Gérard Berry

Collège de France
Chaire Informatiques et sciences numériques
gerard.berry@college-de-france.fr
http://www.college-de-france.fr/site/gerard-berry/index.htm

Pages personnelles : http://www-sop.inria.fr/members/Gerard.Berry

Résumé général du cours 2017-2018, « Esterel de A à Z », v1, 21/01/2018

Le langage de programmation parallèle synchrone Esterel est spécifiquement dédié aux systèmes cyber-physiques temps-réels, aux circuits électroniques et, de façon plus générale, aux *systèmes réactifs* qui maintiennent une interaction avec leur environnement par la réception et l'émission de valeurs d'événements : automatismes de tous types, robotique, protocoles de communication, interaction homme-machine, etc. Esterel a été partiellement présenté à plusieurs reprises dans les cours précédents : de 2012 à 2014 dans le cadre plus large du traitement informatique du temps et des événements en informatique, de 2014 à 2016 dans celui de la vérification formelle des programmes et circuits et, enfin, de façon différente en 2014 dans trois cours donnés à l'Inria Sophia-Antipolis sur la relation science-industrie.

Le cours de cette année 2017-2018 étudiera Esterel de façon beaucoup plus approfondie et unifiée, en discutant le design du langage, l'évolution de ses sémantiques formelles, les techniques de compilation qui en découlent directement, et la façon dont elles ont étroitement dirigé la conception de l'outillage académique et industriel pour sa mise en œuvre.

Un *leitmotiv* du cours sera de montrer que le développement d'Esterel a toujours énergiquement refusé les compromis souvent présents dans d'autres langages textuels ou graphiques qui sacrifient quelque peu la clarté sémantique pour traiter les cas particuliers embêtants quand ils se présentent. Au contraire, nous avons toujours fondé son développement universitaire et industriel sur la clarté et la rigueur de sa sémantique mathématique. Nous verrons à plusieurs reprises à quel point cette attitude s'est avérée payante.

En complément des résumés des cours et séminaires, il me paraît utile de présenter ici l'histoire technique d'Esterel en relation avec celle des autres langages synchrones, notamment Lustre et Signal. Cette histoire montre bien pourquoi l'interaction permanente entre design de langage, approche mathématique et applications est indispensable dans des projets ambitieux de ce type. D'autres aspects historiques ont déjà été présentés dans les cours à Sophia-Antipolis précités.

La naissance des langages synchrones

Esterel a été conçu et développé depuis 1982 au sein d'un projet commun entre l'École des Mines et l'Inria à Sophia-Antipolis qui rassemblait des informaticiens et des automaticiens (spécialistes de la théorie mathématique du contrôle des systèmes). Jean-Paul Marmorat, automaticien, et Jean-Paul Rigault, à la formation mixte informatique / automatique, ont joué un rôle essentiel dans sa naissance. Esterel, dont le premier papier publié est [BeMo83], est le plus ancien des *langages parallèles synchrones* [BeBe91], dans lesquels on suppose que les réactions aux événements extérieurs et les communications entre processus parallèles programmés sont conceptuellement instantanées ou, de façon équivalente, que l'ordinateur qui calcule la réaction est infiniment rapide. Cette hypothèse est implicite en automatique : quand on écrit une équation d'évolution temporelles de variables comme $z_t = x_t + z_{t-1}$, on ne parle pas du temps qu'il faut pour

calculer l'addition. Généraliser l'hypothèse synchrone permet de grandement simplifier la spécification et la programmation des systèmes cyber-physiques temps-réels, c'est-à-dire de ceux où l'on commande informatiquement des systèmes physiques sensibles au temps : avions, trains, robots, etc. Le synchronisme garantit en particulier par construction le *déterminisme* des applications qui est le plus souvent une contrainte importante des applications embarquées.

