

Moca-RP^c

Manuel utilisateur 13.21

Philippe Thomas

Copyright © 2013 Total

Table des matières

1. Introduction	5
1.1. Nouveautés : Version 13.21	5
1.2. Nouveautés : Version 13.16	5
1.3. Nouveautés : Version 13.14	5
1.4. Nouveautés : Version 13.12	5
1.5. Nouveautés : Version 13.07	6
1.6. Nouveautés : Version 13.04	6
1.7. Nouveautés : Version 13.01	6
1.8. Nouveautés : Version 13.00	6
1.9. Nouveautés : Version 12.19	7
1.10. Nouveautés : Version 12.16	7
1.11. Nouveautés : Version 12.14	7
1.12. Nouveautés : Version 12.10	8
1.12.1. Résultats pour différents temps de calcul	8
1.12.2. Ajout des priorités sur les transitions	8
1.12.3. Ajout d'opérateurs	8
1.13. Nouveautés : Version 12.0x	8
1.13.1. Modification du pouvoir d'expression	8
1.13.2. Interpréteur de commandes	9
1.13.3. Simulation double-détente	9
1.13.4. Mémorisation des résultats	9
2. Format des Réseaux de Petri de Moca-RP^c	11
2.1. Introduction	11
2.2. Rubrique : Options	11
2.3. Rubrique : Constantes (paramètres nommés)	12
2.4. Rubrique : Variables (et expressions)	12
2.4.1. Déclarations	12
2.4.2. Expressions numériques	12
2.4.3. Expressions booléennes	13
2.4.4. Priorités des opérateurs	14
2.5. Rubrique : Transitions	14
2.5.1. Nom de transition	14
2.5.2. Places en amont/aval	14
2.5.3. Gardes ou Conditions de validation	15
2.5.4. Affectations	15
2.5.5. Loi de délai	15
2.5.6. Loi de tir	16
2.5.7. Priorité & Equiprobabilité	16
2.5.8. Transitions à mémoire	17
2.5.9. Limiter le nombre de tir instantané	18
2.5.10. Histogramme	18
2.6. Rubrique : Tableaux généraux	18
2.7. Rubrique : Places	19
2.8. Rubrique : Initialisations	19
2.9. Rubrique : Etats statistiques (Observateurs)	19
2.9.1. Nom de l'état statistique	19
2.9.2. Expression	19
2.9.3. Drapeaux	20
2.10. Rubrique : Type de statistiques	20
2.11. Exemple d'un réseau de Petri	21
2.12. Résultats attendus	22
3. Interpréteur de commandes	23
3.1. Introduction	23
3.2. Lancement de Moca-RP ^c	23

3.2.1. Exemple d'une session Moca-RP ^c	23
3.2.2. Lancement de Moca-RP ^c en batch	24
3.3. Commandes de base	24
3.3.1. Commentaires	24
3.3.2. Aide en ligne { help ...}	25
3.3.3. Chargement de fichiers { load ...}	25
3.3.4. Sauvegarde de fichier { save ...}	25
3.3.5. Gestion des différents formats de données { ##parser ...}	25
3.3.6. Réinitialisation { clear all ;}	26
3.3.7. Sortie de l'interpréteur { exit ...}	26
3.3.8. Redirection de l'affichage	26
3.4. Commandes de gestion des réseaux	26
3.4.1. Affichage d'un réseau { display net ...}	26
3.4.2. Modification d'un réseau { set net ...}	26
3.4.3. Suppression d'un réseau de Petri { clear net ...}	27
3.4.4. Lancement d'une simulation { simul ...}	27
3.5. Commandes de gestion des résultats	30
3.5.1. Introduction	30
3.5.2. Accès aux différentes banques de résultats	31
3.5.3. Affichage des résultats { result display ...}	31
3.5.4. Chargement et sauvegarde de résultats { result load save ...}	31
3.5.5. Suppression d'une banque de résultats { result clear ...}	31
3.5.6. Fusion de 2 banques de résultats { result merge ...}	31
3.5.7. Ajout d'une banque dans une autre { result add ...}	32
3.5.8. Exemple d'un traitement par lot à l'aide de banques de résultat	32
3.6. Commandes de gestion des formats d'affichage	33
3.6.1. Création d'un format d'affichage { set format ...}	33
3.6.2. Utilisation d'un format d'affichage	40
3.6.3. Affichage d'un format { display format ...}	40
3.6.4. Suppression d'un format d'affichage { clear format ...}	40
3.7. Commandes Utilitaires	41
3.7.1. Gestionnaire de chronomètres { ... timer ...}	41
3.7.2. Trace des commandes et des résultats { set trace ...}	41
3.7.3. Options de l'interpréteur de commandes { display options set ...}	41
3.7.4. Utilisation des commandes systèmes { system ...}	43
3.7.5. Affichage d'une chaîne de caractères { display "<string>" ...}	43
4. Les lois dans Moca-RP^c	44
4.1. Généralités	44
4.2. Lois de délai de Moca-RP ^c	44
4.2.1. Loi Dirac { drc ...}	45
4.2.2. Loi Erlang { erlg ...}	45
4.2.3. Loi Erlang généralisée { erlgg ...}	45
4.2.4. Loi Empirique { empir ...}	45
4.2.5. Loi Exponentielle { exp ...}	46
4.2.6. Loi Exponentielle + wow { expw ...}	46
4.2.7. Loi Instant Prévu à l'Avance { ipa ...}	46
4.2.8. Loi Instant Fixé à l'Avance { ifa ...}	47
4.2.9. Loi Log-Normale { nlog ...}	47
4.2.10. Loi Spéciale { spec ...}	47
4.2.11. Loi Triangulaire { tri ...}	48
4.2.12. Loi Uniforme { unif ...}	48
4.2.13. Loi Weibull { web ... ou Weibull ...}	48
4.2.14. Loi Weibull tronquée { webtr ...}	49
4.3. Lois de tir de Moca-RP ^c	50
4.3.1. Loi de tir par défaut { def }	50
4.3.2. Loi de tir à la sollicitation { sol sol2 ...}	51
4.3.3. Loi spéciale { spec ...}	51

5. Interface de programmation C	52
5.1. Lancement de Moca-RP ^c	52
5.2. Utilisation des fonctions/lois externes	52
5.2.1. Fonctions externes	52
5.2.2. Lois externes (ou spéciales)	53
5.3. Création de la librairie optionnelle MocaAdd.dll	53
5.3.1. Pré-requis	53
5.3.2. Fichiers utilisés	53
5.3.3. Principe général	54
5.3.4. Création de fonctions externes	54
5.3.5. Création de lois externes	56
5.4. Description des types/fonctions de l'API	57
5.4.1. Déclaration des différents types gérés par l'API	57
5.4.2. Liste des différentes fonctions	61
A. Lecture de fichiers au format Moca10	67
B. Lecture de fichiers au format Aralia	68
C. Calcul de la moyenne et de la variance	72
1. Lors de la simulation	72
2. Lors de la fusion de 2 lots	72
2.1. Présentation	73
2.2. Développements mathématiques	73
2.3. Résultats	74

1. Introduction

Le logiciel Moca-RP^c (Monte-Carlo basé sur les Réseaux de Petri) est destiné à la simulation du comportement des systèmes dynamiques complexes dans le but d'obtenir, par un traitement statistique, des résultats concernant leur fiabilité, disponibilité, productivité, ainsi que tout autre paramètre probabiliste. Le modèle du système à étudier est réalisé sous la forme d'un réseau de Petri stochastique interprété qui sert de support à une simulation de Monte-Carlo classique.

En 1996, une collaboration entre ELF Exploration-Production, le Laboratoire Bordelais de Recherche en Informatique (LaBRI) et la société IXI, a conduit au développement de la version 10 du logiciel Moca-RP^c. Cette version se distingue de la précédente par son langage d'implémentation (le langage C ANSI au lieu du Fortran) qui a facilité l'intégration de nouvelles fonctionnalités.

En 2002, une collaboration entre Total-Fina-Elf et la société GFI Consulting a conduit au développement de la version 12 du logiciel Moca-RP^c. Cette version intègre principalement, les messages et états statistiques complexes, un interpréteur de commande et un début de simulation double-détente.

De 2005 à 2009, une collaboration entre Total et Dassault-Data-Services a permis de faire évoluer Moca-RP^c jusqu'à l'aboutissement d'une version 13 :

- La version 12.10 intègre la possibilité d'afficher les résultats des états statistiques à différents temps.
- La version 12.14 permet de charger des arbres de défaillance au format Aralia afin de calculer efficacement la fiabilité associée aux événements redoutés décrits à l'aide de ces arbres de défaillances.
- La version 13.00 intègre un simulateur pas à pas exploitable par des interfaces utilisateur comme GRIF.
- La version 13.04 intègre un ensemble de mécanisme simplifiant la modélisation utilisateur comme les priorités dynamiques, l'évaluation séquentielle des affectations, les transitions à mémoire dynamique, la gestion équiprobable des conflits, ...

Depuis 2010, le développement de la version 13 de Moca-RP^c se poursuit à travers une collaboration entre Total et SaToDev.

- La version 13.07 intègre une notion de tableau et la possibilité d'exécuter du code lors du tir d'une transition. TBC ...

1.1. Nouveautés : Version 13.21

Ajout d'une loi Weibull basée sur le paramètre d'échelle.

L'opérateur $e1 \% e2$ accepte maintenant des arguments réels.

Les statistiques sur les places et les transitions ne sont plus obligatoires.

1.2. Nouveautés : Version 13.16

Lancer Moca-RP^c avec une priorité inférieure à la normale sous Windows.

1.3. Nouveautés : Version 13.14

Afficher la graine du générateur de nombre aléatoire à la fin de chaque histoire

Pouvoir faire plus de 2147483647 histoires avec la version 64 bits de Moca-RP^c.

1.4. Nouveautés : Version 13.12

Ajout des fonctions random() et crc(...)

Amélioration de la commande format (nouvelle directive d'affichage Cst et Var, Utilisation de joker dans les sélecteurs, drapeau XML dans la directive Net)

1.5. Nouveautés : Version 13.07

TBC

1.6. Nouveautés : Version 13.04

Jusqu'à présent les affectations lors du tir d'une transition se faisaient de manière parallèle de la même manière que les déplacements de jetons (on retire tous les jetons des places amonts, puis on ajoute tous les jetons aux places avalées). Sous la pression de nombreux utilisateurs, il est maintenant possible d'évaluer les affectations de manière séquentielle (comme dans un langage de programmation). Pour plus d'informations, Cf. Section 2.5.4, « Affectations ».

Dans certain cas, par exemple afin de prendre en compte une politique de maintenance différente suite à une reconfiguration, il est nécessaire de pouvoir modifier en cours de simulation la priorité de certaines transitions. Les priorités dynamiques ont été introduites pour répondre à ce besoin (Cf. Section 2.5.7, « Priorité & Equiprobabilité »).

Il est maintenant possible de choisir de manière équiprobable une transition parmi les transitions en conflit. Cette nouvelle politique de gestion des transitions en conflit exploite également la notion de priorité et peut être utilisé conjointement à politique actuelle. Cf. Section 2.5.7, « Priorité & Equiprobabilité ».

Certaines modélisations posent des problèmes de boucles instantanées qui obligent les utilisateurs à mettre en place des subterfuges afin de les empêcher (via des messages dans le meilleur des cas, via des lois dirac epsilon ou via ...). Il est maintenant possible de préciser le nombre de tir autorisé sans incrément du temps avant qu'une transition ne devienne invalide. Cf. Section 2.5.9, « Limiter le nombre de tir instantané ».

Cette version apporte également un mécanisme pour prendre en compte ou non, de manière dynamique, la mémoire d'une transition. Il est alors possible de modéliser un composant sollicité avec maintenance corrective et préventive de manière plus juste. Cf. Section 2.5.8, « Transitions à mémoire ».

Pour finir, un nouveau type de statistique a été ajouté afin de calculer la valeur moyenne entre 2 temps de calcul (typiquement cela permet de calculer facilement une production moyenne pour chaque année de production sans être obligé de créer un RdP utilitaire). Cf. Section 2.10, « Rubrique : Type de statistiques ».

1.7. Nouveautés : Version 13.01

Les formats d'affichage ont été étendus afin de permettre d'afficher les résultats au format XML. Pour plus d'informations, Cf. Section 3.6, « Commandes de gestion des formats d'affichage ».

Afin de permettre de contrôler le déroulement d'une simulation et en particulier arrêter la simulation sans envoyer de signal SIGINT (ce qui n'est pas possible en Java par exemple), il est maintenant possible d'arrêter la simulation lorsque la présence d'un fichier lock est avérée (Cf. Section 3.4.4.5, « Arrêt de la simulation de manière externe {set simul locker ...} »).

Il est maintenant possible d'afficher la progression d'une simulation en cours en pourcentage du nombre total d'histoire à réaliser. Cela permet de se rendre compte de l'avancement de la simulation. Pour plus d'informations, Cf. Section 3.4.4, « Lancement d'une simulation {simul ...} ».

1.8. Nouveautés : Version 13.00

La version 13.00 intègre un simulateur pas à pas avec possibilité de revenir en arrière, de tirer une transition à un temps spécifique, de visualiser l'échéancier, la valeur d'une variable ou le marquage d'une place. Ce simulateur pas à pas est directement utilisé par les interfaces graphiques (comme GRIF) afin de permettre de déboguer un réseau de Petri.

Lorsqu'une exception intervient lors de la simulation (comme une division par zéro, un nombre de tir instantané supérieur à la limite, ...) le message d'erreur intègre maintenant la valeur de la graine du générateur aléatoire au début de l'histoire ayant engendré l'erreur. Cette information a son importance depuis qu'il est possible de déboguer un réseau de Petri. En effet, lors du lancement de la simulation graphique, il suffira de préciser la graine du générateur aléatoire pour se positionner dans le même cadre que lors de l'apparition de l'exception et ainsi en comprendre l'origine.

Le calcul à différents temps pour les états statistiques fonctionnait jusqu'à présent à partir des valeurs des états statistiques avant le tir de la transition. Cela pouvait poser des problèmes de compréhension des résultats lorsqu'une transition devait être tirée de manière déterministe à un instant t et qu'un état statistique devait être calculé également à t . La valeur retenue pour l'état statistique correspondait à la valeur avant le tir de la transition. Ce n'est maintenant plus le cas. La valeur retenue correspond maintenant à la valeur après le tir de toutes transitions devant être tirées à l'instant considéré. Les états statistiques n'étant que des observateurs, ils sont donc considérés comme moins prioritaires qu'une quelconque transition.

En revanche, il est toujours intéressant de connaître la valeur d'un état statistique juste avant un tir d'une transition déterministe. Cela permet de capter les phénomènes de discontinuités déterministes. Il est donc possible de calculer un état statistique à $t - \epsilon$ (c'est à dire juste avant t). Pour plus d'informations, voir la syntaxe associée aux temps de calcul.

1.9. Nouveautés : Version 12.19

Des fonctions de gestion des banques de résultat ont été ajoutées afin de permettre la parallélisation de la simulation à l'extérieur de Moca-RP^c. Pour plus d'informations, voir Section 3.5, « Commandes de gestion des résultats ».

Un nouveau format d'affichage des histogrammes a été mis en place afin d'afficher les SIL (Safety Integrity Level) d'une grandeur et plus généralement d'afficher un histogramme avec des intervalles choisis. Pour plus d'informations, voir Section 3.6.1.8, « Directive pour les histogrammes des états statistiques ».

Une seconde loi de tir à la sollicitation a également vu le jour afin de permettre d'associer explicitement les probabilités d'occurrence aux places. Pour plus d'informations, voir Section 4.3.2, « Loi de tir à la sollicitation {sol|sol2 ...} ».

1.10. Nouveautés : Version 12.16

Il est maintenant possible d'associer un nom aux tableaux au sein de la description d'un réseau de Petri (Cf. Section 2.6, « Rubrique : Tableaux généraux »).

L'API de programmation a été mise à jour afin de pouvoir définir des fonctions externes à Moca. Ces fonctions doivent être écrites en C/C++ et compilées au sein d'une librairie dynamique (MocaAdd.dll sous Windows). Pour plus d'informations, voir Section 5, « Interface de programmation C ».

1.11. Nouveautés : Version 12.14

De plus en plus d'études sont réalisés afin de déterminer les SIL (Safety Integrity Level) d'une installation ou d'un système de protection. Ces études sont réalisées à l'aide de Aralia WorkShop. Le calcul de SIL est basé sur une estimation de la disponibilité à chaque instant de la vie du système.

Les arbres de défaillance ne permettent de calculer que la disponibilité du système et éventuellement une approximation de sa fiabilité lorsque tous les événements ont un comportement de panne et de réparation défini à l'aide de loi exponentielle. Dans les autres cas, Aralia n'est pas capable de fournir de manière relativement sûre la fiabilité du système.

Le module de conversion de formule Aralia a été industrialisé afin de calculer facilement la fiabilité des variables sommets (à priori les événements redoutés) des formules Aralia.

En particulier, les différentes lois de test périodique, ainsi que la loi Weibull ont été traduites en réseau de Petri équivalent.

1.12. Nouveautés : Version 12.10

1.12.1. Résultats pour différents temps de calcul

Il est parfois utile de suivre l'évolution d'un état statistique afin de tracer des courbes de disponibilité, productivité, fiabilité, ... en fonction du temps. Préalablement, les calculs statistiques ne se faisaient qu'au temps de mission du système. Pour réaliser de telles courbes, il fallait relancer autant de fois Moca-RP^c que l'on souhaitait de points intermédiaires ce qui conduisait à des temps de calcul parfois très longs. Ce temps est proportionnel au nombre de points souhaité soit parfois 2h pour un calcul unitaire de l'ordre de 2 à 3 minutes.

Il est maintenant possible de spécifier les différents temps de calcul, soit à l'aide d'une liste de temps, soit en précisant un temps initial, un temps final et un pas de temps pour l'ensemble des états statistiques ou pour uniquement un état statistique considéré.

1.12.2. Ajout des priorités sur les transitions

La priorité est utilisée pendant la simulation lors de conflit possible sur plusieurs transitions tirables au même instant. Auparavant, seul le rang de description de la transition permettait de déterminer la priorité de la transition. Il est maintenant possible de spécifier un entier positif précisant le degré de priorité de la transition.

1.12.3. Ajout d'opérateurs

Les opérateurs suivant ont été ajoutés comme opérateur valide dans les descriptions Moca-RP^c :

- `delay()` correspondant à la durée de la simulation (équivalent à une constante égale à l'option `duration` du réseau de Petri)
- `@(e1, ..., en)` (où `e1, ..., en` sont des expressions booléennes) est équivalent à l'opérateur `#(e1, ..., en)` du langage AltaRica DataFlow (Il a été ajouté afin de pouvoir traduire des descriptions AltaRica en Moca-RP^c)

1.13. Nouveautés : Version 12.0x

1.13.1. Modification du pouvoir d'expression

Les messages manipulés dans la version de 10 de Moca-RP^c ne sont que booléens. La version 12 enrichit le pouvoir d'expression du modèle, en permettant l'utilisation des variables réelles, entières ou booléennes. À partir de ces variables et constantes, il est possible de construire, à l'aide d'opérateurs, des expressions complexes qui sont utilisées dans différentes parties de descriptions (en particulier les émissions/réceptions de messages).

De plus, la version 12 de Moca-RP^c scinde clairement les lois qui déterminent le délai de la transition de celles qui spécifient la manière de tirer la transition. Cette idée permet de définir beaucoup plus simplement (et proprement) les défaillances à la sollicitation. En effet, les défaillances à la sollicitation modifient les règles de tir habituelles d'une transition puisque seul le marquage d'une place avale est modifié.

Le format de description des réseaux de Petri a donc été modifié afin de pouvoir :

- Déclarer des constantes et des variables booléennes, entières ou réelles
- Définir ces variables à l'aide d'expressions complexes prenant en compte les opérateurs mathématiques et logiques usuels (+, -, *, / &, |, !, ite, ...) ainsi que des opérateurs spécifiques aux réseaux de Petri manipulés dans Moca-RP^c (`#i` pour le marquage de la place `i`, `min(...)` et `max(...)` pour les fonctions retournant le minimum et le maximum de leurs opérateurs, `nlog(m, e)` et `unif(min, max)` pour des générateurs de nombres aléatoires utilisés dans la simulation double-détente, ...).
- Définir la liste de transition (dans un format proche de celui actuellement utilisé dans la version 10.04) précisant pour chaque transition : son nom, ses arcs amonts et avals, ses gardes (une liste d'expressions devant être différentes de zéro pour rendre la transition valide), ses affectations (une liste de variables prenant une expression comme valeur au moment où la transition est tirée), ses lois de délai et de tir définis à l'aide de paramètres

(eux mêmes décrits à l'aide d'expressions), des éventuels drapeaux permettant de spécifier si la transition est à mémoire ou pas et si l'utilisateur souhaite obtenir un histogramme ou non.

- Spécifier le nom de chacune des places du réseau
- Définir le marquage initial des places, ainsi que l'affectation initiale des variables
- Définir les états statistiques à partir d'expressions complexes.

Les paramètres des lois (de délai et de tir) sont des expressions dont la valeur dépend du moment où la transition est valide (loi de délai) ou tirée (loi de tir). Ceci généralise les lois optionnelles de la version 10 de Moca-RP^C.

1.13.2. Interpréteur de commandes

La version 10 de Moca-RP^C fonctionne de manière batch. On spécifie, sur la ligne de commande appelant le logiciel, le fichier d'entrée à traiter, le fichier de sortie et un certain nombre de directives optionnelles. Le logiciel traite alors le fichier d'entrée, génère le fichier de sortie et rend la main. Pour refaire une simulation, il faut modifier le fichier d'entrée et relancer le calcul.

Les différents projets de ces dernières années (Aralia, Hévée, Aloès, ...) montrent l'intérêt des interpréteurs de commandes, tant en terme d'utilisation (choix du déroulement des opérations, modifications des données, ...) que de programmation (facilité d'ajout de fonctionnalités, de format d'affichage, ...).

La version 12 de Moca-RP^C intègre un interpréteur de commandes permettant :

- La création et la gestion de plusieurs réseaux de Petri
- Le chargement des réseaux au format Moca-RP^C version 10.04 et précédents
- Le chargement des réseaux au format Moca-RP^C version 12
- La sauvegarde (et / ou l'affichage) des réseaux au format Moca-RP^C version 12
- L'arrêt et la reprise d'une simulation permettant la vérification d'un RdP sur un petit nombre d'histoires, puis le lancement de la simulation pour un grand nombre d'histoires (sans perdre les résultats précédents).
- L'affichage de tout ou d'une partie des résultats vers un ou plusieurs fichiers (les histogrammes pouvant être redirigés vers un fichier spécifique afin de les traiter sans filtrer les autres données)
- Une aide en ligne, une commande générique afin d'accéder au système, une gestion de chronomètres, ...

1.13.3. Simulation double-détente

Le but d'une simulation double-détente est d'évaluer l'influence des incertitudes des paramètres d'entrée sur la dispersion du (des) résultat(s) de sortie.

Comme proposer en [8], la première idée pour tenir compte de la dispersion des paramètres d'entrée dans une simulation de Monte Carlo d'un RdP, est de faire deux tirages au hasard en cascade : un pour choisir le jeu de paramètres et la seconde pour la simulation proprement dite.

Cela va consister à tirer un jeu de paramètres au hasard et de réaliser n histoires avec ce jeu, puis recommencer m fois avec m autres jeux de paramètres. Ainsi on obtient m échantillons de n valeurs chacun dont on va pouvoir estimer les paramètres statistiques.

Comme cette implémentation correspond à un premier essai de simulation double-détente, on ne conserve actuellement que la moyenne de la seconde simulation. On obtient alors un échantillon de m moyennes sur lequel on estime les paramètres statistiques.

Afin d'aller plus loin, il est possible d'afficher une trace des m résultats intermédiaires (moyenne et variance de chaque échantillon de n valeurs). Il est alors possible de faire un post-traitement.

1.13.4. Mémorisation des résultats

Un des scénarios d'utilisation de Moca-RP^C consiste à l'utiliser en « Batch ». On lance une simulation pour un réseau de Petri avec des paramètres de simulation donnés (nombre d'histoires, temps de simulation maximum, ...), puis on sort de l'interpréteur.

Il est possible que les résultats obtenus ne satisfassent pas l'utilisateur dans la mesure où ils ne sont pas assez précis. Une des deux raisons suivantes explique ce fait : le nombre d'histoires était insuffisant ou le temps de simulation maximum a été atteint avant la fin de la simulation.

Comme la simulation a été lancée en « batch », il n'est pas possible de reprendre la simulation là où elle s'était arrêtée (contrairement à une utilisation en mode interprété).

L'idée consiste à pouvoir mémoriser les résultats dans un fichier afin de pouvoir les recharger par la suite et poursuivre la simulation.

Cette idée a été mise en œuvre dans la gestion des banques de résultats.

