Teckel

Dick Grune
dick@dickgrune.com

October 23, 2014; DRAFT

mene teckel upharsin
counted, weighed, and split
Daniel 5:25

1 Introduction

Teckel is an interpreter for non-deterministic and deterministic algorithms written in a mathematical style and expressed in LaTeX. Examples of such "abstract algorithms" can be found in Section 3. Teckel tries hard to find solutions, avoid getting sidetracked by infinite dead ends, and produce at least some results in finite time. On the other hand, speed has consistently been sacrificed for power.

Teckel allows the same LaTeX text to be run and published, thus cutting out one conversion layer and a lot of programming. It is also a convenient means for running non-deterministic programs.

Teckel works on the LaTeX input text rather than on the LaTeX output. This means that Teckel has to be used carefully: it is quite possible to construct a Teckel program that runs correctly and shows up on paper as nonsense, and vice versa.

2 Features

The most salient features are:

2.1 Finite and Infinite Sets

Finite and infinite sets with their operators are available. The Teckel program

$$Q \equiv \{n^2, n \in \mathbb{N}\}$$
$$C \equiv \{n^3, n \in \mathbb{N}\}$$
$$Q \cap C$$

starts printing the infinite set of natural numbers that are both squares and cubes: 0, 1, 64, ...

If an operator that is not defined for sets is applied to sets, it is distributed over the components of the set: $\{2,3\} \times \{3,4\} = \{2 \times 3, 2 \times 4, 3 \times 3, 3 \times 4\} = \{6,8,9,12\}$. This also applies to the (invisible) concatenation operator:

$\{x, xy\}\{y, yy\} = \{xy, xyy, xyy, xyyy\} = \{xy, xyy, xyyy\}$, where x and y are atoms.

In short, $S_1 \otimes S_2 \equiv \{a \otimes b, a \in S_1, b \in S_2\}$ for any $\otimes$ defined on members of S_1 and S_2. ˚

˚ This isn't true for Boolean operators; why?

There is no coercion from a value to a singleton containing that value. More in particular, $3 \times \{3, 4\}$ does not work; use $\{3\} \times \{3, 4\}$ instead.

2.2 Arithmetic

Numbers are represented as rationals with the numerator and the denominator implemented as machine integers with as large as possible accuracy. Rationals are implemented and printed with non-negative denominators. Teckel accepts $+1/0$ for $+\infty$ ("infinity") and $-1/0$ for $-\infty$, but not $0/0$ for "indefinite", since it breaks the transitivity of the $=$ relation: $a = b \wedge b = c \Rightarrow a = c$ does not hold for $b = 0/0$.

There is $\mathbb{N}$ (`\mathbb{N}`) for the set of natural numbers $\{1..\infty\}$; likewise $\mathbb{Z}$ (`\mathbb{Z}`) stands for $\{0..\infty\}$. Given the nature of Teckel there is no support for floating point computation.

2.3 Filters

A filter is defined using the vertical bar `|`:

$$\{a^2, a \in \mathbb{N} \mid a \bmod 2 = 0\}$$

defines the set of squares of even numbers. The $a \in \mathbb{N}$ is a declaration. A declaration can also serve as a formal parameter itself:

$$\{a \in \mathbb{N} \mid a \bmod 2 = 0\}$$

is the set of all even numbers.

2.4 Relations are Sets of Pairs of Values

Example:

$$\rho \equiv \{(a, b) \in \mathbb{N} \mid a = b^2\}$$

defines ρ as the relation of numbers and their squares: $3\ \rho\ 9$ is equivalent to $(3, 9) \in \rho$ and yields *true*; $0\ \rho\ 5$ yields *false*.˚

˚ implementation problem

Relations can be composed:

$$\rho \circ \sigma \equiv \{(a, c) \mid \exists(b) a\rho b \wedge b\sigma c\}$$

See also Section 2.10.

2.5 Variables, Constants and Atoms

Identifiers occurring in the left-hand side of definitions are understood as constants, operators, functions, or formal parameters, depending on their positions. Identifiers occurring in the left-hand side of assignments (see 4.4) are understood as variables. Identifiers occurring in the program but not understood as any of the above are understood as atoms, and represent themselves. The only operation on atoms value is comparison for equality (=).

