Projet de recherche

Programmation spatiale d'un médium biologique

Antoine Spicher

Mon projet de recherche s'inscrit dans le domaine des langages de programmation non conventionnels. Il est motivé par la modélisation et la simulation de processus biologiques complexes, l'utilisation de nouveaux supports de calculs, en particulier biologiques, et par de nouvelles applications où il faut obtenir un comportement global dans une grande population d'entités irrégulièrement et dynamiquement interconnectées. L'objectif à long terme est de développer un formalisme permettant de spécifier et d'analyser un comportement spatial global et de le compiler automatiquement dans le comportement local des entités constituant le système afin d'en exploiter les propriétés émergentes. La cible privilégiée de la compilation est une population de bactéries programmées par modifications génétiques.

Le plan de cette partie est le suivant : nous présentons dans les deux premières sections les thématiques développées par la programmation spatiale, le calcul amorphe et la biologie synthétique, fondements de mon projet. La section 3 développe concrètement la proposition d'un compilateur de bactéries synthétiques.

1 Programmation spatiale et calcul amorphe

L'évolution des technologies, l'apparition de nouveaux besoins, d'idées originales de services nous emmènent inéluctablement vers une vision des systèmes d'informations très éloignée de la station de travail traditionnelle. Les unités de calcul monolithiques laissent peu à peu la place à des supports constitués d'un grand nombre d'entités qui interagissent et coopèrent. La programmation de tels supports consistent à maîtriser leur comportement global à partir de la seule spécification de ces interactions locales et décentralisées.

Espace et programmation. L'une de mes motivations majeures en informatique est la prise en compte de l'espace dans les modèles de calcul, et plus particulièrement dans les langages de programmation. Ces travaux se placent dans le cadre de la *programmation spatiale* [DGG06], un domaine dont l'objectif est le développement de modèles de calcul, de langages ou encore d'architectures tenant compte de l'espace afin de dépasser le modèle de von Neumann. Ce dernier atteint des limites dues à l'accès séquentiel à la mémoire ; la prise en compte de l'espace permet alors d'échapper à cette séquentialité en considérant que plusieurs évènements peuvent se produire « en même temps » mais à des « endroits différents ».

Un ordinateur spatial [DGG06] est un modèle de calcul où la notion de position joue un rôle important pour les performances et la fonctionnalité, et est intégrée dans un espace dont la géométrie est définie par un coût de communication. De tels calculateurs sont des collections spatialement distribuées d'éléments dont la puissance de calcul n'est pas obligatoirement équivalente, et où le coût (temps/énergie) de communication entre deux éléments distants est au moins proportionnel à la longueur du plus court chemin qui les sépare. Les automates cellulaires [vN66] sont un exemple de calculateur spatial avec une architecture régulière, statique (voisinages de Moore, de von Neumann, etc.) et une communication synchrone.

L'espace peut, dans les calculs, jouer deux rôles différents :

1. soit parce qu'il est une *ressource* de calcul,
2. soit qu'il est le *résultat* d'un calcul.

L'espace est une ressource de calcul lorsqu'il permet de prendre en compte des systèmes distribués aussi bien au niveau de l'architecture (avec par exemple les FPGAs) qu'au niveau des paradigmes de calcul comme les langages à parallélisme de données. Dans ces derniers, les données sont spatialement distribuées entre des unités de calcul. L'espace vu précédemment comme une ressource, peut aussi être

considéré comme le résultat d'un calcul avec par exemple les systèmes de Lindenmayer [RS92] utilisés dans la modélisation de croissance des plantes ou les langages de modélisation géométrique utilisés pour spécifier des comportements physiques [PS93].

Ces deux points de vue sont unifiés et largement abordés dans [GMCS05] où les structures de données des langages de programmation sont considérées comme des espaces topologiques, la relation de voisinage étant définie par les accès logiques faits durant le calcul. Ce contexte soulève alors la question de la programmation (construction et manipulation) intentionnelle d'entités spatialement structurées :

- **des entités spatialement organisées** sont des agrégats d'éléments structurés par des relations de voisinage, d'obstructions, de bords, etc. Dans ces agrégats, seuls des éléments voisins peuvent interagir.
- **intentionnel** signifie que ces agrégats sont manipulés comme un tout sans référence à leurs éléments constituants. La manipulation intentionnelle des structures de données a déjà été considérée comme un élément crucial des langages de programmation de haut niveau [WA85, AFJW95].

