Approfondissement de nuSMV, nuSMV-a et PyBoolNet

1. Les sorties de nuSMV-a et de nuSMV avant la transformation par PyBoolNet :

La fonction qui permet d'afficher la réponse de NuSMV et de NuSMV-a dans PyBoolNet est **nusmv_handle**. En effet, lorsqu'on **a des états acceptant, PyBoolNet utilise NuSMV-a** et lorsqu'il n'y en a pas, **PyBoolNet utilise NuSMV**.

```python
def nusmv_handle(Command, Process, Output, Error, DisableCounterExamples, AcceptingStates):
    """
    **arguments**:
        * *Command* (list): list of commands that was used to call subprocess.Popen.
        It is only used for creating an error message if the process does not finish correctly.
        * *Process* (subprocess.Popen): the object returned by subprocess.Popen
        * *Output* (Popen.communicate): the object returned by Popen.communicate
        * *DisableCounterExamples* (bool): whether a counterexample should be returned if the query is false
        * *AcceptingStates* (bool): whether information about the accepting states should be returned

    **returns**:
        * *Answer* (bool): whether NuSMV returns "is true"
        * *CounterExample* (list): a counterexample if NuSMV returns "is false" and DisableCounterExamples==False.
        * *AcceptingStates* (dict): information about the accepting states, if *DisableAcceptingStates==False*.
    """

    if Process.returncode == 0:
        if Output.count('specification')>1:
            print('SMV file contains more than one CTL or LTL specification.')
            raise Exception

        if 'is false' in Output:
            answer = False
        elif 'is true' in Output:
            answer = True
        else:
            print(Output)
            print(Error)
            print('NuSMV output does not respond with "is false" or "is true".')
            raise Exception

    else:
        print(Output)
        print(Error)
        print('NuSMV did not respond with return code 0')
        print('command: %s'%' '.join(Command))
        raise Exception

    if DisableCounterExamples and not AcceptingStates:
        return answer
    result = [answer]

    if not DisableCounterExamples:
        counterex = output2counterexample(NuSMVOutput=Output)
        result.append(counterex)

    if AcceptingStates:
        accepting = output2acceptingstates(NuSMVOutput=Output)
        result.append(accepting)

    return tuple(result)
```

Cette fonction **ne modifie pas** la réponse de nuSMV (ou de nuSMV-a), elle se contente de la réécrire plus « proprement ».

```
CTLSPEC:              AG !(proc1.state = critical & proc2.state = critical)
INIT:                 !(semaphore | (proc1.state.1 | (proc1.state.0 | (proc2.state.1 | (proc2.state.
0)))))
INIT_SIZE:            1
ACCEPTING:            TRUE
ACCEPTING_SIZE:       32
INITACCEPTING:        !(semaphore | (proc1.state.1 | (proc1.state.0 | (proc2.state.1 | (proc2.state.
0)))))
INITACCEPTING_SIZE:   1
ANSWER:               TRUE
```