Mais l'automatique n'est pas l'informatique : notre hypothèse de synchronisme parfait est exactement opposée à celle du dogme standard de l'informatique parallèle qui a tendance à identifier parallélisme et asynchronisme de l'exécution des processus et de leur communication, imposant ainsi un non-déterminisme intrinsèque des modèles et langages. L'hypothèse synchrone fut donc longtemps jugée iconoclaste. Elle s'est au contraire avérée remarquablement féconde. En évitant dès l'écriture toute interférence entre le temps du processus à contrôler et celui pris par l'exécution du programme, elle conduit à des sémantiques mathématiques délicates mais qui s'avèrent en pratique nettement plus simples et contrôlables que celles de l'asynchrone, garantissant de plus le déterminisme par construction. Le synchronisme permet ainsi une spécification et une programmation bien plus élégantes, un débogage beaucoup plus facile, une génération de codes objets bien plus efficaces et temporellement contrôlables qu'avec les approches asynchrones, ainsi qu'une une connexion facile avec les systèmes de vérification automatique étudiés dans le cours 2015-2016. Mais il n'y a pas d'opposition entre les deux types de parallélisme : l'approche synchrone est d'application moins générale que l'approche asynchrone, et synchronisme et asynchronisme s'avèrent tout à fait compatibles si on agrandit le dessin. Nous le verrons pour la commande de robot où un contrôleur synchrone commande le comportement mécanique des organes du robot, lui naturellement asynchrone, et pour l'orchestration de service Web, où une logique synchrone permet de mieux contrôler et coordonner des activités asynchrones situées sur des machines distantes.

Lustre et Signal : langages flots de données

En parallèle avec Esterel, la même hypothèse de synchronisme parfait a été adoptée par Lustre, langage développé à Grenoble dans l'équipe de Paul Caspi et Nicolas Halbwachs [HaCaPi91], et le langage Signal [GuGa91], développé à Rennes par celle de Paul Le Guernic et Albert Benveniste. Comme pour Esterel, leur conception était conjointe entre des informaticiens et des automaticiens. Les trois langages étaient complémentaires : Esterel visait le contrôle discret dans lequel le système étudié change constamment d'état, alors que Lustre et Signal visaient plutôt le contrôle continu dont le but est de réaliser une transformation complexe mais assez constante de flots de données entrants vers des flots de données sortants. Prenons un exemple en avionique : la gestion complexe des phases de sortie d'un train d'atterrissage ou celle des écrans multiplexés d'un cockpit s'expriment bien en Esterel, celle des lois de pilotage s'exprime naturellement en Lustre, et les opérations subtiles de traitement du Signal sont bien adaptées au langage éponyme, qui est plus général mais plus complexe que Lustre. Au cours des années 2000, les styles d'Esterel et Lustre ont pu être unifiés dans les outils industriels Esterel v7 pour les circuits et SCADE 6 pour les systèmes de haute sûreté développés chez d'Esterel Technologies dont je fus le directeur scientifique de 2001 à 2009.

Le cours et les séminaires

Le cours parlera surtout d'Esterel et Lustre, les résumés de chaque session étant disponibles sur le site https://www.college-de-france.fr/site/gerard-berry/course-2017-2018.htm. Signal et son extension Polychrony seront toutefois abordés en séminaire par Albert Benveniste. De nombreux autres langages synchrones sont apparus après les trois premiers. Les séminaires présenteront les principaux dans la lignée d'Esterel : Reactive ML [MaPo05] de Louis Mandel et Marc Pouzet qui intègre dans la programmation fonctionnelle les idées initiales du langage Reactive C de Frédéric

Boussinot [Bou91], HipHop [SmMo17] de moi-même, Manuel Serrano et Colin Vidal, déjà présenté dans le cours 2015-2016, et enfin ScEst [SmMo17] et Scl de l'équipe de Reinhard von Hanxleden, qui étendent les principes d'Esterel et les appliquent à C en construisant une voie d'avenir très prometteuse. La vérification formelle des sémantiques sera à la fête : le séminaire de Tim Bourke présentera la preuve en machine de la correction d'un compilateur Lustre en utilisant l'assistant de vérification de preuves Coq, et le séminaire final de Lionel Rieg présentera les preuves Coq associées aux différentes sémantiques d'Esterel – travail toujours en cours.