Supports de calcul amorphes. Une voie prometteuse pour le développement d'un paradigme de programmation spatiale, est l'étude des *systèmes amorphes*. Il s'agit de systèmes composés d'un grand nombre (d'un ordre de grandeur supérieur à 10^7) d'unités de calcul autonomes, possiblement défaillantes et interconnectées de façon irrégulière et dynamique. Chaque unité de calcul est capable d'acquérir des informations sur son voisinage et d'interagir uniquement avec des entités spatialement voisines. De tels systèmes naturels ou artificiels constitués d'une myriade d'unités élémentaires et dont la structure est dynamique, sont à l'heure actuelle disponibles et peuvent servir de support de calcul. On y trouve entre autres les réseaux pair-à-pair, les réseaux de senseurs (ceux du projet *smart dust* par exemple [KKP99]), les *smart matters* ou MEMS [HH98], les cellules d'un tissu biologique, les populations de bactéries (biofilm, *quorum sensing*), les nanostructures auto-assemblées [RPW04], etc.

Un support de calcul amorphe [AAC+00] est donc un système hautement redondant, spatialement organisé et asynchrone ; par l'absence d'hypothèse sur la précision des connexions ou sur l'organisation des parties, ces systèmes constituent l'environnement idéal pour explorer les problématiques de la programmation spatiale : une conception robuste dans un contexte présentant une topologie dynamique, une organisation locale variable et arbitraire, et un couplage local avec le monde physique environnant.

Deux points importants soulevés par ces systèmes sont ceux de la *décentralisation* et de l'*autonomie*. En effet, il n'est pas raisonnable de concevoir un programme nécessitant une intervention extérieure pour se (re)configurer. De la même manière, ces configurations du calcul ne peuvent se faire de manière centralisée sans mettre en péril le passage à l'échelle de l'application. Ces points impliquent la mise en place de techniques d'auto-diagnostic, d'auto-réparation et d'auto-organisation (l'exécution doit être autonome).

Calcul non-conventionnel et systèmes complexes. Une façon originale d'appréhender cette problématique est d'étudier ces nouveaux supports et leur programmation comme la réalisation de systèmes complexes. Ils en possèdent en effet les caractéristiques principales :

- *Localité* : comme nous l'avons déjà précisé, les entités ne peuvent interagir qu'avec leurs voisines. Établir une propriété globale sur un tel système revient à spécifier le comportement de chaque individu de telle sorte que la propriété *émerge* des interactions.
- *Aucune centralisation* : le très grand nombre d'unités composant les supports empêche l'intervention d'un opérateur extérieur qui nécessiterait une trop grande quantité de ressources afin de gérer une représentation complète du système (imaginez par exemple un serveur devant connaître en temps réel la constitution du réseau dans le cas d'une application pair-à-pair). Des niveaux d'organisations peuvent apparaître et se stabiliser mais uniquement de façon émergente.
- *Autonomie* : le comportement agrégé des entités élémentaires composant le système doit induire des propriétés permettant l'*auto-configuration*, l'*auto-réparation*, l'*auto-optimisation* et l'*auto-protection*. L'*autonomic computing initiative* [Hor01] propose de reposer sur ces propriétés pour concevoir des ordinateurs et des applications qui s'auto-régulent en s'adaptant eux-mêmes à leur environnement immédiat de façon totalement non-supervisée. Néanmoins, il est difficile de concevoir des systèmes respectant ces propriétés auto-*.

