EasyCell: A Language for Describing Cellular Signalling Pathways

Cole Trapnell and Michael Lam

May 20, 2008

Abstract

Cellular biologists strive to accurately simulate intracellular communication pathways using computational models. This kind of simulation allows for study and experimentation without the expense of a wetlab, but is tedious because of the complexity of biological systems and the high resolution of current modeling techniques. We have created a new language called EasyCell for describing cellular pathways using a simple yet powerful syntax, and we have written a translator and runtime using a graph rewriting back-end. We hope this work will eventually enable systems biologists to more easily build and test simulations of cellular pathways.

1 Introduction

One major objective of molecular and cellular biology is the construction of comprehensive models of the mechanisms that drive cells. Understanding the normal operation of the internal workings of cells in various tissues and organisms is key to treating disease. Unfortunately, these mechanisms are difficult to observe in operation in the cells, or *in vivo*. While there is a vast array of 'wetlab' techniques available to probe the workings of part of a cell, these techniques are time-consuming and expensive to execute. Some techniques involve replicating parts of a biochemical mechanism, such as a signalling pathway, in a test tube. These *in vitro* models are easier and cheaper to manipulate but still require time, skilled labor, and costly reagents and equipment to build.

These small-scale cellular and molecular biological methods have been steadily developed and successfully executed for many years. However, within the last two decades, enormous advances in DNA sequencing and high-throughput protein assays have produced an explosion of biological data. The entire genomic sequences for many complex organisms are now known, including those of thousands of bacteria. The network of protein-protein interactions (or "proteomes") are beginning to be measured with high-throughput assays such as yeast two-hybrid and tandem mass spectrometry. [11, 7] The challenge for life scientists is to analyze this enormous volume of data and use it to build a more complete model of the workings of cells. To use a programming metaphor, imagine that we have obtained a copy of the binary executable for an enormously complex program. The challenge faced by life scientists is like trying to figure out exactly how that program works simply by looking at the sequence of ones and zeros in the executable, without knowing the instruction set architecture of the machine.

A promising approach to building good cell models is to synthesize genomic and proteomic data into a computer program that simulates the workings of a particular cellular mechanism. The emerging field of computational systems biology attempts to build so-called *in silico* models. These simulations take as input a description of the mechanics of the cell along with some initial conditions, and produce as output a state or a set of reachable states for that cell. The mechanics are often described as equations that govern how populations of proteins and molecules, termed *species* by the literature, change over time and interact with each other. [8] The output in such models is the set of populations of these species. A potential drawback to equation-oriented, high-level population models of cell mechanics is that one must be fluent in systems of differential equations in order to build them.

In this paper, we propose a new domain-specific programming language, EasyCell, and corresponding runtime environment for expressing signalling pathways in a user-friendly way. This language is a prototype; it is not yet expressive enough to describe all known signalling pathways. However, EasyCell can express many pathways very simply, using keywords and grammatical constructs that should be familiar to a cell biologist.

The paper is organized as follows: we first briefly review previous modelling approaches, with a description of the classic epidermal growth factor (EGF) signalling pathway. We then describe EasyCell's grammar and implementation. We then present highlights from a case study of a complex biological pathway in EasyCell. We conclude with a description of future work and enumerate some specific features that will increase EasyCell's expressive power.

2 Background

2.1 EGF Pathway

The epidermal growth factor (EGF) pathway is often a first target for *in silico* modeling systems. We will use this pathway as an example, so we introduce it here for the readers who are not familiar with it. Figure 1 shows the general layout of components in the first part of the pathway.

The EGF pathway features a chain of protein interactions and binding events that control an array of cell functions and determine the fate of the cell. Notably, the pathway is initiated by the binding of the hormone EGF to a receptor on the surface of the cell and culminates in the transcription and expression of numerous genes, including the gene for EGF itself. The EGF pathway has been implicated in numerous cancers including breast tumors and small cell carcinoma of the lung, and the EGF receptor is overexpressed in the majority of solid tumors. [6]

