NTCC model checker for Oz

The following is the specification of the functionality exposed in this module. Presently, there are 4 interfaces implemented: *Linear Time Logic formulae*, *NTCC processes*, *NTCC kripke structures* and *NTCC model checker*.

Usage

You need to include the module at the beginning of any code that uses it so *mozart* can direct the calls to the pre-compiled library:

```
functor
import NTCC at 'ntccModelChecker.ozf'
define
    Formula = {New NTCC.ltlFormula init( ...  )}
    Process = {New NTCC.ntccProcess init( ...  )  }
    MC = {New NTCC.ntccModelChecker init( Formula Process )
    Answer = {MC modelCheck( $ )}
end
```

Remember to place the `ozf`-extension library into the location specified after the `at` statement(in the above case the same directory as the oz code).

ltlFormula Interface

'`<=`' stands for default parameter value and '`$`' means the method produces an output.

`init`

```
            {NTCC.ltlFormula init( LTLFormula <= preposition(constraint:'x=0'#[x '=:'  0]))
)}
```

initializes the internal formula with the constructed formula *LTLFormula*. The value of *LTLFormula* is by default the constraint preposition $x = 0$, a constraint is constructed as a pair where the first element is the string representation of the constraint and the second is a triplet of the form [*variable relation value*]. The possible relations are `=:`, `\\=:`, `>:`, `<:`, `>=:` and `=<:`.

`disjunction`

```
{NTCC.ltlFormula disjunction( $ FormulaLeft<=@formula FormulaRight )}
```

outputs a disjunction($\vee$) between *FormulaLeft* and *FormulaRight*, both are constructed LTL formulas. *FormulaLeft* is by default the internal formula.

conjunction

```
{NTCC.ltlFormula conjunction( $ FormulaLeft<=@formula FormulaRight )}
```

outputs a conjunction($\wedge$) between *FormulaLeft* and *FormulaRight*, both are constructed LTL formulas. *FormulaLeft* is by default the internal formula.

implication

```
{NTCC.ltlFormula implication( $ FormulaLeft<=@formula FormulaRight )}
```

outputs an implication($\rightarrow$) between *FormulaLeft* and *FormulaRight*, both are constructed LTL formulas. *FormulaLeft* is by default the internal formula.

negation

```
{NTCC.ltlFormula negation( $ Formula<=@formula )}
```

outputs the negation($\neg$) of the constructed LTL formula *Formula*. The internal formula is the default value of *Formula*.

next

```
{NTCC.ltlFormula next( $ Formula<=@formula )}
```

outputs the constructed LTL formula *Formula* being true in the next time unit($\circ$). The default value of *Formula* is the internal formula.

eventually

```
{NTCC.ltlFormula eventually( $ Formula<=@formula )}
```

outputs the constructed LTL formula *Formula* being true eventually in the future($\Diamond$). The default value of *Formula* is the internal formula.

always

{NTCC.ltlFormula always($ *Formula<=@formula*)}

outputs the constructed LTL formula *Formula* being true in all present and future time units(□). The default value of *Formula* is the internal formula.

getFormula

{NTCC.ltlFormula getFormula($)}

outputs the current internal formula.

setFormula

{NTCC.ltlFormula setFormula(*Formula*)}

sets the internal formula to *Formula*.

showFormula

{NTCC.ltlFormula showFormula($ *Formula<=@formula*)}

outputs the constructed LTL formula *Formula* in a human readable form. The default value of *Formula* is the internal formula.

removeSquare

{NTCC.ltlFormula removeSquare($ *Formula<=@formula*)}

outputs the constructed LTL formula *Formula* with every *always* operators(□) removed. Remember that $\Box f \equiv \neg(\Diamond \neg f)$.

removeDisjunction

{NTCC.ltlFormula removeDisjunction($ *Formula<=@formula*)}

outputs the constructed LTL formula *Formula* with every *disjunction* operators(∨) removed. Remember that $f_1 \vee f_2 \equiv \neg(\neg f_1 \wedge \neg f_2)$.

removeImplication

{NTCC.ltlFormula removeImplication($ *Formula<=@formula*)}

outputs the constructed LTL formula *Formula* with every *implication* operators($\rightarrow$) removed. Remember that $f_1 \rightarrow f_2 \equiv \neg f_1 \vee f_2$.

