THE UNIVERSITY
of EDINBURGH

A modelling language for biology with applications

Argyris Zardilis

Doctor of Philosophy

School of Biological Sciences

University of Edinburgh

2019

Abstract

Understanding the links between biological processes at multiple scales, from molecular regulation to populations and evolution along with their interactions with the environment, is a major challenge in understanding life. Apart from understanding this is also becoming important in attempts to engineer traits, for example in crops, starting from genetics or from genomes and at different environmental conditions (genotype $\times$ environment $\rightarrow$ trait). As systems become more complex relying on intuition alone is not enough and formal modelling becomes necessary for integrating data across different processes and allowing us to test hypotheses. The more complex the systems become, however, the harder the modelling process becomes and the harder the models become to read and write. In particular intuitive formalisms like Chemical Reaction Networks are not powerful enough to express ideas at higher levels, for example dynamic environments, dynamic state spaces, and abstraction relations between different parts of the model. Other formalisms are more powerful (for example general purpose programming languages) but they lack the readability of more domain specific approaches.

The first contribution of this thesis is a modelling language with stochastic semantics, Chromar, that extends the visually intuitive formalisms of reactions, in which simple objects, called agents, are extended with attributes. Dynamics are given as stochastic rules that can operate on the level of agents (removing/adding) or at the level of attributes (updating their values). Chromar further allows the seamless integration of time and state functions with the normal set of expressions – crucial in multi-scale plant models for describing the changing environment and abstractions between scales. This leads to models that are both formal enough for simulations and easy to read and write.

The second contribution of this thesis is a whole-life-cycle multi-model of the growth and reproduction of *Arabidopsis Thaliana*, FM-life, expressed in a declarative way in Chromar. It combines phenology models from ecology to time developmental processes and physical development, which allows to scale to the population and address ecological questions at different genotype $\times$ environment scenarios. This is a step in the path for mechanistic links between genotype $\times$ environment and higher-level crop traits.

Finally, I show a way of using optimal control techniques to engineer traits of plants by controlling their growth environmental conditions. In particular we explore (i) a direct problem where the control is temperature – assuming homogeneous growth

conditions and (ii) indirect problem where the control is the position of the plants – assuming inhomogeneous growth conditions.

Lay Summary

Living organisms are often viewed as hierarchical. Entities at one level (e.g. cells) function and organise giving rise to properties of entities at the next level (e.g. tissues) and they in turn to the next until we get to the level of a living organism. If we want to understand life, which is the most fundamental question in biology, we need to understand the processes and the links between them across these levels. As the systems we are trying to understand become more complex, holding all the interactions in our heads as mental models becomes difficult (or impossible). One way to solve this is by using formal models written in some technical language (e.g. mathematical equations). The advantage is that these can be manipulated in predefined ways, for example setting them in motion inside a computer, without the danger of missing anything. When trying to understand all the scales that make up an organism though, some of these languages become hard to read and write and can no longer serve as documentations of our understanding.

In this thesis I first present a formal language, *Chromar*, which has some features that make writing a class of multi-scale models easier. Some of these features include mixing discrete with continuous entities, for example you can describe how cells are created or destroyed (discrete) but also how the size of cells changes (continuous). It also allows the mixing of deterministic with stochastic dynamics.

I then use this language to describe in a model the whole lifecycle (from seed to seed) of the plant *Arabidopsis Thaliana*, a commonly used organism in plant research, by combining or extending existing models. Such a model allows then multiple generations and the observation over time of populations of plants in environmental conditions of our choice. We can then compare populations of different types of plants in different scenarios and ask which type is more succesful in reproducing in some conditions. This might give us some hints on why some types of plants adapted to specific locations.

Finally, since there are now solutions that allow more precise control over the surrounding environment (think more precise and controlled greenhouses), one might ask if it is possible to find and use optimal environmental conditions (e.g. temperature) to achieve specific trais of plants. Here I use one of these comprehensive plant models along with other mathematical techniques to address this question.

Acknowledgements

I would like to thank everyone that has helped along the way. It has been a journey with many ups and downs that has transformed me in many ways for the better.

First I would like to thank my supervisor Andrew Millar for being an inspiration both as a scientist and as a person and for believing in me even when things did not look great. This thesis would not have been possible without Gordon Plotkin and Ricarco Honorato-Zimmer. Apart from technical discussions and help, their enthusiasm and appetite for new ideas have sparked my own enthusiasm when I seemed to had forgotten it. Their clarity of mind has shaped my own thinking. Filippo Menolascina also, who I have worked with the last months, that with his positivity also reminded of the fun of doing research.

I would like to thank my parents that with their presence throughout my life have shaped my beliefs and attitude in life. Their continuous support along with my sister has been immensely helpful also during this PhD and especially these last few months that were very difficult for me. I cannot imagine how it would have been without them.

I feel very grateful to my friends Alex, Panicos, Kyriacos, and Constantinos. Even though they were physically very far, they were always close. Also my friends in Edinburgh: Ira, Jeanne, Manuel, Kristyna, Anik, Uriel and everyone. They have kept me positive and reminded of the point of it all when I sometimes kept forgetting it. I feel very lucky to have met you and even if I picked up a tiny amount of you through our interactions it has made me a much better person. Also the greek gang, Iordanis, Chris and Chris, Andreas, Pigi, Giannis, Irene. I cannot imagine Edinburgh without you. I am especially grateful to my flatmate Andreas that with his humour kept me in touch with reality.

