NAME

mcl, MCL – Model Checking Language version 5 (probabilistic value-passing modal mu-calculus)

DESCRIPTION

This manual page presents the version 5 of *MCL* (*Model Checking Language*), which is the temporal logic accepted as input by **evaluator5**(LOCAL). *MCL* version 5 conservatively extends *MCL* version 4 with a probabilistic operator specifying the probability measure of transition sequences characterized by regular formulas. In the remainder of this page, "*MCL*" denotes version 5 of *MCL*; see **mcl**(LOCAL) for other versions of *MCL*.

A description of the probabilistic operator of *MCL* can be found in article [MR18], which also presents the verification method implemented in version 5.0 of EVALUATOR. This method is based on translating the problem into the resolution of a Linear Equation System (LES) and a Parameterized Boolean Equation System (PBES), which are carried out simultaneously on the fly.

MCL formulas are interpreted over a PTS (Probabilistic Transition System) [LS91] *<S, A, T, P, s0>*, where: *S* is the set of *states*, *A* is the set of *actions* (transition labels), *T* is the *transition relation* (a subset of $S * A * S$), *P* is the *probability labeling* (a function from *T* to the range]0..1]), and *s0* is the *initial state*. A transition *(s1, a, s2)* of *T*, also written *s1-a->s2*, indicates that the system can move from state *s1* to state *s2* by performing action *a* with probability *P (s1, a, s2)*. For each state *s*, the sum of the probabilities associated to its outgoing transitions *(s, a, s')* is equal to 1. Note that this forbids the presence of sink states (deadlocks) in the PTS, i.e., every state must have at least one outgoing transition.

The PTS must contain only labelled probabilistic transitions of the form "*label***; prob** *%p*", where *%p* denotes a floating-point number in the range]0..1] (see the **bcg_steady**(LOCAL) manual page for details). An LTS can be converted into a PTS on the fly by renaming its transition labels using the **-rename** option of **evaluator5**(LOCAL) to append "**; prob** *%p*" suffixes. If after renaming transition labels, the sum of the probabilities associated to the transitions going out of a state is different from 1, these probabilities are normalized to bring this sum to 1. If no **-rename** option is given, the LTS is converted into a PTS by considering that, for every state, all its outgoing transitions have equal probabilities, the sum of which is 1.

PROBABILISTIC OPERATOR

The probabilistic operator belongs to the *state formulas* of MCL. Its syntax is defined by the following grammar:

"prob"
  R
"is"
  op ["?"] *E*
"end" "prob"

where *R* is a regular formula on transition sequences, *op* is a comparison operator among "<", "<=", ">", ">=", "=", "<>", and *E* is a data expression of type **`real`** denoting a probability.

Let *m (s, R)* be the probability measure of the transition sequences going out of a state *s* and having a prefix satisfying the regular formula *R*. A state *s* of the PTS satisfies the "prob" formula above iff *m (s, R) op v* holds, where *v* is the value of expression *E*.

If the optional "?" clause is present, the evaluation of the "prob" formula on a state *s* will also display the value *m (s, R)* in addition to the Boolean verdict (TRUE or FALSE) displayed by **evaluator5**(LOCAL).

The regular formula *R* and the expression *E* may contain occurrences of free data variables defined outside the "prob" formula. The data variables exported by *R* are neither visible in *E*, nor exported outside the whole "prob" formula.

EXAMPLES OF PROBABILISTIC PROPERTIES

DATALESS PROPERTIES

The "prob" operator expresses constraints on the probability of certain sequences described using regular formulas. The formula below specifies that the probability to send a message along an unreliable channel and receive it finally (possibly after a finite number of retries) is at least 90%:

```
prob
    SEND . (true* . RETRY)* . RECV
is
    >= 0.9
end prob
```

By combining the modalities of PDL and the "prob" operator, one can express quantitative response properties. The formula below specifies that every request to access a resource will be granted with probability 1 (i.e., almost surely):

```
[ true* . REQUEST ]
    prob
        true* . GRANT
    is
        >= 1
    end prob
```

DATA-HANDLING PROPERTIES

MCL enables the formulation of probabilistic properties involving data values, for instance using regular formulas with counting. The formula below, concerning the Bounded Retransmission Protocol (BRP) [DJJL01,MR18], evaluates the probability that the sender reports an unsuccessful transmission (action INPUT_NOK) after more than eight chunks of the packet have been transmitted (action REC_L):

```
prob
    ((not REC_L)* . REC_L) { 8 ... } .
    (not (REC_L or INPUT_NOK))* . INPUT_NOK
is
    >= ? 0
end prob
```

where the regular subformula "*R* { 8 ... }" denotes the repetition of *R* at least 8 times (see **mcl4**(LOCAL) for details). This formula will be evaluated to TRUE and **evaluator5**(LOCAL) will display the computed probability, as required by the "?" clause.