Le meilleur article court pour comprendre Esterel et sa science est [Be00a]. Le livre *Compiling Esterel* [PoEd07] écrit principalement par Dumitru Potop et Stepen Edwards (auteur du très novateur Columbia Esterel Compiler) est bien sûr beaucoup plus complet. Le livre Web [Beo2] présente les sémantiques du noyau d'Esterel et la traduction en circuits digitaux et le *Primer* [Be00b] donne une bonne idée du style de programmation.

Les styles synchrone, textuels ou graphiques

Les langages Esterel, Lustre et Signal ont une forme textuelle qui conduit à des styles de programmations nouveaux, étudiés dans le premier cours. Mais les ingénieurs aimant bien les styles graphiques, souvent utilisés cependant avec des sémantiques quelque peu confuses, Esterel, Lustre et Signal furent donc rapidement suivis de versions graphiques compatibles avec les versions textuelles originales et tout aussi rigoureuses. Pour Esterel, ce fut les SyncCharts [An96], développés à l'Université de Nice par Charles André en reprenant les idées très novatrices du formalisme Statecharts [Ha87] de David Harel et en l'adaptant à la sémantique synchrone d'Esterel. Les SyncCharts furent repris dans les ateliers industriels Agel d'Ilog puis Esterel Studio d'Esterel Technologies, en donnant aux utilisateurs la possibilité de mélanger texte et graphique pour plus de souplesse. Pour Lustre, ce fut l'atelier industriel SCADE (*Safety Critical Application Development Environment*), fondé lui sur une syntaxe graphique des *block diagrams* classique en contrôle de processus. Ce fut ensuite Argos, développé par Florence Maraninchi pour adapter les idées des Statecharts au monde flot de données. Enfin, comme décrit plus loin, l'atelier SCADE 6 d'Esterel Technologies a unifié tous ces aspects. Le premier cours illustrera ces formalismes et montera à quel point il est facile de faire des diagrammes qui ont l'air de faire ce qu'il faut mais dont la sémantique pose des problèmes parfois insolubles ; cela a malheureusement été le cas pour de nombreux systèmes industriels, y compris ceux créés par des sociétés industrielles d'importance majeure.

Les sémantiques mathématiques

Plusieurs cours présenteront l'étude mathématique d'Esterel. Elle est fondée sur les techniques désormais classiques de *sémantique opérationnelle structurelle* (SOS), introduites par Gordon Plotkin dans les années 1980, qui traite la définition sémantique par des règles de logique formelle. Dans le monde synchrone, le problème principal est celui de la *causalité*, c'est-à-dire de la propagation de l'information pendant une réaction du programme. En effet, les programmes synchrones peuvent entraîner des paradoxes logiques analogues au fameux « je mens », exemple de cycle de causalité : l'affirmation devient fausse si elle est vraie et vraie si elle est fausse. Une façon simple d'éviter ce type de paradoxe cyclique est d'exiger une stratification des actions internes vérifiable simplement par un tri topologique. C'est la méthode employée par Lustre et SCADE ; elle est bien adaptée aux programmes flots de données et facilite la certification industrielle des compilateurs qui a toujours été une des grandes forces de SCADE. Introduite pour Esterel avec Laurent Cosserat en 1983 [BeCo84], elle a servi de base au compilateur Esterel v2 que j'ai écrit en 1985 avec Philippe Couronné. Elle s'avère cependant vite insuffisante pour Esterel où la composition de processus parallèles peut provoquer de façon naturelle des cycles tout à fait sains.

