MODULE *Md2LaTeXCorrectness*

For now we only aim at checking the correctness of the *JSON Prefs*
entities are: user (singleton), the *JSONFile*, the checker

VARIABLE *entityState*

To be expressive:

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

$$SetOfEntityStates \triangleq [\ user : \{\text{"working"}, \text{"done"}\},\ prefs : \{\text{"not checked"}, \text{"checked"}\} \times \{\text{"compliant"}, \text{"not compliant"}\},\ checker : \{\text{"working"}, \text{"done"}\}]$$

$$InitCorrectness \triangleq \land\ entityState \in SetOfEntityStates$$

$NextCorrectness \triangleq$

checker is working:
checker simply achieves processing.

$$\land\ entityState.checker = \text{"working"}$$
$$\land\ entityState' = [entityState\ \text{EXCEPT}\ !.checker = \text{"done"}]$$

checker is done:
1. user is working: user achieves all current tasks

$$\lor\ \land\ entityState.user = \text{"working"}$$
$$\land\ entityState.checker = \text{"done"}$$
$$\land\ entityState' = [entityState\ \text{EXCEPT}\ !.user = \text{"done"}]$$

checker is done:
2. user is done, checker is done: user goes back to work

$$\lor\ \land\ entityState.user = \text{"done"}$$
$$\land\ entityState.checker = \text{"done"}$$
$$\land\ entityState' = [entityState\ \text{EXCEPT}\ !.user = \text{"working"}]$$

$$isDone \triangleq \land\ entityState.user = \text{"done"} \land\ entityState.checker = \text{"done"}$$

* Modification History
* Last modified *Wed Nov* 25 15:41:10 *CET* 2020 by *gcordier*
* Created Sun *Nov* 22 02:11:18 *CET* 2020 by *gcordier*

MODULE *Md2LaTeXSystemDesign*

EXTENDS *Md2LaTeXCorrectness*, *FiniteSets*

CONSTANTS *ANY*, *PATH* Any object, any path

CONSTANTS
    STRING_ALPH, The words of the latin alphabet
    STRING_ALPH_NONEMPTY, $\triangleq$ *STRING_ALPH* \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 ${}.*preferences.json*

   (see CONSTANT *SET_OF_PREFERENCES*),

even if no semantics push on that.

isPreferencesFileCompliant${} keep track of preferences compliance.

* *

VARIABLES *preferences*,
    isPreferencesFileCompliantConjectured,
    isPreferencesFileCompliantProved,
    isPreferencesFileCompliant

Convenient operator.

Recall that Yes = True and that No = False

$JSON\_BOOL \triangleq JSON\_YES \cup JSON\_NO$

YesOrNo policy: BEGINNING ————————————————————

* *

So, here is the specification of a file ${}.*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 $YesOrNo$.
* *

$$isFollowingYesOrNoPolicy(f) \triangleq$$

```
    ∧ $Y\_N \in \textsc{domain}\ f$                              the *YesOrNo* switch button
    ∧ $Cardinality(\textsc{domain}\ f) = 2$                      See Statement 4
    ∧   ∨      $f[Y\_N] \in JSON\_NO$                            It is No
        ∨   ∧ $f[Y\_N] \in JSON\_YES$                            It is Yes, and we "do well":
            ∧ $\forall\ key \in (\textsc{domain}\ f) \setminus \{Y\_N\}$ :
                ∧ $f[key] \in EXCLUDED\_BY\_YES\_OR\_NO\_POLICY$
```

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

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

```
$isCompatibleWithYesOrNoPolicy(f) \triangleq XOR($
    $isFollowingYesOrNoPolicy(f),$
    $Y\_N \notin \textsc{domain}\ f)$
```

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

```
$isPreferencesFollowingSpec \triangleq$
        First, only a specific range for the keys:
    ∧ $\textsc{domain}\ preferences \subseteq DOMAIN\_OF\_PREFERENCES$
        Next, every "subrecords" must be compatible with YesOrNo.
    ∧ $\forall\ key \in \textsc{domain}\ preferences$ :
        $isCompatibleWithYesOrNoPolicy(preferences[key])$
```

* *
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:
* *

```
$isOptional(record) \triangleq$
    IF
    $isFollowingYesOrNoPolicy(record)$
    THEN TRUE ELSE FALSE
```

* *
Initial state
* *

```
$InitPreferences \triangleq$
    ∧ $preferences \in SET\_OF\_PREFERENCES$

$InitSystemDesign \triangleq$
    ∧ $InitCorrectness$
```