2. Format des Réseaux de Petri de Moca-RP^c

Cette première section décrit le format d'entrée des réseaux de Petri pour la version actuelle de Moca-RP^c. Il est à noter qu'il est possible d'ouvrir à l'aide de cette version des fichiers provenant des versions 8, 9 et 10 de Moca-RP^c. Veuillez-vous référer à l'annexe A pour plus d'informations sur ce sujet.

2.1. Introduction

Les réseaux de Petri sont chargés au sein de l'interface à l'aide de la commande **load Moca "<file>"**; où <file> est le nom (entouré de guillemets) du fichier à charger contenant un ou plusieurs réseaux de Petri.

Ce format de données est le seul qui permet de définir un réseau en prenant en compte toutes les possibilités de Moca-RP^c. Les réseaux en mémoire sont affichés dans ce format lorsque l'on utilise la commande **display net <net>**; (Cf. Section 3.4.1, « Affichage d'un réseau {**display net ...**} »).

Un fichier de description Moca-RP^c peut contenir la description de plusieurs réseaux de Petri. Chaque réseau de Petri est décrit à l'aide d'un bloc ayant la forme suivante :

```
net <id-net> {  
  /* Différentes rubriques pour constituer le RdP */  
};
```

où <id-net> est le nom du réseau de Petri défini dans le bloc qui suit. Ce dernier est constitué de différentes rubriques qui ne sont pas forcément ordonnées. La seule limite consiste à déclarer une variable avant de l'utiliser.

2.2. Rubrique : Options

Cette rubrique permet d'ajouter des informations spécifiques et inclassables, tel qu'un titre (ou une description) associé au réseau de Petri ou la durée d'une histoire.

Chacune des options est précédée du mot-clef général **OPT:** suivi d'un mot-clef spécifique à l'option et de sa valeur. Seules les options suivantes sont permises :

- **title** suivi d'une chaîne de caractères entre guillemets permet d'associer un titre au réseau de Petri
- **duration** suivi d'un entier ou d'un réel permet de spécifier la durée d'une histoire. Cette valeur doit évidemment être cohérente avec les lois de délai du réseau.
- **times <times>** permet de préciser les différents temps de calcul pour les états statistiques. La directive <times> peut s'écrire de deux manières :
 - **at t_1 , ..., t_i , ..., t_n** : pour préciser une liste de temps explicite. Le ou les t_i sont des nombres réels positifs ou nuls. Ils peuvent être suivis du caractère '-' pour demander explicitement à faire le calcul avant le tir des transitions devant se faire à t_i .
 - **from min to max step inc** : pour générer une liste de temps de min à max par pas de inc. L'incrément inc peut être suivi du caractère '-' pour des observations à $t - \epsilon$.
- **add_censured_hst** suivi d'un booléen permet d'activer/(désactiver) l'ajout des données censurées au sein des histogrammes des transitions.

Exemple 1 :

```
OPT: title      "DEFAULT TITLE" ;  
OPT: duration   100.00 ;  
OPT: times from 0.0 to 100.0 step 10.0 ;
```

Exemple 2 :

```
OPT: times at 0, 5, 10-, 10, 15, 20-, 20;
```

Si des tirs de transitions ont lieu à $t = 10$ et/ou à $t = 20$, les états statistiques seront calculés juste avant et juste après le tir de ces transitions.

2.3. Rubrique : Constantes (paramètres nommés)

Il est possible de déclarer des constantes qui peuvent être booléennes (`bool`), entières (`int`) ou réelles (`float`). La déclaration d'une constante s'effectue en respectant la syntaxe suivante :

```
const <type> <ident> = <expr> ;
```

où `<type>` est un des mots-clés `bool`, `int` ou `float`, `<ident>` est le nom (identificateur) de la constante et `<expr>` est une expression constante, c'est à dire ne contenant pas de variable.

Exemple :

```
const float LAMBDA1 = 1e-5 ; /* un paramètre de loi générique */  
const int JETONS = 12 ; /* pour l'initialisation d'une place */
```

L'avantage de ces paramètres nommés est qu'ils peuvent être utilisés pour paramétrer le réseau. On peut les employer en particulier pour définir le poids des arcs afin de modifier le comportement du réseau de Petri d'une simulation à l'autre (contrairement aux variables).

2.4. Rubrique : Variables (et expressions)

Les paragraphes Section 2.4.2, « Expressions numériques » et Section 2.4.3, « Expressions booléennes » donnent la syntaxe des expressions pouvant être utilisées pour les gardes, affectations et états statistiques du réseau.

2.4.1. Déclarations

Avant d'être utilisées, les variables doivent être déclarées en utilisant une des syntaxes suivantes :

```
<type> <ident> ;  
<type> <ident> = <expr> ;
```

où `<type>` est un des mots-clés `bool`, `int` ou `float`, `<ident>` est le nom (identificateur) de la variable.

Dans la seconde syntaxe, on définit la variable `<ident>` à partir de l'expression `<expr>`.

Cette affectation peut aussi se faire en deux temps :

```
<type> <ident> ;  
<ident> = <expr> ;
```

Exemple : L'exemple qui suit déclare quatre variables booléennes, la dernière étant définie en fonction des trois premières.

```
bool a; bool b; bool c;  
bool r = (a & b) | (!a & c);
```

2.4.2. Expressions numériques

Elles sont formées à partir des variables et constantes numériques (c.-à-d. entières ou réelles) et des opérateurs `+`, `-` (binaire), `-` (unaire), `*`, `/`, `#`, des parenthèses et de fonctions mathématiques.

Une expression numérique peut être :

- une constante entière ou réelle (Cf. Section 2.3, « Rubrique : Constantes (paramètres nommés) ») ;
- une variable entière ou réelle (Cf. Section 2.4.1, « Déclarations ») ;
- un nombre entier ou réel ;
- `int(expr)` (où `expr` est une expression numérique) est une conversion d'une expression en une expression entière. La valeur de `int(expr)` est la valeur de `expr` arrondie à sa partie entière inférieure ;
- `float(expr)` (où `expr` est une expression numérique) est une conversion de l'expression `expr` en une expression réelle ;
- `-expr` (où `expr` est une expression numérique) a pour valeur l'opposée de celle de `expr` ;
- `(expr)` est une expression numérique qui a la même valeur que `expr` ;
- `#i` (où `i` est un entier > 0) qui est égale au marquage de la place numéro `i` du réseau ;
- Les opérations arithmétiques usuelles `e1+e2`, `e1-e2`, `e1*e2` et `e1/e2` (où `e1` et `e2` sont deux expressions du même type c'est à dire soit entières, soit réelles) ;
- L'opérateur `e1%e2` (où `e1` et `e2` sont deux expressions entières) a pour valeur le reste de la division entière de `e1` par `e2` ;
- `unif(e1,e2)` (où `e1` et `e2` sont deux expressions réelles constantes) est un générateur de nombres aléatoires suivant une distribution uniforme dans l'intervalle `[e1,e2]` (voir ci-dessous) ;
- `nlog(m,e)` (où `m` et `e` sont deux expressions réelles constantes) est un générateur de nombres aléatoires suivant une distribution lognormale centrée en la valeur moyenne `m` et se trouvant avec une probabilité de 90% dans l'intervalle `[m/e,m*e]` (voir ci-dessous) ;
- `min(e1, ..., en)` et `max(e1, ..., en)` (où `e1, ..., en` sont des expressions du même type) sont des fonctions calculant le minimum ou le maximum des expressions passées en paramètre ;
- `ln(e1)`, `exp(e1)`, `logx(e1, e2)`, `pow(e1, e2)` (où `e1` et `e2` sont deux expressions réelles) sont des fonctions calculant le logarithme népérien, l'exponentiel, le logarithme base `e2` ou la puissance de `e1` ;
- `ite(e1, e2, e3)` (où `e1` est une expression booléenne et `e2` et `e3` deux expressions du même type) a pour valeur `e2` si `e1` (`e1 == vrai` ou `e1 != 0`) sinon elle a pour valeur `e3`. La syntaxe suivante `e1 ? e2 : e3` définie en langage C existe aussi pour cet opérateur ;
- `time()` est le temps courant de la simulation.
- `story()` est le numéro de l'histoire courante.
- `delay()` est la durée de la simulation (équivalent à une constante égale à l'option `duration` du réseau de Petri)
- `@(e1, ..., en)` (où `e1, ..., en` sont des expressions booléennes) est égale au nombre d'argument valant vrai. Cet opérateur a été ajouté afin d'être compatible avec l'opérateur `#(e1, ..., en)` du langage AltaRica DataFlow.

Une expression peut faire intervenir des générateurs de nombres aléatoires ; ceux-ci seront utilisés lors d'une simulation double détente. Sur l'exemple ci-dessous, la variable `LAMBDA` peut être utilisée comme paramètre d'une loi.

```
const float MIN_L 1e-3 ;
const float MAX_L 1e-4 ;
float LAMBDA unif(MIN_L,MAX_L) ;
```

2.4.3. Expressions booléennes

Elles sont formées à partir des variables et constantes booléennes (vrai et faux), des parenthèses et des opérateurs `|`, `ou`, `&`, `et`, `!`, `non`, `==`, `!=`, `<`, `>`, `<=`, `>=` et `@`.

Une expression booléenne peut être :

- une constante booléenne (Cf. Section 2.3, « Rubrique : Constantes (paramètres nommés) ») ;
- une variable booléenne (Cf. Section 2.4.1, « Déclarations ») ;
- une des constantes `vrai` ou `faux` ;
- `e1 | e2`, `e1 ou e2` (où `e1` et `e2` sont des expressions booléennes) est le OU logique de `e1` et `e2` ;
- `e1 & e2`, `e1 et e2` (où `e1` et `e2` sont des expressions booléennes) est le ET logique de `e1` et `e2` ;

- `!expr, non expr` (où `expr` est une expression booléenne) est le NON logique de `expr` ;
- les opérateurs d'égalité et d'inégalité usuels `e1 == e2, e1 != e2, e1 < e2, e1 > e2, e1 <= e2, e1 >= e2` (où `e1` et `e2` sont deux expressions numériques du même type ; c'est à dire soit entières, soit réelles) ;
- `@(i)(e1, ..., en)` (où `i` est un entier positif et `e1, ..., en` sont des expressions booléennes) vaut vrai si au moins `i` des `e1, ..., en` valent vrai ;
- `@(i,j)(e1, ..., en)` (où `i` et `j` sont des entiers positifs et `e1, ..., en` sont des expressions booléennes) vaut vrai si au moins `i` et au plus `j` des `e1, ..., en` valent vrai.

2.4.4. Priorités des opérateurs

Comme dans tous les langages avec des expressions mathématiques, il existe des priorités lors de la quantification des opérateurs (la multiplication est prioritaire par rapport à l'addition).

Lorsque deux opérateurs ont la même priorité, ils sont quantifiés de la gauche vers la droite.

Les priorités sont données dans le tableau ci-contre de la moins prioritaire à la plus prioritaire.

1. Opérateur tertiaire : `... ? ... : ...`
2. Opérateur booléen : `|`, `ou`
3. Opérateur booléen : `&`, `et`
4. Opérateur d'égalité/inégalité : `==`, `!=`
5. Opérateur de comparaison : `>`, `>=`, `<`, `<=`
6. Opérateur arithmétique : `+`, `-`
7. Opérateur arithmétique : `*`, `/`, `%`
8. Opérateur unitaire : `-`, `!`
9. Autres expressions

2.5. Rubrique : Transitions

La déclaration d'une transition utilise la syntaxe suivante :

```
TR: [<nom>] [AM <arcs-en-amont>] [AV <arcs-en-aval>]
    [?? <conditions>] [(!! <affectations>) | (!! <instructions> )!!]
    [LOI <loi-de-délai>] [TIR <loi-de-tir>]
    [PRIO <expression>] [EQP <expression>]
    [MEM <expression>] [LIMIT <entier>] [HST] ;
```

2.5.1. Nom de transition

Le nom de la transition est un identificateur optionnel ; si celui-ci n'est pas spécifié alors le nom par défaut est `Tri` où `i` est le numéro d'apparition de la transition dans la description.

2.5.2. Places en amont/aval

Les arcs en amont d'une transition sont décrits après le mot-clef `AM`. Ils consistent en une liste de couples (numéro de place, poids) séparées par des virgules. Le poids peut être une expression constante entière respectant la syntaxe donnée à la section 0. A l'instar des anciennes versions de Moca-RP, les arcs inhibiteurs sont indiqués avec un poids négatif et un arc de poids 0 vide la place associée à l'arc.

Les arcs en aval se décrivent de manière analogue mais en utilisant le mot-clef `AV`. Un arc aval ne peut pas avoir de poids négatif. Un avertissement est affiché lorsqu'un arc aval a un poids nul.

Dans l'exemple suivant, une constante `C` est utilisée pour le poids des arcs : le poids de l'arc amont avec la place 2 est de 2 et le poids de l'arc aval avec la place 3 est de 4.

```
const entier C 2 ;
```

```
TR: AM 1 2, 2 C AV 3 2*C ;
```

2.5.3. Gardes ou Conditions de validation

Dans la description d'une transition il est possible de spécifier après le mot-clef ?? (double point d'interrogation pour le différencier de l'opérateur [si]&[alors]:[sinon]) une liste d'expressions booléennes qui doivent être nécessairement vérifiées pour que la transition soit valide. Ces expressions sont séparées par des virgules.

La description ci-dessous définit un message booléen m1 qui doit être vrai pour que la transition Tr1 soit valide (condition nécessaire) mais il faut de plus que le marquage de la place 3 soit supérieur à 2 ; d'une certaine manière la condition #3 >= 2 simule un arc testeur de poids 2 sur la place 3 (i.e. qui ne consomme pas de jeton).

```
bool m1;
TR: Tr1 AM 1 1 AV 2 1 ?? (#3 >= 2), m1;
```

2.5.4. Affectations

Lors du tir d'une transition, il est possible de modifier la valeur d'une (ou plusieurs) variable(s) en lui (leurs) affectant la valeur d'expression(s).

Lorsqu'il y a plusieurs affectations, il est possible de réaliser ces affectations de manière parallèle (en même temps) ou de manière séquentielle (l'une après l'autre, dans l'ordre).

Le mot-clef !! (double point d'exclamation pour le différencier de l'opérateur de négation) introduit une liste d'affectations de variables. Ces affectations ont lieu, en parallèle, avant la production de jetons et consistent en une liste de couples (variable, expression) séparés par des virgules où variable et expression sont du même type.

Le mot-clef ! { introduit une liste d'instructions. Cette liste doit se conclure par le mot-clef }!. Les instructions sont exécutées les unes après les autres dans l'ordre de définition. Actuellement, les instructions possibles sont :

- L'affectation : <variable> = <expression> ; où variable et expression sont du même type (attention au point-virgule en fin de ligne)
- La liste d'instruction : { <instructions> } pour regrouper un ensemble d'instructions.
- Le branchement conditionnel : if (<expression>) <instruction> [else <instruction>] si l'expression est vraie alors la première instruction est exécutée sinon (si elle existe) la seconde instruction est exécutée.

Remarque : Dans les 2 cas, il n'est pas possible d'affecter un nombre de jetons à une place.

À chaque tir de la transition TrDEF ci-dessous la variable NB_DEF est incrémentée ; NB_DEF sert ainsi à compter le nombre de tirs de cette transition.

```
int NB_DEF;
bool m1;
TR: TrDEF AM 1 1 AV 2 1 !! NB_DEF=NB_DEF+1, m1=faux;
```

2.5.5. Loi de délai

La loi, qui permet de spécifier le délai au bout duquel la transition valide sera tirée, est spécifiée après le mot-clef LOI. Nous listons ici l'ensemble des lois possibles ainsi que leurs paramètres. Cf. Section 4.2, « Lois de délai de Moca-RP^c » pour une description plus détaillée.

- drc δ est la loi de Dirac de délai δ ;
- erlg m, β est la loi Erlang de moyenne m et d'ordre β ;

- `erlgg β , λ_1 , ..., λ_β` est la loi d'Erlang généralisée d'ordre β et $\lambda_1, \dots, \lambda_\beta$ sont les paramètres des exponentielles en série ;
- `empir c_1, \dots, c_n` est la loi empirique à n classes c_1, \dots, c_n ;
- `exp λ` est la loi exponentielle de taux λ ;
- `expow λ, δ` est une loi exponentielle de taux λ à laquelle on ajoute le délai δ ;
- `ipa τ` est la loi "instants prévus à l'avance" ayant pour délai entre deux tirs τ ;
- `ifa δ, t_0` est la loi "instants fixés à l'avance" ayant pour délai entre deux tirs le paramètre δ et comme premier instant de tir t_0 ;
- `nlog m, e` est la loi log-normale de moyenne m et de facteur d'erreur e ;
- `spec n, p_1, \dots, p_m` est la $n^{\text{ième}}$ loi spéciale qui prend p_1, \dots, p_m en paramètres. Les lois spéciales permettent d'ajouter de nouvelles lois directement en C à l'aide d'une API de programmation (Cf. Section 5, « Interface de programmation C »)
- `tri a, b, c` est la loi triangulaire[3, 4] de minimum a , de maximum b et de mode c .
- `unif \min, \max` est la loi uniforme de minimum \min et de maximum \max ;
- `Weibull η, β` est la loi de Weibull de paramètre d'échelle η et de paramètre de forme β ;
- `web m, β` est la loi de Weibull de moyenne m et de paramètre de forme β ;
- `webtr m, β, α` est la loi de Weibull tronquée de moyenne m , de paramètre de forme β et d'âge α à l'instant $t = 0$;

Tous les paramètres des lois sont des expressions réelles. Lorsqu'un paramètre est défini par une expression non constante, sa valeur dépend donc de l'instant où la transition est valide. Ce principe permet de définir les lois optionnelles (`drpop`, `expow` et `webop`) relativement facilement.

2.5.6. Loi de tir

La loi, qui permet de spécifier la manière de tirer la transition est spécifiée après le mot-clef `TIR`. Actuellement, il existe trois lois différentes de tir:

- `def` est la loi de tir par défaut (généralement employé dans les réseaux de Petri)
- `sol $\gamma_1, \gamma_2, \dots, \gamma_n$` est une loi de tir à la sollicitation pour $n+1$ arcs avals.
- `sol2 $\gamma_1, \#P_1, \gamma_2, \#P_2, \dots, \gamma_n, \#P_n$` est une seconde loi de tir à la sollicitation pour $n+1$ places avales. Cette seconde loi permet de spécifier explicitement la place qui est associée à chaque gamma.
- `spec n, p_1, \dots, p_m` est la $n^{\text{ième}}$ loi spéciale qui prend p_1, \dots, p_m en paramètres. Les lois spéciales permettent d'ajouter de nouvelles lois directement en C à l'aide d'une API de programmation (Cf. Section 5, « Interface de programmation C »)

Cf. Section 4.3, « Lois de tir de Moca-RP^C » pour une description plus détaillée.

2.5.7. Priorité & Equiprobabilité

La priorité est utilisée pendant la simulation afin d'ordonner les transitions en conflit. Deux transitions sont en conflit si elles doivent être tirées au même instant. Si deux transitions restent en conflit malgré les priorités, elles restent ordonnées suivant leurs rangs de description.

Plus la priorité est grande, plus la transition est prioritaire. Par défaut, la priorité d'une transition est égale à 0. Il est possible de spécifier une priorité négative afin de définir des transitions moins prioritaire que les transitions par défaut.

La priorité est spécifiée après le mot-clef `PRIO` à l'aide d'une expression entière. La priorité peut donc être dynamique (dépendre de l'instant où elle est utilisée).

Les transitions équiprobables offrent une alternative dans la gestion des conflits. Jusqu'à présent, la gestion des conflits de transitions était déterministe. Lorsque la première transition de l'échéancier est de type équiprobable, l'échéancier est parcouru pour récupérer toutes les transitions équiprobables en conflit et une de ces transitions est choisie de manière aléatoire.

Un poids de pondération peut être associé à chaque transition équiprobable. Plus ce poids est grand, plus la probabilité de choisir cette transition parmi les transitions en conflit est importante. Par défaut, le poids de pondération d'une transition équiprobable est égale à 1.

Une transition est déclarée comme équiprobable à l'aide du mot-clef EQP éventuellement suivi d'une expression réelle (supérieure à zéro) correspondant à son poids de pondération.

2.5.8. Transitions à mémoire

Au cours de la simulation d'un système, il arrive fréquemment qu'une transition valide à un instant donné soit inhibée (suite au tir d'autres transitions) avant d'avoir été tirée. Il est important alors de savoir s'il s'agit de la disparition pure et simple de l'événement correspondant ou seulement de la suspension de celui-ci car il faudra en tenir compte si cette transition redevient valide au cours de la même histoire :

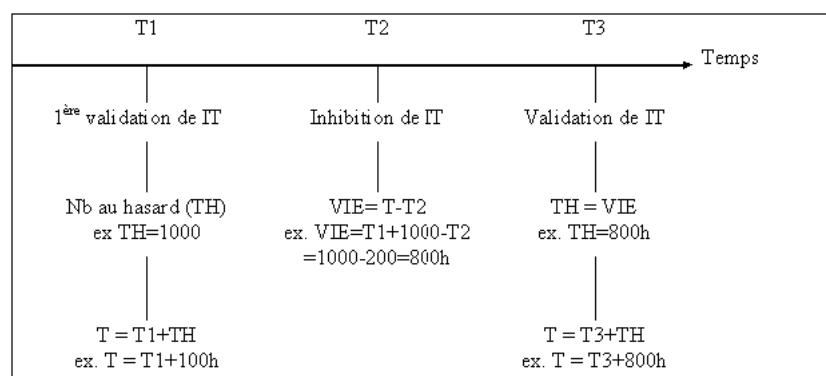
- Si la transition a été inhibée suite à la disparition de l'événement correspondant alors une nouvelle validation correspond à un nouvel événement indépendant du précédent et le délai doit être évalué par un nouveau tirage de nombre au hasard ;
- Par contre, si l'inhibition correspond seulement à la suspension de l'événement correspondant, la nouvelle validation correspond seulement à la réactivation de cet événement et il faut utiliser comme délai celui qui restait à courir au moment de la suspension.

Pour montrer plus explicitement ce que nous venons de décrire, nous allons prendre l'exemple d'une transition (IT) relative à la défaillance d'un composant.

Soit T1 l'instant où on commence à utiliser le composant ; le composant est alors neuf et IT devient valide pour la première fois (cf. figure ci-dessous) ; par tirage d'un nombre au hasard, on évalue alors TH le délai d'utilisation au bout duquel le composant tombera en panne et la transition sera effectivement tirée (par exemple, TH=1000h). La transition IT devrait donc être tirée à l'instant $T = T1 + 1000h$.

Considérons maintenant un instant T2 (p.ex. $T2 = T1 + 200h$) où, momentanément, on n'a plus besoin du composant considéré, on arrête alors de le faire fonctionner et cela se traduit dans le réseau de Petri par l'inhibition de la transition IT. À cet instant le composant a été utilisé pendant 200h et il lui reste normalement $1000 - 200 = 800h$ à vivre (soit T-T2).

Quelques temps plus tard (T3), on a de nouveau besoin du composant et la transition (IT) redevient valide. Comme elle correspond toujours à la défaillance du même composant c'est le délai de TH=800h ci-dessus qu'il faut utiliser et non réévaluer une nouvelle durée de vie TH (qui correspondrait au remplacement du composant par un composant neuf) par le tir d'un nouveau nombre au hasard.



Le processus d'inhibition-validation peut se poursuivre plusieurs fois avant que IT soit effectivement tirée. Lorsqu'elle a été tirée effectivement, le processus reprend à zéro (la validation suivante est considérée comme une première validation).

Cela étant dit, il est des fois nécessaire d'inhiber le mécanisme de la mémoire en cours de simulation. Cela revient à se poser la question de l'utilisation du temps restant à chaque validation de la transition. Il est alors possible de

choisir au moment de la validation de la transition si un nouveau délai doit être calculé ou si le délai restant est utilisé. La transition devient alors une transition à mémoire dynamique.

Une transition est déclarée "à mémoire" à l'aide du mot-clef MEM. Pour les transitions à mémoire dynamique, le mot-clef MEM est suivi d'une expression booléenne. Cette expression doit être, au moment de la validation de la transition, :

- à `true` si le délai restant doit être utilisé (mécanisme de mémoire);
- à `false` si un nouveau délai doit être calculé.

2.5.9. Limiter le nombre de tir instantané

Il est possible de limiter le nombre de tir instantané sans incrément du temps. Cela permet de traiter facilement les transitions ne devant être tirés qu'une seule fois à un temps donnée (comme les transitions avec une loi "Instant Prévu à l'Avance").

Ce mécanisme est spécifié à l'aide du mot-clef LIMIT suivi d'un entier (généralement égal à 1) qui précise le nombre de tir autorisé sans incrément du temps. Lorsque le temps de la simulation change, le compteur du nombre de tir autorisé est remis à zéro.