Le travail fondamental de Georges Gonthier [BeGo92] au milieu des années 1980 a permis d'aller beaucoup plus loin en fournissant une analyse statique permettant d'accepter beaucoup de cycles sains. Il a servi de base au compilateur Esterel v3. Mais cette détection restait encore incomplète. La traduction en circuits booléens étudiée aux deux cours suivants a permis d'aller beaucoup plus loin, avec une caractérisation exacte des bons cycles fondée sur des idées de logique constructive. La sémantique constructive présentée dans [Be02] est maintenant le standard pour Esterel. L'étude a culminé avec la caractérisation des bons cycles électriques dans les circuits par la logique booléenne constructive, déjà présentée en détail dans le cours du 26 mars 2014 et décrite dans [MeSh12].

De leur côté, Lustre et Signal sont fondés sur une vision fonctionnelle (pour Lustre) ou relationnelle (pour Signal) manipulant des flots infinis, c'est-à-dire des suites discrètes infinies de valeurs infinies. La sémantique de Lustre est très simple car la causalité par stratification s'y avère suffisante en pratique et facilite la certification des programmes. La sémantique de Signal est plus subtile, comme l'illustrera Albert Benveniste.

Toutes ces sémantiques formeront une partie très importante du cours et les séminaires de Tim Bourke et Lionel Rieg montreront qu'on peut complètement automatiser la preuve formelle de leur propriété, poussant ainsi les théorèmes de correction le plus loin possible.

Les premiers compilateurs

Après un prototype Esterel v1 développé par Laurent Cosserat pour mettre en pratique ses premières idées sur les problèmes de causalité, le premier compilateur Esterel v2 a été développé en 1985 par Philippe Couronné et moi-même, en utilisant le système Le_Lisp de Jérôme Chailloux et sa couche Ceyx développée par Jean-Marie Hullot, qui allait ensuite se rendre célèbre par les développements chez Apple de la machine NeXT, du système macosx et de son interface, et de l'iPhone (voir son séminaire dans le cours de Didier Roux pour la chaire d'innovation technologique Liliane Bettencourt, https://www.college-de-france.fr/site/didier-roux/seminar-2017-03-31-11h00.htm). À partir du terme Esterel initial, le principe est de calculer efficacement pour chaque entrée possible le terme résultant capable de réaliser les réactions suivantes, en utilisant directement la sémantique formelle du langage, ici avec une causalité stratifiée simple. Une fois ce processus convergent itéré jusqu'à plus soif, on obtient un automate dont les états sont indexés par ces textes de programme. Il suffit alors de jeter ces textes pour obtenir un automate standard en gardant simplement des numéros d'états. Celle méthode est élégante et facile à programmer, même avec les ressources limitées de l'époque. Mais elle est lourde et chère à l'exécution. Esterel v2 a néanmoins été utilisé pour de premières évaluations industrielles.

Un progrès essentiel a été dû à une rencontre avec Ravi Sethi, chercheur aux *Bell Laboratories*, qui connaissait parfaitement les langages de programmation et la théorie des automates. En combinant nos connaissances, nous avons pu définir une nouvelle façon beaucoup plus efficace de traduire les expressions régulières en automates non-déterministes que l'on peut ensuite déterminiser par la méthode usuelle [BeSe89]. L'innovation majeure pour Esterel fut apportée peu après par Georges Gonthier, que j'ai eu l'honneur d'avoir comme thésard et qui s'est ensuite rendu célèbre par ses preuves en Coq du théorème des 4 couleurs puis du grand théorème de Feit-Thompson sur la classification des groupes d'ordre impair. Gonthier a construit une nouvelle sémantique par analyse statique fine de causalité, bien plus fine que la stratifiée, et des algorithmes de compilation d'un ordre de grandeur plus efficace en généralisant l'algorithme de Berry-Sethi [BeGo92]. Il a enfin défini avec moi l'architecture d'un nouveau compilateur Esterel v3 que nous avons écrit en équipe. Il a ensuite été embauché aux Bell Labs dans l'équipe de Ravi Sethi, où il a construit des optimiseurs puissants pour Esterel qui ont été utilisés pour des protocoles de centraux téléphoniques.