Réponse de NuSMV-a sur le sémaphore 1

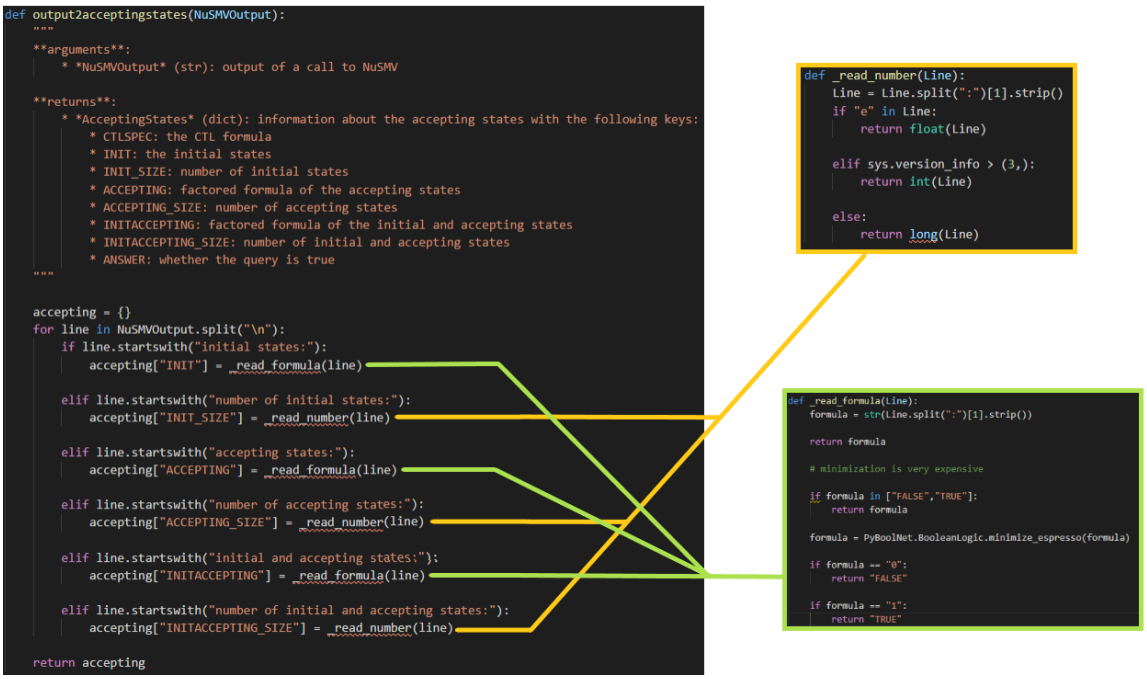
```
Test sur le fichier : semaphoreCTL1.smv

(True, {'INIT': '!(semaphore | (proc1.state.1 | (proc1.state.0 | (proc2.state.1
| (proc2.state.0))))))', 'INIT_SIZE': 1, 'ACCEPTING': '!(semaphore & (proc1.state
.0 & (proc2.state.0)) | !semaphore & (proc1.state.0 | (proc2.state.0)))', 'ACCEP
TING_SIZE': 16, 'INITACCEPTING': '!(semaphore | (proc1.state.1 | (proc1.state.0
| (proc2.state.1 | (proc2.state.0))))))', 'INITACCEPTING_SIZE': 1})
```

Réponse après nusmv_handle sur le sémaphore 1

On peut constater que les deux réponses sont les mêmes, simplement, grâce à PyBoolNet les réponses sont plus facilement compréhensibles par Python. Par exemple, on constate que le 'TRUE' de NuSMV-a, qui est considéré par Python comme une String, devient le True de Python.

Elle fait appel à deux autres fonctions de PyBoolNet : **output2acceptingstates** et **output2counterexample**.



```
def output2acceptingstates(NuSMVOutput):
    """
    **arguments**:
        * *NuSMVOutput* (str): output of a call to NuSMV

    **returns**:
        * *AcceptingStates* (dict): information about the accepting states with the following keys:
            * CTLSPEC: the CTL formula
            * INIT: the initial states
            * INIT_SIZE: number of initial states
            * ACCEPTING: factored formula of the accepting states
            * ACCEPTING_SIZE: number of accepting states
            * INITACCEPTING: factored formula of the initial and accepting states
            * INITACCEPTING_SIZE: number of initial and accepting states
            * ANSWER: whether the query is true
    """

    accepting = {}
    for line in NuSMVOutput.split("\n"):
        if line.startswith("initial states:"):
            accepting["INIT"] = _read_formula(line)

        elif line.startswith("number of initial states:"):
            accepting["INIT_SIZE"] = _read_number(line)

        elif line.startswith("accepting states:"):
            accepting["ACCEPTING"] = _read_formula(line)

        elif line.startswith("number of accepting states:"):
            accepting["ACCEPTING_SIZE"] = _read_number(line)

        elif line.startswith("initial and accepting states:"):
            accepting["INITACCEPTING"] = _read_formula(line)

        elif line.startswith("number of initial and accepting states:"):
            accepting["INITACCEPTING_SIZE"] = _read_number(line)

    return accepting
```

```
def _read_number(Line):
    Line = Line.split(":")[1].strip()
    if "e" in Line:
        return float(Line)

    elif sys.version_info > (3,):
        return int(Line)

    else:
        return long(Line)
```

```
def _read_formula(Line):
    formula = str(Line.split(":")[1].strip())

    return formula

    # minimization is very expensive

    if formula in ["FALSE","TRUE"]:
        return formula

    formula = PyBoolNet.BooleanLogic.minimize_espresso(formula)

    if formula == "0":
        return "FALSE"

    if formula == "1":
        return "TRUE"
```

