md2LaTeX specifications in TLA+

Gabriel Cordier
admin@gcordier.eu

December 2, 2020

Contents

1 Overview **2**

2 What specifying means **2**
2.a Our story of the universe 2
2.b Pitching the script 2

3 Definitions **2**
3.a Mapping 2
3.b The user's scope 3
3.c The axiom of the user's choice 3
3.d Dictionaries, Maps, Records 3
3.e Keys 3
3.f Some improvised algebra 3

4 Around the specifications **4**

5 The specifications **4**

6 Aknowledgments **16**

Bibliography **16**

1 Overview

`md2LaTex` (this current *forked* version) needs two inputs:

1. A `.md` file, of course;
2. A preferences file (in practice, a `.json`) that encodes the user's choice (titlepage, table of contents, language, graphic design,...).

This document specifies the format of the preferences file.

2 What specifying means

2.a Our story of the universe

According to Leslie Lamport, in [1],

> *A specification does not describe the correct behavior of a system. Rather, it describes a universe in which the system and its environments are behaving correctly.*

So, all we have to do is falling into the right universe! How to figure it out? Maybe a short tale will help us to grasp all that abstraction...

2.b Pitching the script

Here is the script: The user is expected to submit a form, as we are expected to read it. Every field of the form may be filled by the user. This is how we know about the user's *choice*. In practice, the submitted form is encoded as a preferences file. And from our cave, we see nothing but shadows (preferences files). Implementing the user's will is the only pupose we can serve. Who the user is is off scope and we only assume that all user's inputs are "choices". Hence, we do not want to ensure that they are actually intentional or consistent: We only deal with syntax, not with semantics.

3 Definitions

3.a Mapping

A function $\texttt{f}[\texttt{x} \in \texttt{X}]$ is a *mapping iff* $\texttt{DOMAIN f} = \texttt{X}$. In `TLA+`, **functions are actually mappings**. For instance,

$$\texttt{record} = [\texttt{fileA} \mapsto 1729] \tag{1}$$

is not in the records set

$$[\texttt{fileA} : \texttt{HashSpace}, \texttt{fileB} : \texttt{HashSpace}], \tag{2}$$

since we do not know what record.fileB is.

3.b The user's scope

Since we deal with forms, we define the user's *scope* as a sequence

$$(\texttt{Scope}_1, \ldots \texttt{Scope}_i, \ldots)\,. \tag{3}$$

Formally, you may define a form field as

$$\texttt{Field}^{(i)} \triangleq \{\texttt{Scope}_i\}. \tag{4}$$

Remark that $\texttt{Field}^{(i)}$ is nonempty. From now on, we only consider $\texttt{Scope}_i$.

3.c The axiom of the user's choice

Statement. From the user's point of view, any element of

$$\texttt{Possible} \triangleq \texttt{Scope}_1 \times \cdots \times \texttt{Scope}_i \times \cdots \tag{5}$$

is a *possible world*. In other words, whatsoever the possible world you consider, **someday** a user **will** *reach* it.

QA version. There is no "We don't check that because it's unlikely to happen".

A choice function. Hence, a user's choice is a *choice function* U that satisfies

$$\texttt{choice}_i \triangleq \texttt{U}[\texttt{Scope}_i] \in \texttt{Scope}_i \tag{6}$$

whenever $\texttt{Possible} \neq \emptyset$.

3.d Dictionaries, Maps, Records

TLA+ defines *records* as the special mappings

$$[\texttt{keyAsStringA} \mapsto \ldots, \texttt{keyAsStringB} \mapsto \ldots, \ldots] \tag{7}$$

It is quite sure that implementations of your favorite language already include such type as a standard, *e.g.* `Map` in Java, `dict` in Python. From now on, we will use "dictionary" to denote any implementation of records.