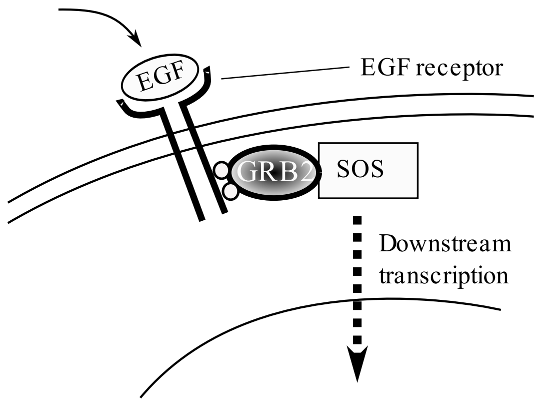


Figure 1: A partial view of the classic epidermal growth factor pathway. Signalling is intiated by binding of the hormone epidermal growth factor (EGF) to EGF receptor (EGFR), which triggers dimerization and phosphorylation of EGFR. Proteins GRB2 and SOS complex with dimerized EGFR and ultimately intiate downstream transcription and expression of numerous genes. The EGF pathway regulates cell proliferation, differentiation, and survival.

As presented below, we are able to represent the first portion of the EGF pathway very simply in EasyCell (see Figure 3).

2.2 Systems Biology Models

Some of the earliest systems biological models were purely analytic - often complex systems of differential equations that drew mainly on population dynamics literature. Ecologists and population biologists modeled the dynamics of rabbits and foxes, and immunologists similarly tried to describe the populations of B cells and T cells in response to antigen. More recently, researchers have attempted to build more general simulations based on varied techniques, including cellular automata [1], petri nets, [5] and rewriting systems [3, 2].

While each approach has advantages and disadvantages, the rewriting-based systems are particularly attractive because they allow very simple expression of complex system dynamics. Rewriting systems such as Kappa [3] and BioNetGen [2] express cell dynamics as a series of rules, each of which is structured as an operation to be performed on a set of elements such as proteins.

Kappa [3] is a rule- and agent-based language abstraction for describing protein interactions. The language provides mechanisms for describing how the sites of various proteins can bind and under what conditions these bindings can take place. The authors of Kappa believe that the granularity this language provides is sufficient to handle most current practical applications, and provide an example using the EGF pathway.

The BioNetGen converter [2] is implemented in Perl and takes as input a description file that defines 1) rate constants and concentrations, 2) molecular components, 3) reaction rules, and 4) output functions. It produces a chemical reaction network in a format that is readable by several ordinary differential equation (ODE) solvers and simulators. Using this system, the authors were able to generate models for antigen-related signaling events, EGF, mitogen-activated protein kinase cascades in yeast, among others.

2.3 Graph Rewriting

EasyCell runs on top of GrGen [4], a generic C# framework for fast graph rewriting. It provides the ability to manipulate graphs deterministically using patterns and transformation rules. Figure 2 shows an example of a GrGen rule. The GrGen language provides great flexibility but at a much finer granularity than is necessary for cellular biology. We believe that by wrapping GrGen in a higher-level descriptive language, we can provide a system that is both powerful and easy to use for systems biologists.

```
// EGF binds to its receptor
rule ActiveEGFR {
  e1:EGF;
  e2:EGFR;
  negative {
    e1 <-- temp1:EGFR;
  }
  negative {
    e2 --> temp2:EGF;
  }
  modify {
    e2 --> e1;
  }
}
```

Figure 2: This GrGen example rule shows the first step of the EGF pathway (an EGF protein binds to a receptor). The pattern specifies that the rule matches a node of type EGF and a node of EGFR that are not already bound to nodes of the other type. If this match occurs, the rule will fire and the "modify" command will create a new link between the two nodes.

3 Language

We designed the EasyCell language with the primary goal of making it easy to understand without extensive programming background. Previous computational systems biology tools have a high technical barrier to entry; one must be an expert C++ programmer, for example, or be very comfortable building systems of differential equations. EasyCell keywords and phrasing are drawn from molecular and cellular biology (ex. "bind" or "transfer") wherever possible.

An EasyCell program is structured as a set of protein and molecule declarations, followed by a series of rules that specify how those proteins interact. There are currently three main rule constructs in EasyCell: bindings, generators, and environments. All of the rules operate on *domains*, which represent a physical molecular interface. All proteins have at least one domain. Some proteins, which we call compound proteins, have more than one domain. To specify a particular domain in a protein or a group of bound proteins, the programmer may use the "in" clause. To add conditions or constraints to a rule, a programmer may use the "where" clause.