La fonction **output2acceptingstates** ci-dessus se contente de réécrire les réponses de NuSMV-a en les rendant **réutilisable en Python**.

La fonction **output2couterexample** (ci-dessous) réécrit une partie de la réponse de NuSMV. Cette fonction fonctionne parfaitement sur un cas simple où **chaque état est libellé {TRUE, FALSE}**.

```
def output2counterexample(NuSMVOutput):
    """
        Converts the output of a NuSMV call into a sequence of states that proves that the query is false.

    **arguments**:
        * *NuSMVOutput* (str): output of a call to NuSMV

    **returns**:
        * *CounterExample* (list): a sequence of states that disprove the query.
    """

    if 'Trace Type: Counterexample' not in NuSMVOutput:
        return None

    assert(NuSMVOutput.count('Trace Type: Counterexample')==1)
    counterexample = []
    last_state = False

    blocks = NuSMVOutput.split('Trace Type: Counterexample')[1]

    blocks = blocks.split('-> ')

    for block in blocks:
        lines = block.split('\n')
        lines = [x.strip() for x in lines]
        lines = [x for x in lines if '=' in x]
        lines = [x for x in lines if '_IMAGE' not in x]
        lines = [x for x in lines if '_STEADY' not in x]
        lines = [x for x in lines if not x.startswith('SUCCESSORS')]
        lines = [x for x in lines if not x.startswith('STEADYSTATE')]
        lines = [x for x in lines if x!=[]]

        if lines:
            if last_state:
                state = last_state.copy()
            else:
                state = {}

            for line in lines:
                name, value = line.split(' = ')
                assert(value in ['TRUE','FALSE'])
                state[name] = 1 if value== 'TRUE' else 0

            counterexample.append(state)
            last_state = state

    return tuple(counterexample)
```

Mais dans un cas plus compliqué , comme celui ci-dessous, d'autres libellés sont possibles. Et alors, cette fonction ne peut pas le gérer. C'est ainsi qu'on arrive à l'erreur que nous avons obtenue lors des derniers essais. Nous arrivons donc à une **limite de PyBoolNet**.

```
-- specification AG (proc1.state = entering -> AF proc1.state = critical)  is
lse
-- as demonstrated by the following execution sequence
Trace Description: CTL Counterexample
Trace Type: Counterexample
  -> State: 1.1 <-
    semaphore = FALSE
    proc1.state = idle
    proc2.state = idle
  -> Input: 1.2 <-
    _process_selector_ = proc1
    running = FALSE
    proc2.running = FALSE
    proc1.running = TRUE
  -- Loop starts here
  -> State: 1.2 <-
    proc1.state = entering
  -> Input: 1.3 <-
    _process_selector_ = proc2
    proc2.running = TRUE
    proc1.running = FALSE
  -- Loop starts here
  -> State: 1.3 <-
  -> Input: 1.4 <-
  -> State: 1.4 <-
    proc2.state = entering
  -> Input: 1.5 <-
  -> State: 1.5 <-
    semaphore = TRUE
    proc2.state = critical
  -> Input: 1.6 <-
    _process_selector_ = proc1
    proc2.running = FALSE
    proc1.running = TRUE
  -> State: 1.6 <-
  -> Input: 1.7 <-
    _process_selector_ = proc2
    proc2.running = TRUE
    proc1.running = FALSE
  -> State: 1.7 <-
    proc2.state = exiting
  -> Input: 1.8 <-
  -> State: 1.8 <-
    semaphore = FALSE
    proc2.state = idle
```