2.5.10. Histogramme

Il est possible de demander la mémorisation de l'histogramme des tirs d'une transition à l'aide du mot-clef HST.

2.6. Rubrique : Tableaux généraux

Le nombre de paramètres maximum des lois mises en œuvre dans les versions précédentes de Moca-RP^c était fixé à 3. Ce nombre est largement suffisant dans la plupart des cas mais pour la mise en œuvre de certaines lois complexes il était nécessaire de pouvoir indiquer un plus grand nombre de paramètres. Pour régler ce problème on utilisait alors un des trois paramètres précédents pour indiquer un numéro de tableau et le tableau en question était ensuite décrit sous une autre rubrique.

Dans la version actuelle de Moca-RP^c, le nombre de paramètres maximum des lois n'a pas été limité et le nombre de lois utilisant plus de trois paramètres a été réduit à 4. Nous avons pourtant souhaité conserver un mécanisme similaire afin de simplifier la saisie des différentes lois.

Il est possible de déclarer des tableaux (qui sont d'ailleurs plutôt des listes d'expressions) et de les utiliser lors de la description des lois des transitions. Ces tableaux sont identifiés à l'aide d'un numéro et éventuellement d'un nom.

Un tableau est déclaré de la manière suivante :

```
TB: [name] n expr1, expr2, ..., exprk;
```

où `name` est un nom optionnel, `n` est le numéro du tableau en question et `expr1, expr2, ..., exprk` sont les `k` expressions du tableau `n`.

Ne s'agissant que d'un raccourci d'écriture, ils peuvent s'utiliser à la place de toute liste d'expressions (expressions séparées par des virgules), c'est-à-dire comme paramètres des lois de délai et de tir, comme arguments des expressions `@(k[, l])(...)`, `min(...)` et `max(...)` et comme gardes des transitions.

Pour les utiliser, il suffit d'utiliser l'opérateur `$` suivi du nom ou du numéro de tableau concerné.

Exemple :

```
TB: 1 1e-3, 1e-2, 1e-3; /* Definition d'un tableau */
TB: solP 2 0.1, 0.2; /* Definition d'un autre tableau */
```

```
TR: Tr1 AM 1 1 AV 2 1, 3 1, 4 1 LOI erlgg 3,$1 TIR sol $solP; /* Utilisation des
tableaux */
```

2.7. Rubrique : Places

Les places sont automatiquement déclarées lors des descriptions des transitions et/ou des expressions utilisant le marquage d'une place. Cette rubrique ne sert qu'à associer des noms aux places du réseau de Petri. Elle est repérée par le mot-clef **PL** : . La suite consiste en une liste de la forme suivante :

```
PL: Nom1, P2 Nom2...
```

où P_i est un numéro de place et Nom_i le nom qui lui est associé.

Exemple :

```
PL: 1 Out_OK, 2 Out_NOK,
    10 A_OK, 11 A_NOK,
    20 B_OK, 21 B_NOK,
    30 C_OK, 31 C_NOK;
```

2.8. Rubrique : Initialisations

L'initialisation d'un réseau de Petri se résume en l'initialisation de ses places et de ses variables. L'initialisation commence par le mot-clé **AI** : suivi d'une liste d'affectations:

- **#<place>** = **<expr>** où **<place>** est le numéro de la place et **<expr>** est une expression constante entière. Par défaut une place contient 0 jeton.
- **<var>** = **<expr>** où **<var>** est le nom d'une variable et **<expr>** une expression constante. **<var>** et **<expr>** doivent être du même type. Par défaut une variable booléenne vaut faux et une variable entière ou réelle 0.

Exemple :

```
bool b ; /* une variable booléenne */
int c ; /* une variable entière */
TR: AM 1 1 AV 2 1;
AI: #1=1, c=3 ;
/* par défaut b vaut faux et la place 2 contient 0 jeton */
```

2.9. Rubrique : Etats statistiques (Observateurs)

La déclaration d'un état statistique utilise la syntaxe suivante :

```
ES: <name> = <expr> [<flags>];
```

2.9.1. Nom de l'état statistique

Le nom **<name>** de l'état statistique est un identificateur utilisé lors de l'affichage des résultats.

2.9.2. Expression

L'expression **<expr>**, permettant de savoir à tout instant si nous sommes ou non dans un état statistique donné, est décrit à l'aide d'une expression numérique définie dans la Section 2.4.2, « Expressions numériques ».

2.9.3. Drapeaux

Les drapeaux <flags> sont tous facultatifs, mais ils doivent être indiqués dans l'ordre ci-dessous.

- Le drapeau HST permet d'obtenir l'échantillon complet des valeurs tirées pour l'état concerné. Ceci est utile lorsque l'on ne se contente pas des résultats obtenus par défaut.
- Le drapeau CHRO permet d'obtenir un chronogramme moyen de la valeur de l'état concerné. Cela permet de visualiser le profil de vie de l'état statistique.
- Le drapeau at <float>[-]? [, <float>[-]?]* ou from <float> to <float> step <float>[-]? permet de spécifier des temps de calcul spécifique à l'état statistique considéré. Si ce drapeau n'est pas présent les temps de calcul considérés sont ceux définies à l'aide de l'option times.
- Le drapeau [TS: <num> [, <num>]*] permet de spécifier de type de statistique spécifique à l'état statistique considéré. Si ce drapeau n'est pas présent les types de statistiques considérés sont ceux définies dans la rubrique TS:

2.10. Rubrique : Type de statistiques

Moca-RP^c évalue systématiquement un certain nombre de résultats mais il est de plus possible, pour les états statistiques décrits à la rubrique précédente (2.2.8), d'obtenir des résultats spécifiques.

La présente rubrique, repérée par le mot-clef TS : , permet d'indiquer quels types de résultats doivent être évalués sur ces états.

Moca-RP^c autorise 7 types de statistiques :

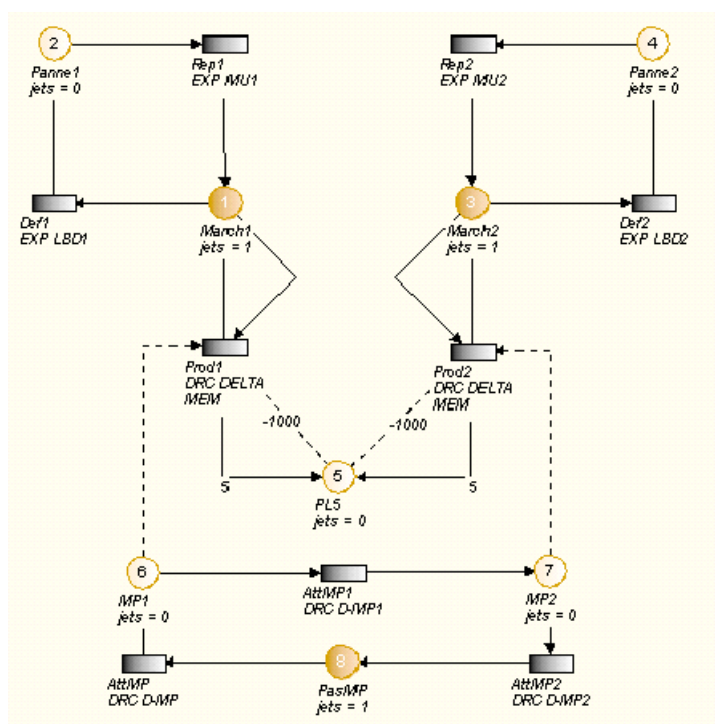
1. Temps moyen de séjour cumulé passé dans les états : De loin le plus utilisé car il permet, en particulier, les évaluations de disponibilité de production (productivité) des installations.
2. Moyenne de présence d'états à la fin d'une histoire : Destiné à la réalisation des calculs de fiabilité ; Dans ce cas, on regarde si l'état de panne (absorbant) est présent en fin d'histoire.
3. Valeur moyenne de l'expression de l'état à la fin d'une histoire : Le pendant pour un état statistique du nombre moyen de jetons en fin d'histoire pour une place.
4. Nombre de passages moyens par les états au cours d'une histoire : Destiné au dénombrement des occurrences d'événements particuliers au cours d'une histoire.
5. Valeur moyenne de l'expression de l'état au cours d'une histoire : Le pendant pour un état statistique du marquage moyen pour une place.
6. Date moyenne de première arrivée dans les états et nombre total de données non censurées, c'est-à-dire de dates inférieures à la durée d'une histoire : Permet d'obtenir des renseignements sur l'instant où un système tombe en panne pour la première fois. Ceci permet de réaliser des calculs de fiabilité et d'évaluer le temps moyen de bon fonctionnement du système.
7. Valeur moyenne de l'expression de l'état pour chaque intervalle de temps : A été intégré afin d'afficher la production moyenne pour chaque année de production. Doit être utilisé avec une liste de temps (dans le cas contraire, le résultat est identique à un type de statistique 5). La différence entre 2 temps doit être strictement supérieure à epsilon (une liste de temps incluant t et t-epsilon rendra impossible le lancement de la simulation).

Il est possible de préciser plusieurs types de statistiques sur tous les états statistiques en séparant les types de statistiques par une virgule.

L'exemple suivant définit les trois types de statistique 1, 2 et 4.

```
TS: 1, 2, 4 ;
```

2.11. Exemple d'un réseau de Petri



```
net RdpTyp1 {

/* Description des options */
  OPT: title "1er exemple de réseaux de Petri typiques (JPS)" ;
  OPT: duration      8760.00 ; /* Une année */

/* Données générales */
  const float DELTA = 1.; /* Vitesse de production */

/* Premier train de production */
  const float LBD1 = 1e-3;
  const float MU1 = 1e-2;
  TR: Rep1 AM 2 1 AV 1 1 LOI exp MU1 ;
  TR: Def1 AM 1 1 AV 2 1 LOI exp LBD1 ;
  TR: Prod1
    AM 1 1, 5 -1000, 6 -1
    AV 1 1, 5 5
    LOI drc DELTA MEM;
  PL: 1 March1, 2 Panne1;
  AI: #1 = 1;

/* second train de production */
  const float LBD2 = 1e-3;
  const float MU2 = 1e-2;
  TR: Rep2 AM 4 1 AV 3 1 LOI exp MU2 ;
  TR: Def2 AM 3 1 AV 4 1 LOI exp LBD2 ;
  TR: Prod2
    AM 3 1, 5 -1000, 7 -1
    AV 3 1, 5 5
    LOI drc DELTA MEM;
  PL: 3 March2, 4 Panne2;
  AI: #3 = 1;

/* Maintenance préventive */
  const float D-MP = 800.;
  const float D-MP1 = 100.;
  const float D-MP2 = 100.;
  TR: AttMP1 AM 6 1 AV 7 1
```

```

      LOI drc D-MP1 ;
TR: AttMP2 AM 7 1 AV 8 1
      LOI drc D-MP2 ;
TR: AttMP AM 8 1 AV 6 1
      LOI drc D-MP ;
PL: 6 MP1, 7 MP2, 8 PasMP;
AI: #8 = 1;

/* Description des autres places */
      PL: 5 Stock;
};

```

2.12. Résultats attendus

Les résultats fournis par le logiciel sont les moyennes, écart-types et intervalles de confiance à 90% des mesures suivantes :

1. Le nombre de tirs pour chaque transition (résultat fourni par défaut) ;
2. Le temps de séjour cumulé dans chaque place (résultat fourni par défaut) ;
3. Le marquage de chaque place (résultat fourni par défaut) ;
4. Le marquage en fin d'histoire de chaque place ;
5. Le temps cumulé dans un état statistique ;
6. La date de première occurrence d'un état statistique ;
7. Le nombre d'occurrences d'un état statistique ;
8. La présence d'un état statistique en fin d'histoire ;
9. L'histoire des mesures sur les états et transitions ;

Pour les états statistiques, des échantillons sont constitués au fur et à mesure du déroulement des histoires. Pour chaque état spécifié, on obtient ainsi un échantillon qui peut être caractérisé par sa moyenne (m) et sa variance (σ^2). La racine carrée de la variance constitue l'écart-type (σ) de l'échantillon. Au fur et à mesure que le nombre d'histoires grandit, l'échantillon grandit aussi et les paramètres m et σ convergent vers des valeurs limites qui correspondent aux résultats recherchés. Plus les valeurs de l'échantillon sont dispersées et plus σ est grand. Si on représente l'échantillon sous la forme d'un histogramme, celui-ci sera d'autant aplati que σ sera grand.

Un autre paramètre indispensable à connaître pour appréhender la qualité de la simulation réalisée est l'évaluation de la confiance que l'on peut attribuer au résultat. Si on considère la moyenne m de l'échantillon, il s'agit d'une estimation de la moyenne *vraie* M . Au fur et à mesure que l'échantillon s'étoffe, l'estimation m doit se rapprocher de plus en plus de la valeur *vraie* M . Comme M est une valeur déterministe, la qualité de l'estimation peut être mesurée par l'intervalle dans lequel se trouve l'estimation m par rapport à la valeur *vraie* M . Ceci s'exprime à l'aide de la notion de confiance $C(e)$ à $\alpha\%$:

$$C(e) = \text{Probabilité}\{M - e \leq m \leq M + e\} = \alpha$$

La formule ci-dessus exprime que la probabilité que la moyenne estimée m se trouve dans l'intervalle $[-e, +e]$ autour de la valeur *vraie* M est de 0.9.

Plus l'intervalle $[-e, +e]$ sera petit, plus l'estimation m de M sera précise. En théorie, pour un nombre infini d'histoires cet intervalle devient nul puisque M est une valeur déterministe.

La fonction $C(e)$ s'exprime en fonction de la fonction $erf(.)$ tabulée dans les tables relatives à la loi de Gauss. Cela permet de démontrer que $C(e)$ est égale à 90% si l'on pose $e = 1.64\sigma/\sqrt{N}$. Dans cette formule σ est l'écart-type de l'échantillon, N est le nombre d'histoires réalisées, et 1.64 un coefficient donné par des tables.

Moca-RP^c évalue systématiquement la valeur de e correspondant à un intervalle de confiance à 90% afin que l'utilisateur puisse avoir une idée de l'erreur liée à la simulation. Il est à noter que e varie selon la racine carrée du nombre d'histoires, donc très lentement. Il en résulte que pour obtenir des résultats précis pour des événements peu probables, un très grand nombre d'histoires sera nécessaire.

3. Interpréteur de commandes

3.1. Introduction

La version 10 de Moca-RP^C fonctionne de manière batch. On spécifie, sur la ligne de commande appelant le logiciel, le fichier d'entrée à traiter, le fichier de sortie et un certain nombre de directives optionnelles. Le logiciel traite alors le fichier d'entrée, génère le fichier de sortie et rend la main. Pour refaire une simulation, il faut modifier le fichier d'entrée et relancer le calcul.

Les versions 12 (et supérieures) de Moca-RP^C dialogue avec les utilisateurs à l'aide d'un interpréteur de commandes.

Celui-ci permet :

- La création et la gestion de plusieurs réseaux de Petri
- Le chargement des réseaux au format Moca-RP^C version 10.04 et précédents
- Le chargement des réseaux au format Moca-RP^C
- La sauvegarde (et / ou l'affichage) des réseaux au format Moca-RP^C
- L'arrêt et la reprise d'une simulation permettant ainsi la vérification d'un RdP sur un petit nombre d'histoires, puis le lancement de la simulation pour un grand nombre d'histoires (sans perdre les résultats précédents).
- L'affichage de tout ou d'une partie des résultats vers un ou plusieurs fichiers
- Une aide en ligne, une commande générique afin d'accéder au système, une gestion de chronomètres, ...

3.2. Lancement de Moca-RP^C

Pour lancer le logiciel, il faut à partir d'un shell (fenêtre de commandes Dos dans le cas d'un système Windows) lancer l'exécutable Moca13 (Moca13.exe sous Windows). L'invite de l'interpréteur de commandes Moca-RP^C s'affiche alors à l'écran.

```
Moca-RPc : propriete du groupe Total
Version 13.21 (Sep 14 2009 - 16:35:17)

  Pour obtenir de l'aide, saisissez la commande "help;"

Moca >
```

Vous pouvez à partir de là exécuter des commandes Moca.

3.2.1. Exemple d'une session Moca-RP^C

Un interpréteur de commandes permet à l'utilisateur d'effectuer des requêtes au logiciel qui répond en fonction de la nature de la question posée et des commandes qu'il a déjà traitées.

L'exemple ci-dessous correspond à une copie de la console DOS. Les commandes saisies par l'utilisateur sont affichées en gras.

```
01 D:\BASIC\TEST>Moca13
02
03 Moca-RPc : propriete du groupe Total
04 Version 13.21 (Sep 14 2009 - 16:35:17)
05
06   Pour obtenir de l'aide, saisissez la commande "help;"
07
08
09 Moca > help;
```

```

10 |-----
11 | Moca [Version 13.21] : aide en ligne
12 |-----
13 | L'interface utilisateur de Moca est un interpreteur de commandes.
14 | Les reseaux sont charges a l'aide de "parser" externe non
15 | documente dans cette aide en ligne.
16 | Des commandes permettent de manipuler ces donnees (afficher,
17 | supprimer, ...).
18 |
19 | Les commandes principales sont les suivantes :
20 |   exit, load, save, set, clear, timer, system
21 |   simul, display, redirection,
22 |
23 | D'autres mots clefs sont aussi utilises
24 |   trace, seed, of, stop, start, variable, prompt, information
25 |   precision, language, format, options, result
26 |
27 | Voir 'help <commande>;' pour une aide sur la commande donnee.
28 |
29 | Moca peut etre appele avec des noms de fichiers comme arguments.
30 | Ces fichiers sont lus avant que l'interpreteur de commande ne
31 | démarre.
32 |-----
33 |
34 | Moca > load "drc.mok";
35 | Chargement du fichier drc.mok
36 | ## parser Moca at line 2
37 | ## endparser at line 51
38 |
39 | Moca > display net all;
40 | display net ;
41 | /*
42 |   1 drc
43 | */
44 |
45 | Moca > simul drc {story 100};
46 | Temps de simulation : 3.98 (3.99)
47 |
48 | Moca > display result drc > "File.res";
49 |
50 | Moca > exit;
51 |
52 | D:\BASIC\TEST>

```

Dans un premier temps, on exécute le logiciel à partir du chemin en cours (ligne 1). L'interpréteur de commandes est alors opérationnel. Comme cela est spécifié dans la bannière de lancement de Moca, la commande **help** (ligne 9) permet d'afficher le début de l'aide en ligne. Le chargement du fichier `drc.mok` est effectué sur la ligne 34. La commande **display net all;** (ligne 39) nous informe qu'un réseau de Petri nommé `drc` a été chargé en mémoire. Cent histoires sont alors simulées sur le réseau `drc` à l'aide de la commande **simul** (ligne 45). Les résultats statistiques de ces simulations sont imprimés vers le fichier `File.res` à l'aide de la commande **display result** (ligne 48). Pour finir, la commande **exit** permet de quitter l'interpréteur de commandes.

3.2.2. Lancement de Moca-RP^C en batch

Lorsque l'on souhaite lancer Moca-RP^C en batch, il suffit d'écrire dans un fichier (par exemple `cmd.mok`) toutes les commandes que l'on souhaite exécuter, puis on lance Moca-RP^C à l'aide de la commande **Moca1301 < cmd.mok**

Il convient de ne pas oublier de mettre la commande **exit;** à la fin du fichier de commande.

3.3. Commandes de base

3.3.1. Commentaires

Les commentaires dans Moca-RP^C ont la même syntaxe que dans le langage C. Ils commencent par les caractères `/*` et finissent sur les caractères `*/`. Ils peuvent être étendus sur plusieurs lignes.

3.3.2. Aide en ligne {help ...}

Une aide en ligne est disponible directement dans Moca-RP^c en saisissant la commande **help**. Elle est segmentée en différentes rubriques correspondant aux mots clefs du langage.

La commande **help display**; affiche l'aide en ligne associé aux commandes **display**.

3.3.3. Chargement de fichiers {load ...}

On peut charger un fichier de commande à l'aide de la commande **load "<file>"**. Les commandes de ce fichier doivent avoir la même syntaxe que les commandes Moca-RP^c. Le nom du fichier doit être mis entre guillemets.

3.3.4. Sauvegarde de fichier {save ...}

On a la possibilité de sauvegarder l'ensemble des réseaux et des options de Moca-RP^c en mémoire à l'aide de la commande **save "<file>"**;

Les données sauvegardées sont les suivantes :

- Les réseaux au format Moca-RP^c
- Les différentes options

Le nom du fichier doit être mis entre guillemets.

3.3.5. Gestion des différents formats de données {##parser ...}

L'interpréteur de commandes ne gère pas directement le chargement des réseaux de Petri. Il se décharge pour cette opération sur des "Parseurs" spécifique à un format de donnée.

Actuellement, il existe trois parseurs supporté par Moca-RP^c :

- Le parseur Moca12/Moca permettant de charger des fichiers de versions 12 et suivante de Moca-RP^c (Cf. Section 2, « Format des Réseaux de Petri de Moca-RP^c » pour plus d'informations sur ce format),
- Le parseur Moca10 permettant de charger des fichiers des précédentes versions de Moca-RP^c (Ce format est décrit dans [5]. Cf. Annexe A, *Lecture de fichiers au format Moca10* pour la description des différentes options d'ouverture du fichier).
- Le parseur Aralia permettant de charger des fichiers au format Aralia. (Cf. Annexe B, *Lecture de fichiers au format Aralia* pour la description de différentes options d'ouverture du fichier).

La commande **display parser**; permet d'afficher tous les parseurs de données disponibles dans Moca-RP^c avec dans la mesure du possible un rappel sur la syntaxe de la ligne de commande.

Pour ouvrir un fichier dans un format, il faut utiliser la commande **load <parser> "<file>" ["<options>"]**; où <parser> est un parseur supporté par Moca-RP^c, <file> le nom du fichier à charger et "<options>" des options spécifiques au parseur.

Il est également possible d'ouvrir directement au sein de l'interpréteur une session afin de saisir un réseau de Petri à l'aide d'un parseur dans un format spécifique. L'ouverture de la session se fait à l'aide de la commande **##parser <parser> [<options>]** où <parser> est un parseur supporté par Moca-RP^c (la fin de ligne permet de spécifier des options de création spécifique au parseur). Il est alors possible de saisir directement le réseau de Petri au format spécifié. La commande **##endparser** permet de fermer cette session. Le texte saisi pendant la session est alors interprété et des éventuels messages d'avertissement ou d'erreur sont affichés.

La session suivante permet de définir un réseau de Petri nommé **First** composé de 2 transitions et de 2 places.

```
Moca > ## parser Moca

net First {
  TR: def AM 1 1 AV 2 1 LOI exp 1e-3;
  TR: rep AM 2 1 AV 1 1 LOI exp 1e-2;
```

```
};  
  
## endparser  
## parser Moca at line 1  
## endparser at line 8  
  
Moca >
```

3.3.6. Réinitialisation {clear all;}

Il est possible de réinitialiser l'interpréteur à l'aide de cette commande. Tous les réseaux chargés en mémoire, toutes les banques de résultats (Cf. Section 3.5, « Commandes de gestion des résultats »), tous les formats d'affichage (Cf. Section 3.6, « Commandes de gestion des formats d'affichage ») sont supprimés. Par contre, les options de l'interpréteur (Cf. Section 3.7.3, « Options de l'interpréteur de commandes {display options |set ...} ») sont conservées.

3.3.7. Sortie de l'interpréteur {exit ...}

Pour finir correctement une session Moca-RP^C, il faut utiliser la commande **exit**;

3.3.8. Redirection de l'affichage

Toutes les commandes d'affichage (display) peuvent être redirigées vers un fichier ASCII.

Si le point-virgule terminant la commande est précédé par > "<file>" (respectivement par >> "<file>"), les résultats sont affichés (respectivement ajoutés) dans le fichier indiqué. Sinon ils sont affichés sur la sortie standard.

display result drc > "result.res"; crée un fichier `result.res` (l'écrase s'il existait déjà) et affiche dans celui-ci les résultats courants du réseau drc.

3.4. Commandes de gestion des réseaux

3.4.1. Affichage d'un réseau {display net ...}

Il est possible d'afficher le nom de l'ensemble des réseaux en mémoire à l'aide de la commande **display net all**;

Il est de plus possible d'afficher un réseau de Petri en mémoire à l'aide de la commande **display net <id-net>**; où <id-net> est le nom d'un réseau de Petri en mémoire.

3.4.2. Modification d'un réseau {set net ...}

Il est intéressant et souvent nécessaire de devoir modifier à la volée un réseau en mémoire à l'aide de l'interpréteur de commandes.

Notons que la structure du réseau de Petri ne peut pas être modifiée.

En revanche, les données suivantes sont modifiables :

- Le nom et le titre d'un réseau,
- La durée de simulation,
- La valeur des paramètres nommés,
- L'état initial (marquage initial des places et valeur initiale des variables)
- Les états statistiques qui ne sont que des observateurs du réseau

Pour modifier un réseau de Petri, il faut entrer dans un mode particulier (un sous-shell) à l'aide de la commande **set net <id-net>**; où <id-net> est le nom d'un réseau de Petri à modifier.