2 Un support de calcul biologique

À l'origine, les développements dans le domaine du calcul amorphe sont d'inspiration biologique : le fonctionnement d'un organisme pluricellulaire (ou plus généralement d'une population d'individus en interaction réalisant une tâche commune) est l'un des meilleurs exemples de système amorphe, à grande échelle et présentant une robustesse et une autonomie surprenante. Le langage ECOLI [Coo99, Nag01] illustre parfaitement cette corrélation. Il s'agit d'un langage orienté évènements permettant la programmation de chacune des entités de calcul composant le milieu amorphe. Ce type de langage correspond exactement au comportement de *sensor/actuator* des cellules biologiques. On note d'ailleurs que les premiers travaux sur la programmation des systèmes amorphes étaient très proches de cette analogie et consistaient en des diffusions/réactions de substances chimiques abstraites.

Un nouveau domaine d'activité, la *biologie synthétique* [WBH+03], permet de s'intéresser de près à la programmation de cet exemple naturel de système amorphe.

La biologie synthétique. La biologie synthétique combine biologie et ingénierie génétique pour concevoir et construire de manière *systématique* de nouveaux systèmes biologiques (ou biochimiques) qui exhibent des fonctionnalités nouvelles. Ce domaine de recherche, impulsé par des informaticiens, est émergent aux États-Unis (Princeton, Harvard, MIT, ...) et apparaît en Europe (ETH Zurich, Université de Rome, projet européen PACE, IISB, ...). Un certain nombre d'applications ont été développées lors de la compétition internationale iGEM[1] organisées aux États-Unis entre de grandes universités : film bactérien photosensible [KKA+04] ou biosenseur permettant de détecter des explosifs [Gib04]. Si pour le biologiste, il y a là une voie d'accès à l'étude de l'origine de la vie, la biologie synthétique fournit aux informaticiens un nouveau médium de calcul. À terme les techniques développées permettront de créer des systèmes biologiques dédiés, de grande échelle et composés d'un grand nombre d'entités. La programmation d'une telle architecture, aussi peu conventionnelle soit elle, relève bien des problématiques de la programmation spatiale.

Une des caractéristiques de ce domaine est l'importance accordée à l'aspect « ingénierie *réutilisable* », c'est-à-dire à la spécification et au développement d'un ensemble de *composants standards*[2] comme par exemple les *BioBricks* [WBH+03] aux caractéristiques et performances bien définies, utilisables dans tous les contextes, pour concevoir des systèmes biologiques. La constitution de cette bibliothèque pose les questions de la compositionnalité (la plus orthogonale possible) des entités biologiques, du développement d'outils d'aide à la conception et de simulation de ces systèmes, et de l'ingénierie inverse des « modules biologiques existants » pour les adapter à nos besoins. Bien qu'il soit émergent, ce domaine est plein essor et l'on prévoit qu'il va suivre une croissance équivalente à la loi de Moore en informatique [Car03].

Des problématiques fondamentales. Outre le fait d'étudier d'un point de vue ingénieur l'élaboration de systèmes biologiques, la biologie synthétique adresse des problématiques habituellement absentes lors de la programmation de systèmes électroniques plus traditionnels. On retrouve alors dans un même contexte des notions habituellement opposées :

- *Global - local* : bien que le comportement *global* d'un organisme multicellulaire résulte des interactions *locales* entre les cellules qui le composent, la distinction entre les deux points de vue, global et local, n'est pas aisée. Ils s'inscrivent dans un cycle où *les cellules s'auto-organisent spontanément en une structure* qui *détermine un espace impliquant de nouvelles relations entre les cellules* qui *s'auto-organisent*... Le niveau d'organisation globale est défini à travers les éléments qui le composent mais ceux-ci ne sont souvent viables que dans cette structure globale. La rétro-action du niveau global sur le comportement local des individus est également appelée *immergence* et se rapproche de la notion de *système dynamique à structure dynamique* développée dans le projet MGS.
- *Discret - continu* : les systèmes biologiques mêlent dans un cadre unique et souvent à la même échelle, des entités intrinsèquement *continues* comme le temps ou assez nombreuses pour qu'elles

[1] Concours iGEM (*intercollegiate Genetically Engineering Machine*). Voir `http://parts2.mit.edu/iGEM`

[2] voir par exemple le « Registry of Standard Biological Parts » à l'adresse `http://parts.mit.edu/`

puissent être capturées par une abstraction continue comme les concentrations, et d'autres très faiblement représentées comme la molécule d'ADN ou encore des mécanismes de compartimentation qu'il est naturel de décrire discrètement. L'enjeu n'est pas ici de confronter ces deux points de vue mais de développer des outils de modélisation pour chaque paradigme et de réussir à coupler leur utilisation.