2.6 Sequences

Sequences can be indexed by subscripting (e.g. x_1) or by square brackets (e.g. $x[1]$). Subsequences are indicated by ranges: $x_3...x_5$. They are concatenated by juxtaposition (the invisible operator): $x\ S\ x$.

2.7 Arrays

Arrays are somewhat different from sequences in that not all elements in a range need to exist; an array a can have the elements $a[1]$, $a[100]$, $a[10000]$, and have no other elements. It takes the space of 3 elements rather than of 10000.

Taking the value of a non-existing array element yields `NO_RESULT`; assigning to a non-existing array element creates it.

2.8 Scoping

New scopes are provided by formal parameters of function definitions and set filters `|`.

Functions are identified by name and shape, which includes the number of arguments; $F(a, b, c)$, $F_{a,b}(c)$ and $F(a, b, c, d)$ all call different functions. The types of the arguments are not considered. Rather, the types of the arguments of all calls to a given function with a given name and shape are unified. If the unification fails, `NO_RESULT` is returned.

There are no formal or local functions.

2.9 Shorthands

Teckel features a number of shorthands usual in abstract algorithms:

Form	Meaning
$x_{1\cdots5}$	$x_1x_2x_3x_4x_5$
$x_1 \cdots x_5$	idem
$x_1 + \cdots + x_5$	$x_1 + x_2 + x_3 + x_4 + x_5$, using most binary operators(+, ∧, ∨, etc.)
$x_1 + x_3 + \cdots + x_5$	in steps of 2
$x_1 + x_3 + \cdots + x_6$	erroneous because the sequence misses the x_6
$x_1 \leq \ldots \leq x_n$	$(x_1 \leq x_2) \wedge (x_2 \leq x_3) \wedge \cdots \wedge (x_{n-1} \leq x_n)$
$x \leq y = z \leq w$	$(x \leq y) \wedge (y = z) \wedge (z \leq w)$, using any comparison operator

2.10 Transitive Closure

◦◦

◦ rewrite
◦ what?

3 Teckel Examples

3.1 First steps

◦

◦ Test these for syntax

– aap noot comment

$(1 + 2 \times 3 + 3^8 + 5 + 3) \times (1 - 0)$

3.2 A toy example

From E. C. Freuder,*A Sufficient Condition for Backtrack-Free Search*, J. ACM (29(1):24-32 (1982).

$X \equiv \{5, 2, 4, 6\}$

$Y \equiv \{2, 4, 6, 10\}$

$Z \equiv \{5, 2, 4, 6\}$

$\{(x, y, z), x \in X, y \in Y, z \in Z \mid x\%z = 0 \ \wedge \ y\%z = 0\}$

3.3 Pythagorean triples

The program:

$\{(a, b, c), a, b, c \in \mathbb{N} \mid a^2 + b^2 = c^2\}$

3.4 CF Recognition

The production mechanism:

$\mathcal{L}(a \in T) \equiv \{a\}$

$\mathcal{L}(A \in N) \equiv \bigcup_{(A \to \alpha) \in P} (\mathcal{L}(A \to \alpha))$

$\mathcal{L}((A \to a_1 \dots a_n) \in P) \equiv \mathcal{L}(a_1) \dots \mathcal{L}(a_n)$

The grammar:

$P \equiv \{\mathrm{S} \to \mathrm{xSx}, \mathrm{S} \to \varepsilon\}$; $T \equiv \{\mathrm{x}\}$; $N \equiv \{\mathrm{S}\}$

The program:

$\mathrm{xx} \in \mathcal{L}(\mathrm{S})$

$\mathrm{xxx} \in \mathcal{L}(\mathrm{S})$

Note that the second expression requires obtaining a negative membership result over an infinite set.

3.5 Sorting

$\mathrm{SORT}(x) \equiv 1$

$\mathrm{PERM}(x_1 \cdots x_n) \equiv \bigcup_{1 \le i \le n} (x_i \ \mathrm{PERM}(x_{1..(i-1)} x_{(i+1)..n}))$

3.6 Hamming Problem

° ° Not sorted!