Le sous-shell affiche un prompt de type `Modif[<id-net>]` et accepte les commandes suivantes :

- **exit**; permet de sortir du sous-shell

- **display**; affiche le réseau actuel
- **duration** <float>; modifie la durée de simulation
- **name** <id-net>; modifie le nom du réseau
- **title** "<string>"; modifie le titre du réseau (attention <string> est une chaîne de caractère entouré de guillemets)
- **state add** [<id-name>] "<expr>" [hst]; ajoute un état statistique (nommé <id-name>) définit à l'aide de l'expression <expr>. Pour plus d'information concernant la syntaxe de l'expression, cf. Section 2.4.2, « Expressions numériques ».
- **state add-expr** [hst]; ajoute un état statistique pour chaque variable du modèle.
- **state del** {all | <int>}; supprime tous les états statistiques (all) ou le <int>^{ème} état statistique.
- **state hst** {all | <int>} <value>; modifie l'histogramme de tous les états statistiques (all) ou le <int>^{ème} état statistique suivant <value> (si <value> = 0 on supprime l'histogramme, sinon on l'ajoute).
- **state times** {all | <int>} <times>; modifie les temps de calcul de tous les états statistiques (all) ou le <int>^{ème} état statistique. Cf. option times pour la syntaxe de <times>
- **state type** {all | <int>} <num> [, <num>]*; modifie les types de statistiques de tous les états statistiques (all) ou le <int>^{ème} état statistique. (Cf. Section 2.10, « Rubrique : Type de statistiques »)
- **init** <id-var> = <value>; modifie la valeur initiale d'une variable ou la valeur d'un paramètre nommé (constante).
- **init** #<num> = <value>; modifie le marquage initial de la place <num>.

3.4.3. Suppression d'un réseau de Petri {clear net ...}

Pour supprimer un réseau de Petri, il suffit d'utiliser la commande **clear net** <id-net>; où <id-net> est le nom d'un réseau de Petri à supprimer.

Pour supprimer tous les réseaux de Petri en mémoire, il faut lancer la commande **clear net all**;

3.4.4. Lancement d'une simulation {simul ...}

Cette commande permet de lancer une simulation de Monte Carlo de N histoires sur un réseau de Petri donné. Les paramètres de cette commande sont nombreux mais, en général, optionnels.

A tout instant l'utilisateur peut suspendre la simulation en cours afin de consulter les résultats. Pour se faire, l'utilisateur doit taper la séquence de touches **CTRL-C** ; le programme termine alors l'histoire courante.

3.4.4.1. Syntaxe de la commande

```
simul <id-net> [= <result>] {
  story [until] <nbr-stories>
  [delay <delay>]
  [max-loop <max-loop>]
  [seed <seed>]
  [tries [until] <nbr-tries> {
    [seed <seed-tries>]
    [inc-seed <inc-seed-tries>]
    [trace <trace-tries>]
    [merge-histo]
  } ]
  [batch <size-batch> [{
    [trace <trace-batch>]
    [inc-seed <inc-seed>]
  } ] ]
  [progress [<nbr-class>] ["<format>"]]
} ;
```

Les paramètres obligatoires sont :

- <id-net>(identificateur) est le nom d'un réseau de Petri à simuler

- `<nbr-stories>`(entier) est le nombre d'histoires que l'on souhaite simuler. S'il est précédé du mot-clé `until`, cela signifie que l'on souhaite simuler jusqu'à la `<nbr-stories>` histoire.

Les paramètres généraux optionnels sont :

- `<result>`(identificateur) est la banque où seront stockés les résultats. (Cf. Section 3.5, « Commandes de gestion des résultats »). Lorsque cette option n'est pas spécifiée, la banque de résultats par défaut `def` est utilisée. Si elle n'existe pas, elle est créée.
- `<delay>`(entier) permet de spécifier le temps d'exécution maximal de Moca-RP^c pour effectuer cette simulation. Au-delà de ce temps, la commande termine l'histoire courante et arrête la simulation.

La valeur par défaut de ce paramètre est fixée dans les options de l'interpréteur (Cf. Section 3.7.3, « Options de l'interpréteur de commandes { **display options** | **set ...** } »)

- `<max-loop>`(entier) permet de spécifier le nombre de transitions qui peuvent être tirées successivement sans que le temps de l'histoire soit incrémenté. Au-delà de ce nombre la simulation est brutalement interrompue, la boucle et l'échéancier sont affichés.

La valeur par défaut de ce paramètre est fixée dans les options de l'interpréteur (Cf. Section 3.7.3, « Options de l'interpréteur de commandes { **display options** | **set ...** } »)

- `<seed>`(réel) permet de spécifier la graine du générateur de nombres au hasard. Cette donnée permet en particulier de rendre reproductible la simulation. Il est conseillé d'utiliser un nombre réel assez grand et impair.

Lorsque la simulation courante n'est que la suite d'une précédente simulation (banque de résultats ayant un nombre d'histoires supérieur à 0) ce paramètre est ignoré.

La valeur par défaut de ce paramètre est fixée dans les options de l'interpréteur (Cf. Section 3.7.3, « Options de l'interpréteur de commandes { **display options** | **set ...** } »)

- Il est possible d'afficher la progression de la simulation à l'aide du mot-clé `progress`. Les options de ce paramètre sont le nombre de classe d'affichage (`<nbr-class>` par défaut cette valeur est fixée à 100) et le format d'affichage du message de progression (`<format>` par défaut égal à "`%P%`" ; `%P` sera remplacé par l'état d'avancement en pourcentage et `%` sera remplacé par le caractère `'%`').

3.4.4.2. Cas d'une simulation double-détente

Dans Moca-RP^c, une simulation double-détente consiste à tirer un jeu de paramètres au hasard et de réaliser `<nbr-stories>` histoires avec ce jeu, puis de recommencer `<nbr-tries>` fois avec `<nbr-tries>` autres jeux de paramètres. Ainsi on obtient un échantillon de `<nbr-tries>` moyennes que l'on traite de manière classique.

Nous avons décidé de scinder le générateur de nombres aléatoires utilisé pour générer le jeu de paramètres (`<seed-tries>`) du générateur de nombre aléatoire utilisé pour réaliser le nombre d'histoires (`<seed>`). De plus, afin de rendre facilement parallélisable ce type de simulation, un incrément de la graine initiale (`<inc-seed-tries>`) a été ajouté aux paramètres de ce type de simulation.

L'algorithme lors d'une simulation double-détente est alors le suivant :

```
Seed_tries = <seed-tries> ;
Seed_story = <seed> ;
Pour i de 0 à <nbr-tries> :
    GenRandom_SetSeed(Seed_tries) ;
    Pour chaque expression faisant intervenir des générateurs de nombres aléatoires
    (nlog, unif ou norm) :
        Initialiser la valeur de l'expression en fonction de sa distribution et de ses
        paramètres
        GenRandom_SetSeed(Seed_story) ;
        Pour j de 0 à <nbr-story>
            Faire une histoire
        Seed_story += <inc-seed-tries> ;
    Seed_tries += <inc-seed-tries> ;
```

Les fonctions `GenRandom_GetSeed()` et `GenRandom_SetSeed(seed)` permettent respectivement de récupérer et de fixer la graine du générateur de nombres aléatoires.

Notons que la graine du générateur de nombres aléatoires utilisé pour générer le $i^{\text{ème}}$ jeu de paramètres est égal à `<seed-tries> + (i-1)*<inc-seed-tries>`.

Paramètres utilisés lors d'une simulation double-détente tries { ... } :

- `<nbr-tries>`(entier) est le nombre de jeux de paramètres que l'on souhaite simuler. S'il est précédé du mot-clé `until`, cela signifie que l'on souhaite simuler jusqu'au `<nbr-tries>` jeux de paramètres.
- `<seed-tries>`(réel) permet de spécifier la graine du générateur de nombres au hasard permettant de générer les jeux de paramètres. Cette donnée permet en particulier de rendre reproductible la simulation. Il est conseillé d'utiliser un nombre réel assez grand et impair.
- `<inc-seed-tries>`(réel) permet de spécifier l'incrément de la graine du générateur de nombres aléatoires utilisé pour générer chaque nouveau jeu de paramètres.
- `<trace-tries>`(format d'affichage) permet d'afficher les résultats intermédiaires issus de chaque jeu de paramètres ; c'est à dire une fois les `<nbr-stories>` histoires simulées ; dans le format spécifié (Cf. Section 3.6, « Commandes de gestion des formats d'affichage »).
- `<merge-histo>` permet d'ajouter l'histogramme des états statistiques des `<nbr-stories>` histoires au sein de l'histogramme du jeu de paramètres courant. Dans le cas contraire, seule la valeur moyenne de l'état statistique est ajoutée à l'histogramme du jeu de paramètres courant.

3.4.4.3. Cas d'une simulation par lots

Il est possible dans Moca-RP^c de faire des simulations par lots. Dans l'absolu, cette fonctionnalité n'apporte strictement rien. En effet, au lieu de faire une simulation *linéaire*, on découpe le nombre d'histoires à simuler `<nbr-stories>` en lots de taille identique `<size-batch>`. Pour chaque lot, on fait une simulation classique et on regroupe les résultats provenant des différents lots afin d'obtenir le résultat final de la simulation.

En fait l'intérêt de cette approche est de pouvoir facilement paralléliser la simulation. Il est effectivement possible de distribuer la simulation à différentes ressources de calcul (processeur et/ou poste de calcul) en leur demandant de traiter différents lots d'histoires. C'est la raison qui nous a poussés à implémenter ce type de simulation, en local dans un premier temps, en distribué dans un second temps.

L'algorithme lors d'une simulation par lot est le suivant :

```
Nbr_batch = <nbr-story> / <size-batch> ;
Seed_story = <seed> ;
Pour i de 0 à Nbr_batch :
    GenRandom_SetSeed(Seed_story) ;
    Pour j de 0 à <nbr-story>
        Faire une histoire
    Ajouter au résultat courant le résultat de ce lot d'histoire
    Seed_story += <inc-seed> ;
```

La manière de calculer la moyenne et la variance pour des échantillons par lots utilisée dans Moca-RP^c est décrite en annexe (Cf. Section 2.3, « Résultats »)

Paramètres utilisés lors d'une simulation par lot { ... } :

- `<size-batch>`(entier) est le nombre d'histoire par lot.
- `<inc-seed>`(réel) permet de spécifier l'incrément de la graine du générateur de nombre aléatoire utilisé lors de la simulation de chaque lot.

- `<trace-batch>`(format d'affichage) permet d'afficher les résultats intermédiaires issus de chaque lot d'histoires dans le format spécifié (Cf. Section 3.6, « Commandes de gestion des formats d'affichage »).

3.4.4.4. Cas d'une simulation double-détente par lot

L'idée est de coupler les deux types de simulation précédents.

L'algorithme mis en œuvre est alors le suivant :

```
Seed_tries = <seed-tries> ;
Nbr_batch = <nbr-story> / <size-batch> ;
Pour i de 0 à <nbr-tries> :
    GenRandom_SetSeed(Seed_tries) ;
    Pour chaque expression faisant intervenir les générateurs de nombres aléatoires
    (nlog, unif ou norm) :
        Initialiser la valeur de l'expression en fonction de sa distribution et de ses
        paramètres
        Seed_story = <seed> + i * <inc-seed-tries>;
        Pour j de 0 à Nbr_batch :
            GenRandom_SetSeed(Seed_story) ;
            Pour k de 0 à <nbr-story>
                Faire une histoire
            Ajouter au résultat courant le résultat de ce lot
            Seed_story += <inc-seed> ;
        Seed_tries += <inc-seed-tries> ;
```

Tous les paramètres ont déjà été présentés.

Notons que le paramètre `<inc-seed-tries>` sert :

- d'incrément pour le générateur déterminant le jeu de paramètres
- d'incrément pour le générateur déterminant la simulation classique.

3.4.4.5. Arrêt de la simulation de manière externe {set simul locker ...}

Afin de permettre d'arrêter la simulation sans envoyer de signal SIGINT (équivalent d'un Ctrl+C - ce qui est impossible à faire en Java par exemple), il est maintenant possible d'arrêter la simulation lorsque la présence d'un fichier lock est avérée.

Pour activer cette fonctionnalité, il faut utiliser la commande **set simul locker "<file>"**; où `<file>` est le nom d'un fichier entouré par des guillemets. Si ce fichier existe la simulation s'arrête.

Pour désactiver cette fonctionnalité, il suffit de lancer la commande **set simul locker;** (sans nom de fichier).

3.5. Commandes de gestion des résultats

3.5.1. Introduction

Les différentes évolutions de Moca-RP^c ont conduit à introduire des banques de résultats. Une banque de résultats est une structure de données contenant suffisamment d'informations pour poursuivre une simulation.

Les informations stockées au sein de ces banques de résultats sont :

- Les informations sur l'état de la simulation à terminaison ; A savoir le nombre d'histoire jouée, la graine du générateur de nombre aléatoire lors de l'arrêt de la simulation, ...
- L'ensemble des estimateurs statistiques (moyenne, écart type, donnée censuré) pour les transitions, places et observateurs aux différents temps observés.
- L'ensemble des histogrammes (transitions et observateurs).
- L'ensemble des chronogrammes (observateurs).

Dans les versions précédentes la version 12.17, Moca-RP^C utilisait ces structures de données

- pour mémoriser les résultats de chaque lot d'histoires (traitement batch)
- pour mémoriser les résultats issus de la simulation d'un jeu de paramètres lors d'une simulation double-détente.
- pour sauvegarder/charger ces résultats sous la forme de fichier binaire utilisé lors des échanges de données entre les serveurs de calcul et le client lors d'un calcul distribué.

Le mécanisme de calcul distribué interne à Moca-RP^C (à l'époque basé sur le protocole Corba) n'a pas été maintenu.

En revanche, les commandes permettant d'enregistrer/charger des banques de résultat, ainsi que de synthétiser les différentes banques de résultats ont été conservées.

Un réseau de Petri peut donc utiliser une ou plusieurs banques de résultats. Il n'y a pas d'interdépendance entre ces banques.

3.5.2. Accès aux différentes banques de résultats

L'accès à une banque de résultats ne se fait qu'à travers le réseau dont elle dépend. Pour accéder à la banque de résultat `bank1` du réseau `PNet1`, il suffit d'écrire `PNet1 => bank1`.

3.5.3. Affichage des résultats {result display ...}

Pour afficher les résultats associés à une banque de résultats, il faut utiliser la commande **result display** `<access-result>` [`<format-result>`]; où `<access-result>` est un accès à une banque de résultats et `<format-result>` est un format d'affichage (Cf. Section 3.6, « Commandes de gestion des formats d'affichage »). Si ce dernier paramètre n'est pas présent, le format d'affichage par défaut est utilisé.

Il est également possible de connaître toutes les banques de résultats d'un réseau à l'aide de la commande **result display** `<id-net> all`; où `<id-net>` est le nom d'un réseau de Petri en mémoire.

3.5.4. Chargement et sauvegarde de résultats {result load/save ...}

Il est possible de sauvegarder/charger une banque de résultats dans un fichier binaire. L'intérêt de cette approche réside dans la possibilité d'arrêter une simulation en cours et de pouvoir la reprendre plus tard sans perdre les simulations déjà effectuées.

Pour sauvegarder une banque de résultats, il suffit d'utiliser la commande **result save** `<access-result>` "`<file>`"; où `<access-result>` est un accès à une banque de résultats et `<file>` un nom de fichier entouré de guillemets.

Pour recharger une banque de résultats, il faut avoir au préalable le réseau en mémoire et utiliser la commande **result load** `<id-net>` "`<file>`"; où `<id-net>` est le nom du réseau de Petri et `<file>` un nom (entouré de guillemets) du fichier binaire contenant une banque de résultats du réseau considéré.

3.5.5. Suppression d'une banque de résultats {result clear ...}

Pour supprimer une banque de résultat, il suffit d'utiliser la commande **result clear** `<access-result>`; où `<access-result>` est un accès à la banque de résultats à supprimer.

Pour supprimer toutes les banques de résultats d'un réseau, il faut lancer la commande **result clear** `<id-net> all`; où `<id-net>` est le nom du réseau de Petri considéré.

3.5.6. Fusion de 2 banques de résultats {result merge ...}

Cas d'une simulation par lot

La commande **result merge** `<id-net>=><id-result1> <id-result2>`; (où `<id-net>` est le nom du réseau de Petri considéré et `<id-result1>` et `<id-result2>` 2 banques de résultats) permet de fusionner les 2 banques de résultats. Le résultat de cette fusion est stocké dans la première banque.

La fusion est réalisée en fonction de la nature des données à fusionner.

- Pour les estimateurs statistiques (moyenne, écart type, donnée censuré), il est possible de fusionner ces informations sans connaître la valeur de chaque histoire de chaque lot. Pour plus d'information, voir Section 2, « Lors de la fusion de 2 lots »
- Pour les histogrammes, il suffit de mettre bout à bout les histogrammes des 2 lots.
- Pour les chronogrammes, une fonction a été développée pour fusionner 2 chronogrammes de poids différents (en effet le nombre d'histoire associé à chaque lot n'est pas forcément le même). La diminution du nombre de points du chronogramme se fait après la fusion à proprement parlé.

3.5.7. Ajout d'une banque dans une autre {result add ...}

Cas d'une simulation double-détente

La commande **result add <id-net>=><id-result1> <id-result2>**; (où <id-net> est le nom du réseau de Petri considéré et <id-result1> et <id-result2> 2 banques de résultats) permet d'ajouter les résultats de la seconde banque dans la première.

L'objectif étant de traiter le cas d'une simulation double-détente, les 2 banques de résultat n'ont pas la même signification. La première banque permet de stocker les résultats des différents jeux de paramètres. La seconde correspond aux résultats associés à un jeu de paramètres.

L'ajout est réalisé en fonction de la nature des données à prendre en compte.

- Pour les estimateurs statistiques (moyenne, écart type, donnée censuré), la moyenne est utilisée comme valeur unitaire du jeu de paramètre.
- Les histogrammes des transitions provenant du jeu de paramètres ne sont pas pris en compte. Les histogrammes des états statistiques du jeu de paramètre sont en revanche pris en compte.
- Le chronogramme moyen issu d'un jeu de paramètres est vu comme le chronogramme d'une seule histoire. Il est donc fusionné avec un poids de 1.

3.5.8. Exemple d'un traitement par lot à l'aide de banques de résultat

L'exemple qui suit montre l'utilisation des banques de résultats pour simuler une simulation par lot, c'est-à-dire faire la même chose qu'une simulation par lot (Cf. Section 3.4.4.3, « Cas d'une simulation par lots »), mais avec uniquement des simulations standards et des manipulations sur les banques de résultats.

```
01 simul PetriNet3=>Lot1 {
02   story 500
03   seed 12345679};
04 result merge PetriNet3=>Result1 Lot1;
05 simul PetriNet3=>Lot2 {
06   story 500
07   seed 12345802};
08 result merge PetriNet3=>Result1 Lot2;
09 simul PetriNet3=>Lot3 {
10   story 500
11   seed 12345925};
12 result merge PetriNet3=>Result1 Lot3;
13
14 simul PetriNet3=>Result2 {
15   story 1500
16   seed 12345679
17   batch 500 {inc-seed 123} };
```

Les commandes ligne 01, 05 et 09 permettent de faire des simulations de 500 histoires avec des graines de nombre aléatoire différentes. Les résultats de ces simulations sont stockés dans les banques de résultats Lot1, Lot2 et Lot3.

Les commandes ligne 04, 08 et 12 permettent de synthétiser les différentes banques de résultat au sein de la banque Result1.

La commande ligne 14 est la commande équivalente.

L'intérêt est de pouvoir faire la simulation de chaque lot sur différentes ressources de calcul et de pouvoir regrouper ces résultats.

3.6. Commandes de gestion des formats d'affichage

Au vu des nombreuses demandes de modification des affichages des résultats dans les versions précédentes de Moca-RP^c, nous proposons maintenant un format d'affichage personnalisable en terme de types de tabulations, de précisions, ...

L'interpréteur de commandes gère une liste de format et il est possible d'afficher une banque de résultats en utilisant un des formats d'affichage de la liste. Il existe toujours un format d'affichage par défaut (nommé `default`).

Les formats d'affichage provenant de la version 10.07 de Moca-RP^c ont été traduits au format d'affichage de la version courante. Ces formats d'affichage sont définis au sein du fichier `Moca13 .ini` (fichier faisant parti du package de Moca-RP^c).

3.6.1. Création d'un format d'affichage {set format ...}

La commande suivante permet de créer un format d'affichage :

```
set format <id-format> {  
... /* Différentes directives d'affichage */  
} ;
```

où `<id-format>` est le nom du format d'affichage.

Un format d'affichage est donc composé d'une ou plusieurs directives d'affichage. Chaque directive commence par un mot-clef l'identifiant et par un certain nombre d'options.

Parmi ces options, on retrouve pratiquement toujours une chaîne de caractères (entourée de guillemets) précisant le texte à afficher. Ce texte, de la responsabilité de l'utilisateur, contient des champs dans un format spécifique. Lors de l'affichage des résultats ces champs sont remplacés par l'information qu'ils codent.

Un champ est composé d'une suite de balise obligatoire ou optionnel dans l'ordre suivant :

```
% [flags] [width] [.precision] $ type
```

Chaque balise est un caractère simple ou un nombre ayant une signification particulière. Le champ le plus simple contient seulement le signe de pourcentage, le signe '\$' (dollars) et un caractère spécifiant le type d'information à afficher. Le type dépend forcément de la directive d'affichage en cours. Il est remplacé à l'affichage par un caractère, une chaîne de caractères ou un nombre. Si le signe '\$' est suivi par un caractère ne correspondant pas à la directive d'affichage, une erreur s'affiche à l'écran.

Pour afficher le signe de pourcentage, utilisez %%.

Les balises optionnelles, qui apparaissent avant le signe \$ contrôle le format d'affichage du champ.

- `flags` : caractère(s) optionnel(s) contrôlant
 - la justification de la sortie : Si '-' est présent la justification se fera à droite, par défaut elle se fait à gauche.
 - l'affichage de signe : Si '+' est présent, les nombres sont précédés du signe '+' s'ils sont positifs.
 - le caractère de remplissage. Si `width` est supérieur à l'affichage normal, des caractères sont affichés en plus. '0' et ' ' (espace) sont des flags qui forcent ce remplissage à l'aide de zéros ou d'espaces.
 - l'affichage de virgules décimales : Si '#' est présent, le point décimal sera toujours affiché.
 - l'affichage au format XML : Si 'x' est présent, les caractères {&, <, >, ', " } sont remplacés par les chaînes de caractères {&#amp;, <, >, ', "}.
- `width` : nombre optionnel spécifiant le nombre minimum de caractères de l'affichage.
- `precision` : nombre optionnel spécifiant soit le nombre maximum de caractères à afficher dans le cas d'une chaîne de caractères ou d'un entier, soit le nombre de chiffre significatif pour un nombre réel. Ce nombre peut

être remplacé par le caractère '_', auquel cas la précision par défaut (option de l'interpréteur de commandes) sera utilisée.

Après cette introduction, les paragraphes qui suivent présentent les différentes directives d'affichage.

3.6.1.1. Directive pour les informations générales

Objectif : Afficher des informations générales sur la simulation

Syntaxe : **Info "<format>"**

où <format> est une chaîne de caractères acceptant les types de champs suivants :

- T : Titre du réseau (chaîne de caractères)
- t : Nom du réseau (chaîne de caractères)
- B : Nom de la banque de résultat (chaîne de caractères)
- S : Nombre d'histoires simulées (entier)
- C : Graine initiale du générateur de nombres aléatoires (réel)
- L : Graine courante du générateur de nombres aléatoires (réel)
- D : Durée d'une histoire (réel)
- @ : Différents temps de calcul pour les états statistiques
- 1 | 2 | 3 | 4 | 5 | 6 : Type de statistique pris en compte dans la simulation ("ON" | "OFF")

3.6.1.2. Directive pour les transitions

Objectif : Pour chaque transition (sélectionnée), afficher son nom, numéro, moyenne, écart-type et/ou intervalle de confiance.

Syntaxe : **Tr <select> "<format>" ["<title>"]**

où <select> est un sélecteur de transitions, <title> est un titre à afficher et <format> est une chaîne de caractères acceptant les types de champs suivants :

- n : Numéro de la transition (entier)
- N : Nom de la transition (chaîne de caractères)
- M : Nombre moyen de tir de la transition (réel)
- S : Ecart-type sur le tir de la transition (réel)
- I : Intervalle de confiance à 90 % sur le tir de la transition (réel)

Cette directive n'affiche rien si le sélecteur de transitions est vide.

3.6.1.3. Directive pour les places

Objectif : Pour chaque place (sélectionnée), afficher son nom, numéro, et/ou (moyenne, écart-type, intervalle de confiance) des différentes grandeurs statistiques calculées.