More complex properties about transition sequences having certain cumulated costs can be expressed using data-handling action predicates and the general iteration operator "loop" on regular formulas. The following example deals with a mutual exclusion protocol that manages the accesses of *n* processes *P1*, ..., *Pn* to a shared resource [MS13,MR18]. The MCL formula below computes the probability that a process *Pi* performs memory accesses of a total cost *max* before entering its critical section (*Pi* and *max* are assumed to be defined outside of the formula). The regular subformula expresses that, after executing its non critical section for the first time (action predicate "{ NCS !i }"), process *Pi* begins its entry section and, after a number of memory accesses (action predicate "{ MU ... ?c:nat !i }"), enters its critical section (action predicate "{ CS !"ENTER" !i }"):

```
prob
    (not { NCS !i })* . { NCS !i } .
    loop (total_cost:nat:=0) in
        (not ({ MU ... !i } or { CS !"ENTER" !i }))* .
        if total_cost < max then
            { MU ... ?c:nat !i } .
            continue (total_cost + c)
```

```
            else
                exit
            end if
        end loop .
        { CS !"ENTER" !i }
    is
        >= ? 0
    end prob
```

The "loop" subformula captures the entry section of *Pi* and requires that it terminates when the cost of all memory accesses performed by *Pi* (accumulated in the iteration variable *total_cost*) exceeds a given value *max*. The costs present on transitions are captured in the *c* variable of the action predicate "{ MU ... ?c:nat !i }" and used in the "continue" subformula to update the value of *total_cost*. The other processes can execute freely during the entry section of *Pi*, in particular they can overtake *Pi* by accessing their critical sections before it.

RELATION TO OTHER PROBABILISTIC LOGICS

The "prob" operator of MCL generalizes naturally the Until operators of classical probabilistic branching-time logics. The Until operator of PCTL (Probabilistic Computation Tree Logic) [HJ94] without discrete time can be expressed in MCL as follows:

[*F1* **U** *F2* **]{>=** *p***} =**
  prob
    (HOLDS (*F1***) . true)* . HOLDS (***F2***)**
  is
    >= *p*
  end prob

where "HOLDS (*F*)" is the testing operator of PDL (Propositional Dynamic Logic) [FL79], defined by the MCL regular formula below:

HOLDS (*F***) = if not (***F***) then false end if**

Similarly, probabilistic versions of the two Until operators of ACTL (Action-based Computation Tree Logic) [DV90] can be defined in MCL as follows:

[*F1***{***A1***}** **U** *F2* **]{>=** *p***} =**
  prob
    (HOLDS (*F1***) .** *A1***)* . HOLDS (***F2***)**
  is
    >= *p*
  end prob

[*F1***{***A1***}** **U{***A2***}** *F2* **]{>=** *p***} =**
  prob
    (HOLDS (*F1***) .** *A1***)* . HOLDS (***F1***) .** *A2* **. HOLDS (***F2***)**
  is
    >= *p*
  end prob

The full Until operator of PCTL, as well as its action-based counterparts derived from ACTL, can be expressed as follows (where *t* >= 0 is the number of ticks until *F2* holds):

[*F1* **U{<=** *t***}** *F2* **]{>=** *p***} =**
  prob

(HOLDS (*F1***) . true) { 0 ...** *t* **} . HOLDS (***F2***)**
is
>= *p*
end prob

[*F1*{*A1*} **U{<=** *t***}** *F2* **]{>=** *p***} =**
prob
(HOLDS (*F1***) .** *A1***) { 0 ...** *t* **} . HOLDS (***F2***)**
is
>= *p*
end prob

[*F1*{*A1*} **U{<=** *t***}{***A2***}** *F2* **]{>=** *p***} =**
prob
(HOLDS (*F1***) .** *A1***) { 0 ...** *t* **} .**
HOLDS (*F1***) .** *A2* **. HOLDS (***F2***)**
is
>= *p*
end prob

where the regular subformula "*R* { 0 ... *t* }" denotes interval counting, i.e., the repetition of *R* between 0 and *t* times (see **mcl4**(LOCAL) for details).