$\land$ *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:

$\land$ *isPreferencesFileCompliantConjectured* = TRUE

Of course, up to now, nothing has been proved:

$\land$ *isPreferencesFileCompliantProved* = FALSE
$\land$ *isPreferencesFileCompliant* = TRUE

* *
Next step
* *

$NextSystemDesign \triangleq$
$\quad \land NextCorrectness$
$\quad \land isPreferencesFollowingSpec$
$\quad \land isPreferencesFileCompliantProved' = isPreferencesFollowingSpec$
$\quad \land isPreferencesFileCompliantConjectured' =$ FALSE
$\quad \land isPreferencesFileCompliant' = XOR($
$\qquad isPreferencesFileCompliantConjectured',$
$\qquad isPreferencesFileCompliantProved')$
$\quad \land$ UNCHANGED *preferences*

* *
Invariants
* *

* *
Properties
* *
We can assume that our preferences comply with all policies:
Under the specs:
$\Box[isPreferencesFileCompliant]_{-}\langle$
$\quad isPreferencesFileCompliantConjectured,$
$\quad isPreferencesFileCompliantProved$
$\rangle$
Check with *TLC* must be *OK*.
I consider it as an invariant, even if it's not syntactically true,
since *isPreferencesFileCompliant*\${} variables are primed.

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 $preferences\_as\_dict$

If it is No, then no setting.
So, current key is off $preferences\_as\_dict$.

First, filter:

```
filteredKeys ≜ {
    key ∈ DOMAIN preferences :
        ∧ isFollowingYesOrNoPolicy(preferences[key])
        ∧ preferences[key][Y_N] ∉ JSON_NO
}
```

Next, stir up:

$parsing \triangleq [key \in filteredKeys \mapsto preferences[key]]$

Initial state

```
InitAlgorithms ≜
    ∧ InitSystemDesign
    ∧ preferences_as_dict = preferences
```

Next state

```
NextAlgorithms ≜
    ∧  NextSystemDesign
    ∧ preferences_as_dict′ = parsing
```

```
MODULE Md2LaTeXSpecifications
EXTENDS Md2LaTeXAlgorithms

Init  ≜ InitAlgorithms

Next  ≜ NextAlgorithms


Spec  ≜ Init ∧ □[NextAlgorithms]⟨
    entityState,
    preferences,
    isPreferencesFileCompliantConjectured,
    isPreferencesFileCompliantProved,
    isPreferencesFileCompliant,
    preferences_as_dict⟩
```

MODULE *Md2LaTeXSystemDesignPreferencesFile*

```
* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
So, here is the specification of a file ${}.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 .
* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
The preferences as a mapping:
First, Domain:
DOMAIN_OF_PREFERENCES ≜ {
    “documentclass”,
    “import_packages”,
    “fancy”,
    “import_titlepage”,
    “table_of_contents”,
    “fonts”,
    “colors”,
    “language”,
    “custom”,
    “foreword”,
    “annex”,
    “sources”,
}

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 : {“${}.fancy.tex”, “”}],
    import_titlepage : [
        Y_N : JSON_BOOL,
        path : PATH
    ],
    table_of_contents : [
        Y_N : JSON_NO ∪ JSON_YES,
        renewcommand : STRING_LATEX],
    fonts : [
      main : STRING_ALPH_NONEMPTY,
      fixed_width : STRING_ALPH_NONEMPTY,
      LARGE : NAT,
      Large     : NAT],
     'colors' is a record of (key, value) pa\definecolor{'s}{HTML}{'s}
    colors : [
        Y_N : JSON_BOOL,
        definition : RECORD],
    language : [
      main : STRING_ALPH_NONEMPTY,
      date   : STRING_LATEX,
      page_numbering : STRING_ALPH,
      nameForTableOfContents : STRING_ALPH],
    custom :   [
      section :   [
        color : STRING_ALPH,
        renewcommand : STRING_LATEX],
      subsection :   [
        renewcommand : STRING_LATEX]],
    foreword : [
      Y_N : JSON_BOOL,
      path : {"${}.foreword.tex", ""}],
    annex   : [
      Y_N : JSON_BOOL,
      section :   [
        renewcommand : STRING_ALPH],
      path : {"${}.annex.tex", ""}],
    sources : [
      root : {"./"},
      images : {"img"}]
]
```