Syntaxe : **Pl <select> "<format>" ["<title>"]**

où <select> est un sélecteur de places, <title> est un titre à afficher et <format> est une chaîne de caractères acceptant les types de champs suivants :

- n : Numéro de la place (entier)
- N : Nom de la place (chaîne de caractères)
- K : Marquage moyen en fin d'histoire de la place (réel)
- L : Marquage moyen de la place (réel)
- M : Temps moyen de séjour dans la place (réel)
- Q : Ecart-type sur le marquage moyen en fin d'histoire de la place (réel)
- R : Ecart-type sur le marquage moyen de la place (réel)
- S : Ecart-type sur le temps moyen de séjour dans la place (réel)

- G : Intervalle de confiance à 90 % sur le marquage moyen en fin d'histoire de la place (réel)
- H : Intervalle de confiance à 90 % sur le marquage moyen de la place (réel)
- I : Intervalle de confiance à 90 % sur le temps moyen de séjour dans la place (réel)
- P : Pourcentage de temps total passé avec un jeton dans la place (réel).

Cette directive n'affiche rien si le sélecteur de places est vide.

3.6.1.4. Directive pour les états statistiques (Ancienne version)

Objectif : Pour chaque état statistique (sélectionné), afficher son nom, numéro, et/ou (moyenne, écart-type, intervalle de confiance) des différentes grandeurs statistiques calculées.

Syntaxe : **St** <type> <select> "<format>" ["<title>"]

où <type> correspond à un type de statistique spécifique (1,2,3,4,5 ou 6), <select> est un sélecteur d'états statistiques, <title> est un titre à afficher et <format> est une chaîne de caractères acceptant les types de champs suivants :

- n : Numéro de l'état (entier)
- N : Nom de l'état (chaînes de caractères)
- M : Valeur moyenne pour cet état et ce type de statistique (réel)
- S : Ecart-type pour cet état et ce type de statistique (réel)
- I : Intervalle de confiance à 90 % pour cet état et ce type de statistique (réel)
- P (uniquement pour le type de statistique 1): Pourcentage de temps total passé dans cet état statistique (réel).
- C (uniquement pour le type de statistique 6): Nombre de données non-censurées (entier).

Cette directive n'affiche rien si le sélecteur d'états statistiques est vide.

3.6.1.5. Directive pour les états statistiques (Nouvelle version)

Objectif : Afficher les résultats des états statistiques en prenant en compte la dimension temporelle.

Par rapport à l'ancienne version, nous passons d'un système à deux dimensions (données [Etat statistiques] x valeurs [moyenne, écart type, intervalle de confiance, ...]) à un système à trois dimensions (données x valeurs x temps). Le format d'affichage restant en deux dimensions, il y a 6 combinaisons possibles d'affichage.

Le format d'affichage gère 3 sous-ensembles : Times, Select et Values représentant les temps de calcul, les variables et les grandeurs calculés.

Le premier sous-ensemble saisi correspond la boucle externe d'affichage, le deux autres sous-ensemble sont utilisés pour afficher un tableau à 2 dimensions de manière classique.

Le format d'affichage de chaque sous-ensemble sera directement précisé dans la commande ainsi que le libellé associé à la grandeur considérée.

Le format a donc avoir la syntaxe suivante :

```
Xt <type>
Times [<times>] "<format>" ["<title>"]
Select <select> "<format>" ["<title>"]
Values "<format>" ["<title>"] [, "<format>" ["<title>"]]*
"<string>"
```

où <type> correspond à un type de statistique spécifique (1,2,3,4,5 ou 6), <select> est un sélecteur d'états statistiques, <times> est une liste de temps (Cf. option times [Chapitre 2.2.1] pour la syntaxe de <times>), <title> est un titre à afficher et <format> est une chaîne de caractères acceptant les types de champs suivants :

- T : Temps considéré
- n : Numéro de l'état (entier)
- N : Nom de l'état (chaînes de caractères)

- M : Valeur moyenne pour cet état et ce type de statistique (réel)
- S : Ecart-type pour cet état et ce type de statistique (réel)
- I : Intervalle de confiance à 90 % pour cet état et ce type de statistique (réel)
- P (uniquement pour le type de statistique 1): Pourcentage de temps total passé dans cet état statistique (réel).
- C (uniquement pour le type de statistique 6): Nombre de données non-censurées (entier).

L'ordre entre Times, Select et Values précise le préfixe, les abscisses et les ordonnées.

Si le préfixe est unique, le message (<title>) est affiché en préambule. Dans le cas contraire, il est affiché directement dans le tableau.

Cette directive n'affiche rien si le sélecteur d'états statistiques est vide.

Les exemples suivant permettent de mieux comprendre ce format d'affichage. (Calcul pour le réseau nommé samples sur les états statistiques E1, E2 et E3 au temps 10 et 20).

- ```
display result samples {Xt 1 Times at 10 "%_.$t" "Times"
 Select * "% 5$N" "State"
 Values "%_.$M" "Temps Moyen", "%_.$S" "Ecart-Type"
 "};";
Times;10
State;Temps Moyen;Ecart-Type
E1;2.12345;1.98765
E2;4.12345;2.98765
E3;6.12345;3.98765
```
- ```
display result samples {Xt 1 Select * "% 5$N" "State"
    Times "%_.$t" "Times"
    Values "%_.$M" "Temps Moyen", "%_.$S" "Ecart-Type"
    "};";
State;Times;Temps Moyen;Ecart-Type
E1;10;2.12345;1.98765
E1;20;4.12345;2.98765
E2;10;4.12345;2.98765
E2;20;8.12345;4.98765
E3;10;6.12345;3.98765
E3;20;12.1234;6.98765
```
- ```
display result samples {Xt 1 Values "%_.$M" "Temps Moyen"
 Select * "% 5$N" "State"
 Times "%_.$t" "Times"
 "};";
Temps Moyen
State;10;20
E1;2.12345;4.12345
E2;4.12345;8.12345
E3;6.12345;12.1234
```
- ```
display result samples {Xt 1 Values "%_.$M" "Temps Moyen"
    Times "%_.$t" "Times"
    Select * "% 5$N" "State"
    "};";
Temps Moyen
Times;E1;E2;E3
10;2.12345;4.12345;6.12345
20;4.12345;8.12345;12.1234
```

3.6.1.6. Directive pour le réseau

Objectif : Afficher la description du réseau.

Syntaxe : **Net**

Avertissement

Il n'y a pas d'affichage XML pour cette directive.

3.6.1.7. Directive d'affichage d'une chaîne de caractères

Objectif : Afficher une chaîne de caractères.

Syntaxe : **Str** "<str>"

où <str> est une chaîne de caractères.

Avertissement

Il n'y a pas d'affichage XML pour cette directive. Il faut donc écrire la chaîne de caractères au format XML.

3.6.1.8. Directive pour les histogrammes des états statistiques

Objectif : Pour chaque état statistique (sélectionné) ayant le drapeau HST actif, afficher les résultats intermédiaires (résultat à chaque fin d'histoire).

Syntaxe : **HstSt** <type> <action> <select> [XML] ["<title>"]

où <type> correspond à un type de statistique spécifique (1,2,3,4,5 ou 6), <select> est un sélecteur d'états statistiques, <title> est un titre à afficher et <action> permet de préciser le traitement à réaliser sur l'histogramme. Le drapeau XML permet de spécifier un affichage au format XML.

Il existe quatre traitements :

- **Display** : Affichage des données en brute

```
HISTOGRAMME DE L'ETAT No 'St001' No 2
3,37977E+00
3,45643E+00
4,91471E+00
5,69483E+00
6,60019E+00
7,04654E+00
8,35086E+00
8,72988E+00
9,78460E+00
9,88095E+00
1,03942E+01
1,09530E+01
1,10085E+01
...
```

- **EqProba** [<nbrclass>] : Affichage de classes équiprobables (équivalent à l'ancien utilitaire classes de la distribution moca10XY permettant de réutiliser un histogramme pour la loi empirique)

```
/* Classes generation for state 'St001'
   Number of classes : 10
   Size of histogram : 1000
*/
3,379767E+00 ,
5,058298E+01 ,
8,994500E+01 ,
1,344299E+02 ,
1,766010E+02 ,
2,245854E+02 ,
2,710736E+02 ,
3,221753E+02 ,
```

```
3,664302E+02 ,
4,151745E+02 ,
4,989649E+02
```

- **FixStep** [**<nbrclass>**] : Affichage d'histogramme de taille fixe, la taille d'un intervalle est déterminé en fonction de la valeur maximum et minimum de l'histogramme, ainsi que du nombre de classe demandé ((max-min)/nbrclass)

```
/* Histogram generation for state 'St001'
   Number of classes : 10
   Size of histogram : 1000
*/
--
--<3,3797666393220425E+000 0
>=3,3797666393220425E+000 <5,2938283545430750E+001 108
>=5,2938283545430750E+001 <1,0249680045153946E+002 112
>=1,0249680045153946E+002 <1,5205531735764816E+002 125
>=1,5205531735764816E+002 <2,0161383426375687E+002 116
>=2,0161383426375687E+002 <2,5117235116986558E+002 92
>=2,5117235116986558E+002 <3,0073086807597429E+002 106
>=3,0073086807597429E+002 <3,5028938498208299E+002 102
>=3,5028938498208299E+002 <3,9984790188819170E+002 114
>=3,9984790188819170E+002 <4,4940641879430041E+002 73
>=4,4940641879430041E+002 <=4,9896493570040911E+002 52
>4,9896493570040911E+002 -- 0
```

- **DefineBorne** ['[''] **<list-of-borne>** : Affichage d'histogramme dont les intervalles sont définis par l'utilisateur ; **<list-of-borne>** est une liste de réel séparé par des virgules qui permet de spécifier les bornes des intervalles. Les bornes -∞ et +∞ sont toujours ajoutés. Le caractère optionnel [permet de changer les règles d'inclusion des bornes des intervalles.

```
display result pnet {HstSt 3
  DefineBorne [ 1E-4,1E-3,1E-2,1E-1 Stat1];

/* Define borne Histogram for state 'Stat1'
   Size of histogram : 15000
   Minimum of histogram : 3.0000000000000001E-005
   Maximum of histogram : 2.0000000000000001E-001
*/
-- <1.000000E-004 739 4.9267%
>=1.000000E-004 <1.000000E-003 2229 14.86%
>=1.000000E-003 <1.000000E-002 7508 50.053%
>=1.000000E-002 <1.000000E-001 2971 19.807%
>=1.000000E-001 -- 1553 10.353%
```

Si le drapeau XML a été spécifié, la sortie est différente et dépend du traitement effectué :

- Les données de l'histogramme sont affichés de la manière suivante :

```
<histogram-data type='St' name='St001' id='2'>
  <data history='1' value='3.116854E+002' />
  <data history='2' value='5.473069E+002' />
  ...
</histogram-data>
```

- Les histogrammes de taille fixe, de classes équiprobable ou d'intervalles définies sont affichés de la même manière (l'attribut **typehst** peut prendre les valeurs **fixstep**, **eqproba** ou **defineborne** afin de rappeler le type d'histogramme affiché) :

```
<histogram typehst='defineborne' type='St1' name='State' id='1' size='627'
  minimum='1.53406350E+000'
  maximum='5.99560058E+002'>
  <interval number='13' ratio='2.0734%'
    beg='-Infinity' include-beg='false'
```

```

end='1.00000000E+001' include-end='true' />
<interval number='17' ratio='2.7113%'
  beg='1.00000000E+001' include-beg='false'
  end='2.00000000E+001' include-end='true' />
<interval number='11' ratio='1.7544%'
  beg='2.00000000E+001' include-beg='false'
  end='3.00000000E+001' include-end='true' />
<interval number='8' ratio='1.2759%'
  beg='3.00000000E+001' include-beg='false'
  end='4.00000000E+001' include-end='true' />
<interval number='11' ratio='1.7544%'
  beg='4.00000000E+001' include-beg='false'
  end='5.00000000E+001' include-end='true' />
<interval number='567' ratio='90.431%'
  beg='5.00000000E+001' include-beg='false'
  end='+Infinity' include-end='false' />
</histogram>

```

Cette directive n'affiche rien si le sélecteur d'états statistiques est vide.

3.6.1.9. Directive pour les histogrammes des transitions

Objectif : Pour chaque transition (sélectionnée) ayant le drapeau HST actif, afficher la liste des temps où cette transition a été tirée (histoire par histoire).

Syntaxe : **HstTr** <action> <select> [XML] ["<title>"]

où <action> permet de préciser le traitement à réaliser sur l'histogramme <select> est un sélecteur de transitions, et <title> est un titre à afficher. Le drapeau XML permet de spécifier un affichage au format XML.

Cette directive n'affiche rien si le sélecteur de transitions est vide. Dans le cas contraire, l'affichage est très proche de l'histogramme des états statistiques.

3.6.1.10. Directive pour les chronogrammes des états statistiques

Objectif : Pour chaque état statistique (sélectionné) ayant le drapeau CHRO actif, afficher le chronogramme moyen.

Syntaxe : **Chro** <select> [XML] ["<title>"]

où <select> est un sélecteur d'états statistiques et <title> est un titre à afficher. Le drapeau XML permet de spécifier un affichage au format XML.

Cette directive n'affiche rien si le sélecteur d'états statistiques est vide.

Dans le cas contraire, il affiche le chronogramme

- soit sous une forme tabulé (si le drapeau XML n'a pas été précisé) :

```

CHRONOGRAMME DE L'ETAT No 'Stck' No 1
TEMPS          VALEUR
0.0000000000000000E+000  0.0000000000000000E+000
1.0000000000000000E+001  2.1699999999999990E+000
2.0000000000000000E+001  4.0300000000000002E+000
3.0000000000000000E+001  5.9199999999999999E+000
4.0000000000000000E+001  1.5470000000000004E+001
5.0000000000000000E+001  1.7099999999999994E+001
6.0000000000000000E+001  1.8630000000000010E+001
7.0000000000000000E+001  2.0239999999999998E+001
8.0000000000000000E+001  2.1779999999999987E+001
9.0000000000000000E+001  2.3270000000000003E+001
1.0000000000000000E+002  3.2570000000000000E+001
1.1000000000000000E+002  3.3799999999999997E+001
...

```

- soit dans un format XML

```
<chronogram name='St001' id='1'>
  <point time='0.000000E+000' value='1.900000E-002' />
  <point time='1.000000E+001' value='4.100000E-002' />
  <point time='2.000000E+001' value='6.700000E-002' />
  <point time='3.000000E+001' value='7.500000E-002' />
  <point time='4.000000E+001' value='1.040000E-001' />
  <point time='5.000000E+001' value='1.290000E-001' />
  ...
  <point time='1.000000E+003' value='3.682000E+000' />
</chronogram>
```

3.6.1.11. Sélecteur de données

La plupart des directives d'affichage prennent un ensemble d'objets (transitions, places ou états statistiques) en paramètre. Cet ensemble d'objets est construit à l'aide d'un sélecteur défini par des primitives de sélection.

Plusieurs primitives sont disponibles pour construire ces sélecteurs :

- `<name>` : sélectionne l'objet de nom donné.
- `{ }` : l'ensemble vide.
- `*` : l'ensemble de référence.
- `{ S }` : l'ensemble de S.
- `S1, . . . , Sk` : l'union des ensembles S1,...,Sk.
- `S1^ . . . ^Sk` : l'intersection des ensembles S1,...,Sk.
- `S1 / . . . / Sk` : la différence des ensembles S1,...,Sk. $((... (S1/S2)/...) / Sk)$.

3.6.2. Utilisation d'un format d'affichage

Un format d'affichage est requis à chaque fois que l'on souhaite afficher un résultat, c'est-à-dire lors de l'utilisation de la commande **display result ...**, mais également lorsque l'on souhaite afficher les résultats intermédiaires d'une simulation double-détente ou d'une simulation par lots.

D'une manière pratique, le format d'affichage n'est pas un champ obligatoire. Dans ce cas, le format par défaut est utilisé (`default`). Pour utiliser un format d'affichage, il suffit de spécifier son nom dans la commande d'affichage.

```
display result net=>res Mocal0Fr1 ;
```

En lieu et place du nom du format, il est également possible d'utiliser directement les directives de format d'affichage en les entourant d'accolades. Cette fonctionnalité est généralement employée pour afficher des informations courtes sur la banque de résultats.

Par exemple, pour afficher le nombre d'histoires qui ont été simulées et mémorisées dans une banque de résultats, il suffit d'employer la commande **display result net=>res {Info "%\$S"};**

3.6.3. Affichage d'un format {display format ...}

Pour afficher la définition d'un format (ces directives d'affichage), il convient d'utiliser la commande **display format <id-format>**; où `<id-format>` est le nom du format à afficher.

3.6.4. Suppression d'un format d'affichage {clear format ...}

Pour supprimer d'un format d'affichage, il faut d'utiliser la commande **clear format <id-format>**; où `<id-format>` est le nom du format à afficher.

Pour supprimer tous les formats d'affichage, il faut lancer la commande **clear format all;**

Le format par défaut ne peut jamais être supprimé. En revanche, lorsque l'on utilise cette commande sur celui-ci, il est reconstruit. C'est utile lorsque l'on souhaite changer de langue.

3.7. Commandes Utilitaires

3.7.1. Gestionnaire de chronomètres {... timer ...}

Moca-RP^c permet de gérer des chronomètres (ou timers) afin de mesurer les temps de chargement, de simulation et/ou d'affichage. Dans la mesure du possible (suivant les systèmes d'exploitation), les données suivantes sont recueillies : temps réel, temps utilisateur, temps associé au noyau (du système d'exploitation) comme le temps pour la gestion de la mémoire "swap".

Les chronomètres sont nommés et définis à l'aide de la commande **set timer <id-timer>**; où <id-timer> est un identificateur d'un chronomètre qui n'existe pas actuellement au sein de Moca-RP^c.

Les commandes **timer <id-timer> start;** et **timer <id-timer> stop;** permettent respectivement de déclencher et d'arrêter un chronomètre.

Il est alors possible d'afficher la valeur courante du chronomètre à l'aide de la commande **display timer ...**. Cette dernière prend en paramètre un format d'affichage défini à l'aide d'une chaîne de caractères. Le caractère balise '%' spécifie que le prochain caractère sera remplacé en fonction de son type et du chronomètre considéré.

Les méta-caractères suivants sont acceptés :

- %R : affiche la durée réelle
- %U : affiche la durée spécifique à la simulation (temps utilisateur)
- %K : affiche la durée associée au noyau
- %T : affiche l'heure actuelle (uniquement sous Windows)
- %D : affiche la date actuelle (uniquement sous Windows)

Les durées sont affichées en seconde avec une précision en millisecondes. La syntaxe pour la commande d'affichage est la suivante **display timer <id-timer> <format>**; où <id-timer> est un identificateur d'un chronomètre préalablement défini et <format> est une chaîne de caractères entourée de guillemets utilisant les méta-caractères précisés ci-dessus. (Par exemple : **display timer TS "Temps de simulation : %U sec";**)

Pour finir, la commande **clear timer <id-timer>**; permet de supprimer un chronomètre en mémoire.

3.7.2. Trace des commandes et des résultats {set trace ...}

Toutes les commandes saisies par l'utilisateur (directement ou indirectement en lisant un fichier de commande) et toutes les sorties (résultats d'une commande ou message d'erreur) peuvent être automatiquement redirigées vers un fichier de trace.

Pour activer cette fonctionnalité, il faut utiliser la commande **set trace "<file>"**; où <file> est le nom du fichier de trace entouré par des guillemets. Si ce fichier existait préalablement, il est ouvert en ajout (il n'est donc pas détruit).

Pour désactiver la trace, il suffit de lancer la commande **set trace;** (sans nom de fichier).

3.7.3. Options de l'interpréteur de commandes {display options /set ...}

L'interpréteur de commandes utilise un certain nombre d'options qui permet de préciser son fonctionnement. Citons deux exemples : la langue utilisée pour l'affichage des messages et la précision par défaut utilisée lors de l'affichage d'un nombre réel.

Toutes les options de l'interpréteur peuvent être affichées à l'aide de la commande **display options;**. Les différentes options et leurs valeurs par défaut sont les suivantes.

```
01      Moca > display options;
02      set language french ;
03      set precision 6 ;
04      /* no trace active */
```

```

05      set prompt "
06      Moca > " ;
07      set seed      12345681.00 ;
08      set delay 30 ;
09      set max-loop 1000 ;
10      set max-histo 1000 ;
11      set max-chrono 1000 ;
12      set adj-chrono 1.0000000000000000E+003 ;
13      set simul timer "Temps de simulation : %U (%R)
14      " ;

```

Notons que ce format d'affichage peut être utilisé directement comme fichier de commande. Il est possible de rediriger cet affichage dans un fichier (par exemple **display options > "Moca.ini"**), de modifier ce fichier à notre convenance et de le réutiliser immédiatement ou dans une prochaine session à l'aide de la commande **load "Moca.ini"**;

Les différentes options sont :

1. La langue utilisée pour l'affichage des messages : Seules deux langues sont disponibles au sein de Moca-RP^C : le français (french) et l'anglais (english). Il est relativement aisé d'ajouter de nouvelles langues au moteur de calcul (même si cela entraîne la recompilation et la génération d'une nouvelle version de Moca-RP^C). Actuellement, seuls les messages en français sont à jour. Pour modifier cette option, il faut utiliser la commande **set language {french | english}**;
2. La précision par défaut utilisée lors de l'affichage d'un nombre réel : Cette précision peut être utilisée dans les formats d'affichage des résultats en mettant pour le champ de précision le caractère '_'. Cette technique permet de modifier l'affichage des résultats dynamiquement sans créer un nouveau format d'affichage. Pour modifier cette option, il faut utiliser la commande **set precision <int>**; où <int> est un entier.
3. La trace, c'est-à-dire la redirection des commandes et des réponses de l'interpréteur : Pour plus d'explication, voir Section 3.7.2, « Trace des commandes et des résultats {set trace ...} ». Lorsqu'aucune trace n'est active, la commande **display options**; affiche `/* no trace active */`.
4. Le prompt, c'est-à-dire le texte qui invite l'utilisateur à saisir une nouvelle commande : Chacun peut personnaliser son prompt. Il existe quelques meta-caractères permettant d'ajouter automatiquement la date courante (%D), l'heure au moment de l'affichage (%T) ou le caractère de pourcentage (%%). Pour modifier cette option, il faut utiliser la commande **set prompt <prompt>**; où <prompt> est une chaîne de caractères entourée de guillemets (les retours à la ligne sont acceptés).
5. La graine par défaut du générateur de nombres aléatoires : Cette donnée est utilisée lors de nouvelles simulations, lorsque la banque de résultats vient d'être créée. Pour modifier cette option, il faut utiliser la commande **set seed <seed>**; où <seed> est un nombre réel supérieur à zéro. Il est conseillé d'utiliser un nombre réel assez grand et impair.
6. Le temps de simulation maximum autorisé pour effectuer une simulation : Au-delà de ce temps, la commande termine l'histoire courante et arrête la simulation. Pour modifier cette option, il faut utiliser la commande **set delay <delay>**; où <delay> est un nombre entier supérieur à zéro représentant le temps de simulation maximum en seconde.
7. Le nombre de transitions qui peuvent être tirées successivement sans que le temps de l'histoire soit incrémenté. Au-delà de ce nombre la simulation est brutalement interrompue, la boucle et l'échéancier sont affichés. Pour modifier cette option, il faut utiliser la commande **set max-loop <int>**; où <int> est un nombre entier supérieur à zéro.
8. Le nombre d'histogrammes possibles pour chaque état statistique ou chaque transition. Ce nombre limite le nombre de valeur mémorisée pour chaque histogramme de la simulation. Pour modifier cette option, il faut utiliser la commande **set max-histo <int>**; où <int> est un nombre entier supérieur à zéro ou égal à '-1'. Dans le dernier cas, il n'y a pas de limite en dehors de la mémoire de l'ordinateur. Pour les histogrammes des transitions, plusieurs valeurs peuvent être mémorisées pour une seule histoire. La valeur '-1' est utilisée afin de préciser que l'on traite une nouvelle histoire. Dans ce cas, le nombre maximum d'histogrammes pour les transitions est supérieur au nombre de valeurs affichées.

9. Les options de chronogramme permettent de diriger l'algorithme de composition des chronogrammes.

10. L'affichage du temps de simulation : A la fin d'une simulation, Moca-RP^c affiche le temps qu'il a mis pour effectuer cette simulation. Pour cela, il utilise un chronomètre (ou timer). Pour modifier cette option, il faut utiliser la commande **set simul timer <display>**; où <display> est un format d'affichage de chronomètres entouré de guillemets (Cf. Section 3.7.1, « Gestionnaire de chronomètres {... timer ...} »).