EasyCell systems are implemented as dynamic graphs. Nodes for proteins are introduced by the `environment` statement, and then edges between them are added by binding and generator rules. New nodes are also added and deleted by these rules to

```
domain SH2;
domain SH3;
domain PS;

protein EGF;
protein EGFR;

protein GRB2 {
  domain SH2[2];
  domain SH3;
};

protein SOS;

binding EGF binds EGFR produces
  complex ActiveEGFR;

binding ActiveEGFR binds ActiveEGFR
  produces complex EGFRDimer;

generator
  EGFR in ActiveEGFR in EGFRDimer
  produces PS[3];

binding
  SH3 in GRB2
  binds PS in eg:EGFR in ActiveEGFR in EGFRDimer
  produces complex EGFRGRB2
  inhibited by ps:PS in eg where ps in EGFRGRB2;

binding
  (SH2,SH2) in gg:GRB2
    where sh:SH3 in gg and sh in EGFRGRB2
  bind SOS
  produces complex EGFRGRB2SOS;

environment {
  EGF[2];
  EGFR[2];
  GRB2[6];
  SOS[6];
};
```

Figure 3: Representation of the EGF pathway in the EasyCell language

represent groupings of proteins into complexes. We have attempted to hide the graph dynamic underpinnings of EasyCell from the grammar itself, so that as the language's implementation evolves, the grammar will be unchanged.

Figures 7-10 give the complete parser grammar for EasyCell. The lexer rules are straightforward and have been omitted here for brevity.

3.1 Bindings

This construct represents the binding of two proteins or molecules to create a new *complex.* Binding events are central to the functioning molecular and cellular mechanisms. As proteins bind together, the biological system changes state, transports protein and molecular products across the system, or simply conveys information at a molecular level. Binding events are thus central to the way in which biologists discuss pathways. In the EGF pathway, expression of genes is activated through a sequence of binding events.

From a graph dynamics viewpoint, a binding creates a new parent node for both involved entities. This parent node represents the complex formed by the two entities and provides a central access point. We have also provided the capability of producing "free" molecules or proteins (ie. unbound to the complex node or either of the binding nodes).

3.2 Generators

When binding events occur, there may be side effects that modify the participating proteins, or the generation of small free molecules. While these effects could be modeled using binding rules, doing so would be clumsy, so we have introduced *generators* as a convenience.

A generator is a reflexive binding. The pattern specifies a single node that produces other nodes (either free or bound to the generator). In our examples, this is used to model *phosphorylation*, in which a given protein "sprouts" new phosphor groups under given conditions (i.e. the protein has been bound into a certain type of complex).

3.3 Environments

The environment specification allows the programmer to dictate the starting point of the simulation. Each simulation should contain exactly one environment. Syntactically, an environment consists of a list of proteins, compound proteins, and molecules, with an optional indicator of how many of them should be instantiated.

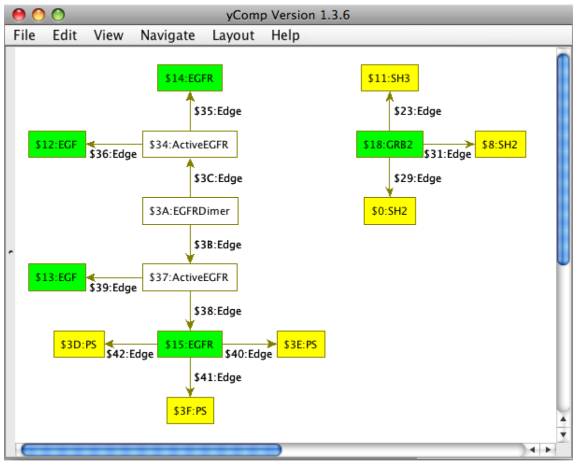


Figure 5: EGF pathway simulation in YComp with an activated, dimerized and partially phosphorlyated EGF receptor visible on the left and an unbound GRB2 protein visible on the right.

4 Translator and Runtime

Using the ANTLR [10] parser generator, we wrote a translator from EasyCell descriptions into the three files needed by GrGen: 1) a model file containing domain, molecule, and protein declarations, 2) a rule file containing rewrite patterns and modifications, and 3) a GrShell script file for loading and running the simulation.