– *Logiciel - matériel* : encore une fois, la distinction traditionnelle des systèmes informatiques en une architecture permettant l'exécution d'applications disparaît. On dit souvent que le code génétique contenu dans l'ADN est un programme exécutée par la machinerie cellulaire. Cependant, il est tout aussi envisageable de comprendre l'ADN comme une architecture interprétant les états physico-chimiques de la cellule. Cette problématique est à rapprocher des études développées dans le domaine des architectures reconfigurables (comme les FPGA) ou des langages de programmation réflexifs permettant la modification de structures internes de haut niveau *à l'exécution*.

Ces problématiques montrent la richesse que promet l'étude de la programmation spatiale d'un système amorphe biologique. Cependant elles exhibent également la difficulté d'aborder des systèmes de calcul de nature aussi complexe.

3 Un langage pour programmer des bactéries

Afin d'aborder concrètement la programmation spatiale d'un médium amorphe biologique et en s'appuyant sur les résultats de la programmation spatiale, de la biologie synthétique et du calcul amorphe, nous proposons de développer à court et moyen terme une tour de langages de programmation et les outils d'analyse et de compilation associés. Cet outil permettrait de traduire automatiquement les descriptions du langage de plus haut niveau spécifiant le comportement global d'une population de bactéries dans le comportement local des bactéries instancié et réalisé par modifications génétiques.

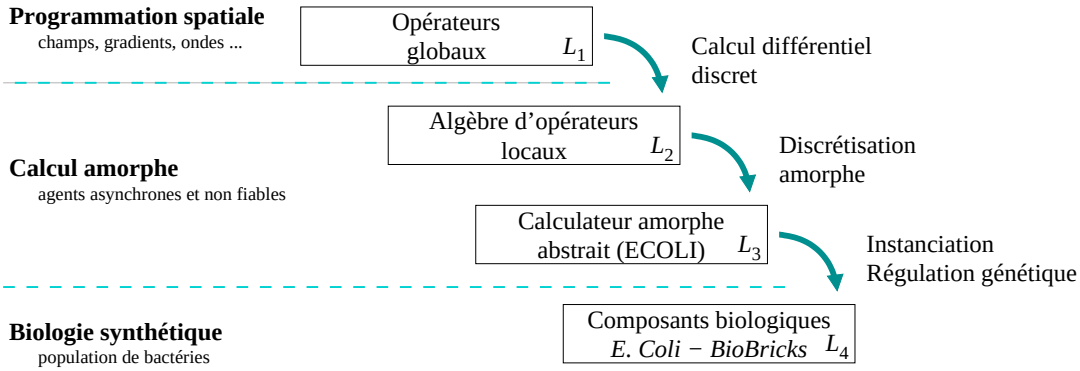

FIG. 1 – Tour de langages

L_1 **:** ce niveau d'expression permet la spécification de comportements spatiaux globaux à l'aide de primitives de base qu'il est possible de composer. Le jeu de primitives est inspiré des patrons de conception élaborés pour la morphogenèse (mise en place de gradients, propagation d'ondes, etc. [Mei82, PLH+90, Law92, Wol02]) ou encore des langages développés dans le cadre des systèmes multi-agents réactifs.

L_2 **:** ce langage, extension de mes travaux de thèse, est fondé sur l'utilisation des opérateurs du calcul différentiel pour la spécification intentionnelle d'entités spatialement structurées. Cette approche, abstraite, permet la compréhension et le contrôle de l'espace et du temps, et la manipulation globale de nouvelles structures complexes (agrégats d'éléments corrélés spatialement et/ou temporellement).

L_3 **:** ce niveau d'expression intermédiaire consiste en des processus organisés spatialement et échangeant des messages uniquement avec leurs voisins. Ce niveau de description est celui abordé dans les problématiques du calcul amorphe. On pense en particulier au langage de description ECOLI qui permet la conception d'un automate d'états spécifiant le fonctionnement local de chaque entité.