equalFormulas

```
{NTCC.ltlFormula equalFormulas( $ Formula1 Formula2<=@formula Process<=nil )}
```

outputs whether *Formula1* is equal to *Formula2* by using the constraint entailment of the NTCC process *Process*. By default *Formula2* is the internal formula and if *Process* is `nil` then prepositions are compared by list equality.

calculateClosure

```
{NTCC.ltlFormula calculateClosure( $ Formula<=@formula )}
```

outputs the closure of the constructed LTL formula *Formula*. The closure are all the sub-formulas whose truth value can influence the truth value of the formula. By default *Formula* is the internal formula.

getClosure

```
{NTCC.ltlFormula getClosure( $ )}
```

outputs the internal closure.

setClosure

```
{NTCC.ltlFormula setClosure( Closure )}
```

sets the internal closure to *Closure*.

calculateBasicFormulas

```
{NTCC.ltlFormula calculateBasicFormulas( $ Closure<=@closure )}
```

outputs the basic formulas of the closure *Closure*. Basic formulas are formulas that are satisfied in a specific time unit, mainly prepositions and formulas inside *next* operators. The default value of *Closure* is the internal closure.

getBasicFormulas

```
{NTCC.ltlFormula getBasicFormulas( $ )}
```

outputs the internal basic formulas.

setBasicFormulas

{NTCC.ltlFormula setBasicFormulas(*BasicFormulas*)}

sets the internal basic formulas to *BasicFormulas*.

calculateCombinations

{NTCC.ltlFormula calculateCombinations(*$ Formulas*)}

outputs all possible combinations of the basic formulas *Formulas*.

getCombinations

{NTCC.ltlFormula getCombinations(*$*)}

outputs the internal combinations of some basic formulas.

setCombinations

{NTCC.ltlFormula setCombinations(*Combinations*)}

sets the internal combinations to *Combinations*.

prepositionInCombination

{NTCC.ltlFormula prepositionInCombination(*$ Preposition Combination Process*)}

outputs whether the preposition *Preposition* is entailed by the formulas in the list *Combination* using the constraint entailment of the NTCC process *Process*.

formulaConsistentCombination

{NTCC.ltlFormula formulaConsistentCombination(*$ Formula Combination Process*)}

outputs whether the formula *Formula* is consistent(including entailment of its prepositions) with the list of formulas *Combination* using the constraint entailment of the NTCC process *Process*.

formulaInCombination

{NTCC.ltlFormula formulaInCombination(*$ Formula Combination Process*)}

outputs whether the formula *Formula* is inside the list of formulas *Combination* using the constraint entailment of the NTCC process *Process*.

nonBasicFormulasInCombination

{NTCC.ltlFormula nonBasicFormulasInCombination(*$ Closure<=@closure Combinations<=@combinations Process*)}

outputs the list of formulas *Combinations* plus the non-basic formulas of the closure *Closure* that are consistent with all the formulas from the combination. It uses the constraint entailment of the NTCC process *Process*. By default, the values of *Closure* and *Combinations* are the internal closure and combinations respectively.

ntccProcess Interface

init

{NTCC.ntccProcess init(*Process<=tell([x ':' 0] Variables<=[x] Range<=0#1*)}

initializes the internal process with the constructed NTCC process *Process*. The default value of *Process* is telling the constraint $x = 0$. The constraint is a triplet with the same characteristics as the constraints in the *ltlFormula* interface.

when

{NTCC.ntccProcess when(*$ Constraint<=[x ':' 0] ProcessRight<=@process*)}

outputs the NTCC process *ProcessRight* conditioned by the constraint *Constraint*. *Constraint* defaults to the constraint $x = 0$ and *ProcessRight* defaults to the internal process.

unless

{NTCC.ntccProcess unless(*$ Constraint<=[x ':' 0] ProcessRight<=@process*)}

outputs the NTCC process *ProcessRight* conditioned by the absence of the constraint *Constraint*. *Constraint* defaults to the constraint $x = 0$ and *ProcessRight* defaults to the internal process.

parallel

```
{NTCC.ntccProcess parallel( $ Process<=@process Processes )}
```

outputs the NTCC process *Process* in parallel execution with the list of NTCC processes *Processes*. *Process* defaults to the internal process.