$H \equiv \{1\} \cup (\{2, 3, 5\} \times H)$

4 The Input Language

4.1 Program Identification

° ° rewrite

We would like to allow the user to pass the entire LaTeX file (or even a list of them) to Teckel. This raises the question how to identify the Teckel code among the rest of the LaTeX text. The (obvious) solution is to have directives like

```
%Teckel on
```

and

```
%Teckel off
```

to demarcate the Teckel text.

```
%Teckel end
```

Note that these are ignored automatically by LaTeX.

Since one paper can contain several programs, the `%Teckel` directive can be given an identifier, to be specified in the command line:

```
%Teckel on [<progname>,<progname>,...]
```

All text with the same `<progname>` is combined into one text and handed to Teckel. `<Progname>` can be any sequence of characters not including white space, commas or semicolons.

4.2 Program Style

To be written° °

4.3 Lexical Form

$$2B \vee \neg 2B$$
Hamlet, 3.1.64

The input consists of Teckel syntax symbols, numbers, identifiers, and Teckel commands. They are represented by single characters (for example `a` or `{`) and by LaTeX commands (for example `\leq` or `\Tassign`). There are also Teckel directives, but strictly speaking these are not part of the Teckel program; they also have a different shape in that they start with a `%` character.

4.3.1 Syntax Symbols

Teckel has the following printing syntax symbols:

Symbol	Example	Usage
`,`	a, b	list separator
`( )`	(a, b)	for expressions, tuples, parameter lists
`[ ]`	$A[i]$	for indexes
`{ }`	S_{i+1}	as LaTeX parentheses
`\{ \}`	$\{a, b\}$	for sets
`|`	$\mid i > 1$	for set filters, "such that"
	$\|V\|$	the size of set V
`? :`	$a \geq 0 ? a : -a$	conditional expression
`..`	$1..10$	ellipsis
`...`	$x_1 ... x_5$	ellipsis
`^`	a^n	superscript, power
`*`	a^*	in `^*`, for Kleene star
`+`	a^+	in `^+`, for Kleene plus
`_`	a_b	subscript, index
`%`	%`foo`	start of LaTeX comment
`\forall \exists`	$\forall\ \exists$	quantifiers
`\in`	$a \in T$	type definition
`\not`	$\neq$	negation of Boolean operators
`\ldots`	$\ldots$	ellipsis
`\cdots`	$\cdots$	ellipsis
`\frac`	$\frac{i}{i+1}$	rational fractions

The commands

```
<sp> <tab> <newline> ~ \quad
\<sp> \<tab> \<newline> \! \, \: \; \\
```

are layout. They are ignored by Teckel, except that they separate lexical items that would otherwise glue together.

Additionally the Teckel command `\Tcont` is supposed to produce only layout and continue the text in the next line, with the proper indentation, in LaTeX; it is ignored by Teckel.

4.3.2 Numbers

Numbers are in decimal and may contain a decimal point, for example `3.16`, which represents $\frac{316}{100} = \frac{79}{25}$. They may also be presented as fractions: `\frac{355}{133}`, which prints as $\frac{355}{133}$.

4.3.3 Identifiers

Identifiers identify atoms, constants, variables, operators, and functions. They are represented by single letters, a few other characters, and LaTeX commands. A LaTeX command can have parameters, which are then an integral part of the identifier; it cannot have optional (`[...]`) parameters. Identifiers can be extended with postfixed primes (`'`). So, `a`, `A`, `\rho`, `@`, `\clubsuit`, `\mathbb{N}`, and `\stackrel{a'}{\rightarrow}` are identifiers, and print as a, A, ρ, @, ♣, $\mathbb{N}$, and $\stackrel{a'}{\rightarrow}$, respectively. The user can use any identifier not predefined in Teckel to identify any of the above classes.

Since LaTeX considers all letters in math text separately, Teckel does the same: `xyz` is a sequence of three identifiers, `x`, `y`, and `z`, separated by the

invisible juxtaposition operator. To obtain a more complex identifier a LaTeX command can be used: `\mathit{Rhs}` is a single identifier *Rhs*. The result does not have to be alphanumeric; there would be no way to check that anyway. So `\rightarrow` is a perfectly good identifier, and if it is not defined otherwise it is an atom and denotes itself. And then $S \rightarrow Sx$ is a sequence of four identifiers.