Réponse de NuSMV sur le sémaphore 2

```
Traceback (most recent call last):
  File "off_test_lol.py", line 12, in <module>
    print(check_smv("semaphoreCTL2.smv"))
  File "off_test_lol.py", line 9, in check_smv
    return pm.check_smv_with_counterexample(FnameSMV)
  File "/usr/local/lib/python3.6/dist-packages/PyBoolNet/ModelChecking.py", line
 336, in check_smv_with_counterexample
    return nusmv_handle(cmd, proc, out, err, DisableCounterExamples=False, Accep
tingStates=False)
  File "/usr/local/lib/python3.6/dist-packages/PyBoolNet/ModelChecking.py", line
 747, in nusmv_handle
    counterex = output2counterexample(NuSMVOutput=Output)
  File "/usr/local/lib/python3.6/dist-packages/PyBoolNet/ModelChecking.py", line
 600, in output2counterexample
    assert(value in ['TRUE','FALSE'])
AssertionError
```

Erreur provoquée par nusmv_handle et plus précisément par output2counterexample sur le sémaphore 2

Ainsi, on arrive à la conclusion que récupérer la réponse de NuSMV ou de NuSMV-a avant la traduction par PyBoolNet peut être **intéressante dans les cas compliqués** que nusmv_handle n'arrive pas à gérer. Mais en **règle générale**, la traduction de PyBoolNet nous permet de **réutiliser la réponse en Python sans avoir à la modifier**.

2. Compréhension de nuSMV-a :

Tout d'abord, NuSMV-a est une librairie Python codée en C. Nous avons cherché à comprendre quel était son but, et comment elle fonctionnait.

Nous pensions que NuSMV-a avait pour but de **trouver les états acceptants d'un réseau de booléens**. Ne trouvant pas l'algorithme correspondant sur le GitHub de la librairie, nous avons pris la décision d'entrer en contact avec l'un de ses créateurs : le Professeur Hannes Klarner.

Nous avons obtenu la réponse suivante :

« Surprisingly, there is **no algorithm to find the accepting states**. The **accepting states are already computed by NuSMV**. The developers of NuSMV have decided to return True/ False for a CTL Query based on whether the initial states are contained in the accepting states or not. NuSMV already computes the accepting states. I think every CTL

model checking software computes accepting states. **My contribution is simply to return the accepting states to the user**. »