The translator consists of an LL(1) parser/lexer and code generator. Each rule that involves parsing graph information returns a data structure that roughly represents the corresponding graph pattern. The code generation is currently inline for simplicity and easy of development, although we believe it would be relatively simple to move the code generation into its own module.

One interesting aspect of writing the translator was dealing with compound proteins. These are proteins containing multiple domains that must be matched separately, but must be treated as a whole in generations. We address this problem by keeping a list of compound proteins (with their subcomponents) and referring to this list whenever it becomes necessary to instantiate a compound protein.

Currently, the EasyCell translator creates a GrShell script file to run the simulation. This script file uses a Java-based debugger interface provided with GrGen called YComp to visualize the simulation and allow the user to step through the rewriting process. Figure 5 shows a screenshot of the EGF pathway simulation in YComp.

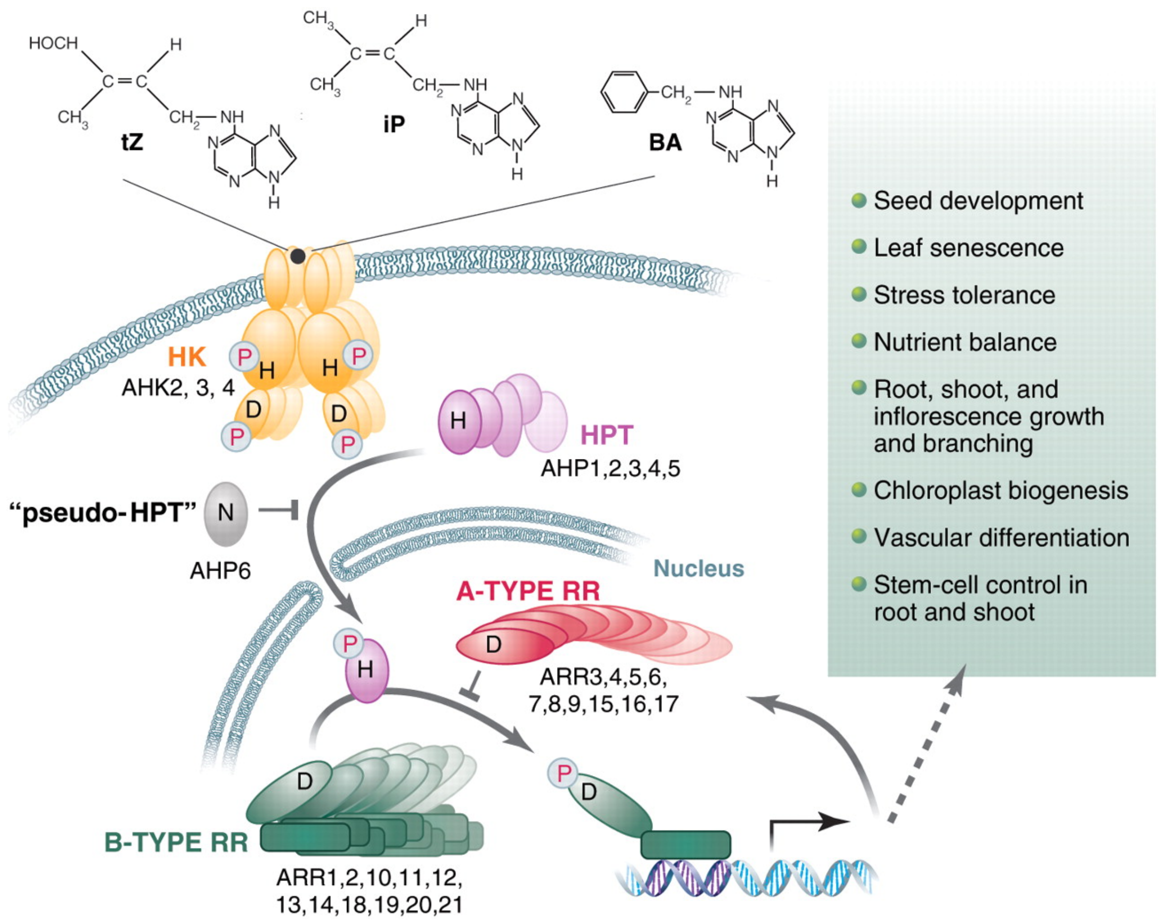


Figure 4: A high level view of the cytokinin signal transduction pathway in *A. thaliana*. Image from [9], copyright AAAS, used with permission.

