md2LateX specifications in TLA+

Git Cordier
admin@gcordier.eu

November 26, 2020

Contents

1 Overview

md2LaTex (this "forked" version) needs two inputs: the `.md` file, of course, and to preferences file (in practice, a JSON) that encodes the user choices (titlepage, table of contents, language, graphic design,...). This document specifies format and parsing process of such a preferences file.

2 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* - think of it as a main function in a C programm.
- NAME.md.md:The markdown main document (recursive imports are allowed.
- NAME. preferences.json: the preferences file

and so on. This convention is followed in the `.tla` documents.

3 The specifications

Following the Use of Formal Methods at Amazon Web Services,

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

I always beared that in mind as I was writing the specs and you may read them as if your were standing on this ladder: Starting from the highest bar and stepping down to the lowest.

MODULE Md2LaTeXCorrectness

EXTENDS FiniteSets

CONSTANTS ANY, PATH Any object, any path

CONSTANTS

STRING_ALPH, The words of the latin alphabet

STRING_ALPH_NONEMPTY, $\triangleq$ STRING_ALPH \ {‘’}

STRING_LATEX All LaTeX Markups/commands, including ‘.

CONSTANT RECORD Any record

CONSTANT NAT Any integer 0, 1, 2, ...

The preferences define a unique record

CONSTANTS DOMAIN_OF_PREFERENCES, SET_OF_PREFERENCES

* *

“yes”, “on”, and “true” are synonyms;

“no”, “off”, and “false” are synomyms.

The way we express “yes”, “no” in a JSON file.

* *

CONSTANTS Y_N, JSON_YES, JSON_NO, EXCLUDED_BY_YES_OR_NO_POLICY

* *

The preferences are identified with a file ’preferences’.

In practice, this is a JSON file NAME.preferences.json

(see CONSTANT SET_OF_PREFERENCES),

even if no semantics push on that.

isPreferencesFileCompliant keeps track of preferences compliance.

* *

VARIABLES preferences, isPreferencesFileCompliant

Convenient operators.

Recall that Yes = True and that No = False

JSON_BOOL $\triangleq$ JSON_YES $\cup$ JSON_NO

XOR(a, b) $\triangleq$ $(a \vee b) \wedge (\neg b \vee \neg a)$

YesOrNo policy: BEGINNING

* *

So, here is the specification of a file NAME.preferences.json .

See CONSTANTS DOMAIN_OF_PREFERENCES, SET_OF_PREFERENCES;

or Md2LaTeXSystemDesignPreferencesFile

Such a file must implement, or at least “follow”, a specific policy,

that I named “YesOrNo”.

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:

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.

Statement:
1. It is Yes XOR No (see above definitions 2, 3).
1.1.1. If you do not say anything, then it is No.
1.1.2. If you say “emptyset” (None, NULL, ‘’, ...), then it is No.
1.1.3. If you say “No”, then it is No.
1.2. If you say “Yes”, then you do value of key right now.
4. You do not neither do nor say anything else.

Implementation
The “yes or no” key Y_N
(see Statement 1 for existence, Statement 4 for uniqueness)
is always the String “Y/N”.
Moreover, we expect you 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 countable but 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 value
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.

Test / action’isFollowingYesOrNoPolicy(f)′
We expect the atom f to be a “first-degree subrecord” of preferences
(documentclass $\mapsto$ ..., import_packages $\mapsto$ ..., and so on).
isFollowingYesOrNoPolicy(f) is TRUE if.f f follows YesOrNo.
* *

$$
\begin{aligned}
&\text{isFollowingYesOrNoPolicy(f)} \triangleq \\
&\quad \wedge \text{Y\_N} \in \text{DOMAIN f} && \text{the YesOrNo switch button} \\
&\quad \wedge \text{Cardinality}(\text{DOMAIN f}) = 2 && \text{See Statement 4} \\
&\quad \wedge \ \vee \quad \text{f[Y\_N]} \in \text{JSON\_NO} && \text{It is No} \\
&\qquad \vee \ \wedge \text{f[Y\_N]} \in \text{JSON\_YES} && \text{It is Yes, and we “do well”:} \\
&\qquad\quad \wedge \forall \text{ key} \in (\text{DOMAIN f}) \setminus \{\text{Y\_N}\} : \\
&\qquad\qquad \wedge \text{f[key]} \in \text{EXCLUDED\_BY\_YES\_OR\_NO\_POLICY}
\end{aligned}
$$

YesOrNo policy: END ———————————————————————————————

* *
Either you want to implement YesOrNo (see above),
either you want to do something entirely different.
* *

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

```
* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
isPreferencesFollowingSpec =TRUE if.f
preferences follow the specs.
* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
isPreferencesFollowingSpec ≜

    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

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

InitCorrectness ≜
  ∧InitPreferences

    IF  we do not believe that our current preferences file is legal,
    then, there is no process at all, we just go nack to work:
    Of course, up to now, nothing has been proved:
  ∧isPreferencesFileCompliant =TRUE

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

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

* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
Properties
* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
We can assume that our preferences comply with all policies:
Under the specs:
□[isPreferencesFileCompliant]_⟨
```

isPreferencesFileCompliant

⟩

Check with TLC must be OK.