L_4 **:** ce dernier niveau correspond au support final du calcul. Dans un premier temps, nous envisageons d'utiliser, de programmer et de composer les *BioBricks* du concours iGEM permettant l'intégration de nouveaux comportements chez *E. Coli*.

La première étape de traduction ($L_1 \rightarrow L_2$) permet de passer à une spécification locale des comportements globaux ; l'objectif est la mise en place d'un *calcul différentiel discret* qui permet de transformer un primitive globale, telle qu'une onde, en une équation différentielle *discrète* exprimant les mouvements d'informations dans l'espace. La deuxième étape ($L_2 \rightarrow L_3$) permet de transformer ces mouvements de données décrits en L_2 vers un comportement individuel de chaque unité de calcul, perçue alors comme un agent réactif répondant à son environnement et communiquant avec ces voisins. Enfin, la dernière traduction ($L_3 \rightarrow L_4$) consiste à implanter par modifications génétiques le comportement abstrait des agents de la couche L_3 dans des bactéries *Escherichia coli*.

La faisabilité de ces trois phases de compilation dépend largement des primitives disponibles à chaque niveau de description. En particulier, ces travaux nécessitent de répondre aux questions concernant :

- les relations entre les langages et simulateurs amorphes, et l'approche topologique des formes différentielles ;
- la conception d'un langage de programmation fondé sur les notions de champs de données, d'opérateurs du calcul différentiel et d'homomorphismes de structures de données ;
- les relations entre calcul à parallélisme de donnée et calcul amorphe ;
- le développement d'applications appropriées ;
- la certification de *BioBricks* génériques et effectives autorisant la composition fonctionnelle.

Ce dernier point est en particulier crucial pour valider ma proposition.

La forme des langages L_3 et L_4 est moins problématique que celle de L_1 et L_2 ; L_3 correspond à des langages tels que GPL [Coo99], ECOLI, OSL [Nag01] largement étudiés dans le cadre du calcul amorphe, et L_3 aux *BioBricks* des concours iGEM. En revanche, les langages L_1 et L_2 sont largement à inventer. De même, les trois phases de traduction sont nouvelles (bien que connaissant L_3 et L_4, cela ne permet pas de préjuger le travail de compilation). Les deux paragraphes suivants mettent en avant les niveaux d'expression L_1 et L_2 ainsi que la traduction $L_1 \rightarrow L_2$, les autres n'étant pas assez matures pour être présentés (en particulier les travaux effectués en collaboration avec D. Coore lors de sa visite au laboratoire IBISC en juillet 2007) ou ayant déjà été décrits par ailleurs (comme par exemple les *BioBricks* présentées dans le cadre du projet SMB dans la partie « travaux effectués » du dossier).

Morphogenèse. La conception du langage L_1 et l'analyse de ses programmes est un des enjeux majeurs de ce projet. Étant donné la nature spatiale de la programmation au niveau L_1, il est possible de l'envisager comme la mise en place de *formes* particulières dans un espace déterminé. Nous avons déjà abordé cette notion en précisant qu'un calcul pouvait se présenter comme la création d'un nouvel espace.

D'un point de vue plus pratique il me semble intéressant de s'inspirer des études faites dans le domaine de la morphogenèse pour développer cette proposition par l'utilisation des notions de gradients morphogénétiques, de processus de diffusion et de propagation d'ondes, de brisures de symétrie utilisées dans les modélisations des biologistes du développement. Ces notions permettent en effet de décrire de manière abstraite et globale le comportement spatialement et temporellement corrélé d'une population d'entités à travers la notion de *patron de conception*. Ce sont ces patrons qui sont à l'origine des primitives globales offertes au niveau L_1. On peut citer les travaux de L. Wolpert [Wol02], P. Lawrence [Law92], H. Meihnardt [Mei82], ou encore P. Prusinkiewicz [PLH$^+$90].

Le développement de cette thématique est possible à travers mes collaborations avec Ch. Godin du CIRAD et P. Prunsinkiewicz de l'université de Calgary pour leurs travaux sur la morphogenèse des plantes, et auprès de P. Bourgine au CREA développant une thématique forte sur l'embryogenèse animale.