Les premières industrialisations

L'industrialisation d'Esterel v3 commença dès la fin des années 1980, d'abord avec Cisi Ingéniérie et ILOG, qui vendirent le compilateur universitaire accompagné d'un environnement de développement sophistiqué et d'un langage graphique inspiré des SyncCharts de Charles André [An96]. Le tout fut utilisé intensivement par Dassault Aviation pour diverses parties critiques de l'avion Rafale, et plus modestement par d'autres industriels (Thomson, Bertin, etc.). Mais le compilateur Esterel v3 souffrait d'un mal qui devenait vite rédhibitoire quand les applications grandissaient : si les automates étaient engendrés rapidement et eux-mêmes très rapides, ils pouvaient devenir très gros dès que le nombre d'entrées et la taille du code source augmentaient notablement. En fait, dans le pire cas, la taille de l'automate était exponentielle dans la taille du source. Même si le synchronisme réduisait cette taille par rapport à celle obtenue par les approches asynchrones, le problème se trouvait vite sérieux, voire bloquant pour les grands utilisateurs. Il fallait changer complètement de méthode ; nous allons voir comment dans la section suivante

En parallèle, dans les années 1990, Lustre fut industrialisé par la société Verilog sous la forme de l'environnement graphique SCADE précité, doté d'un compilateur vers C certifié au plus haut niveau requis pour l'avionique. Cela permettait de considérablement simplifier la certification des codes engendrés et de leurs évolutions. SCADE fut utilisé par Merlin-Gérin pour des applications nucléaires, par Airbus pour les commandes de vol (en particulier pour l'une des deux chaînes de pilotage de l'A380), ainsi que pour des applications ferroviaires.

Signal fut lui industrialisé par la société TNI sous la forme de l'atelier Sildex, avec toutefois un succès plus modeste.

Le passage aux circuits électroniques

C'est encore une fois une rencontre avec d'autres scientifiques qui nous permit de tordre le coup à l'explosion exponentielle d'Esterel v3. Au début des années 1990, en collaboration avec l'équipe de mon ami Jean Vuillemin au *Digital Equipment Paris Research Lab*, j'ai développé de nouvelles méthodes de compilation d'Esterel produisant directement des circuits électroniques de taille quasi-linéaire dans la taille du code Esterel et, de plus, très efficaces après optimisation adéquate. Les résultats s'avéraient en fait souvent meilleurs que ceux que savaient construire les ingénieurs avec les méthodes classiques. Un code logiciel efficace était aussi engendré sous la forme d'un simulateur C du circuit engendré, lui aussi de taille linéaire dans la taille du source mais moins rapide que l'automate v3 qui fut donc conservé comme alternative pour pouvoir varier les codes générés.

La traduction en circuits sera présentée dans le troisième cours du 8 mars 2018. Ce travail s'appuyait directement sur la sémantique constructive précitée en faisant aussi grand usage des avancées majeurs en calcul booléen que j'ai présentées dans le cours 2015-2016 pour l'optimisation et la vérification formelle des programmes Esterel. Le nouveau compilateur Esterel v4, à causalité stratifiée, a été rapidement industrialisé par ILOG et utilisé par ses clients, la version universitaire l'étant aussi par les sociétés américaines leaders de la CAO électronique Cadence Design Systems et Synopys, cette dernière ayant d'ailleurs acheté une licence source et intégré Esterel dans un produit de design système. J'ai appris beaucoup en travaillant avec ces sociétés autant qu'avec l'Université de Berkeley, toute proche.