Voici la fonction utilisée pour récupérer, dans la réponse de NuSMV, les états acceptants. (https://github.com/hklarner/NuSMV-a/blob/master/NuSMV-2.6.0/NuSMV/code/nusmv/core/mc/mcAc.c)

```
void print_accepting_states(NuSMVEnv_ptr env, Prop_ptr prop, DDMgr_ptr dd, BddEnc_ptr enc,
          const OptsHandler_ptr opts, const StreamMgr_ptr streams, bdd_ptr init, bdd_ptr accepted)
{
  /* needed for printing to commandline/file */
  const MasterPrinter_ptr wffprint = MASTER_PRINTER(NuSMVEnv_get_value(env, ENV_WFF_PRINTER));
  OStream_ptr stream = StreamMgr_get_output_ostream(streams);
  FILE * out = StreamMgr_get_output_stream(streams);

  /* get ctlspec as char* */
  node_ptr p = Prop_get_expr(prop);
  const char* ctlspec = (char*) sprint_node(wffprint, p);

  /* get file name for output file */
  const char * file_name = get_print_accepting(opts);

  /* holds names of input variables */
  const char** inames;
  int lev;

  /* get index of ctlspec */
  int index_of_spec = Prop_get_index(prop);

  /* hold size of bdd's */
  double init_size, acc_size, inacc_size;

  /* get size of decision diagram */
  const int dd_size = dd_get_size(dd);

  /* compute initial and accepting states */
  bdd_ptr init_and_accepted = bdd_dup(init);
  bdd_and_accumulate(dd, &init_and_accepted, accepted);

  /* get number of initial, accepting, initial and accepting states */
  init_size = BddEnc_count_states_of_bdd(enc, init);
  acc_size = BddEnc_count_states_of_bdd(enc, accepted);
  inacc_size = BddEnc_count_states_of_bdd(enc, init_and_accepted);

  /* get input names */
  inames = ALLOC( const char*, dd_size);
  nusmv_assert((const char**) NULL != inames);

  for(lev = 0; lev < dd_size; ++lev) {
    int index = lev;

    if(BddEnc_has_var_at_index(enc, index)) {
      inames[lev] = (const char*) sprint_node(wffprint,BddEnc_get_var_name_from_index(enc, index));
    }
    else {
      inames[lev] = (const char*) NULL;
    }
  }
```

```
  /* if 'print' is specified as filename, write states to commandline */
  if(strcmp(file_name, "print") == 0) {
    OStream_printf(stream, "\ninitial states: ");
    Cudd_DumpFormula_modified(dd, 1, &init, inames, out);
    OStream_printf(stream, "\nnumber of initial states: %.20g", init_size);
    OStream_printf(stream, "\n");

    OStream_printf(stream, "accepting states: ");
    Cudd_DumpFormula_modified(dd, 1, &accepted, inames, out);
    OStream_printf(stream, "\nnumber of accepting states: %.20g", acc_size);
    OStream_printf(stream, "\n");

    OStream_printf(stream, "initial and accepting states: ");
    Cudd_DumpFormula_modified(dd, 1, &init_and_accepted, inames, out);
    OStream_printf(stream, "\nnumber of initial and accepting states: %.20g", inacc_size);
    OStream_printf(stream, "\n\n");
  }

  /* write to file, if filename is specified */
  else {
    if (index_of_spec != 0){
      /* append, if file already exists*/
      out = fopen(file_name, "a");
    }

    else{
      out = fopen(file_name, "w");
    }

    fprintf(out, "CTLSPEC:              ");
    fprintf(out, "%s", ctlspec);
    fprintf(out, "\n");

    fprintf(out, "INIT:                 ");
    Cudd_DumpFormula_modified(dd, 1, &init, inames, out);
    fprintf(out, "\nINIT_SIZE:            %.20g", init_size);
    fprintf(out, "\n");

    fprintf(out, "ACCEPTING:            ");
    Cudd_DumpFormula_modified(dd, 1, &accepted, inames, out);
    fprintf(out, "\nACCEPTING_SIZE:       %.20g", acc_size);
    fprintf(out, "\n");

    fprintf(out, "INITACCEPTING:        ");
    Cudd_DumpFormula_modified(dd, 1, &init_and_accepted, inames, out);
    fprintf(out, "\nINITACCEPTING_SIZE:   %.20g", inacc_size);
    fprintf(out, "\n");

    if (Prop_get_status(prop) == Prop_True) {
      fprintf(out, "ANSWER:               TRUE\n\n");
    }
    else{
      fprintf(out, "ANSWER:               FALSE\n\n");
    }
    fclose(out);
  }
```

Ainsi, **les fonctions que nous cherchions pour trouver les états acceptant se trouvent dans NuSMV**. Plus précisément, dans le package **mc** de NuSMV.

3. Utilisation de NuSMV-a sur des fichiers non-booléens :

Nous nous intéresserons ici à des fichiers contenant des **réseaux qui ne sont pas booléens** (True, False).

Lorsque nous avons appliqué la commande nusmv-a sur le fichier, cela semble marcher.

```
CTLSPEC:              (operon = 0 -> AG !(operon = 2))
INIT:                 TRUE
INIT_SIZE:            8
ACCEPTING:            TRUE
ACCEPTING_SIZE:       8
INITACCEPTING:        TRUE
INITACCEPTING_SIZE:   8
ANSWER:               TRUE

CTLSPEC:              (operon = 2 -> AG !(operon = 0))
INIT:                 TRUE
INIT_SIZE:            8
ACCEPTING:            TRUE
ACCEPTING_SIZE:       8
INITACCEPTING:        TRUE
INITACCEPTING_SIZE:   8
ANSWER:               TRUE
```

Réponse obtenue par la commande nusmv-a