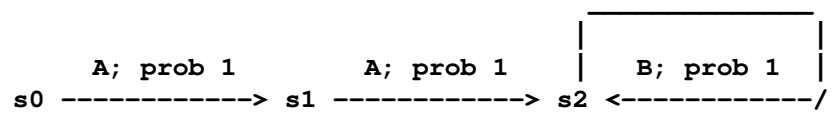
DETERMINIZATION OF PROBABILISTIC FORMULAS

To ensure that the verification of a "prob" formula on a PTS is translated correctly into the resolution of a LES, the regular subformula *R* must be *deterministic*, meaning that it must satisfy two conditions:

(a) if *R* matches a transition sequence in the PTS, it cannot match also one of its prefixes.

(b) if *R* matches a transition sequence in the PTS, it does this in a unique manner.

We illustrate these two conditions by considering the following PTS:



For condition (a), consider the following formula:

```
prob
    true* . A
is
    >= 0
end prob
```

The regular subformula "true* . A" is nondeterministic, since it matches both the sequence s0--A-->s1--A-->s2 and its proper prefix s0--A-->s1. A deterministic version of this regular subformula is "(not A)* . A".

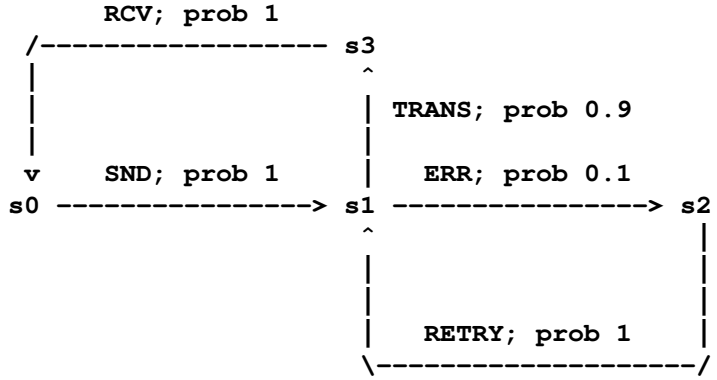
For condition (b), consider the following formula:

```
prob
    A* . (not B)* . B
is
    >= 0
end prob
```

The regular subformula "A* . (not B)* . B" is nondeterministic, since it matches the sequence s0--A-->s1--A-->s2--B-->s2 both as "A* . B" and as "(not B)* . B". A deterministic version of this regular subformula is "A* . (not (A or B))* . B".

The **evaluator5**(LOCAL) model checker determinizes automatically the dataless formulas, and thus the two formulas above are correctly handled. However, the tool in its current version does not determinize data-handling formulas, but only detects the presence of nondeterminism and signals it by a specific warning message. It is the user's responsibility to constrain the regular subformulas of "prob" operators so as to remove all warning messages concerning nondeterminism, and thus to ensure the correctness of verification.

To illustrate the warning messages concerning nondeterminism, consider the following PTS modeling a simple communication protocol:



and the MCL formula below, supposedly stored in a file "p.mcl" (with the line numbers indicated for clarity):

```
 1  prob
 2      SND .
 3      (
 4          (not (RETRY or RCV))* .
 5          RETRY
 6      ) { 0 ... 2 } .
 7      (not RCV)* .
 8      RCV
 9  is
10      < ? 0.9999
11  end prob
```

This formula expresses that the probability that a message sent (action SND) is received (action RCV) after at most two retries (action RETRY) caused by transmission errors is less than 99,99%. The invocation of **evaluator5**(LOCAL) for checking this formula on the PTS above produces the following output:

```
warning during optimisation phase:
    possibly nondeterministic probabilistic formula;
    check that the two sequences below do not overlap
    (and, if needed, ensure this by adding constraints
    on action formulas):

SND . (* [p.mcl:2] *)
  not (RCV) (* [p.mcl:7] *)

SND . (* [p.mcl:2] *)
  RETRY (* [p.mcl:5] *)
```

```
1.11

FALSE
```

The warning message above indicates that the regular subformula can match a transition sequence s0--SND--> ... --RETRY-->s1 in two different ways, caused by the fact that the action predicates "not (RCV)" and "RETRY" overlap (they are both satisfied by the RETRY action). Indeed, the following transition sequence:

s0--SND-->s1--ERR-->s2--RETRY-->s1--TRANS-->s3--RCV-->s0

in the PTS is matched by the regular subformula either as "SND . (not RCV)∗ . RCV" (corresponding to 0 retries) or as "SND . (not (RETRY or RCV))∗ . RETRY . (not RCV)∗ . RCV" (corresponding to one retry).