3.e Keys

We record the user's `choice` in a dictionary `choice_`. Each key of `choice_` is identified with a specific field, in the sense that

$$\texttt{choice\_}[\texttt{Key}^{(i)}] = \texttt{choice}_i. \tag{8}$$

3.f Some improvised algebra

It follows from (6, 8) that $\texttt{Key}^{(i)}, \texttt{Scope}_i$ satisfies

$$\texttt{Key}^{(i)}[\texttt{U}] = \texttt{U}[\texttt{Scope}_i], \tag{9}$$

for all `i`. Conversely, (9) defines the very format of the preferences file. Moreover, (9) can be stated with a *bra* and a *ket* , as follows,

$$\underbrace{\langle \texttt{U}|\texttt{Key}^{(\texttt{i})}\rangle}_{\texttt{recording}} = \underbrace{\langle \texttt{Scope}_\texttt{i}|\texttt{U}\rangle}_{\texttt{choice}}, \tag{10}$$

i.e

$$|\texttt{Key}^{(\texttt{i})}\rangle = \langle \texttt{Scope}_\texttt{i}| \, ; \tag{11}$$

which is a very pedantic way of saying that

i. The parsing scope should be the **minimal set** of all `write` $\texttt{Key}^{(\texttt{i})} : \texttt{Scope}_\texttt{i}$;

ii. There is no hidden dependency between the keys: We are not dealing with semantic, *e.g.* "Is that choice making sense?"

4 Around the specifications

NAME stands for the project's name. The naming convention (but it is off specs) is

- NAME.md.pdf: The pdf output, md stands for *main document*, see below;
- NAME.md.md:The markdown main document (recursive imports are allowed. You may think of it as the main function of a C program.
- NAME. preferences.json: the preferences file

and so on. This naming convention is followed in the `.tla` documents, and is nonoptional, *i.e* `md2latex` is not intented to work successfully whether you do not follow the standard.

5 The specifications

Following [2], from Amazon's point of view,

> *In `TLA+`, correctness properties and system designs are just steps on a ladder of abstraction, with correctness properties occupying higher levels, systems designs and algorithms in the middle, and executable code and hardware at the lower levels.*

I always beared that in mind as I was writing the specs. You may read them with your own mind on the ladder bars: Starting from the highest one and stepping down to the lowest.

The preferences must be an element of SET_OF_PREFERENCES.
They must be univoquely encoded in a preferences file.
Conversely, the format of such file is the mappings space
SET_OF_PREFERENCES.

The current module specifies the preferences file.
* *

* *
Since this module should be read first, we declare global naming
conventions:

NAME stands for the project's name.
The naming convention (but it is off specs) is
NAME.md.pdf: The pdf output, md stands for 'main document' -
think of it as a main function in a C programm.
NAME.md.md: The markdown main document (recursive imports are OK
NAME. preferences.json: the preferences file
and so on.
* *

* *
So, here is the specification of a file NAME.preferences.json .
Such a file must implement, or at least "follow", a specific policy,
that I named "YesOrNo".
Such a policy is istancied with
  DOMAIN_OF_PREFERENCES and
  SET_OF_PREFERENCES.

Further explanations below.

This file only exists for the sake of readability,
DOMAIN_OF_PREFERENCES and SET_OF_PREFERENCES should be regarded as
CONSTANTS.
* *

* *

The YesOrNo policy:
Goal: The very purpose of all that verbose is about implementing a
key -namely, "Y/N" - you can see as a switch on/off button.

Definitions:
'Yes' or 'No' is always about a sequence of logical action(s).
encoded as a dictionary value.
1. No: Means "no action"; which we define as follows:
    i. If you do something, then it is discarded.
    ii.If you announce something, then it is disregarded.
2. Saying "No": The current key is mapped to some value in JSON_NO.

3. Saying “Yes”:The current key is mapped to some value in JSON_YES.

Statements:
1. It is only Yes XOR No (see above definitions 2, 3).
2. You say it once, whith a single value.
3.1.1. If you do not say anything, then it is No.
3.1.2. If you say “emptyset” (None, NULL, ‘’, ...), then it is No.
3.1.3. If you say “No”, then it is No.
3.2. If you say “Yes”, then you do ’value for key′ right now.

