:xs) &= 0 : f(f(xs) & & \end{aligned}$$

both other approaches fail, while our approach succeeds. This specification in our format (introducing g) and its observational variant were presented in Section 4 as the third example.

Conversely, there are examples where the other approaches succeed and our approach fails. For instance, productivity of M of the specification

$$\begin{aligned} M &= 1 : \mathsf{even}(\mathsf{zip}(M, M)) \\ \mathsf{zip}(a:s,t) &= a : \mathsf{zip}(t,s) \\ \mathsf{even}(a:s) &= a : \mathsf{odd}(s) \\ \mathsf{odd}(a:s) &= \mathsf{even}(s) \end{aligned}$$

is easily proved by the tool of [1], while our approach fails to prove well-definedness.

References

1. J. Endrullis, C. Grabmayer, and D. Hendriks. Data-oblivious stream productivity. In *Proceedings of the 11th International Conference on Logic for Programming, Artificial Intelligence, and Reasoning (LPAR'08)*, volume 5330 of *Lecture Notes in Computer Science*, pages 79–96. Springer, 2008. webinterface tool: `http://fspc282.few.vu.nl/productivity/`.
2. J. Giesl et al. Automated program verification environment (AProVE). Available at `http://aprove.informatik.rwth-aachen.de/`.
3. M. Korp, C. Sternagel, H. Zankl, and Aart Middeldorp. Tyrolean termination tool 2. In R. Treinen, editor, *Proceedings of the 20th Conference on Rewriting Techniques and Applications (RTA)*, Lecture Notes in Computer Science. Springer, 2009. Tool available at `http://colo6-c703.uibk.ac.at/ttt2/`.
4. D. Lucanu and G. Rosu. CIRC: A circular coinductive prover. In *CALCO'07*, volume 4624 of *Lecture Notes in Computer Science*, pages 372 – 378, 2007.
5. H. Zantema. Well-definedness of streams by termination. In R. Treinen, editor, *Proceedings of the 20th Conference on Rewriting Techniques and Applications (RTA)*, volume 5595 of *Lecture Notes in Computer Science*, pages 164–178. Springer, 2009. Available at `www.win.tue.nl/~hzantema/str.pdf`.