Teckel has no predefined atoms, variables or functions, but it has predefined constants and operators.

Teckel knows the following zeroadic operators (constants):

Input	Prints as	Meaning
`\emptyset`	$\emptyset$	an empty set of any type
`\varepsilon`	ε	an empty sequence of any type
`\mathbb{N}`	$\mathbb{N}$	the set of natural numbers
`\mathbb{Z}`	$\mathbb{Z}$	nonnegative numbers
`\infty`	∞	infinity

Other zeroadic operators in LaTeX are (Table 16)[1]:

$\aleph$ $\clubsuit$ $\diamondsuit$ ℓ $\hbar$ $\heartsuit$ $\imath$ $\jmath$ $\spadesuit$

Teckel knows the following monadic prefix operators:

`+`	$+$	no arithmetic operation
`-`	$-$	arithmetic inversion
`\neg`	$\neg$	Boolean negation

Other monadic prefix operators in LaTeX are (Table 16):

$\Im$ ∇ $\Re$ $\wp$

Teckel knows the following monadic postfix operator:

`!`	!	factorial

Other monadic postfix operators in LaTeX:

$^\circ$

Teckel knows the following dyadic operators:$^\circ$ $^\circ$ mod , % ?

`+`	$+$	arithmetic addition
`-`	$-$	arithmetic subtraction
`\times`	$\times$	arithmetic multiplication
`/`	$/$	arithmetic division
`\div`	$\div$	arithmetic division ????
`\bmod`	mod	arithmetic modulo ????
`\%`	%	arithmetic modulo
`\circ`	$\circ$	relation composition
`\cap`	$\cap$	set intersection
`\cup`	$\cup$	set union
`\setminus`	$\setminus$	set difference
`\lor \vee`	$\vee$	Boolean or
`\land \wedge`	$\wedge$	Boolean and

[1] The table numbers refer to *The Comprehensive LaTeX Symbol List* by Scott Pakin, David Carlisle, and Alexander Holt (2001)

Multiplication can also be expressed by juxtaposition: $2x$ means 2 times x. The factors must come in the order

`number`? `identifier`* `parenthesized_expression`*

So $2xy(y+1)$ is allowed, $2x(y+1)y$ is not.

Other dyadic operators in LaTeX are (Table 12):

$\amalg * \bigcirc \bigtriangledown \bigtriangleup \bullet \cdot \dagger \ddagger \diamond \lhd \mp \odot \ominus \oplus \oslash \otimes \pm \sqcap \sqcup \star \triangleleft \triangleright \uplus \wr \angle \triangle$

Teckel knows the following built-in relational operators:

`<`	$<$	
`=`	$=$	
`>`	$>$	
`\leq`	$\leq$	
`\geq`	$\geq$	
`\neq`	$\neq$	
`\in`	$\in$	set membership
`\ni`	$\ni$	inverse set membership
`\subset`	$\subset$	
`\subseteq`	$\subseteq$	
`\supset`	$\supset$	superset
`\supseteq`	$\supseteq$	

Teckel knows the following user-definable relational operators (Table 13):

$\approx \asymp \bowtie \cong \dashv \doteq \equiv \frown \gg \ll \mid \models \parallel \perp$

$\prec \preceq \propto \sim \simeq \smile \sqsubseteq \sqsupseteq \succ \succeq \vdash$

Teckel knows the following repetition operators:

`\bigcup`	$\bigcup$	repeated set union
`\bigvee`	$\bigvee$	repeated Boolean or
`\bigwedge`	$\bigwedge$	repeated Boolean and
`\prod`	$\prod$	repeated multiplication
`\sum`	$\sum$	repeated addition

Other big operators in LaTeX are (Table 17):

$$\bigcap \bigodot \bigoplus \bigotimes \bigsqcup \biguplus \coprod \int \oint$$

but currently Teckel has no way of using them.

Other miscellaneous symbols in LaTeX (Table 16):

$\bot \flat \natural \partial \prime \sharp \surd \top$

The symbol `\backslash` is indistinguishable from `\setminus` and is treated like the latter.