Structures topologiques et déplacement de données. Les primitives présentes au niveau L_1 sont très expressives pour la manipulation globale. Elles peuvent de plus être exprimées à travers des systèmes d'équations différentielles. Elles s'intègrent donc dans une formulation mathématique utilisant les opérateurs du calcul différentiel. Les perspectives de ma thèse mettent en avant le fait que

ces opérateurs correspondent à des déplacements atomiques de données sur des espaces topologiques discrets. Nous proposons d'élaborer le niveau d'expression L_2 sur ces déplacements et de développer la traduction $L_1 \rightarrow L_2$ comme un calcul différentiel permettant d'établir les équations différentielles décrivant les primitives du niveau L_1.

L'introduction de concepts topologiques dans les langages de programmation amène naturellement à considérer les structures de données comme des espaces topologiques : c'est la notion de *collection topologique*. Une collection topologique est un champ de données sur un espace topologique et peut être formalisée à travers des concepts de topologie algébrique. L'idée fondamentale de cette branche des mathématiques est l'étude des espaces topologiques en leur associant des objets algébriques (groupes, espaces vectoriels, etc.) dont certaines propriétés (appelées *invariants algébriques*) caractérisent la nature de l'espace représenté [RS97]. Une formalisation possible des collections topologiques est fondée sur la notion de *chaînes topologiques* [Mun84]. Il paraît alors naturel d'utiliser les *cochaînes topologiques* (des homomorphismes de chaînes [Mun84, DKT06]) pour manipuler les collections. En approfondissant cette idée, les cochaînes étant considérées comme le pendant discret des formes différentielles [Hir03], l'utilisation du calcul différentiel discret peut être envisagée pour calculer à l'aide des collections topologiques.

L'apport mathématique important (différentiation, théorème de Stockes, ...) de cette idée dans les langages de programmation est à l'origine de résultats profonds et non triviaux. Par exemple, le calcul différentiel ouvre la voie à une notion d'homomorphisme avec l'étude d'une algèbre fondée sur ses opérateurs (bord, différentielle, étoile de Hodge, ...), qui pourrait se faire dans la lignée des algèbres que R.S. Bird et B.M. Merteens ont développées pour les listes [Bir87]. Cela permettrait à terme de programmer les algorithmes du calcul spatial de façon purement intentionnelle (à l'aide de réductions et de *maps*) faisant alors totalement abstraction du parcours des structures de données sous-jacentes aux modèles utilisés. De plus, le style d'écriture de ces algorithmes, proche de la physique discrète [Ton74, PS93] et du calcul différentiel discret [Hir03], simplifierait l'expression des programmes en l'établissement d'équations « différentielles », exprimant des propriétés d'invariances, de symétries, de conservations, d'équilibres, ...

La présentation de ces travaux est actuellement en cours de soumission dans la revue *Physica D*.

Références

[AAC+00] Harold Abelson, Don Allen, Daniel Coore, Chris Hanson, George Homsy, Thomas F. Knight, Radhika Nagpal, Erik Rauch, Gerald Jay Sussman, and Ron Weiss. Amorphous computing. *CACM: Communications of the ACM*, 43, 2000.

[AFJW95] Edward A. Ashcroft, A. Faustini, R. Jagannathan, and W. Wadge. *Multidimensional Programming*. Oxford University Press, February 1995. ISBN 0-19-507597-8.

[Bir87] Richard S. Bird. An introduction to the theory of lists. In M. Broy, editor, *Logic of Programming and Calculi of Discrete Design, NATO ASI Series, vol. F36*, pages 217–245. sv, 1987.

[Car03] Robert Carlson. The pace and proliferation of biological technologies. *Biosecur Bioterror*, 1(3):203–214, 2003.

[Coo99] Daniel Coore. *Botanical Computing: A Developmental Approach to Generating Interconnect Topologies on an Amorphous Computer*. PhD thesis, Massachusetts Institute of Technology, 1999.

[DGG06] André DeHon, Jean-Louis Giavitto, and Frédric Gruau. Computing media and languages for space-oriented computation, 2006. Seminar.