5 Case Study: cytokinin in *A. thaliana*

The model organism *Arabidopsis thaliana*, or thale cress, was one of first complex organisms to be sequenced. *A. thaliana* is believed to share a signaling pathway that is common to most plants. The cytokinin signal transduction pathway controls various developmental functions and responses to external stimuli, and is much more complex than the EGF pathway. We chose to implement a high-level view of this pathway in EasyCell as a test of EasyCell's expressiveness. We have identified several limitations of the language, which we will address as development continues. [9]

The cytokinin pathway, shown in Figure 3.3, features several biomolecular processes not found in the EGF pathway. First, the pathway uses a *phosphorelay*, a mechanism in which proteins pass a phosphor group from one to another as a means of transmitting information. This phosphor group originates at a transmembrane receptor and is passed from protein to protein until it binds with a transcription factor. Once phosphorylated, this transcription factor allows the expression of an array of genes. This mechanism requires complexes that are transient—once bound, proteins must at some point dissociate from each other. As discussed below, EasyCell lacks an adequate means of expressing dissociation.

Much of the pathway was representable with no changes to EasyCell. Other parts of the pathway were expressible after only minor changes to the language or its implementation. For example, the expression of new genes necessitated the creation of new *free* (uncomplexed) proteins into the environment. The `produces` statement was enriched to recognize an optional `free` keyword, indicating that the product was not a part of a complex.

The cytokinin pathway also specifies that certain proteins compete with other for binding partners. Conversely, some proteins may bind with multiple

```
// many proteins become phosphorylatable
// at some point
property Phosphorylatable {
  generator self produces PHOSPHOR[3];
}
...
EGFR in ActiveEGFR is Phosphorylatable;
```

Figure 6: Example of "properties" idea (generic rules)

other partners, and may prefer binding with one type of protein over another. So called *preferential binding* is a hallmark of both intracellular pathways and networks of the interactions of whole cells. EasyCell cannot express preferential binding, and will need to do so before it is able to completely simulate complex systems such as the Arabidopsis cytokinin pathway.

6 Future Work

Our initial implementation of EasyCell lacks a means of dissociating complexed proteins from one another. We plan to implement an `unbinds` statement that takes a complex and splits it into two pieces. This is not necessarily an inverse operation to the `binds` operator, since the `unbinds` operator may split complexes arbitrarily. This necessitates a somewhat complex topological change, and we have not finalized a clean, expressive syntax for `unbinds`. However, we attempted to implement a simpler statement to specify that a member of a complex moves from one complex to another. We wanted to be able to transfer phosphor groups as part of a phosphorelay (see Section 5). Unfortunately, `transfer` quickly became similarly complicated, and we plan to revisit its syntax in the future. In principle, `transfer` may be implemented as an `unbind` followed by a `bind`, so we may include `transfer` as a convenience expression for that idiom, rather than a first-class element of the language.

Although we did not have time to implement it, we also believe it would be relatively simple and very beneficial to add biological "properties" (ex. phosphorylation) in the form of the ability to parameterize rules, similar to templates in C++ or generics in Java. This would allow the programmer to specify common reactions and behaviors, instantiating them for given proteins as necessary. This promotes code reuse and lowers the amount of work for the end user. See Figure 6 for an example of how properties might be useful.

A generic EasyCell rule would look like a normal rule, but with the addition of a special "self" keyword. The generic rule itself would not output any code, but whenever a particular protein is imbued with a particular behavior using the "is" keyword, a copy of the rule is generated with all occurrences of "self" replaced with that protein.

The usefulness of EasyCell could be greatly improved by the addition of a standard library of common biological components and rules. This library might take advantage of generic rules (see above) for even more expressiveness and power. The availability of a standard library would ease the burden on the end user and provide additional stability (assuming it is well-written and thoroughly debugged).

The most significant weakness of our current language and translator is that there is currently no way of easily expressing preferential or non-deterministic bindings. This would likely require major changes to the EasyCell runtime, as it would need finer control over GrGen. Such control is possible, but will require significant engineering.