Implementation
The “yes or no” key Y_N
(see Statement 1 for existence, Statement 2 for uniqueness)
is always the String “Y/N”.
Moreover, we expect you to actually do something relevant/nontrivial
This latter requirement cannot be implemented from a general case,
since:
(a) “relevant” and “nontrivial” are context-dependent.
(b) The context space is infinite.
A complementary approach is about defining
EXCLUDED_BY_YES_OR_NO_POLICY
as the minimal set of what is either trivial or irrelevant.
This set is not constructed ;) .
(In practice, EXCLUDED_BY_YES_OR_NO_POLICY should contain,
at least, boolean and numerical values).
Hence, we cannot guarantee that the YesOrNo policy is implemented.
But we can check that the policy is “followed”, in the sense that:
i. The policy is partially implemented and:
ii.If the provided content is actually relevant,
then the policy is (nonprovably) implemented.

* *

CONSTANTS PATH Any path
CONSTANTS
STRING_ALPH, The words of the alphabet {a, , z, A, , Z}.
STRING_ALPH_NONEMPTY, $\triangleq$ STRING_ALPH \ {‘’}
STRING_LATEX All LaTeX Markups/commands, including ‘’.

CONSTANT HTML_COLORS The set of of all dictionaries
{(k, v)} where every evaluation
\definecolor{k}{HTML}{v}
sets a HTML color in LaTeX.

CONSTANT NAT {0, 1, 2, }
CONSTANT EXCLUDED_BY_YES_OR_NO_POLICY

* *

JSON_X: Set of all possible encoding(s) of x in a JSON file.
“yes”, “on”, and “true” are synonyms;
“no”, “off”, and “false” are synomyms.

```
\* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
CONSTANTS JSON_BOOL, JSON_NO, JSON_YES, Y_N

The preferences as a mapping:
First, Domain:
\* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
A compliant set of preferences is a necessarily (⇒)
i. A mapping of domain DOMAIN_OF_PREFERENCES;
\* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
DOMAIN_OF_PREFERENCES ≜ {
   'documentclass',
   'import_packages',
   'fancy',              To overwrite default settings
   'import_titlepage',   make your own page
   'table_of_contents',
   'fonts',              XeLaTeX: Any font managed by the OS is OK.
   'colors',
   'language',           Encompasses all language-dependent settings.
   'custom',             Misc. settings, e.g. section color
   'foreword',
   'annex',
   'sources'             Additional path, e.g. /src/img
}
\* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
Bear in mind that it is all about MAPPINGS: The domain is rigid,
it is always the same DOMAIN_OF_PREFERENCES, which means that
every key from DOMAIN_OF_PREFERENCES must exist in the record.
Hence, they is NO OPTIONAL KEY.
\* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *

Next, the function space:
\* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
A compliant set of preferences is a necessarily (⇒)
i. A mapping of domain DOMAIN_OF_PREFERENCES, more specifically:
ii. An element of SET_OF_PREFERENCES, see below.
In practice, the converse of (ii) is not true, since we expect some
common sense, e.g. setting fonts.LARGE >fonts.large.
But common sense is off scope.
\* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *

SET_OF_PREFERENCES ≜ [
   documentclass:[
      class:STRING_ALPH_NONEMPTY,
      options:[
         paper_size :STRING_ALPH,
         draft_mode:{'draft', ''},
         titlepage:{'titlepage', 'notitlepage', ''}]],
   import_packages:[
      Y_N:JSON_BOOL,
```

```
    path:PATH],
  fancy:[
    Y_N:JSON_BOOL,
    path:{'NAME.fancy.tex', ''}],
  import_titlepage:[
    Y_N:JSON_BOOL,
    path:PATH],
  table_of_contents:[
    Y_N:JSON_BOOL,
    renewcommand:STRING_LATEX],
  fonts:[
   main:STRING_ALPH_NONEMPTY,
   fixed_width:STRING_ALPH_NONEMPTY,
   LARGE:NAT,
   Large:NAT],
  colors:[
    Y_N:JSON_BOOL,
    definition:HTML_COLORS],
  language:[
   main:STRING_ALPH_NONEMPTY,
   date:STRING_LATEX,
   page_numbering:STRING_ALPH,
   nameForTableOfContents:STRING_ALPH],
  custom: [
   section: [
    color:STRING_ALPH,
    renewcommand:STRING_LATEX],
   subsection: [
    renewcommand:STRING_LATEX]],
  foreword:[
   Y_N:JSON_BOOL,
   path:{'NAME.foreword.tex', ''}],
  annex:[
   Y_N:JSON_BOOL,
   section: [
    renewcommand:STRING_ALPH],
   path:{'NAME.annex.tex', ''}],
  sources:[
   root:{'./'},
   images:{'img'}]
]
```