4.3.4 Other Symbols

The above leaves a number of possible symbols undefined. They can be classified as follows.

The following symbols draw compile-time error messages from Teckel for various reasons, mainly because they are forbidden in math mode or do not print.

`#`	–	forbidden in math mode
`$`	–	math delimiter
`&`	–	forbidden in math mode
`;`	–	too similar to a statement separator
`\`	–	LaTeX command former
`\(`	–	math delimiter
`\)`	–	math delimiter
`\[`	–	math delimiter
`\]`	–	math delimiter
`\"`	–	accent, (redefinable)
`\'`	–	accent, (redefinable)
`\.`	–	accent, (redefinable)
`\=`	–	accent, tabbing command, (redefinable)
`\^`	–	accent, (redefinable)
`\‘`	–	accent, (redefinable)
`\~`	–	accent, (redefinable)
`\+`	–	tabbing command, (definable)
`\-`	–	tabbing command, hyphenation
`\<`	–	tabbing command, (definable)
`\>`	–	tabbing command, (redefinable)
`\@`	–	misc., (redefinable)
`\/`	–	??, (redefinable)
`\*`	–	??, (redefinable)
`\0`		
⋮	–	(definable)
`\9`		
`\?`	–	(definable)

Although many of the tabbing and accents commands are definable or redefinable, that feature is undocumented and Teckel rejects them.

The remaining 9 symbols are specified below; at present they are just identifiers.˚ They have no obvious uses, but some are suggested in the table. Each can be defined freely, for example as atom, prefix/infix operator, or postfix operator; like any identifier they can even be used as variable names, but it seems unwise to do so.

˚ is that wise? Yes. Implement

Symbol	prints as	possible use
"	”	doubtful
@	@	
‘	‘	
\#	#	length operator?
\$	$	constant?
\%	%	percentage operator?
\&	&	
_	_	atom?
|	‖	Boolean infix operator?

4.3.5 LaTeX syntax commands

There are a large number of LaTeX commands that serve to control its syntax rather than to produce output. Examples are `\begin`, `\newlength`, `\section`, etc. With the exception of the layout commands below, Teckel accepts them as identifiers, but using them in that way will almost certainly draw LaTeX errors.

The following LaTeX layout commands are ignored by Teckel:

```
~
%...
\quad
\begin{...}
\end{...}
\hspace{...}
\hspace*{...}
\vspace{...}
\vspace*{...}
```

If required, other LaTeX forms can be ignored by defining them away using the macro mechanism or by surrounding them with `%Teckel off` and `%Teckel on` lines.

4.4 Syntax

The context-free grammar of Teckel is shown in the appendix. It is neither the union nor the intersection of all features of abstract algorithms, nor is it innovative or very orthogonal. It rather represents a middle-of-the-road view of abstract algorithms. For example, although arrays can be indexed using subscripts (a_i) or square brackets ($a[i]$), functions can use subscripts only: $F_j(a)$ is a function call, but $F[j](a)$ is not accepted, in accordance with normal usage.˚

˚ check implementation

4.4.1 Teckel commands

The syntax structure of Teckel is defined by the following LaTeX commands.

```
\Tdefine{definee}{defining expression}
\Tassign[optional operator]{destination}{source}
\Tfunction{head}{parameters}{body}

\Tif{condition}{corresponding action}
```

```
\Telseif{condition}{corresponding action}
\Telse{default action}
\Tendif

\Tfor{generator}{action}
\Tendfor

\Twhile{condition}{action}
\Tendwhile

\Treturn{expression}
\Toutput{expression}
\Tcomment{comment text}

\Ttrue
\Tfalse

\Tsc
\Tcont
```

The parameterless commands `\Tendif`, `\Tendfor`, and `\Tendwhile` serve to allow pretty-printing packages to print block closers. They are required syntactically by Teckel but otherwise ignored.

A comment is ignored by Teckel and may contain any valid LaTeX text.