7 Conclusion

We have presented EasyCell, a prototype language for expressing cellular signalling pathways. EasyCell runs on top of GrGen, a fast graph rewriting system, and includes a translator, runtime, and basic visualization for building, running, and viewing simulations. While EasyCell has some significant limitations, none are insurmountable. The language is already able to express a wide array of pathways, such as the classic epidermal growth factor pathway, and with some additional improvements will be able to express much more complex pathways. *In silico* experiments are not yet practical, primarily due to the difficulty of working with existing modeling systems. We believe that EasyCell is significantly easier to use than existing tools, and will allow the inexpensive investigation of complex pathways.

References

[1] Advait Apte, John Cain, Danail Bonchev, and Stephen Fong. Cellular automata simulation of topological effects on the dynamics of feed-forward motifs. *Journal of Biological Engineering*, 2(1):2, 2008.

[2] Michael L. Blinov, James R. Faeder, Byron Goldstein, and William S. Slavacek. Bionetgen: software for rule-based modeling of signal

transduction based on the interactions of molecular domains. *Bioinformatics*, 20(17):3289–3291, 2004.

[3] Vincent Danos, Jerome Feret, Walter Fontana, Russell Harmer, and Jean Krivine. Rule-based modelling of cellular signalling. In *18th International Conference on Concurrency Theory, Proceedings*, September 2007.

[4] R. Greiß, G. V. Batz, D. Grund, S. Hack, and A. M. Szalkowski. GrGen: A fast SPO-based graph rewriting tool. In *3rd International Conference on Graph Transformations (ICGT 2006), Proceedings*, September 2006.

[5] Simon Hardy and Pierre N. Robillard. Petri net-based method for the analysis of the dynamics of signal propagation in signaling pathways. *Bioinformatics*, page btm560, 2007.

[6] Roy S. Herbst. Review of epidermal growth factor receptor biology. *International Journal of Radiation Oncology*, 59(2):S21–S26, June.

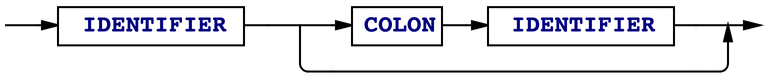
[7] Lan Huang et al. The Identification of Protein-Protein Interactions of the Nuclear Pore Complex of Saccharomyces cerevisiae Using High Throughput Matrix-assisted Laser Desorption Ionization Time-of-Flight Tandem Mass Spectrometry. *Mol Cell Proteomics*, 1(6):434–450, 2002.

[8] Hiroaki Kitano. Computational systems biology. *Nature*, 420, November 2002.

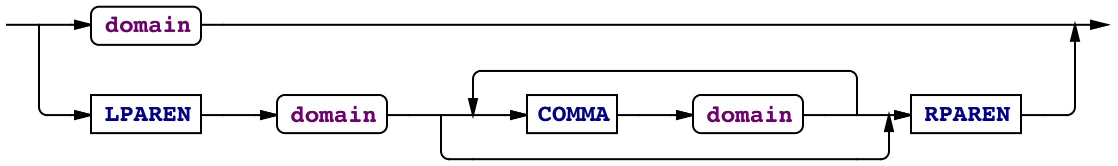
[9] Bruno Muller and Jen Sheen. Advances in Cytokinin Signaling. *Science*, 318(5847):68–69, 2007.

[10] Terence J. Parr and Russell W. Quong. ANTLR: A predicated-LL(k) parser generator. *Software: Practice and Experience*, 25(7):789–810, 1995.

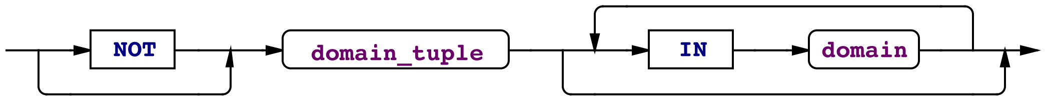
[11] Peter Uetz et al. A comprehensive analysis of proteinprotein interactions in saccharomyces cerevisiae. *Nature*, 403:623–627, February 2000.