MODULE Md2LaTeXCorrectnessPreferencesFile

EXTENDS Md2LaTeXCorrectness, FiniteSets

* *

The preferences are identified with a file NAME.preferences.json

isPreferencesFileCompliant keeps track of preferences compliance.

* *

VARIABLES preferences, isPreferencesFileCompliant

Convenient operators

$XOR(a,\ b) \triangleq (a \lor b) \land (\neg b \lor \neg a)$

* *

The YesOrNo policy (2nd part):

Test/action isFollowingYesOrNoPolicy(f):

Definition:

isFollowingYesOrNoPolicy(f) is TRUE if.f f follows YesOrNo.

We expect the atom f to be a "first-degree subrecord" of preferences

(e.g. documentclass $\mapsto$ ..., import_packages $\mapsto$...).

Conversely, preferences record is flat

(see SET_OF_PREFERENCES definition).

No recursive check, then.

* *

$$
\begin{array}{l}
isFollowingYesOrNoPolicy(f) \triangleq \\
\quad \text{IF } Y\_N \in \text{DOMAIN } f \qquad \text{the YesOrNo switch button} \\
\quad \text{THEN} \\
\qquad \land Cardinality(\text{DOMAIN } f) = 2 \qquad \text{See Statement 2} \\
\qquad \land XOR(f[Y\_N] \in JSON\_NO, \qquad \text{It is No} \\
\qquad\quad \land f[Y\_N] \in JSON\_YES \qquad \text{It is Yes, and we "do well":} \\
\qquad\quad \land \forall\, key \in (\text{DOMAIN } f) \setminus \{Y\_N\} : \\
\qquad\qquad f[key] \notin EXCLUDED\_BY\_YES\_OR\_NO\_POLICY) \\
\quad \text{ELSE FALSE}
\end{array}
$$

YesOrNo policy: END

* *

Either you want to implement YesOrNo (see above),

either you want to do something entirely different.

* *

$$
\begin{array}{l}
isCompatibleWithYesOrNoPolicy(f) \triangleq XOR(\\
\quad isFollowingYesOrNoPolicy(f), \\
\quad Y\_N \notin \text{DOMAIN } f)
\end{array}
$$

* *

isPreferencesFollowingSpec = TRUE if.f

preferences follow the specs.

* *

$isPreferencesFollowingSpec \triangleq$

First, only a specific range for the keys:

```
    ∧DOMAIN preferences ⊆DOMAIN_OF_PREFERENCES
        Next, every "subrecord" must be compatible with YesOrNo.
    ∧∀key ∈DOMAIN preferences :
        isCompatibleWithYesOrNoPolicy(preferences[key])

 ****************************************
 Remark: If it is YesOrNo, then it is optional,
 since you cannot turn off a mandatory feature.
 In other words, we have the following criterion:
 ****************************************
isOptional(record) ≜
    IF
   isFollowingYesOrNoPolicy(record)
     THEN TRUE
     ELSE FALSE

 ****************************************
 The isOptional term is clearly misleading... there is no contradiction
 with the fact that all keys are nonoptional. The isOptional operator
 is about the features themselves. TODO: (!) Make that clear.
 ****************************************

 ****************************************
 Initial state
 ****************************************
InitPreferences ≜
    ∧preferences ∈SET_OF_PREFERENCES

InitCorrectness ≜
    ∧InitPreferences

        If the user does not believe that the preferences file is OK,
        then, there is no process at all, (s)he just goes back to work:
        Of course, up to now, nothing has been proved:
    ∧isPreferencesFileCompliant = TRUE  Conjecture

 ****************************************
 Next step
 ****************************************
NextCorrectness ≜
    ∧isPreferencesFollowingSpec
    ∧isPreferencesFileCompliant′ =isPreferencesFollowingSpec
    ∧UNCHANGED preferences

 ****************************************
 Invariants
 ****************************************

 ****************************************
 Properties
 ****************************************
```