This nondeterminism makes the probabilities labeling the actions of this sequence to be counted twice, which yields an incorrect value 1.11 of the probability.

A deterministic version of the formula above is obtained by replacing the subformula "(not RCV)∗" by the more constraining formula "(not (RETRY or RCV))∗", which reflects the intended meaning of the initial property (i.e., a message is received after a number of retries between 0 and 2, these retries being captured by the "((not (RETRY or RCV))∗ . RETRY) { 0 ... 2 }" subformula). This revised formula is verified correctly, without any warning message, producing a probability value of 0.999 and a verdict TRUE.

Note: If a "prob" formula triggers warning messages concerning nondeterminism, it is important to eliminate *all* these warnings to achieve a correct verification. Indeed, even if the computed probability is less than 1, the presence of such warnings indicates an ambiguity in the formula, and thus the risk of having computed a probability higher than the real one.

BIBLIOGRAPHY

[DJJL01] P. D'Argenio, B. Jeannet, H. Jensen, and K. Larsen. "Reachability Analysis of Probabilistic Systems by Successive Refinements." Proceedings of the Joint Intetnational Workshop on Process Algebra and Probabilistic Methods, Performance Modeling and Verification PAPM/PROBMIV'01, LNCS v. 2165, p. 39-56, 2001.

[DV90] R. De Nicola and F. W. Vaandrager. "Action versus State based Logics for Transition Systems." Proceedings Ecole de Printemps on Semantics of Concurrency, LNCS v. 469, p. 407-419, 1990.

[EL86] E. A. Emerson and C-L. Lei. "Efficient Model Checking in Fragments of the Propositional Mu-Calculus." Proceedings of the 1st LICS, p. 267-278, 1986.

[FL79] M. J. Fischer and R. E. Ladner. "Propositional Dynamic Logic of Regular Programs." Journal of Computer and System Sciences, no. 18, p. 194-211, 1979.

[HJ94] H. Hansson and B. Jonsson. "A Logic for Reasoning about Time and Reliability." Formal Aspects of Computing, v. 6, no. 5, p. 512-535, 1994.

[Koz83] D. Kozen. "Results on the Propositional Mu-Calculus." Theoretical Computer Science, v. 27, p. 333-354, 1983.

[LS91] K. G. Larsen and A. Skou. "Bisimulation through Probabilistic Testing." Information and Computation, v. 94, no. 1, p. 1-28, 1991.

[Mat06] R. Mateescu. "CAESAR_SOLVE: A Generic Library for On-the-Fly Resolution of Alternation-

Free Boolean Equation Systems." Springer International Journal on Software Tools for Technology Transfer (STTT), v. 8, no. 1, p. 37-56, 2006. Full version available as INRIA Research Report RR-5948. Available from http://cadp.inria.fr/publications/Mateescu-06-a.html

[MR18] R. Mateescu and J. I. Requeno. "On-the-Fly Model Checking for Extended Action-Based Probabilistic Operators." Springer International Journal on Software Tools for Technology Transfer (STTT), v. 20, no. 5, p. 563-587, 2018.

[MS13] R. Mateescu and W. Serwe. "Model Checking and Performance Evaluation with CADP Illustrated on Shared-Memory Mutual Exclusion Protocols." Science of Computer Programming v. 78, no. 7, p. 843-861, 2013.

[MT08] R. Mateescu and D. Thivolle. "A Model Checking Language for Concurrent Value-Passing Systems." Proceedings of the 15th International Symposium on Formal Methods FM'08, LNCS v. 5014, p. 148-164, 2008. Available from http://cadp.inria.fr/publications/Mateescu-Thivolle-08.html

[Str82] R. S. Streett. "Propositional Dynamic Logic of Looping and Converse." Information and Control, v. 54, p. 121-141, 1982.

SEE ALSO

bcg_steady(LOCAL), **evaluator**(LOCAL), **evaluator3**(LOCAL), **evaluator4**(LOCAL), **evaluator5**(LOCAL), **mcl**(LOCAL), **mcl3**(LOCAL), **mcl4**(LOCAL), **regexp**(LOCAL)

Additional information is available from the CADP Web page located at http://cadp.inria.fr

Directives for installation are given in files **$CADP/INSTALLATION_***.

Recent changes and improvements to this software are reported and commented in file **$CADP/HISTORY**.

BUGS

Please report bugs to Radu.Mateescu@inria.fr