3.7.4. Utilisation des commandes systèmes {system ...}

Il est possible d'utiliser des commandes systèmes directement au sein de Moca-RP^c. L'intérêt de cette commande est multiple. Elle permet de connaître facilement le répertoire de travail (**system "pwd";** ou **system "cd";**), d'afficher un fichier de données ou de résultats (**system "type file.res";**), de modifier un fichier de données (**system "notepad file.mok";**) ou un fichier de résultats (**system "excel file.res";**) ou encore de comparer deux fichiers de résultats (**system "windiff file1.res file2.res";**).

Comme les nombreux exemples le montrent la syntaxe de cette commande est la suivante **system "<command>"**; où <command> est la commande système à exécuter. Si vos commandes utilisent des guillemets, il convient de les faire précéder par le caractère anti-slash \.

3.7.5. Affichage d'une chaîne de caractères {display "<string>" ...}

La commande **display "chaîne de caractères"**; permet d'afficher chaîne de caractères sur la sortie usuelle.

Cette commande permet lors du chargement d'un fichier de commande l'affichage d'information permettant de préciser la teneur des commandes jouées. A l'aide de la redirection (Cf. Section 3.3.8, « Redirection de l'affichage »), il permet aussi l'insertion de commentaires dans les fichiers de résultat.

4. Les lois dans Moca-RP^C

4.1. Généralités

A chaque transition du réseau de Petri est attachée :

- la loi de distribution du délai de tir de cette transition c'est-à-dire le délai qui s'écoule entre l'instant de la validation et l'instant du tir effectif.
- La loi de tir de cette transition, c'est-à-dire la manière dont sera tirée la transition

Cette différenciation entre ces deux lois fait que les lois à la sollicitation font maintenant partie des lois de tir des transitions.

Les lois sont désormais repérées uniquement par leur nom (de six caractères au maximum) alors que dans les versions antérieures elles l'étaient soit par leur nom, soit par leur numéro.

Les paramètres des lois sont des expressions quelconques (donc pas forcément constantes). Ces expressions sont évaluées au moment où elles sont utilisées, c'est-à-dire lorsque la transition est valide pour les paramètres des lois de délai et lorsque la transition est tirée pour les paramètres des lois de tir. Le calcul du délai de tir d'une transition peut donc dépendre du moment où elle a été valide. Cette notion a permis de supprimer toutes les lois N optionnelles des versions précédentes et de simplifier (et de généraliser) la loi exponentielle+Wow (attente météo).

D'autre part, la notion de mémoire (Cf. Section 2.5.8, « Transitions à mémoire ») est attachée aux transitions et non plus aux lois (comme c'était le cas dans les versions 8.9 et précédentes).

Notons que par souci de compatibilité ascendante le parseur de fichier 10 offre à l'utilisateur la possibilité d'utiliser les fichiers d'entrée des anciennes versions 8.x, 9.x et 10.x (Cf. Annexe A, *Lecture de fichiers au format Moca10*).

4.2. Lois de délai de Moca-RP^C

Les lois de délai actuellement disponibles sont les suivantes :

- `drc δ` est la loi de Dirac de délai δ ;
- `erlg m, β` est la loi Erlang de moyenne m et d'ordre β ;
- `erlgg β, λ1, ..., λβ` est la loi d'Erlang généralisée d'ordre β et $\lambda_1, \dots, \lambda_\beta$ sont les paramètres des exponentielles en série ;
- `empir c1, ..., cn` est la loi empirique à n classes c_1, \dots, c_n ;
- `exp λ` est la loi exponentielle de taux λ ;
- `expow λ, δ` est une loi exponentielle de taux λ à laquelle on ajoute le délai δ ;
- `ipa τ` est la loi "instants prévus à l'avance" ayant pour délai entre deux tirs τ ;
- `ifa δ, t0` est la loi "instants fixés à l'avance" ayant pour délai entre deux tirs le paramètre δ et comme premier instant de tir t_0 ;
- `nlog m, e` est la loi log-normale de moyenne m et de facteur d'erreur e ;
- `spec n, p1, ..., pm` est la $n^{\text{ième}}$ loi spéciale qui prend p_1, \dots, p_m en paramètres. Les lois spéciales permettent d'ajouter de nouvelles lois directement en C à l'aide d'une API de programmation (Cf. Section 5, « Interface de programmation C »)
- `tri a, b, c` est la loi triangulaire[3, 4] de minimum a , de maximum b et de mode c .
- `unif min, max` est la loi uniforme de minimum \min et de maximum \max ;
- `weibull η, β` est la loi de Weibull de paramètre d'échelle η et de paramètre de forme β ;
- `web m, β` est la loi de Weibull de moyenne m et de paramètre de forme β ;
- `webtr m, β, α` est la loi de Weibull tronquée de moyenne m , de paramètre de forme β et d'âge α à l'instant $t = 0$;

Au niveau de la description du réseau, les lois sont repérées par le mot-clef **LOI** suivi du nom de la loi et des paramètres décrits par l'utilisateur. Exemple : **LOI webtr p1,p2,p3**

4.2.1. Loi Dirac {drc ...}

Densité	Distribution de DIRAC correspondant à un délai déterministe égal à δ
Paramètres	δ = délai au bout duquel la transition est tirée à partir du moment où elle est devenue valide
Simulation	$d = \delta$

Associée à une transition à mémoire (Cf. Section 2.5.8, « Transitions à mémoire »), une loi de DIRAC permet de fabriquer un compteur de temps.

4.2.2. Loi Erlang {erlg ...}

Paramètres	$m = \text{moyenne} = E(d);$ $\beta = \text{ordre de la loi d'Erlang.}$
Pré-calcul	$p_1 = \frac{1}{\lambda} = \frac{m}{\beta}$
Densité	$f(d) = \frac{(\lambda d)^{(\beta-1)}}{(\beta-1)!} e^{(-\lambda d)}$
Simulation	Z_1, Z_2, \dots, Z_β tirés au hasard et équirépartis entre 0 et 1; $d = -p_1 \cdot \ln(Z_1 \cdot Z_2 \dots Z_\beta)$ Distribution d'Erlang.

4.2.3. Loi Erlang généralisée {erlgg ...}

La distribution d'Erlang généralisée est un cas particulier de loi de probabilité dite de TYPE PHASE qui permet de modéliser des réseaux de files d'attente. Une loi d'Erlang généralisée d'ordre β et de paramètres $\lambda_1, \dots, \lambda_\beta$ modélise la somme $T = T_1 + T_2 + \dots + T_\beta$ où les T_i obéissent à des lois exponentielles de paramètres λ_i .

Paramètres	$\beta = \text{ordre de la loi d'Erlang généralisée};$ $\lambda_1, \dots, \lambda_\beta$ correspondent aux β exponentielles en série
Densité	Le produit de convolution de loi exponentielle à paramètres différents.
Simulation	Z_1, Z_2, \dots, Z_β tirés au hasard et équirépartis entre 0 et 1; $d = -\sum_{i=1}^{\beta} \frac{\ln(Z_i)}{\lambda_i}$

4.2.4. Loi Empirique {empir ...}

Paramètres	X_1, X_2, \dots, X_{N+1} où N désigne le nombre de classes de l'échantillon et les $(X_i)_{1 \leq i \leq N+1}$ représentent les bornes des différentes classes équiprobables (Ils sont classés par ordre croissant) :
------------	---

	X_1, X_2 représentent les bornes inférieure et supérieure de la classe 1, ... X_N, X_{N+1} représentent les bornes inférieure et supérieure de la classe N.
Densité	Histogramme à classes équiprobables
Simulation	Z tiré au hasard est équiréparti entre 0 et 1 ; Calcul de la classe i à laquelle appartient Z : $i = \text{INT}(N.Z) + 1$; (INT = partie entière) Calcul du délai par interpolation linéaire entre les bornes de la classe i : $d = X_i + (X_{i+1} - X_i)(N.Z - \text{INT}(i))$.

4.2.5. Loi Exponentielle {exp ...}

Paramètres	λ = taux de défaillance
Densité	$f(d) = \lambda \cdot e^{-\lambda \cdot d}$
Simulation	Z tiré au hasard est équiréparti entre 0 et 1 ; $d = -\frac{\ln(Z)}{\lambda}$

4.2.6. Loi Exponentielle + wow {expw ...}

Il s'agit d'un délai exponentiel auquel on ajoute un délai constant. Dans les versions précédentes du logiciel, ce délai était calculé en fonction du temps courant de la simulation et d'un tableau représentant un délai pour chaque mois de l'année. Dans cette version du logiciel, ce délai peut-être calculé directement à l'aide des expressions du langage.

Paramètres	λ = taux de défaillance ; δ = délai à ajouter
Densité	$f(d_1) = \lambda \cdot e^{-\lambda \cdot d_1}$
Simulation	Z tiré au hasard est équiréparti entre 0 et 1 ; $d_1 = -\frac{\ln(Z)}{\lambda}$: délai exponentiel ; $d = d_1 + \delta$.

4.2.7. Loi Instant Prévu à l'Avance {ipa ...}

Paramètres	τ = délai entre deux tirs potentiels successifs
Densité	Si $(n-1) \cdot \tau < T < n \cdot \tau$ alors le tir est effectué à l'instant $n \cdot \tau$ (T étant le temps courant).
Simulation	$d = \tau - (T \bmod \tau)$.

4.2.8. Loi Instant Fixé à l'Avance {ifa ...}

La loi ifa est similaire à la loi IPA mais le premier intervalle est différent des autres. Ceci est utile, par exemple, pour décaler les instants de test de composants testés à la même fréquence.

Paramètres	τ = délai entre deux tirs potentiels successifs ; θ = délai du 1er tir.
Densité	Si $T < \theta$ alors le tir a lieu à θ . (T étant le temps courant) Si $((n-1).\tau + \theta) < T < (n.\tau + \theta)$ alors le tir est effectué à l'instant $n.\tau + \theta$.
Simulation	Si T est inférieur à θ , $d = \theta$. Dans le cas contraire, $d = \tau - (T \bmod \tau)$.

4.2.9. Loi Log-Normale {nlog ...}

Paramètres	m = moyenne = $E(d)$; q = facteur d'erreur à 5%.
Pré-calcul	$b = \frac{\ln(q)}{1.64}$ <p>Pour un facteur d'erreur à 5%, la valeur de U_a est égale à 1.64 (valeur provenant des tables relatives à la loi normale) ce qui permet de définir un intervalle de confiance à 90% (5% de chaque côté).</p> $a = \ln(m) - \frac{b^2}{2}$
Densité	$f(d') = \frac{1}{d' b \sqrt{2\pi}} \exp \frac{-[\ln(d') - a]^2}{2b^2}$
Simulation	Z_1, Z_2 tirés au hasard et équirépartis entre 0 et 1; $d' = \sqrt{-2\ln(Z_1)} \cdot \cos(2\pi Z_2)$ (Distribution normale réduite); $d = e^{b.d' + a}$ (Distribution log-normale).

4.2.10. Loi Spéciale {spec ...}

Le type spec ne correspond pas à une mais à un ensemble de lois que l'utilisateur peut définir lui-même à l'aide d'un sous-programme C. Un exemple est donné au chapitre : Section 5, « Interface de programmation C ».

Paramètres	n = Numéro de la loi spéciale ; p_1, \dots, p_m = correspondent aux paramètres de la loi.
Densité	Elle est ce que l'utilisateur désire et c'est à lui de la définir
Simulation	d est évalué par un sous-programme écrit par l'utilisateur lui-même

4.2.11. Loi Triangulaire {tri ...}

Paramètres	a = Borne inférieure; b = Borne supérieure; c = mode; avec $a < c < b$.
Densité	f(d) de type triangulaire, comprise entre a et b et étant à son maximum à $t=c$.
Pré-calcul	$F_c = \frac{c-a}{b-a}$: utilisé lors du test sur Z $p_1 = \sqrt{(c-a)(b-a)}$: pente entre a et c $p_2 = \sqrt{(b-c)(b-a)}$: pente entre c et b
Simulation	Z tiré au hasard est équiréparti entre 0 et 1 ; si $Z=0$ $d=a$ sinon si $Z < F_c$ $d = a + p_1 \sqrt{Z}$ sinon si $Z < b$ $d = b + p_2 \sqrt{1-Z}$ sinon $d=b$

4.2.12. Loi Uniforme {unif ...}

A la demande de certains utilisateurs nous avons introduit la loi uniforme qui possède les caractéristiques suivantes:

Paramètres	δ_1 = Borne inférieure du délai δ_2 = Borne supérieure du délai
Densité	f(d) uniforme entre δ_1 et δ_2 .
Simulation	Z tiré au hasard est équiréparti entre 0 et 1 ; $d = (\delta_2 - \delta_1) \cdot Z + \delta_1$: distribution uniforme

4.2.13. Loi Weibull {web ... ou Weibull ...}

Il existe 2 lois Weibull dans Moca-RP^C. L'une, historique, prend comme paramètre principal la moyenne attendu de la loi. La seconde a été ajoutée à la demande des utilisateurs qui disposent généralement du paramètre d'échelle en lieu et place de la moyenne. Il est relativement facile de passer d'un paramètre à l'autre (Cf. plus bas).

4.2.13.1. Loi Weibull historique {web ...}

Paramètres	m = moyenne = E(d);
------------	---------------------

	<p>β = paramètre de forme.</p> <p>A noter qu'il est facile de passer de la moyenne au paramètre d'échelle (η) de la loi Weibull :</p> $m = E(d) = \eta \cdot \Gamma\left(1 + \frac{1}{\beta}\right)$
Pré-calcul	$p_1 = \eta = \frac{m}{\Gamma\left(1 + \frac{1}{\beta}\right)}$ $p_2 = \frac{1}{\beta}$
Densité	<p>Dans la littérature :</p> $f(d) = \frac{\beta}{\eta} \cdot \left(\frac{d}{\eta}\right)^{\beta-1} \cdot e^{-\left(\frac{d}{\eta}\right)^\beta}$ <p>En posant :</p> $\eta = \left(\frac{1}{\lambda}\right)^{\frac{1}{\beta}}$ <p>La densité devient :</p> $f(d) = \lambda \cdot \beta \cdot d^{\beta-1} \cdot \exp(-\lambda \cdot d^\beta)$
Simulation	<p>Z tiré au hasard est équiréparti entre 0 et 1 ;</p> $d = p_1 [-\ln(Z)]^{p_2}$

4.2.13.2. Loi Weibull utilisateur {Weibull ...}

Paramètres	<p>η = paramètre d'échelle;</p> <p>β = paramètre de forme.</p>
Pré-calcul	$p_1 = \eta$ $p_2 = \frac{1}{\beta}$
Simulation	Idem que la loi Weibull historique

4.2.14. Loi Weibull tronquée {webtr ...}

La distribution de la loi Weibull tronquée, a la propriété de correspondre à un taux de défaillance non nul à l'instant initial (âge α du composant) et variant en fonction du temps. Ceci permettra, en particulier de prendre en compte le fait qu'une maintenance préventive ou corrective ne remet pas forcément complètement à neuf les composants

maintenus. Cette loi est complètement décrite dans la note technique DSE/SES- ARF 95063 "Simulation d'une maintenance imparfaite - Loi de Weibull tronquée)".

Paramètres	m = moyenne de la loi non tronquée; β = paramètre de forme; α = âge à $t=0$.
Pré-calcul	$p_1 = \frac{1}{\beta}$ $p_2 = \lambda \cdot \alpha^\beta = \left(\frac{\alpha \cdot \Gamma\left(1 + \frac{1}{\beta}\right)}{m} \right)^\beta$
Densité	$f(d) = \lambda \cdot \beta \cdot (d + \alpha)^{\beta-1} \cdot \exp(\lambda \cdot [\alpha^\beta - (d + \alpha)^\beta])$
Simulation	Z tiré au hasard est équiréparti entre 0 et 1 ; $d = \alpha \cdot \left(\left[1 - \frac{\ln(Z)}{p_2} \right]^{p_1} - 1 \right)$

Rappelons que pour calculer la moyenne à partir du facteur d'échelle, il convient d'utiliser la formule suivante :

$$m = E(d) = \eta \cdot \Gamma\left(1 + \frac{1}{\beta}\right)$$

4.3. Loïs de tir de Moca-RP^c

Les lois de tir actuellement disponibles sont les suivantes :

- `def` est la loi de tir par défaut (généralement employé dans les réseaux de Petri)
- `sol $\gamma_1, \gamma_2, \dots, \gamma_n$` est une loi de tir à la sollicitation pour $n+1$ arcs avals.
- `sol2 $\gamma_1, \#P_1, \gamma_2, \#P_2, \dots, \gamma_n, \#P_n$` est une seconde loi de tir à la sollicitation pour $n+1$ places avales. Cette seconde loi permet de spécifier explicitement la place qui est associée à chaque gamma.
- `spec n, p_1, \dots, p_m` est la $n^{\text{ième}}$ loi spéciale qui prend p_1, \dots, p_m en paramètres. Les lois spéciales permettent d'ajouter de nouvelles lois directement en C à l'aide d'une API de programmation (Cf. Section 5, « Interface de programmation C »)

Au niveau de la description du réseau, les lois sont repérées par le mot-clef **TIR** suivi du nom de la loi et des paramètres décrits par l'utilisateur. Exemple : **TIR sol g1,g2,g3**

4.3.1. Loi de tir par défaut {def}

Cette loi ne prend pas de paramètres et correspond à la manière usuelle de tirer une transition dans un réseau de Petri ; à savoir :

- Suppression pour chaque place amont d'un nombre n_i de jetons (où n_i est le poids de l'arc reliant la place amont i de la transition)
- Ajout pour chaque place avale d'un nombre m_j de jetons (où m_j est le poids de l'arc reliant la place avale j de la transition)

4.3.2. Loi de tir à la sollicitation {sol/sol2 ...}

Au moment où un composant est sollicité pour changer d'état, il existe une certaine probabilité notée γ qu'à ce moment précis, le composant reste dans l'état en cours, tombe en panne, ne change pas d'état correctement. Si on représente ce phénomène à l'aide d'un réseau de Petri, on utilise une transition ayant deux arcs avals, l'un menant vers le changement d'état, l'autre vers un état de panne. Mais, lorsque la transition est tirée, seul un des deux arcs avals transmet un jeton dans sa place avale (l'autre en transmet zéro) et ceci avec une fréquence égale à γ pour l'arc menant vers l'état de panne et $(1-\gamma)$ pour l'autre.

Dans les versions précédentes de Moca-RP^C, pour prendre en compte des défaillances à la sollicitation, on utilisait les lois SOL1, SOL2 et SOL3. Chacune avait des caractéristiques spécifiques (nombre de paramètres, avec ou sans délai, poids des arcs avals strictement égal à 1 ou pas, ...).

La loi de tir à la sollicitation **sol ...** a comme objectif de fournir à l'utilisateur une loi générique pour prendre en compte toutes les défaillances à la sollicitation. Les paramètres associés à cette loi sont les n probabilités de défaillance à la sollicitation $\gamma_1, \gamma_2, \dots, \gamma_n$ pour une transition ayant $n+1$ arcs avals. Cette loi définit implicitement l'attribution des probabilités de distribution aux places avales en fonction de l'ordre du numéro des places avales. La place avale qui porte le plus petit numéro aura une probabilité d'être choisi de γ_1 .

Une seconde loi de tir à la sollicitation **sol2 ...** fonctionne de la même manière que la première loi, mais l'attribution des probabilités aux places avales est explicite. Les paramètres associés à cette loi sont $\gamma_1, \#P_1, \gamma_2, \#P_2, \dots, \gamma_n, \#P_n$ pour une transition ayant $n+1$ places avales. Il doit donc y avoir un arc aval entre chaque place P_i et la transition. La probabilité que la place avale P_i soit choisie est égale à γ_i .

Dans les 2 cas, la somme des γ_i doit être inférieure ou égale à 1.

Le mécanisme de tir ne diffère d'un tir par défaut que pour l'ajout de jetons dans les places avales. Dans cette dernière phase :

- Tir Z au hasard et de manière équiprobable entre 0 et 1
- Calcul de k de la manière suivante :
 - $k = -1$; $i = 1$
 - Tant que (k est égal à -1)
 - Si $i > n$ alors $k = n+1$
 - sinon si ($Z \leq \sum_{k=1}^i \gamma_k$) alors $k = i$
 - $i = i + 1$
- Ajout uniquement pour la $k^{\text{ième}}$ place d'un nombre m_k de jetons (où m_k est le poids de l'arc reliant la $k^{\text{ième}}$ place à la transition).

4.3.3. Loi spéciale {spec ...}

Le type **spec** ne correspond pas à une mais à un ensemble de lois que l'utilisateur peut définir lui-même à l'aide d'un sous-programme C. Un exemple est donné au chapitre : Section 5, « Interface de programmation C ».

Les paramètres de cette loi sont son numéro (n), ainsi que les m paramètres p_1, \dots, p_m dont l'utilisateur aura besoin lors de la simulation.

Le mécanisme de tir de la transition est laissé à la responsabilité de l'utilisateur. Afin de ne pas engendrer des perturbations avec le moteur de calcul, il est recommandé de modifier le marquage d'une place que s'il existe un arc entre la dite place et la transition et il est recommandé de modifier la valeur d'une variable que si on retrouve cette variable dans les affectations de la transition.

5. Interface de programmation C

Ajout de fonctions/lois externes

Afin de permettre l'expérimentation de nouvelles lois (de délai ou de tir) ou d'ajouter des fonctions externes à l'application, Moca-RP^C offre une interface de programmation (API) permettant à l'utilisateur averti d'intégrer de nouvelles lois ou fonctions par un programme. Jusqu'à la version 9.X cette API était réalisée en langage Fortran. À partir de la version 10.X ce programme devra être écrit en langage C.

Dans la version 12, nous n'avons pas jugé utile de maintenir les API d'ajout d'états statistiques. Les expressions offrent de très nombreuses possibilités pour la définition d'états statistiques.

Depuis la version 12.16, cette API a été étendue afin d'intégrer des fonctions externes à Moca. Le principe de lancement de Moca a alors été revu afin de faciliter l'accès à cette API.

5.1. Lancement de Moca-RP^C

La distribution de Moca-RP^C intègre 4 fichiers principaux :

1. `Moca13.dll` est le coeur de l'application. C'est une librairie regroupant l'ensemble des fonctions pour lire les différents formats, interpréter les commandes, faire une simulation, afficher les résultats, ...
2. `Moca13.exe` est le lanceur de l'application (l'exécutable). Il appelle les fonctions d'entrée de la librairie (création et lancement un interpréteur de commandes).
3. `MocaAdd.dll` est une librairie optionnelle chargée dynamiquement par `Moca13.dll`. C'est au sein de cette librairie que des lois ou des fonctions peuvent être ajoutées à l'interpréteur de commande. Cette librairie doit être dans le même répertoire que la librairie principale. Dans le cas contraire, la librairie principale ne pourra pas charger cette librairie optionnelle.
4. `Moca13.ini` est un fichier d'initialisation optionnel lu lors du démarrage de l'application. Il permet à l'aide des commandes de l'interpréteur de commandes la définition de format d'affichage des résultats et/ou la modification des options par défaut de l'interpréteur de commandes

Lors du lancement de l'application, l'interpréteur de commande est créé. Puis différentes opérations sont effectuées dans l'ordre :

- Vérification des options de lancement de l'application (options passées en argument au niveau de la ligne de commande) ;
- Chargement s'il existe du fichier `Moca13.ini` ;
- Chargement s'il existe de la librairie `MocaAdd.dll` (et donc ajout des fonctions/lois externes au sein de Moca-RP^C) ;
- Chargement des fichiers passés au niveau de la ligne de commande ;
- Démarrage d'une session utilisateur (interpréteur de commandes)

5.2. Utilisation des fonctions/lois externes

5.2.1. Fonctions externes

Une fonction externe est définie à l'aide

- *de son nom* : afin de l'identifier et de l'utiliser au sein d'un réseau de Petri. Par convention (bien que cela ne soit d'aucune obligation), le nom d'une fonction externe commence par la lettre x en minuscule (pour eXterne) suivi d'un nom significatif commençant par une majuscule. Cela permet de différencier rapidement les fonctions propres à Moca des fonctions externes définis par l'utilisateur.

- *de sa signature* : la signature d'une fonction précise le nombre d'argument, le type (booléen, entier, réel) de chacun des arguments, ainsi que le type de valeur qui sera renvoyé par la fonction. La signature permet lors de la lecture du modèle d'afficher un éventuel message d'erreur/avertissement si le nombre d'argument n'est pas correct ou si un argument n'est pas du type attendu.
- *d'un éventuel commentaire* permettant de préciser ce que calcule la fonction externe.

Seuls les types booléen (B), entier (I) et réel (F) sont gérés au sein de Moca.

Une fois définie, une fonction externe peut être utilisée dans la description d'un réseau de Petri directement au niveau des expressions du modèle de la même manière qu'une fonction interne à Moca comme `log(n)`. Les fonctions externes s'intègrent naturellement à la description du réseau.