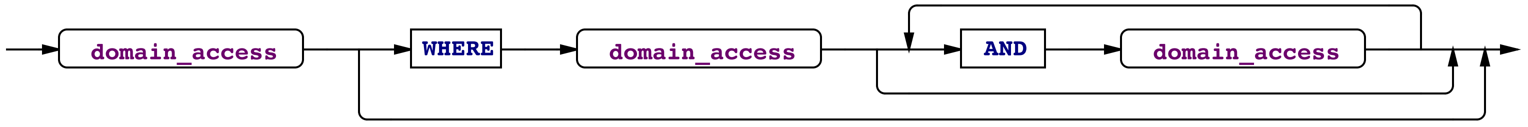
(a) domain



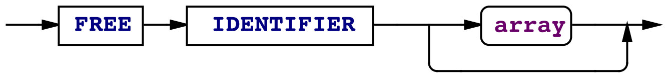
(b) domain_tuple



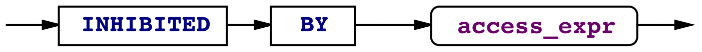
(c) domain_access



(d) access_expr

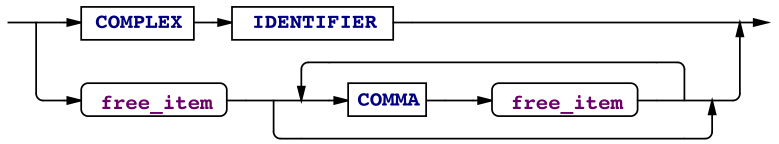


(e) free_item

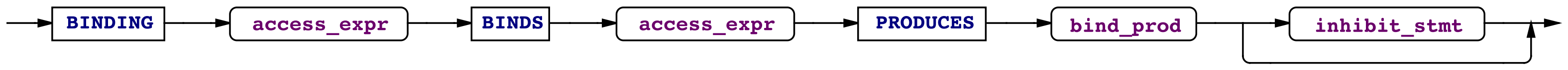


(f) inhibit_stmt

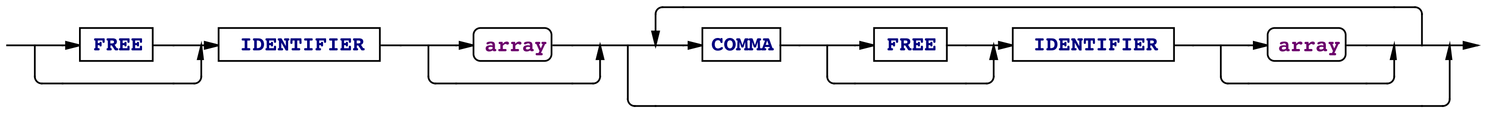
Figure 7: EasyCell Grammar Rules



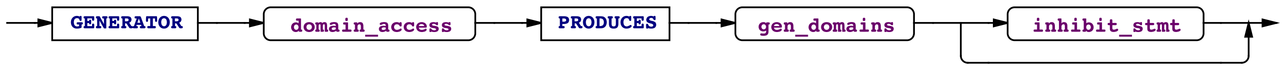
(a) bind_prod



(b) bind_stmt

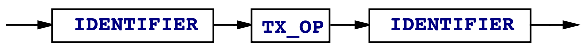


(c) gen_domains