MODULE Md2LaTeXSystemDesign

EXTENDS Md2LaTeXCorrectnessPreferencesFile

* *

These specifications only deals with the preferences themselves
Hence, the way the system interacts with the input files should be
independently specified.
It's a TODO!

* *

$$\texttt{InitSystemDesign} \triangleq \texttt{InitCorrectness}$$

$$\texttt{NextSystemDesign} \triangleq \texttt{NextCorrectness}$$

MODULE Md2LaTeXAlgorithms

```
EXTENDS Md2LaTeXSystemDesign, Functions

 At run time / compile time, the preferences file is parsed,
 which yields a dictionary 'choice_'.
 We below specify the parsing process.
VARIABLE choice_

 If it is No, then no setting.
 So, current key is not part of choice_.

 First, filter:
relevantKeys ≜ {
   key ∈ DOMAIN preferences :
      ∨ key = 'documentclass'
      ∨ ∧isFollowingYesOrNoPolicy(preferences[key])
        ∧preferences[key][Y_N] ∉ JSON_NO
}

 Next, stir up:
parsing ≜ [key ∈ relevantKeys ↦preferences[key]]

 * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
 Initial state
 * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
InitAlgorithms ≜
   ∧InitSystemDesign
   ∧choice_ =preferences

 * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
 Next state
 * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
NextAlgorithms ≜
   ∧NextSystemDesign
   ∧choice_′ =parsing

 * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
 Invariant
 IsParsingOK = TRUE if.f the parsing outputs a dictionary that:
 i. is compatible with the YesOrNo policy, i.e every subrecord is so;
 ii.is 'lean', in the sense that no "turned off" option
    - see Md2LaTeXSystemDesign - keeps existing in the dictionary;
 iii. has no extra key, i.e. all key come from DOMAIN_OF_PREFERENCES
 * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *

IsParsingOK ≜
   ∨InitAlgorithms
   ∨ ∀key ∈ DOMAIN choice_ :
        ∧isCompatibleWithYesOrNoPolicy(preferences[key])
        ∧ XOR( either:
                 ∧¬isFollowingYesOrNoPolicy(preferences[key]),
                  either:
```

$\wedge$`isFollowingYesOrNoPolicy`(`preferences`[`key`])
$\wedge$`choice_`[`key`][`Y_N`] $\in$ `JSON_YES`)

It means some redundant boolean tests, but we do not care about, since
we are are not aiming at optimal predicate computation, only specifications!

MODULE Md2LaTeXSpecifications

```
EXTENDS Md2LaTeXAlgorithms

Init ≜ InitAlgorithms

Next ≜ NextAlgorithms

Spec ≜ Init ∧□[NextAlgorithms]⟨
   preferences,
   isPreferencesFileCompliant,
   choice_⟩
```

6 Aknowledgments

Many thanks to

- Kavin Yao, creator of the `md2latex`, who kindly added a link to my work in his `md2latex Readme.md`.

- Lepture, creator of the Python Markdown parser `mistune`, contributor to Kavin Yao's `md2latex`.

- Leslie Lamport, designer of LaTeX and `TLA+`. The `TLA+` typesetting was performed with the LaTeXpackage `tlatex`. `TLA+` and `tlatex` are documented by their designer in *Specifying Systems*.

References

[1] Leslie Lamport. *TLA, the hyperbook, video 8b*. 2017.

[2] Chris Newcombe Et Al. *How Amazon web services uses formal methods*. ACM, 2015.