Il est possible d'afficher l'ensemble des fonctions externes au sein de l'interpréteur de commandes à l'aide de la commande **display functions**;

5.2.2. Lois externes (ou spéciales)

Moca-RP^c permet d'intégrer de nouvelles lois (de tir et de délai). L'utilisation de ces lois au sein d'un Réseau de Petri est réalisée via la loi spéciale (spec).

Ces lois prennent comme arguments :

- Un numéro qui indique le numéro de la loi externe considérée ; le nombre de nouvelles lois n'est pas borné. Ce numéro est indépendant du type de loi (tir ou délai)
- Un ensemble d'expression correspondant à l'ensemble des paramètres de la loi. Tous les paramètres de la loi sont de type réel. Seule la valeur des expressions sera connue de la librairie. Ces expressions sont évaluées par le Moca-RP^c avant d'être utilisées par la librairie.

5.3. Création de la librairie optionnelle `MocaAdd.dll`

5.3.1. Pré-requis

Avant tout chose, il est nécessaire de disposer d'un compilateur C ANSI.

Dans l'environnement Windows, le compilateur de Microsoft simplifie grandement la tâche (au niveau de la génération de la Dll). Une version gratuite *Microsoft Visual C++ Toolkit 2003* est disponible sur le site de cet éditeur.

Il est également possible de travailler dans l'environnement CygWin/GCC avec les outils spécifiques Windows pour générer une librairie pouvant être chargée dynamiquement.

Dans la suite de ce chapitre, nous supposons que vous avez installé un des compilateurs Microsoft dans un répertoire que vous connaissez.

5.3.2. Fichiers utilisés

Outre les 4 fichiers précédemment décrit, la distribution contient un ensemble de fichiers nécessaires à l'intégration de nouvelles fonctionnalités. Ces fichiers, en général stockés dans le répertoire `api` de la distribution, sont les suivants :

- `MocaAPIType.h` est un fichier entête C. Il contient l'ensemble des déclarations de types utilisés au sein de l'API.
- `MocaAPI.h` est un fichier entête C. Il contient l'ensemble des déclarations de fonctions disponibles au niveau de l'API.
- `MocaAdd.c` est un exemple complet de code permettant de déclarer des fonctions externes.

- `Moca13.lib` est une bibliothèque des fonctions C. Il contient l'ensemble des fonctions exportés au niveau du fichier `MocaAPI.h`. Cette bibliothèque devra être liée à la librairie additionnelle de Moca.
- `MocaMake.bat` est un fichier de commande qui génère la librairie `MocaAdd.dll` dans l'environnement Windows à l'aide d'un compilateur Microsoft.

Nous suggérons à l'utilisateur de faire une copie de l'ensemble de ces fichiers dans son répertoire de travail. Seuls les fichiers `MocaAdd.c` et `MocaMake.bat` doivent être modifiés.

Avant d'utiliser `MocaMake.bat`, il est nécessaire de modifier les lignes 4 et 5 de ce script afin de préciser respectivement le répertoire de destination (là où sera copié le fichier `MocaAdd.dll`) et le répertoire d'installation du compilateur Microsoft à l'aide des variables d'environnement `MOCA_OUT` et `VCPP_PATH`.

```
@echo off
rem -----
rem Visual C++ 9.0 doit être installé et
rem vc9varsxx.bat doit avoir été exécuté avant l'exécution de ce script
rem -----

setlocal

cl /MT /W3 /EHsc /O2 /D "WIN32" /D "NDEBUG" /D "_WINDOWS" /D "__WINDOWS__" /
D "_MBCS" /D "_USRDLL" /D "_CRT_SECURE_NO_WARNINGS" -I. Moca13.lib /LD /
FeMocaAdd.dll MocaAdd.c

del MocaAdd.obj
del MocaAdd.lib
del MocaAdd.exp

endlocal
pause
```

5.3.3. Principe général

Le seul point d'entrée de la librairie additionnelle `MocaAdd.dll` est une méthode C devant être nommée `Moca_LoadExtend` et prenant en paramètre un pointeur sur l'interpréteur de commande de Moca-RP^C. Il est nécessaire d'exporter cette méthode afin que la librairie principale puisse la trouver et l'appeler.

L'exemple ci-dessous est un exemple minimaliste devant afficher sur la sortie standard le message `C'est la bonne librairie` au démarrage de Moca-RP^C.

```
#include "MocaAPI.h"
__declspec(dllexport) void Moca_LoadExtend(mapi_shell *sh)
{
    printf("C'est la bonne librairie");
}
```

Ce point d'entrée permet d'enregistrer au sein de l'interpréteur de commandes les fonctions/lois externes définies par l'utilisateur.

Les fonctions de l'API suivantes peuvent être utilisées au sein de cette méthode :

- `mapi_add_function` permet d'enregistrer une fonction externe.
- `mapi_register_special_law` permet l'enregistrement de tous les lois de délai externe.
- `mapi_register_special_fire` permet l'enregistrement de tous les lois de tir externe.

5.3.4. Création de fonctions externes

La fonction `mapi_add_function` prend en argument l'interpréteur de commande, le nom de la fonction, sa signature, son commentaire et un pointeur de fonction permettant de réaliser le calcul.

L'exemple ci-dessous permet de définir une fonction externe `xSqrt` prenant un paramètre réel et renvoyant la racine carrée du paramètre sous forme d'un réel.

```
#include "MocaAPI.h"
#include <math.h>

static mapi_double
my_fct_sqrt(mapi_net *net, int nbr, mapi_double* args)
{
    if (nbr != 1) {
        mapi_add_warning(net, "xSqrt : le nombre de paramètre n'est pas égal à 1.");
        return mapi_NaN;
    }
    else return sqrt(args[0]);
}

__declspec(dllexport) void Moca_LoadExtend(mapi_shell *sh)
{
    mapi_add_function(sh, "xSqrt", "(F)F",
        "La racine carrée d'un réel", my_fct_sqrt);
}
```

5.3.4.1. Signature de la fonction

La signature de la fonction est décrite sous la forme d'une chaîne de caractères ayant une syntaxe spécifique.

La signature (F)F précise que la fonction prend un argument réel et renvoie un réel. Les parenthèses (qui sont obligatoires) encadrent les arguments de la fonction. Le type de valeur renvoyé par la fonction se précise après les parenthèses.

Les différents types de données admis au niveau de la signature sont :

- B : booléen
- I : entier
- F : réel

Tous les arguments doivent être donnés les uns derrière les autres sans espace. La signature (II)I d'une fonction de division entière (xDivInt) prend 2 arguments entiers et renvoie un entier.

La signature (FFFFFFFFFFFF)F prend 12 arguments réels et renvoie un réel. Il est possible d'utiliser à la place un raccourci d'écriture en utilisant une déclaration de tableau. La signature précédente peut être remplacée par ([F12)F. Le crochet ouvrant précise l'utilisation d'un tableau, le caractère qui suit donne le type (ici un réel) le nombre précise la taille du tableau. Il est possible de d'utiliser conjointement les 2 notations. La signature ([F5FF[F5)F est équivalente aux 2 autres signatures de ce paragraphe.

La signature ([F2+)F définit une fonction prenant 2 ou plus arguments réels. Le caractère '+' ne peut s'employer qu'en dernière position de la déclaration des arguments et en relation avec un tableau. Cela permet donc de définir des fonctions ayant un nombre d'argument indéfini.

La majorité des fonctions n'a pas besoin d'être réévaluée si la valeur des arguments n'a pas été modifiée. Moca-RP^c se basent sur ce principe pour éviter de réévaluer les expressions inutilement. Les fonctions externes peuvent utiliser un nombre généré aléatoirement, le temps courant de la simulation ou encore directement la valeur d'une variable du réseau sans que ces informations ne soient passées en argument. Ces fonctions doivent alors être systématiquement réévaluées. Pour préciser cela, il suffit d'ajouter le caractère '#' à la fin de la signature de la fonction. La fonction xRandom renvoyant un nombre aléatoire entre 0 et 1 est définie à l'aide de la signature ()F#.

Si la syntaxe associée à la signature n'est pas respectée, un message sera affiché sur le flux d'erreur de l'application au moment de l'enregistrement de la fonction.

5.3.4.2. Code de la fonction

La fonction réalisant le calcul prend en paramètre le réseau de Petri, le nombre d'argument et un tableau contenant l'ensemble des arguments. Elle renvoie bien entendu le résultat associé à la fonction externe. Si un problème survient, elle peut renvoyer la valeur mapi_NaN. Le prototype de la fonction est décrit plus en détail (Cf. ???).

Les principales fonctions d'API utilisées au sein du code d'une fonction externes sont :

- `mapi_add_warning` permet d'associer un message d'avertissement au réseau de Petri qui sera affiché à la fin de la simulation.
- `mapi_get_probability` renvoie un nombre aléatoire entre 0 et 1.
- `mapi_get_time` renvoie le temps courant de la simulation.

5.3.5. Création de lois externes

5.3.5.1. Principes

Pour définir de nouvelles lois l'utilisateur doit définir un certain nombre de fonction spécifique au type de loi considéré (loi de délai ou loi de tir). Dans les deux cas, les deux fonctions suivantes doivent être écrites :

1. Une fonction d'initialisation du mécanisme. Cette fonction est appelée par le moteur de calcul avant le lancement de la simulation. Cette fonction a la charge de prévenir l'utilisateur de la prise en charge ou non de la dite loi spéciale. Elle peut aussi se charger de créer les structures de données nécessaires à la mise en œuvre des nouvelles lois. Le prototype de ces fonctions est défini dans le fichier entête `moca_api.h` par le type de fonction `mapi_special_law_init_proc` ou `mapi_special_fire_init_proc`.
2. Une fonction de vérification des paramètres. Cette fonction est appelée par le moteur de calcul afin de valider les paramètres de la loi. La valeur de retour de la fonction permet de savoir si les paramètres sont valides ou non. Dans le second cas, la fonction peut renvoyer un message et un code d'erreur. Le prototype de ces fonctions est défini dans le fichier entête `moca_api.h` par le type de fonction `mapi_special_law_verif_proc` ou `mapi_special_fire_verif_proc`.

Pour les lois de délai, il faut aussi implémenter une fonction qui détermine en fonction du temps courant de la simulation et de la transition considérée quel est le délai avant le prochain tir de la transition. Le prototype de cette fonction est défini dans le fichier entête `moca_api.h` par le type de fonction : `mapi_special_law_delay_proc`.

Pour les lois de tir, il convient de définir deux autres fonctions :

1. Une fonction appelée au moment où la transition devient valide. Cette fonction peut servir pour mémoriser le temps courant par exemple. Le prototype de cette fonction est défini dans le fichier entête `moca_api.h` par le type de fonction `mapi_special_fire_valid_proc`.
2. Une fonction appelée au moment où la transition doit être tirée. Cette fonction doit effectuer toutes les phases de tir de la transition (affectation des variables, suppression de jetons dans les places amonts et ajout de jeton dans les places avalées) en les modifiant éventuellement. Le prototype de cette fonction est défini dans le fichier entête `moca_api.h` par le type de fonction : `mapi_special_fire_trigger_proc`.

Après la construction des fonctions nécessaires aux lois spéciales, l'utilisateur doit enregistrer dans la librairie additionnelle l'ensemble de ces fonctions à l'aide des fonctions `mapi_register_special_law` ou `mapi_register_special_fire`.

5.3.5.2. Exemple de loi de délai externe

L'exemple implémente la loi exponentielle + wow de Moca-RP^c version 10 et précédente comme exemple de loi de délai externe. Cette loi prend 13 paramètres : le premier correspond à un taux de défaillance du composant et les 12 autres à un ajustement du délai de défaillance en fonction des différents mois de l'année. Cette loi a un équivalent dans la version actuelle avec 2 paramètres le taux de défaillance et le délai d'ajustement (généralement calculé à l'aide d'une expression prenant en compte l'opérateur `time()`).

```
#include "MocaAPI.h"
#include <math.h>

static char buffer_mess[128];

static int
```



```

my_law_init(mapi_net *net, mapi_transition *T, int number)
{
    return number == 11 ? 1 : 0;
}

static int
my_law_verif(
    mapi_net *net, mapi_transition *T, int number,
    int nb_arguments, mapi_double *arguments,
    char **mess, int *code)
{
    *code = 999;

    if (number == 11) {
        int i;
        if (nb_arguments != 13) {
            sprintf(buffer_mess, "Le nombre d'argument doit etre egal a 13 "
                "(1 pour le Lbd et 12 pour le delai associe a chaque mois de l'annee)");
            *mess = buffer_mess;
            return 1;
        }
        for (i=0; i< nb_arguments; i++) {
            if (arguments[i] < 0) {
                sprintf(buffer_mess, "L'argument %d ne peut pas etre negatif.", i);
                *mess = buffer_mess;
                return 1;
            }
        }
        return 0;
    }

    *mess = "Pas possible !!";
    return 1;
}

static mapi_double
my_law_delay(
    mapi_net *net, mapi_transition *T, int number,
    int nb_arguments, mapi_double *arguments)
{
    if (number == 11) {
        mapi_double Z = mapi_get_probability(net);
        mapi_double current_time = mapi_get_time(net);
        mapi_double result = -log(Z)/arguments[0];
        int i = ((int)(current_time / 730)) % 12 + 1;
        result += arguments[i];
        return result;
    }
    return -1;
}

__declspec(dllexport) void Moca_LoadExtend(mapi_shell *sh)
{
    mapi_register_special_law(sh,
        my_law_init, my_law_verif, my_law_delay);
}

```

5.4. Description des types/fonctions de l'API

5.4.1. Déclaration des différents types gérés par l'API

5.4.1.1. Types associés à des données/objets

mapi_shell

Type abstrait représentant l'interpréteur de commandes de Moca-RP^c.

mapi_net

Type abstrait représentant un réseau de Petri.

mapi_transition

Type abstrait représentant une transition du réseau de Petri.

mapi_expression

Type abstrait représentant une expression du réseau de Petri.

mapi_table

Type abstrait représentant un tableau du réseau de Petri.

mapi_external_id, mapi_internal_id

Ces types représentent respectivement les numérotations internes et externes des places du réseau de Petri. Dans la mesure où les places peuvent avoir des numéros de description non consécutifs, Moca-RP^c les renumérote en interne.

L'obtention des informations relatives aux places s'effectue via la numérotation interne (*mapi_internal_id*). Le module client a la possibilité de récupérer le numéro interne d'une place en fonction de son numéro externe (celui de la description) via la fonction *mapi_get_place_internal_id*

mapi_double

Type des nombres réels utilisés par Moca-RP^c. Ce type définit la précision de nombres en virgules flottantes. A priori le module client de l'API Moca-RP^c doit utiliser ce type lors de ses calculs.

mapi_NaN

Valeur abstraite d'un nombre en virgule flottante erronée (division par zéro, ...)

5.4.1.2. Prototype de fonctions associées aux fonctions externes

```
mapi_double mapi_special_func_proc(net, nb_arguments, arguments);

mapi_net *net;
int nb_arguments;
mapi_double * arguments;
```

Prototype des fonctions de calcul des fonctions externes définies par routines C.

Ces fonctions reçoivent en argument le réseau de Petri en cours de simulation *net*.

Les arguments de la fonction externe considérée sont donnés dans le tableau *arguments* qui contient *nb_arguments* nombre réels. Le type des arguments a été validé en fonction de la signature de la fonction externe.

la valeur renvoyée est éventuellement modifiée en fonction de la signature, puis utilisée au sein de Moca comme résultat de l'expression.

5.4.1.3. Prototype de fonctions associées aux lois de délai

```
int mapi_special Law_init_proc(net, T, number);

mapi_net *net;
mapi_transition * T;
int number;
```

Prototype des fonctions d'initialisation des lois de délai définies par routines C.

Ces fonctions ont en charge l'initialisation des données du client pour la loi numéro *number* associée à la transition *T* du réseau de Petri *net*.

```
int mapi_special Law_verif_proc(net, T, number, nb_arguments, arguments,
mess, code);

mapi_net * net;
mapi_transition * T;
int number;
int nb_arguments;
mapi_double * arguments;
char **mess;
int *code;
```

Prototype des fonctions de vérification des lois de délai définies par routines C.

Ces fonctions ont en charge la vérification des données du client pour la loi numéro *number* associée à la transition *T* du réseau de Petri *net*. Les arguments de la loi considérée sont donnés dans le tableau *arguments* qui contient *nb_arguments* nombre réels.

Ces fonctions retournent 0 si il n'y a pas de problème, 1 si les paramètres engendrent une erreur irrécupérable, 2 si l'erreur est de type avertissement.

Dans le cas d'une erreur, ces fonctions doivent modifier les paramètres *mess* et *code* afin respectivement d'afficher de manière explicite le message d'erreur et de retourner un code d'erreur via.

```
mapi_double mapi_special Law_delay_proc(net, T, number, nb_arguments,
arguments);

mapi_net *net;
mapi_transition * T;
int number;
int nb_arguments;
mapi_double * arguments;
```

Prototype des fonctions de calcul de délai pour les lois de délai définies par routines C.

Ces fonctions reçoivent en argument le réseau de Petri en cours de simulation *net*, la transition *T* dont on désire connaître le délai avant tir, le numéro *number* de la loi spéciale considérée associée à la transition.

Les arguments de la loi considérée sont donnés dans le tableau *arguments* qui contient *nb_arguments* nombre réels. Ces arguments ont été vérifiés au préalable.

La fonction doit renvoyer le délai entre l'instant de validation de la transition et l'instant de tir de la transition.

5.4.1.4. Prototype de fonctions associées aux lois de tir

```
int mapi_special_fire_init_proc(net, T, number);

mapi_net *net;
mapi_transition * T;
int number;
```

Prototype des fonctions d'initialisation des *lois de tir* définies par routines C.

Cette fonction a en charge l'initialisation des données du client pour la loi numéro *number* associée à la transition *T* du réseau de Petri *net*.

```
int mapi_special_fire_verif_proc(net, T, number, nb_arguments, arguments,
mess, code);

mapi_net *net;
mapi_transition * T;
int number;
int nb_arguments;
mapi_double * arguments;
char **mess;
int *code;
```

Prototype des fonctions de vérification des *lois de tir* définies par routines C.

Ces fonctions ont en charge la vérification des données du client pour la loi numéro *number* associée à la transition *T* du réseau de Petri *net*. Les arguments de la loi considérée sont donnés dans le tableau *arguments* qui contient *nb_arguments* nombre réels.

Ces fonctions retournent 0 si il n'y a pas de problème, 1 si les paramètres engendrent une erreur irrécupérable, 2 si l'erreur est de type avertissement.

Dans le cas d'une erreur, ces fonctions doivent modifier les paramètres *mess* et *code* afin respectivement d'afficher de manière explicite le message d'erreur et de retourner un code d'erreur via.

```
int mapi_special_fire_valid_proc(net, T, number);

mapi_net *net;
mapi_transition * T;
int number;
```

Prototype des fonctions de validation des *lois de tir* définies par routines C.

Ces fonctions sont appelées au moment de la validation d'une transition. Elles permettent la mémorisation d'informations à cet instant (comme le temps courant ou un nombre aléatoire).

```
mapi_double mapi_special_fire_trigger_proc(net, T, number, nb_arguments,
arguments);
```

```

mapi_net *net;
mapi_transition * T;
int number;
int nb_arguments;
mapi_double * arguments;

```

Prototype des fonctions de tir pour les *lois de tir* définies par routines C.

Ces fonctions reçoivent en argument le réseau de Petri en cours de simulation *net*, la transition *T* à tirer, le numéro *number* de la loi spéciale associée à la transition.

Les arguments de la loi sont donnés dans le tableau *arguments* qui contient *nb_arguments* nombre réels. Ces arguments ont été vérifiés au préalable.

La fonction doit effectuer réellement le tir de la transition.

5.4.2. Liste des différentes fonctions

5.4.2.1. Fonctions à utiliser au sein de Moca_LoadExtend

```

void mapi_add_function(sh, name, signature, description, (* funct_proc));

mapi_shell * sh;
char *name;
char *signature;
char *description;
mapi_special_funct_proc (* funct_proc);

```

Cette fonction permet d'ajouter un opérateur/fonction externe au sein de l'interpréteur de commande de Moca-RP^c (*sh*).

Cette fonction est identifiée à l'aide de son nom (*name*), de sa signature (*signature*) et d'un commentaire éventuel (*description* [peut-être être NULL]).

Pour plus d'informations sur la signature d'une fonction, voir ???.

Le pointeur de fonction doit être valide (non NULL), est de type *mapi_special_funct_proc*, est appelée à chaque fois que l'expression doit être recalculée et doit renvoyer un nombre réel correspondant à son résultat en fonction de ces paramètres et de sa signature.

```

void mapi_register_special_law(sh, (* init_proc), (* verif_proc), (*
delay_proc));

mapi_shell * sh;
mapi_special_law_init_proc (* init_proc);
mapi_special_law_verif_proc (* verif_proc);
mapi_special_law_delay_proc (* delay_proc);

```

Cette fonction permet d'ajouter de nouvelles lois de délai au sein de l'interpréteur de commande de Moca-RP^c (*sh*).

Chaque pointeur de fonction doit être valide (non NULL) et doit remplir correctement son contrat.

- (** init_proc*) est un pointeur de fonction de type *mapi_special_law_init_proc*. Cette fonction est appelée lors de la création de la loi spéciale. Elle sert éventuellement à initialiser la loi spéciale. Pour plus d'information, voir *mapi_special_law_init_proc*.

- (** verif_proc*) est un pointeur de fonction de type `mapi_special_law_verif_proc`. Cette fonction est appelée à chaque fois que les arguments de la loi ont été modifiés (une seule fois si l'ensemble des arguments sont constants) afin de vérifier les arguments de la loi. Pour plus d'information, voir `mapi_special_law_verif_proc`.
- (** delay_proc*) est un pointeur de fonction de type `mapi_special_law_delay_proc`. Cette fonction est appelée à chaque fois que la transition devient valide et qu'il faut calculer un nouveau délai de tir. Pour plus d'information, voir `mapi_special_law_delay_proc`.

```
void mapi_register_special_fire(sh, (* init_proc), (* verif_proc), (*
valid_proc), (* trigger_proc));
```

```
mapi_shell * sh;
mapi_special_fire_init_proc (* init_proc);
mapi_special_fire_verif_proc (* verif_proc);
mapi_special_fire_valid_proc (* valid_proc);
mapi_special_fire_trigger_proc (* trigger_proc);
```

Cette fonction permet d'ajouter de nouvelles lois de tir au sein de l'interpréteur de commande de Moca- RP^c (*sh*).

Chaque pointeur de fonction doit être valide (non NULL) et doit remplir correctement son contrat.

- (** init_proc*) est un pointeur de fonction de type `mapi_special_fire_init_proc`. Cette fonction est appelée lors de la création de la loi spéciale. Elle sert éventuellement à initialiser la loi spéciale. Pour plus d'information, voir `mapi_special_fire_init_proc`.
- (** verif_proc*) est un pointeur de fonction de type `mapi_special_fire_verif_proc`. Cette fonction est appelée à chaque fois que les arguments de la loi ont été modifiés (une seule fois si l'ensemble des arguments sont constants) afin de vérifier les arguments de la loi. Pour plus d'information, voir `mapi_special_fire_verif_proc`.
- (** valid_proc*) est un pointeur de fonction de type `mapi_special_fire_valid_proc`. Cette fonction est appelée à chaque fois que la transition devient valide afin de mémoriser d'éventuels informations. Pour plus d'information, voir `mapi_special_fire_valid_proc`.
- (** trigger_proc*) est un pointeur de fonction de type `mapi_special_fire_trigger_proc`. Cette fonction est appelée lorsque la transition doit être tirée. Cette fonction doit effectuer toutes les phases de tir de la transition (affectation des variables, suppression de jetons dans les places amonts et ajout de jeton dans les places aval) Pour plus d'information, voir `mapi_special_fire_trigger_proc`.

5.4.2.2. Fonctions générales sur les réseaux de Petri

```
mapi_double mapi_get_time(net);
```

```
mapi_net * net;
```

Cette fonction retourne le temps courant de la simulation dans l'histoire courante du réseau de Petri *net*.

```
mapi_double mapi_get_probability(net);
```

```
mapi_net * net;
```

Cette fonction retourne un nombre aléatoire réel dans l'intervalle $[0,1[$. Le générateur de nombre au hasard utilisé est celui de la simulation en cours pour le réseau de Petri *net*.

```
void mapi_add_warning(net, warning);
```

```
mapi_net * net;  
char *warning;
```

Cette fonction permet d'ajouter un avertissement *warning* lors de la simulation du r  seau de Petri *net*.

Ces avertissements seront affich  s    la fin de la simulation.