sumation

```
{NTCC.ntccProcess sumation( $ Process<=@process Processes )}
```

outputs the NTCC process of a non-deterministic choice between *Process* and the list of NTCC processes *Processes*. *Process* defaults to the internal process.

next

```
{NTCC.ntccProcess next( $ Process<=@process )}
```

outputs the NTCC process *Process* where its execution is delayed by one time unit. *Process* defaults to the internal process.

replication

```
{NTCC.ntccProcess replication( $ Process<=@process )}
```

outputs the NTCC process *Process* where its execution is replicated on every present and future time units. *Process* defaults to the internal process.

eventually

```
{NTCC.ntccProcess eventually( $ Process<=@process )}
```

outputs the NTCC process *Process* where its execution is delayed by a non-deterministic number of time units. *Process* defaults to the internal process.

getProcess

```
{NTCC.ntccProcess getProcess( $ )}
```

outputs the internal process.

setProcess

```
{NTCC.ntccProcess setProcess( Process )}
```

sets the internal process to *Process*.

showProcess

```
{NTCC.ntccProcess showProcess( $ Process<=@process )}
```

outputs the NTCC process *Process* in a human readable form. *Process* defaults to the internal process.

processToProcedure

```
{NTCC.ntccProcess processToProcedure( $ Process<=@process )}
```

outputs the NTCC process *Process* where all its constraints are transformed to Mozart/Oz procedures. *Process* defaults to the internal process.

constraintEntailment

```
{NTCC.ntccProcess constraintEntailment( $ C1 C2 NotEntailed<=false )}
```

outputs whether the constraint *C1* entails (or not entails, depending on the value of *NotEntailed*) the constraint *C2*. The constraints are in the same form of triplets mentioned above.

ntccStructure Interface

init

```
{NTCC.ntccStructure init( Process )}
```

initializes, creates, reduces and renames the kripke structure based off the NTCC process *Process*.

getStructure

```
{NTCC.ntccStructure getStructure( $ )}
```

outputs the internal structure.

setStructure

{NTCC.ntccStructure setStructure(*Structure*)}

sets the internal structure to *Structure*.

showStructure

{NTCC.ntccStructure showStructure(*Structure<=@structure*)}

prints in the standard output the structure *Structure*. By default the value of *Structure* is the internal structure.

createStructure

{NTCC.ntccStructure createStructure(*$ Process*)}

outputs the kripke structure based off the ntcc process *Process*.

reduceStructure

{NTCC.ntccStructure reduceStructure(*$ ExtendedStructure<=@structure*)}

outputs the minimal kripke structure(external transitions only) from the extended kripke structure *ExtendedStructure*. By default the value of *ExtendedStructure* is the internal structure.

renameStructure

{NTCC.ntccStructure renameStructure(*$ ReducedStructure<=@structure*)}

outputs the renamed(ordered nodes and edges) kripke structure from the reduced kripke structure *ReducedStructure*. By default the value of *ReducedStructure* is the internal structure.

ntccModelChecker Interface

init

{NTCC.ntccModelChecker init(*Formula Process*)}

initializes the model checker object with the LTL goal formula *Formula* and the NTCC base process *Process*.

`getFormula`

```
{NTCC.ntccModelChecker getFormula( $ )}
```

outputs the internal LTL goal formula.

`getProcess`

```
{NTCC.ntccModelChecker getProcess( $ )}
```

outputs the internal base NTCC process.

`setFormula`

```
{NTCC.ntccModelChecker setFormula( Formula )}
```

sets the internal LTL goal formula to *Formula*.

`setProcess`

```
{NTCC.ntccModelChecker setProcess( Process )}
```

sets the internal NTCC base process to *Process*.

`modelCheck`

```
{NTCC.ntccModelChecker modelCheck( $ Structure<=@structure
Formula<=@formula Process<=@process )}
```

outputs the model checking of the LTL goal formula *Formula* on the NTCC base process *Process* w.r.t the reduced and renamed kripke structure *Structure*. The default values of *Formula*, *Process* and *Structure* are the internal formula, process and structure. The result is a pair where the first element is `true` or `false` and the second element is a possible trace of the states(counter-example) if the model check bears false.