˚

˚ TODO: Describe commands

4.4.2 Operator and function definitions

Definitions can be zeroadic, prefix, postfix and infix, with 0, 1, 1, and 2, operands, respectively. Zeroadic operator definitions are equivalent to constant definitions. Examples are:

$$S \equiv \{1..5\}$$
$$\xi\ a \equiv ...$$
$$a\ \ddagger \equiv ..$$
$$a\ \sigma\ b,\ b \in \{0..4\} \equiv ...$$

which define S, ξ, $\ddagger$, and σ as zeroadic, prefix, postfix, and infix operators, respectively.

Formal parameters must be single letters; an operator cannot be a single letter, unless it is a zeroadic operator.˚ This seems to be normal practice and allows us to distinguish between prefix/infix and postfix operator definitions.

˚ update

Formal parameters allow pattern matching:

$$\textbf{function}\ F(n \in \mathbb{N}, aS, a \in T, S \in T^*):$$
$$\quad \textbf{return}\ a\ \textbf{if}\ |S| = n - 1$$

declares F as a function of two parameters. A call of F is only activated if the second argument can be split in two parts, a which must be an element of T and S which must be a sequence of Ts. It then yields a only if the length of S is $n - 1$. If any of the above fails, the result is `NO_RESULT`.

4.4.3 Subscripts as arguments

° ° write

ZZ

4.4.4 Restrictions

There are a few syntactic restrictions which are not reflected in the grammar in the appendix:

0. No restrictions at the moment.

4.5 Macros

The user may have reason to use other symbols for standard purposes than Teckel wants to see. For example, the user may decide that the set difference operator `\setminus`, which prints as $\setminus$, is too thin, and needs to be replaced by `\mathbf{\setminus}`, which prints as $\mathbf{\setminus}$. The macro directive

```
%Teckel macro \mathbd{\setminus} \setminus
```

instructs Teckel to replace all occurrences of `\mathbd{\setminus}` by `\setminus` upon reading the program.

The name to be defined can be any identifier in the sense of Section 4.3; it is replaced by the sequence of symbols that follow it. Possible error messages are given in terms of the replaced result.

5 Output

The basic mode of output of a Teckel program is specified by a Teckel output command:

```
\Toutput{SORT(3,2,1,2)}
```

The output command has an optional integer parameter, limiting the number of elements of a possibly infinite set the user wants to see:

```
\Toutput[3]{Q \cap T}
```

using the definitions from Section 2.1, prints $\{ 0, 1, 64, \ldots \}$. Note that the dots may represent zero elements, since Teckel cannot easily know if there are any more elements to come. An output limit of 0 is ignored.

It is possible to have more than one output command in a Teckel program. Such commands are processed sequentially, and the results are identified by the file name and line number of the LaTeX source file.

6 Why Teckel?

- The first step in the development of a non-trivial algorithm is often an abstract algorithm, a non-deterministic algorithm based on sets, often infinite ones. At present the testing and experimental evaluation of such an algorithm is either impossible or requires substantial programming efforts. It would be convenient to be able to run such programs directly.

- We may presently have the technology to write an interpreter for such abstract programs: lazy evaluation, eager testing, memoization. Whether this is really sufficient remains to be seen.
- Any attempt to write abstract algorithms in the usual mode of program expression –ASCII 96– leads to a very ugly and unappetizing representation, making working with them unattractive. The advent of LaTeX has largely removed this obstacle:
  - LaTeX provides standard notations for standard abstract algorithm concepts, for example `\bigcup_{1 \leq i \leq n}` for
    $$\bigcup_{1 \leq i \leq n}$$
  - Most papers containing algorithms are written in LaTeX these days, so expressing algorithms in it will be natural to users.
- It would be useful and fun to have a way to run non-deterministic programs.
- I probably have both the time and the expertise to write such an interpreter.

7 Why "teckel"?

A teckel (or "dachshund" in English-speaking countries) is a low-profile, very persistent animal that is quite capable of doing both breadth-first and depth-first search of its environment.

The obvious name for a programming language using the LaTeX format would be "Texol". But there are already several firms using that name in one way or another, and in a TeX environment it would soon be pronounced "teckel" anyway.

8 Teckel and the World

There is some similarity to Lamport's *PlusCal*.

¡to be written¿

9 Experience

10 Notes