Enfin, en intégrant tous les résultats obtenus pour la sémantique constructive, nous avons pu résoudre complètement le dernier problème embêtant pour certains de nos utilisateurs, celui des faux cycles de causalité qui peuvent apparaître quand on compose des modules pour construire de grandes applications. Le compilateur Esterel v5 ainsi obtenu par évolution d'Esterel v4 a été utilisé

à grande échelle, en particulier par Dassault Aviation, comme vous pourrez le revoir dans la vidéo du séminaire d'Emmanuel Ledinot donné après mon cours du 16 avril 2013.

Le lecteur peut s'interroger sur le nombre de sémantiques successivement mises en jeu. Changeaient-elles le sens et donc le comportement des programmes ? Pas le moins du monde : chacune acceptait simplement plus de programmes que la précédente, et les programmes acceptés par l'une marchaient toujours pareil dans celle qui l'a suivi. L'approche mathématique est critique pour obtenir ce genre de résultat, essentiel pour les utilisateurs mais qui n'est pas garanti pour tous les langages, loin de là. Et cela prouve aussi que le langage était bien né dès sa première version, avant même qu'il n'ait reçu une sémantique formelle.

Esterel v7

Intel, Texas Instruments et ST Microelectronics devenant intéressés par l'utilisation d'Esterel et de son outillage de compilation et de vérification formelle pour leurs circuits, j'ai défini en 2000 un nouveau langage Esterel v7 bien plus riche que la version précédente, intégrant en particulier les tableaux de données et de processus et les manipulations de données de Lustre. Cette extension a été effectué en collaboration étroite avec Michael Kishinevsky à Portland (Oregon), au sein du Strategic CAD Lab d'Intel. Même si les mathématiques sous-jacentes et les principaux algorithmes restaient très similaires, le nouveau langage intégrait désormais des primitives de traitement de données puissantes, inexistantes en Esterel v5. Il s'est s'avéré bien mieux adapté aux très gros programmes que nous envisagions désormais, à la synthèse de circuits mêlant intimement contrôle et données, et à la génération de codes logiciels efficaces soit pour la simulation des circuits ; soit pour la production de code logiciel embarquable. Esterel v7 a été décrit dans le cours 2014-2015, je n'en parlerai que peu cette année car sa sémantique de base est la même que celle d'Esterel v5.

La création d'Esterel Technologies : Esterel v7 et Scade 6

Au début du siècle, il devenait indispensable de changer de vitesse quant à l'industrialisation. La société Esterel Technologies fut créée en 2001 par Eric Bantegnie, auparavant PDG de la société Simulog qui avait repris l'environnement industriel d'Ilog. J'en fus le Directeur scientifique de 2001 à 2009 et dirigeai toute la construction du compilateur Esterel v7 et de son environnement de vérification formelle – tout en programmant moi-même des parties clefs du compilateur, un plaisir jamais démenti. L'environnement obtenu Esterel Studio fut utilisé avec succès pour des circuits commerciaux par Texas Instruments et ST Microelectronics, et pour des projets de recherches par Intel, NXP (ex Philips Semiconductors) et France Télécom. Son développement et sa commercialisation durent malheureusement s'arrêter en 2009 à la suite de la grande crise de 2008 qui toucha toute l'industrie des circuits et fit disparaître nombre de nos clients pourtant ravis.

Mais Esterel Technologies avait aussi racheté l'activité industrielle de Lustre en 2002. Commença alors le développement du nouveau langage graphique / textuel SCADE 6 et de son atelier SCADE Studio, unifiant également Lustre et Esterel sous une forme un peu plus contrainte à cause des besoins de la certification. SCADE 6 est maintenant leader mondial des applications informatiques certifiées en avionique, ferroviaire, et d'autres secteurs. Il est toujours développé et commercialisé par Esterel Technologies qui a été rachetée par la société américaine Ansys mais reste techniquement indépendante. Voir le séminaire de Bruno Pagano après mon cours du 23 avril 2013 pour en savoir plus sur SCADE 6.