[DKT06] Mathieu Desbrun, Eva Kanso, and Yiying Tong. Discrete differential forms for computational modeling. In *Discrete differential geometry: an applied introduction*, pages 39–54. Schröder, P, 2006. SIGGRAPH'06 course notes.

[Gib04] W. Wayt Gibbs. Synthetic life. *Scientific American*, 2004.

[GMCS05] Jean-Louis Giavitto, Olivier Michel, Julien Cohen, and Antoine Spicher. Computation in space and space in computation. In *Unconventional Programming Paradigms (UPP'04)*, volume 3566 of *LNCS*, pages 137–152, Le Mont Saint-Michel, September 2005. Springer.

[HH98] Tad Hogg and Bernardo A. Huberman. Controlling smart matter. *Smart Materials and Structures*, 7:R1, 1998.

[Hir03] Anil N. Hirani. *Discrete exterior calculus*. PhD thesis, California Institute of Technology, 2003.

[Hor01] Paul Horn. Autonomic computing: IBM's perspective on the state of information technology. Technical report, IBM Research, October 2001. `http://www.research.ibm.com/autonomic/manifesto/autonomic_computing.pdf`.

[KKA+04] Hideki Kobayashi, Mads Kærn, Michihiro Araki, Kristy Chung, Timothy S. Gardner, Charles R. Cantor, and James J. Collins. Programmable cells: Interfacing natural and engineered gene networks. *PNAS*, 101(22):8114–8419, June 2004.

[KKP99] J. M. Kahn, R. H. Katz, and K. S. J. Pister. Next century challenges: Mobile networking for "smart dust". In *Proceedings of the Fifth Annual ACM/IEEE International Conference on Mobile Computing and Networking (MobiCom-99*, pages 271–278, N.Y., August 15–20 1999. ACM Press.

[Law92] Peter Lawrence, editor. *The Making of a Fly*. Blackwell Scientific, 1992.

[Mei82] H. Meinhardt. *Models of biological pattern formation*. Academic Press, New York, 1982.

[Mun84] James Munkres. *Elements of Algebraic Topology*. Addison-Wesley, 1984.

[Nag01] Radhika Nagpal. *Programmable Self-Assembly: Constructing Global Shape using Biologically-inspired Local Interactions and Origami Mathematics*. PhD thesis, Massachusetts Institute of Technology, 2001.

[PLH+90] Przemyslaw Prusinkiewicz, Aristid Lindenmayer, Jim Hanan, et al. *The Algorithmic Beauty of Plants*. Springer-Verlag, 1990.

[PS93] Richard S. Palmer and Vadim Shapiro. Chain models of physical behavior for engineering analysis and design. *Research in Engineering Design*, 5:161–184, 1993. Springer International.

[RPW04] Paul W. K. Rothemund, Nick Papadakis, and Erik Winfree. Algorithmic self-assembly of dna sierpinski triangles. *PLoS Biol*, 2(12):e424, 2004. www.plosbiology.org.

[RS92] Grzegorz Rozenberg and Arto Salomaa. *Lindenmayer Systems*. Springer, Berlin, 1992.

[RS97] Fritz Reinhardt and Heinrich Soeder. *Atlas des Mathématiques*, chapter Topologie algébrique. La pochotèque, 1997.

[Ton74] Enzo Tonti. The algebraic-topological structure of physical theories. In P. G. Glockner and M. C. Sing, editors, *Symmetry, similarity and group theoretic methods in mechanics*, pages 441–467, Calgary, Canada, August 1974.

[vN66] John von Neumann. *Theory of Self-Reproducing Automata*. University of Illinois Press, 1966.

[WA85] William W. Wadge and Edward A. Ashcroft. *Lucid, the Data flow programming language*. Academic Press U.K., 1985.

[WBH+03] R. Weiss, S. Basu, S. Hooshangi, A. Kalmbach, D. Karig, R. Mehreja, and I. Netravali. Genetic circuit building blocks for cellular computation, communications, and signal processing. *Natural Computing*, 2(1):43–84, 2003.

[Wol02] Lewis Wolpert. *Principles of Development*. Oxford University Press, 2nd edition, 2002.