5.4.2.3. Fonctions : Informations sur les transitions

A partir d'une transition, l'API de Moca-RP^C permet d'obtenir les informations concernant les places en amont et en aval de la transition, ainsi que sur le vecteur d'affectation compos   de couple {variable, expression} de la transition.

```
char *mapi_get_transition_name(net, T);
```

```
mapi_net * net;  
mapi_transition * T;
```

Cette fonction retourne le nom de la transition *T* du r  seau de Petri *net* sous forme d'une cha  ne de caract  res.

```
int mapi_get_number_of_input_places(net, T);
```

```
mapi_net * net;  
mapi_transition * T;
```

Cette fonction retourne le nombre d'arcs entrant de la transition *T* du r  seau de Petri *net*.

```
mapi_internal_id mapi_get_nth_input_places(net, T, idx);
```

```
mapi_net * net;  
mapi_transition * T;  
int idx;
```

Cette fonction retourne le num  ro interne de la place associ  e au $(idx-1)^{i  me}$ arc amont de la transition *T* du r  seau de Petri *net*.

```
int mapi_get_weight_nth_input_edge(net, T, idx);
```

```
mapi_net * net;  
mapi_transition * T;  
int idx;
```

Cette fonction retourne le poids du $(idx-1)^{i  me}$ arc amont de la transition *T* du r  seau de Petri *net*.

```
int mapi_get_number_of_output_places(net, T);
```

```
mapi_net * net;  
mapi_transition * T;
```

Cette fonction retourne le nombre d'arcs sortant de la transition T du réseau de Petri net .

```
mapi_internal_id mapi_get_nth_output_places(net, T, idx);
```

```
mapi_net * net;
mapi_transition * T;
int idx;
```

Cette fonction retourne le numéro interne de la place associée au $(idx-1)^{ième}$ arc aval de la transition T du réseau de Petri net .

```
int mapi_get_weight_nth_output_edge(net, T, idx);
```

```
mapi_net * net;
mapi_transition * T;
int idx;
```

Cette fonction retourne le poids du $(idx-1)^{ième}$ arc aval de la transition T du réseau de Petri net .

```
int mapi_get_number_of_affect_variable(net, T);
```

```
mapi_net * net;
mapi_transition * T;
```

Cette fonction retourne le nombre d'affectation associée à la transition T du réseau de Petri net .

Dans le cas d'une transition de type affectation itérative, cette fonction renvoie -1.

```
mapi_expression * mapi_get_nth_affect_variable(net, T, idx);
```

```
mapi_net * net;
mapi_transition * T;
int idx;
```

Cette fonction retourne la $(idx-1)^{ième}$ variable "affectée" à la transition T du réseau de Petri net . Il s'agit de la variable à affecter.

```
mapi_expression * mapi_get_nth_affect_expression(net, T, idx);
```

```
mapi_net * net;
mapi_transition * T;
int idx;
```

Cette fonction retourne la $(idx-1)^{ième}$ expression "affectée" à la transition T du réseau de Petri net . Il s'agit de la valeur de l'affectation.

5.4.2.4. Fonctions : Informations sur les places et les variables

```
mapi_internal_id mapi_get_place_internal_ident(net, P);
```

```
mapi_net * net;
```



```
mapi_external_id P;
```

Cette fonction transforme un numéro de place externe P (décrit par le constructeur du réseau de Petri) en un numéro de place interne au réseau de Petri net .

Si la place externe P n'existe pas, cette fonction retourne -1.

```
int mapi_get_place_marking(net, P);
```

```
mapi_net * net;  
mapi_internal_id P;
```

Cette fonction retourne le marquage courant (nombre de jeton) de la place identifiée par le numéro interne P du réseau de Petri net .

```
mapi_expression * mapi_get_expression(net, name);
```

```
mapi_net * net;  
char *name;
```

Cette fonction retourne la variable ou le paramètre associé au nom $name$ au sein du réseau de Petri net . Si cette variable n'existe pas, elle renvoie NULL.

```
mapi_double mapi_get_expression_value(net, expr);
```

```
mapi_net * net;  
mapi_expression * expr;
```

Cette fonction retourne la valeur courante de l'expression $expr$ du réseau de Petri net .

```
mapi_table * mapi_get_table(net, name);
```

```
mapi_net * net;  
char *name;
```

Cette fonction retourne le tableau associé au nom $name$ au sein du réseau de Petri net . Si ce tableau n'existe pas, elle renvoie NULL.

```
int mapi_get_table_size(net, tab);
```

```
mapi_net * net;  
mapi_table * tab;
```

Cette fonction retourne la taille du tableau tab au sein du réseau de Petri net .

```
mapi_double mapi_get_table_value_at(net, tab, index);
```

```
mapi_net * net;  
mapi_table * tab;  
int index;
```

Cette fonction retourne la $index^{i\grave{e}me}$ valeur courante du tableau *tab* du r  seau de Petri *net*.
index doit   tre compris entre 1 et la taille du tableau *tab*.

5.4.2.5. Fonctions de modification du r  seau

Ces fonctions ne doivent   tre appel  es que lors du tir d'une transition.

```
void mapi_set_place_marking(net, P, mark);  
  
mapi_net * net;  
mapi_internal_id P;  
int mark;
```

Cette fonction fixe au sein du r  seau de Petri *net*. le marquage courant de la place identifi  e par le num  ro interne *P*    la valeur *mark*.

```
void mapi_set_expression_value(net, expr, value);  
  
mapi_net * net;  
mapi_expression * expr;  
mapi_double value;
```

Cette fonction fixe au sein du r  seau de Petri *net*. la valeur de l'expression *expr*    la valeur *value*.

A. Lecture de fichiers au format Moca10

Le format de données de la version actuel de Moca-RP^C n'est pas compatible avec les versions précédentes. Pour des raisons de compatibilité, il est tout de même possible d'ouvrir directement au sein de la version actuelle des fichiers dans l'ancien format.

Dans la suite de cette annexe nous appellerons l'ancien format de données le format Moca10 bien qu'en réalité, il s'agit du format de donnée pour les versions 8.x, 9.X et 10.x du logiciel Moca-RP^C.

Le format Moca10 est décrit en détail dans le manuel utilisateur de la version précédente de Moca-RP^C (Gérald Point. Manuel de l'Utilisateur Moca-RP^C 10.04 (Révision 0). IXI, Réf. : ISI184/PTGD/MU/1004-0, février 2000)

Pour charger un fichier au format Moca10 dans l'interpréteur de commandes, il suffit d'utiliser la commande `load Moca10 "<file-in>" "<options>"` ; où `<file-in>` est le nom du fichier à interpréter encadré par des guillemets et `<options>` des éventuels options.

Cette commande interprète le fichier `<file-in>` et traduit le réseau de Petri au format Moca10 en un réseau de Petri au format actuel nommé `<file-in>`.

La traduction concerne principalement les points suivants :

- Les paramètres nommés sont transformés en constantes.
- Les lois à la sollicitation sont traduites en lois de délai et de tir équivalent.
- Les lois optionnelles sont transformées en loi de base avec des paramètres définis à l'aide d'expression.
- La loi exponentielle + wow à 13 paramètres est traduite dans son équivalent à 2 paramètres avec une expression faisant intervenir l'opérateur `time()`.

Il existe des options supplémentaires lors de l'ouverture du fichier de données :

- `-name <id-net>` : où `<id-net>` est le nom de réseau de Petri (à la place de `<file-in>`).
- `-cmd <file-cmd>` : où `<file-cmd>` est le nom du fichier où sera écrit toutes les commandes permettant de faire une simulation identique à celle qui devait être fait par le fichier au format Moca10.
- `-out <file-out>` : où `<file-out>` est le nom du fichier où sera redirigé l'affichage des résultats. Cette option n'a de sens que si la précédente est également présente.

Ces options doivent être placées entre les guillemets.

Par exemple la commande suivante `load Moca10 "test.mk10" "-name test -cmd cmd.mok -out test.res"` ; ajoute au sein de l'interpréteur de commandes le réseau de Petri nommé `test` issu de la traduction du fichier `test.mk10` et crée le fichier de commande `cmd.mok`. La simulation peut être lancée à l'aide de la commande `load "cmd.mok"` ;. Les résultats doivent être similaires à une simulation du fichier `test.mk10` réalisé à l'aide d'une ancienne version de Moca-RP^C. Ces résultats sont alors affichés dans le fichier `test.res`.

B. Lecture de fichiers au format Aralia

Aralia est un package BDD (Diagramme de Décision Binaire) spécialisé dans le traitement des modèles booléens d'analyse des risques. Son principal avantage est qu'il fournit un résultat exact tant d'un point qualitatif que quantitatif et ceci de manière rapide. Son principal inconvénient est qu'il s'agit d'un logiciel "tout ou rien". Il arrive soit à traiter la formule booléenne dans des temps raisonnables, soit il échoue. De plus, dans le cas de composants réparables, il ne peut calculer que la disponibilité du système. Les calculs d'approximation de la fiabilité ne sont 'valables' que pour des événements ayant des taux de panne et de réparation exponentiel (loi exponentiel et GLM).

Pour des raisons expérimentales et afin d'avoir un moyen correct d'évaluer la fiabilité d'un système de sûreté décrit à l'aide d'un arbre de défaillance, un module de chargement de formule au format Aralia a été développé au sein de Moca-RP^c.

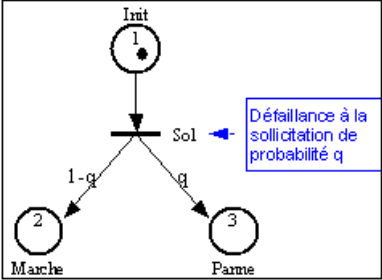

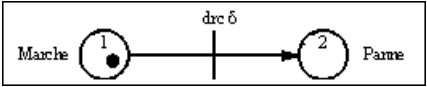
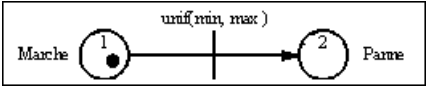
Ce module prend en paramètre une formule booléenne au format Aralia et le traduit en réseau de Petri :

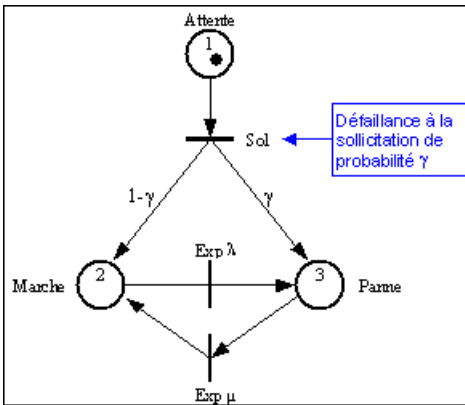
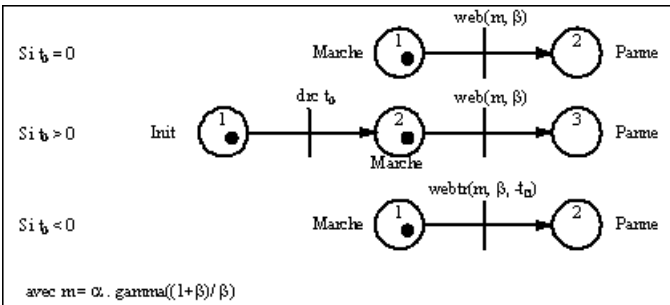
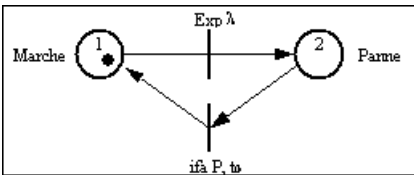
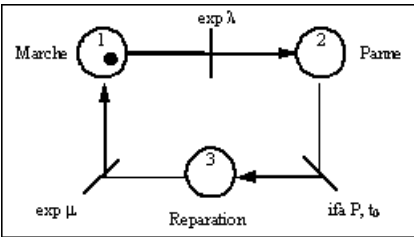
- Les paramètres nommés sont représentés dans Moca-RP^c comme des variables réelles.
- Les événements de base sont simulés à l'aide de petits réseaux de Petri indépendants suivant leur loi de probabilité.
- Les formules booléennes sont définies à l'aide de variables et d'expressions booléennes au sein du réseau de Petri.
- Des observateurs (état statistiques) sont ajoutés pour les variables booléennes.
- Des réseaux de Petri sont ajoutés afin de capturer la fiabilité des événements sommets de l'arbre.

Il existe un certain nombre de limites quand à l'utilisation de ce traducteur :

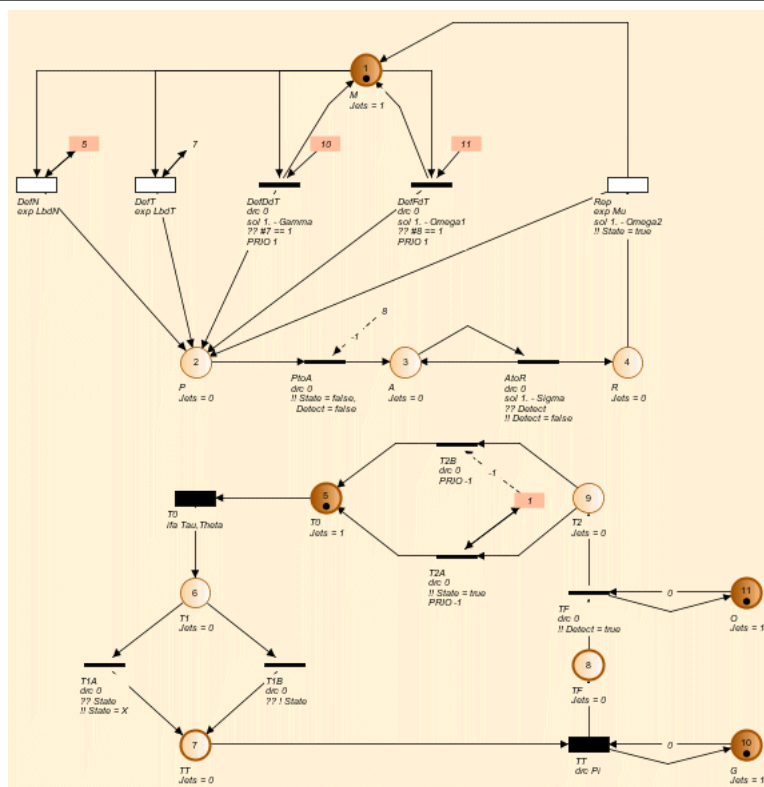
- Les opérateurs booléens suivant ne sont pas pris en compte (ils génèrent une erreur) : les implications, si et seulement si, ou exclusif, quantificateurs existentiel et universel
- Certaines lois de probabilité ne peuvent pas ou non pas été traduites en réseau de Petri : bound-time, factor (utilisé dans les cas de défaillance de cause commune), dormant, plus de nombreuses lois Aralia V4 (non présente au sein de Aralia WorkShop)

La correspondance entre les lois de probabilités et les réseaux de Petri simulant les événements de base utilisant ces lois est défini à l'aide du tableau suivant :

Loi Aralia	Réseau de Petri équivalent
constant q (Et les équivalents : GLM-Asymptotic, NRD, CMT)	
exponential λ	
Dirac d	
uniform min max	
GLM $g \lambda m$	

	
Weibull a, b, t_0	<p>Si $t_0 = 0$</p> <p>Si $t_0 > 0$</p> <p>Si $t_0 < 0$</p> <p>avec $m = \alpha \cdot \text{gamma}((1+\beta)/\beta)$</p> 
periodic-test l, P, t_0	
periodic-test l, m, P, t_0	
periodic-test $l, l^*, m, t, q, g, p, x, s, w$	<p>Traiter comme une loi test périodique à 11 paramètres periodic-test $l, l^*, m, t, q, g, p, x, s, 0, w$</p>

periodic-test 1,
l*, m, t, q, g, p,
x, s, w1, w2



Variables			
Domaine	Nom	Définition/Valeur initi...	Observée
Booléen	Detect	false	
Booléen	State	true	

Paramètres		
Domaine	Nom	Valeur
Réel	Gamma	0.0
Réel	LbdN	0.0010
Réel	LbdT	0.0010
Réel	Mu	0.01
Réel	Omega1	0.1
Réel	Omega2	0.0
Réel	Pi	10.0
Réel	Sigma	1.0
Réel	Tau	100.0
Réel	Theta	100.0
Booléen	X	true

Places		
Numéro	Nom	Marquage initial
1	M	1
2	P	0
3	A	0
4	R	0
5	T0	1
6	T1	0
7	TT	0
8	TF	0
9	T2	0
10	G	1
11	O	1

Pour charger une formule booléenne dans Moca, il suffit d'utiliser la commande load Aralia "<file-in>" "<options>"; où <file-in> contient la formule booléenne considérée (ainsi que les lois des événements de base) au format Aralia et <options> des éventuels options.

Les options supplémentaires sont les suivantes :

- `-name <id-net>` : où `<id-net>` est le nom de réseau de Petri (à la place de Aralia).
- `-all` : permet d'ajouter un observateur pour toutes les variables de la formule booléenne. Dans le cas contraire, un observateur est ajouté uniquement pour les variables sommets.
- `-duration <int>` : permet de spécifier la durée d'histoire du réseau.
- `-hst` : précise qu'un histogramme est associé à chaque observateur du réseau.
- `-times '<times>'` : permet de spécifier les différents temps de calcul pour les états statistiques. Cf. Section 2.2, « Rubrique : Options » pour la syntaxe de `<times>`.
- `-types '<types>'` : permet de spécifier les types de statistiques à effectuer. Cf. Section 2.10, « Rubrique : Type de statistiques » pour une définition des types de calcul et de la syntaxe de `<types>`. Dans le cas d'un calcul de fiabilité et de disponibilité instantanée (équivalence avec Aralia), le type 2 est à prendre en compte.

Après expérimentation, on se rend compte rapidement des limites d'une telle solution pour quantifier un arbre de défaillance.

Cette démarche n'est applicable qu'en cas de probabilité d'occurrence (fiabilité et/ou disponibilité) mesurable ou significative (supérieur à 10^{-4}).

Pour avoir un estimateur statistique correct d'un événement ayant une probabilité d'occurrence de 10^{-6} , il convient de réaliser au minimum 10^{+8} histoires. Cela engendre un temps de calcul déraisonnable et interdit l'utilisation de cette méthode lorsque la probabilité d'occurrence de l'événement considéré est inférieure à cette valeur. Le nombre d'histoires maximum est de la taille d'un entier sur 32 bits, soit $2^{31}-1 = 2\,147\,483\,647 \sim 2 \cdot 10^9$.

C. Calcul de la moyenne et de la variance

Cette annexe précise la manière de calculer la moyenne et la variance lors d'une simulation standard et lors d'une simulation par lots (batch), en prenant en compte les problèmes de précision de calcul qui sont engendrés par un nombre de simulations toujours plus important.

1. Lors de la simulation

Afin de garantir une bonne précision des calculs, il convient de trouver une formule de récurrence pour la moyenne (noté M_N) et la variance (noté V_N) d'un échantillon contenant N données x_1, \dots, x_N . C'est-à-dire, trouvez M_{N+1} et V_{N+1} en ne connaissant que M_N , V_N , N et X_{N+1} .

Pour la moyenne, il est aisé de vérifier l'équation suivante :

$$M_{N+1} = \frac{X_{N+1} + N.M_N}{N+1}$$

La variance V_N s'exprime à l'aide de la formule :

$$V_N = \frac{\sum_{i=1}^N (X_i - M_N)^2}{N-1}$$

En développant cette formule, nous obtenons l'équation suivante :

$$(N-1)V_N = \sum_{i=1}^N (X_i)^2 - N.(M_N)^2$$

En passant à $N+1$, nous obtenons :

$$N.V_{N+1} = \sum_{i=1}^{N+1} (X_i)^2 - (N+1)(M_{N+1})^2 = (X_{N+1})^2 + \sum_{i=1}^N (X_i)^2 - (N+1)(M_{N+1})^2$$

En reprenant la définition de la moyenne et de la variance, nous obtenons :

$$N.V_{N+1} = (X_{N+1})^2 + (N-1)V_N + N.(M_N)^2 - \frac{(X_{N+1} + N.M_N)^2}{N+1}$$

$$N.V_{N+1} = (X_{N+1})^2 + (N-1)V_N + N.(M_N)^2 - \frac{(X_{N+1})^2 + 2.X_{N+1}.N.M_N + (N.M_N)^2}{N+1}$$

$$N.V_{N+1} = \frac{N}{N+1} (X_{N+1})^2 - \frac{2N}{N+1} .X_{N+1}.M_N + \frac{N}{N+1} .(M_N)^2 + (N-1)V_N$$

D'où

$$V_{N+1} = \frac{(X_{N+1} - M_N)^2}{N+1} + \frac{N-1}{N} .V_N$$

2. Lors de la fusion de 2 lots

La parallélisation d'une simulation de Monte-Carlo engendre un traitement par lot. Chaque lot est caractérisé par la graine initiale du générateur de nombre aléatoire, sa taille, la moyenne et la variance de chacun des estimateurs statistiques de la simulation. Il est alors nécessaire de calculer la moyenne et la variance de chacun des estimateurs statistiques pour l'ensemble des lots (et donc des histoires simulées).

2.1. Présentation

Plutôt que de travailler sur l'ensemble des lots calculés, il nous est apparu logique de partir du premier lot et d'ajouter les résultats des autres lots un à un afin de garantir au mieux la précision des calculs (de la manière analogue à une formule de récurrence).

Soit 2 échantillons A et B. L'échantillon A comporte n données a_1, \dots, a_n . L'échantillon B, m données b_1, \dots, b_m . Soit M_A et M_B les moyennes des échantillons A et B ; et V_A et V_B , leurs variances.

Nous rappelons leurs expressions :

$$M_A = \frac{\sum_{i=1}^n a_i}{n} \quad \text{et} \quad V_A = \frac{\sum_{i=1}^n (a_i - M_A)^2}{n-1} = \frac{\sum_{i=1}^n (a_i^2) - n.M_A^2}{n-1}$$

$$M_B = \frac{\sum_{j=1}^m b_j}{m} \quad \text{et} \quad V_B = \frac{\sum_{j=1}^m (b_j - M_B)^2}{m-1} = \frac{\sum_{j=1}^m (b_j^2) - m.M_B^2}{m-1}$$

Nous recherchons la moyenne M_C et la variance V_C de l'échantillon C composé de n+m données $a_1, \dots, a_n, b_1, \dots, b_m$.

2.2. Développements mathématiques

La moyenne M_C ne pose pas de difficultés :

$$M_C = \frac{\sum_{i=1}^n a_i + \sum_{j=1}^m b_j}{n+m} = \frac{n.M_A + m.M_B}{n+m}$$

La variance V_C s'exprime à l'aide de la formule :

$$V_C = \frac{\sum_{i=1}^n (a_i - M_C)^2 + \sum_{j=1}^m (b_j - M_C)^2}{n+m-1}$$

En développant cette formule, nous obtenons les équations suivantes :

$$\begin{aligned} (n+m-1).V_C &= \sum_{i=1}^n (a_i^2 + M_C^2 - 2.a_i.M_C) + \sum_{j=1}^m (b_j^2 + M_C^2 - 2.b_j.M_C) \\ &= \sum_{i=1}^n (a_i^2) + n.M_C^2 - 2.M_C \cdot \sum_{i=1}^n (a_i) + \sum_{j=1}^m (b_j^2) + m.M_C^2 - 2.M_C \cdot \sum_{j=1}^m (b_j) \\ &= \sum_{i=1}^n (a_i^2) + \sum_{j=1}^m (b_j^2) + (n+m).M_C^2 - 2.n.M_A.M_C - 2.m.M_B.M_C \\ &= \sum_{i=1}^n (a_i^2) + \sum_{j=1}^m (b_j^2) + (n+m).M_C^2 - 2.M_C.(n.M_A + m.M_B) \\ &= \sum_{i=1}^n (a_i^2) + \sum_{j=1}^m (b_j^2) + (n+m).M_C^2 - 2.(n+m).M_C^2 \end{aligned}$$

Il convient alors de revenir aux définitions des variances de A et de B :

$$\begin{aligned} (n+m-1).V_C &= \sum_{i=1}^n (a_i^2) - n.M_A^2 + n.M_A^2 + \sum_{j=1}^m (b_j^2) - m.M_B^2 + m.M_B^2 - (n+m).M_C^2 \\ (n+m-1).V_C &= (n-1).V_A + (m-1).V_B + n.M_A^2 + m.M_B^2 - (n+m).M_C^2 \end{aligned}$$

Et pour finir on obtient alors :

$$V_C = \frac{(n-1)V_A + (m-1)V_B + n.M_A^2 + m.M_B^2 - (n+m)M_C^2}{(n+m-1)}$$

2.3. Résultats

A partir de deux échantillons (A et B) dont nous ne connaissons que la taille (respectivement n et m), la moyenne (M_A et M_B) et la variance (V_A et V_B), nous pouvons calculer la taille (n+m), la moyenne (M_C) et la variance (V_C) de l'union des deux échantillons à l'aide des équations :

$$M_C = \frac{n.M_A + m.M_B}{n+m}$$

$$V_C = \frac{(n-1)V_A + (m-1)V_B + n.M_A^2 + m.M_B^2 - (n+m)M_C^2}{(n+m-1)}$$