Vérification formelle et optimisation des programmes Esterel

Dès le début de ses développements, nous avons pris soin d'intégrer la vérification formelle de programmes à l'environnement de développement d'Esterel. Les premières techniques utilisées

étaient fondées sur les calculs d'automates explicites en utilisant la réduction par bisimulation introduite par Milner pour ses calculs de processus. Le système Auto/Autograph [RoSi92] qui réduisait les automates puis permettait de dessiner les résultats a joué un rôle important chez nos utilisateurs. Plus tard, quand sont apparues les techniques BDD (Binary Decision Diagrams) systématisées par Randy Bryant et Kern McMillan aux USA, des chercheurs de Digital Equipment ont développé un système de BDDs nommé TiGeR très bien adapté au besoin d'Esterel, que ce soit pour la simulation ou la vérification des circuits ou codes logiciels issus d'Esterel. J'ai présenté ces principaux algorithmes d'optimisation et de vérification dans mon cours du 9 mars 2016, ils sont détaillés en [BeTo93]. J'ai aussi expliqué dans mes trois premiers cours 2014-2015 à quel point la vérification formelle a été utile pour nos clients. C'est la même chose pour l'optimisation des circuits qui a été critique pour l'intérêt porté par Intel à Esterel et la construction d'Esterel v7 avec Michael Kishinevsky

L'intégration d'Esterel dans des environnements variés

L'intégration des circuits générés est simple dans le domaine des circuits synchrones classiques car l'horloge de base d'Esterel s'identifie à celle du circuit. Elle est néanmoins beaucoup plus subtile dans les environnements plus riches comme les *Systems on Chips* et les logiciels embarqués. Pour les *Systems on Chips*, il faut tenir compte de la présence d'horloges multiples, ce qui a conduit à l'introduction du multi-horloges dans Esterel v7 avec Intel [ArBe06]. Pour les environnements logiciels, il faut déterminer ce qui dans l'environnement doit faire réagir le programme Esterel et comment ses résultats doivent affecter cet environnement. Les modes de fonctionnement sont très variables suivant le contexte, qui peut aller des systèmes embarqués déterministes simples au Web avec tout son asynchronisme. Un cours analysera ses aspects. Il présentera aussi comment Esterel peut contrôler finement des actions asynchrones externes, ce qui est indispensable en robotique et pour le lancement de calculs et d'actions à distance pour des applications Web, réalisant ainsi une pleine compatibilité entre les mondes synchrones et asynchrones.

Bibliographie sommaire sur Esterel (les étoiles indiquent par où commencer)

[BeMo83] *ESTEREL: Towards a Synchronous and Semantically Sound High-Level Language for Real-Time Applications.*
G. Berry, S. Moisan, et J-P. Rigault. Proc. *IEEE Real-Time Systems Symposium*, Arlington, Virginia, IEEE Catalog 83CH1941-4, 30-40 (1983).

[BeCo84] *The Synchronous Programming Language ESTEREL and its Mathematical Semantics*
G. Berry et L. Cosserat. In *Seminar on Concurrency*, Springer-Verlag LNCS 197, 389-448 (1984).

[BeSe86] *From Regular Expressions to Deterministic Automata.*
G. Berry et R. Sethi. *Theoretical Computer Science* 48, 117-126 (1986).

[Be89] *Real Time Programming: Special Purpose or General Purpose Languages.*
G. Berry. *Proc. Information Processing* 89, G.X. Ritter (Ed.), Elsevier Science Publishers B.V., North-Holland, 11-18 (1989).

[Be92] *A Hardware Implementation of Pure Esterel.*
G. Berry. *Sadhana*, Academy Proceedings in Engineering Sciences, Indian Academy of Sciences, vol. 17, part 1, 95-130 (1992).

[RoSi92] Auto / Autograph.
V. Roy et R. de Simone. *Formal Methods in System Design*, Volume 1, Issue 2-3, 239-249 (1992).