I consider it as an invariant, even if it's not syntactically true,

since isPreferencesFileCompliantNAME variables are primed.

MODULE Md2LaTeXSystemDesign

EXTENDS Md2LaTeXCorrectness

* *

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

* *

$\text{InitSystemDesign} \triangleq \text{InitCorrectness}$

$\text{NextSystemDesign} \triangleq \text{NextCorrectness}$

MODULE Md2LaTeXAlgorithms

EXTENDS Md2LaTeXSystemDesign, Functions

At run time / compile time, the preferences file is parsed,
which yields a dictionary (in Python) / HashMap (in Java) object,
namely 'preferences_as_dict'.
We specify the parsing process.

VARIABLE preferences_as_dict

So that preference_[key] is the actual setting 'key':

$$\text{preference\_} \triangleq [\text{key} \in \text{DOMAIN preferences} \mapsto \text{preferences[key]}]$$

If it is No, then no setting.
So, current key is off preferences_as_dict.

First, filter:

$$\begin{array}{l}\text{relevantKeys} \triangleq \{ \\ \quad \text{key} \in \text{DOMAIN preferences} : \\ \qquad \lor\ \text{key} = \text{`documentclass'} \\ \qquad \lor\ \land \text{isFollowingYesOrNoPolicy(preferences[key])} \\ \qquad\quad \land \text{preferences[key][Y\_N]} \notin \text{JSON\_NO} \\ \}\end{array}$$

Next, stir up:

$$\text{parsing} \triangleq [\text{key} \in \text{relevantKeys} \mapsto \text{preferences[key]}]$$

* *
Initial state
* *

$$\begin{array}{l}\text{InitAlgorithms} \triangleq \\ \quad \land \text{InitSystemDesign} \\ \quad \land \text{preferences\_as\_dict} = \text{preferences}\end{array}$$

* *
Next state
* *

$$\begin{array}{l}\text{NextAlgorithms} \triangleq \\ \quad \land \text{NextSystemDesign} \\ \quad \land \text{preferences\_as\_dict}' = \text{parsing}\end{array}$$

* *
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
* *

* *
This is actually repeating what is done with Md2LaTeXSystemDesign,
but this time, it is an invariant!
* *

$$\begin{array}{l}\text{IsParsingOK} \triangleq \\ \quad \lor \text{InitAlgorithms}\end{array}$$

∨ ∀ key ∈ DOMAIN preferences_as_dict :
∧isCompatibleWithYesOrNoPolicy(preference_[key])
∧ XOR(either:
∧¬isFollowingYesOrNoPolicy(preference_[key]),
either:
∧isFollowingYesOrNoPolicy(preference_[key])
∧preferences_as_dict[key][Y_N] ∈ JSON_YES)

MODULE Md2LaTeXSpecifications

EXTENDS Md2LaTeXAlgorithms

$Init \triangleq InitAlgorithms$

$Next \triangleq NextAlgorithms$

$Spec \triangleq Init \wedge \Box[NextAlgorithms]\langle$
preferences,
isPreferencesFileCompliant,
preferences_as_dict$\rangle$

MODULE Md2LaTeXSystemDesignPreferencesFile

* *

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".
Further explanations in Md2LaTeXSystemDesignPreferences.

This file is only here for the sake of completeness;
DOMAIN_OF_PREFERENCES and SET_OF_PREFERENCES are currently set as CONSTANTS .

* *

* *

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 allowed
NAME. preferences.json: the preferences file
and so on.

* *

The preferences as a mapping:
First, Domain:

```
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
}
```

Next, the function space:

```
SET_OF_PREFRENCES ≜ [
    documentclass :[
        class :STRING_ALPH_NONEMPTY,
        options :[
            paper_size  :STRING_ALPH,
            draft_mode :{'draft', ''},
            titlepage :{'titlepage', 'notitlepage', ''}]],
    import_packages :[
        Y_N :JSON_BOOL,
        path :ANY],
    fancy :[
        Y_N :JSON_BOOL,
        path :{'NAME.fancy.tex', ''}],
```

```
    import_titlepage :[
        Y_N :JSON_BOOL,
        path :PATH],
    table_of_contents :[
        Y_N :JSON_NO ∪JSON_YES,
        renewcommand :STRING_LATEX],
    fonts :[
      main :STRING_ALPH_NONEMPTY,
      fixed_width :STRING_ALPH_NONEMPTY,
      LARGE :NAT,
      Large    :NAT],
     'colors' is a record of \definecolor{}{HTML}{}
    colors :[
        Y_N :JSON_BOOL,
        definition :RECORD],
    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 :{'$AME.foreword.tex', ''}],
    annex  :[
      Y_N :JSON_BOOL,
      section : [
        renewcommand :STRING_ALPH],
      path :{'NAME.annex.tex', ''}],
    sources :[
      root :{'./'},
      images :{'img'}]
]
```

4 Aknowledgments

- Lepture

- Kavin Yao

- The TLA typesetting was performed with the`tlatex` LaTeXpackage, written by Leslie Lamport; Documented by the author in *Specifying Systems*.