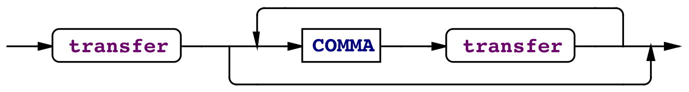


(d) gen_stmt

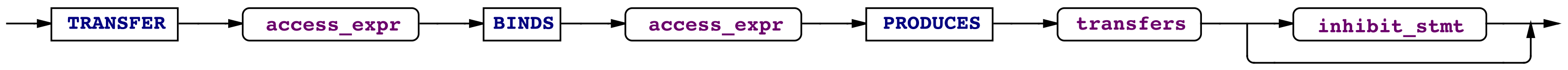
Figure 8: EasyCell Grammar Rules



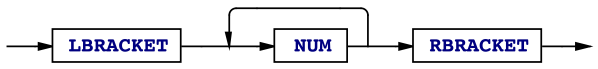
(a) `transfer`



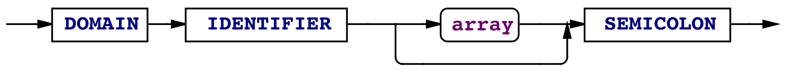
(b) `transfers`



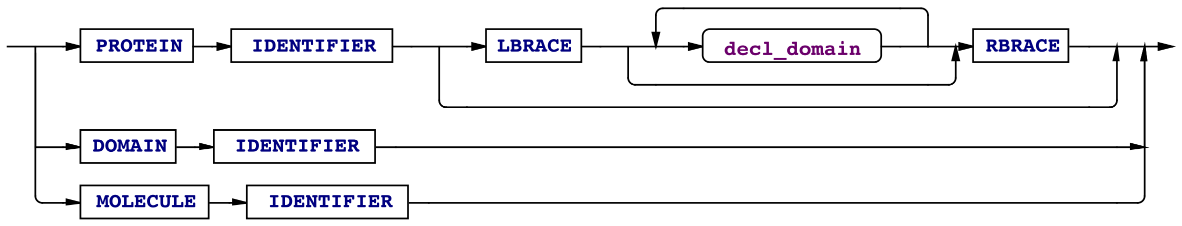
(c) `tx_stmt`



(d) `array`

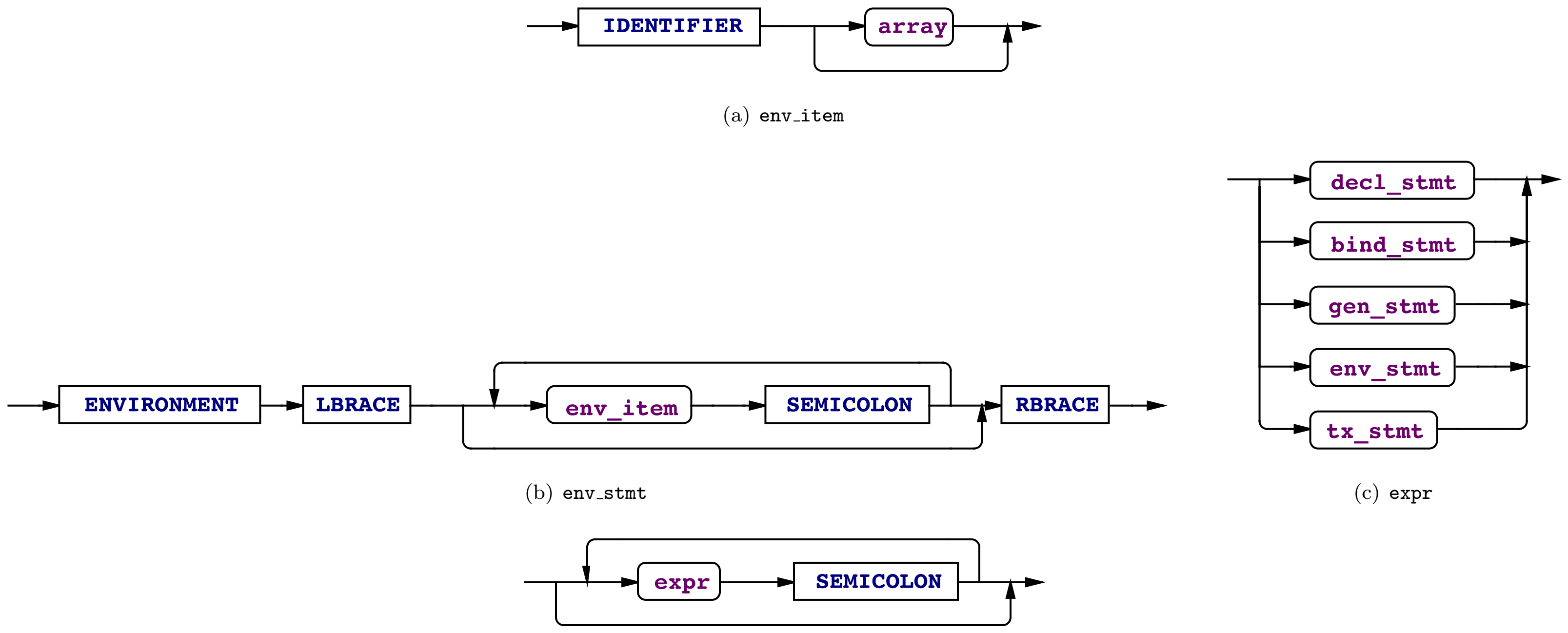


(e) `decl_domain`



(f) `decl_stmt`

Figure 9: EasyCell Grammar Rules



(a) `env_item`

(b) `env_stmt`

(c) `expr`

(d) `start`

Figure 10: EasyCell Grammar Rules