[BeGo93] *The Synchronous Programming Language ESTEREL: Design, Semantics, Implementation.*
G. Berry et G. Gonthier. *Science of Computer Programming*, vol. 19, no. 2, 87-152 (1993).

[BeTo93] *Optimized Controller Synthesis Using Esterel.*
G. Berry et H. Touati. Proc. *Intl. Workshop on Logic Synthesis*, Lake Tahoe, USA (1993).

* [Be00a] *The Foundations of Esterel.*
G. Berry. In *Proof, Language and Interaction: Essays in Honour of Robin Milner*, G. Plotkin, C. Stirling and M. Tofte, editors, MIT Press, Foundations of Computing Series, 425-454 (2000).

* [Be00b] *The Esterel Language Primer, version v5_91*
G. Berry. Current version Esterel v5_91 (2000).

* [Be02] *The Constructive Semantics of Pure Esterel* .
G. Berry. Draft book, current version 3.0, Dec. 16th (2002).

[ArBe06] *Clocking Schemes in Esterel.*
L. Arditi, G. Berry, M. Kishinevsky et M. Perreaut. Proc. *Designing Correct Circuits* DCC'06, Vienna, Austria (2006).

[PoEd07] *Compiling Esterel*
D. Potop, S. Edwards et G. Berry. Springer (2007).

[MeSh12] *Constructive Boolean Circuits and the Exactness of Timed Ternary Simulation*
M. Mendler, T. Shiple and G. Berry. *Formal Methods in System Design*, Vol. 40, No. 3, p. 283-329, Springer (2012). Disponible en ligne à l'adresse DOI 10.1007/s10703-012-0144-6.

Autres langages et formalismes synchrones ou reliés

[Ha87] *Statecharts: a Visual Approach to Complex Systems.*
D. Harel, *Science of Computer Programming*, vol. 8, p. 231-274 (1987).

* [HaCa91] *The Synchronous Dataflow Programming Language Lustre.*
N. Halbwachs, P. Caspi et D. Pilaud. *Another Look at Real Time Programming*, *Proceedings of the IEEE*, *Special Issue* (Sept. 1991).

[GuGa91] *Programming real-time applications with SIGNAL.*
P. Le Guernic, T. Gauthier, M. Le Borgne et C. Le Maire.
Another Look at Real Time Programming, *Proceedings of the IEEE*, *Special Issue* (Sept. 1991).

* BeBe91] *The Synchronous Approach to Reactive and Real-Time Systems*.
A. Benveniste et G. Berry. *Another Look at Real Time Programming, Proceedings of the IEEE*, vol. 79-99, 1270-1282 (1991).
[Bou91] *Reactive C: an Extension of C to Program Reactive Systems.*
F. Boussinot. *Software – Practice and Experience*, vol. 21(4), 401-428 (1991).

[An96] *Representation and Analysis of Reactive Behaviors: A Synchronous Approach*
C. André. *Proc. CESA'96*, IEEE-SMC, Lille(F) (1996).

[MaRe01] Argos, and Automaton-Based Synchronous Language.
F. Maraninchi et Y. Rémond. *Computer Languages* – Oxford -. Elsevier vol 27(1-3), 61-92 (2001).

[MaPo05] *ReactiveML, a Reactive Extension to ML.*
L. Mandel et M. Pouzet, *Proc. Conf. PPDP, Principles and Practice of Declarative Programming* (2005).

[BeSe14] *Hop and HipHop : Multitier Web Orchestration*
G. Berry and M. Serrano. *Proc ICDCIT'2014*, Bubhaneswar, Springer-Verlag Lecture Notes 8337 (2014).

[SmMo17] *SCEst: Sequentially Constructive Esterel*
S. Smyth, C. Motika, K. Rathlev, R. von Hanxleden et Mendler. *ACM Transactions on Embedded Computing Systems* (2017).