Finally, I would like to thank everyone in the Millar group and the RBM group in Informatics especially Yin-Hoon Chew, Daniel Seaton, and Uriel Urquiza. It was great to work and share ideas with you all.

Declaration

I declare that this thesis was composed by myself, that the work contained herein is my own except where explicitly stated otherwise in the text, and that this work has not been submitted for any other degree or professional qualification except as specified.

(*Argyris Zardilis*)

Contents

Chapter 1

Introduction

Life is produced and sustained through the organisation of entities at various levels (e.g. molecules, membranes) that interact to construct other autonomous entities (such as cells). Organisation and structure are therefore fundamental and defining properties of all living systems. Some go as far as to say that (self-)organisation and the resulting self-maintenance are the only common and defining characteristics of living organisms and that they are more important than even natural selection in shaping the forms of life that we see (sometimes referred to as structuralism, for example; Kauffman, 1992; Thompson et al., 1942). Even if we do not take such an extreme view, it is still true that if we want to understand life we have to understand the organisation that sustains it and the links between processes at all scales defined by this organisation – from genes to organisms and ecosystems. Organisation here means both spatial organisation, for example cells in a tissue, but also relations or interactions between processes that give rise to other higher-level phenomena or between processes and the environment.

This view of organisation as a fundamental and defining principle of biology makes, in my opinion, the subject distinct from other natural sciences, like physics for example and precludes a natural mode of enquiry, which is abstraction. The nature of the questions is different. How can we abstract the details of an organism if those details are exactly what makes the organism alive and in fact what we are trying to understand? Since abstraction or at least complete abstraction is not available, in order to answer biological questions we have to take a more holistic and multi-scale view of the natural systems of interest.

Despite the importance of organisation in understanding life, a common focus in biology has been on understanding individual mechanisms at the molecular level. The reductionist view of life is that if the cell is the main unit of life then if we understand

all the processes inside the cell we will understand life, as everything else follows.

There were, however, even if on the fringes or often completely outside mainstream biology, few historical (and perhaps independent) threads of work that recognised the importance of organisation and consequently of the systems view. C.H. Waddington, for example, talked about the importance of the interaction of parts in developmental systems (G. E. Allen, 1977), perhaps inspired by the metaphysics of Alfred North Whitehead (same book; G. E. Allen, 1977, Autobiographical note). While the polarity between constituents of systems and the importance of their organisation (systems view) might have been a topic of discussion in the field at the time (Waddington, 1961), Rashevsky (1954) explained the need for a theoretical study of organisation. He called this organisation the 'relational' aspects of biology that needs to be addressed in theoretical studies to complement usual approaches that follow only the metric aspects of physical systems. This is the usual abstraction in Physics where systems are viewed only by their quantifiable properties (captured by numeric variables). Rosen (1958) developed such a theory of organisation, based on the (meta-)mathematics of category theory, for the relational/organisational aspects of biology and was a proponent of a systems view (Rosen, 1991). Again, thinking about questions on the origin of life, Varela et al. (1974) developed the concept of 'autopoieisis' (self creation through organisation) as a necessary condition for autonomous living entities. Similar questions regarding the origin of macromolecules in cells lead to a study of self-organisation by Eigen (1971). Following this W. Fontana and L. W. Buss (1994) working with Peter Schuster (Eigen's student) highlighted the importance of organisation as the defining characteristic of living systems and revisited the problem of developing a theory of organisation as a basis for biological understanding. For this theory they borrowed from computer science (λ-calculus in particular; Walter Fontana and Leo W Buss, 1996). Finally, the systems view entered the mainstream through the *systems biology* movement (Kitano, 2002) as a way to consolidate increasingly diverse experimental molecular biology datasets.

As the systems we are trying to understand become more complex, which is particularly the case when we take the systems view, intuitive thinking and mental models of the processes are not adequate tools to aid our understanding. A useful tool in these cases is formal models where the systems are represented using some formal language based on a mathematical interpretation, either directly or indirectly. An interesting view is that of formal models as machines that turn the assumptions that we put into them to conclusions (Gunawardena, 2014). The unique advantage of formal models compared to informal ones (mental models) in biology is that this machine is guaranteed to function

reliably therefore providing a good tool for testing our hypotheses (assumptions) just like an experiment would. Mental models however do not provide this guarantee and we therefore do not know if the fault is in our assumption or in the animation of the model in our heads.

Historically, it is no surprise that formal models have been used in systems models, for example for morphogenesis (Turing, 1952) and neuronal activity (Hodgkin and Huxley, 1952) although in some cases good abstraction theories have been used in evolutionary genetics (by abstracting away the organism; Dobzhansky, 1964) and in the small scale for chemical kinetics (Gunawardena, 2012; Michaelis and Menten, 1913), (see also; Gunawardena, 2013). With the movement of systems biology as the intellectual successor to the system and organisation theorists of the 20th century we have seen an increase in mathematical models of cellular processes, from metabolism to the circadian clock.

Taking the systems and modelling view to the extreme and remembering the fundamental goal of understanding living organisms through their organisation, we can take a step further and expand our systems view from the cell to larger scales, for example to the organism and beyond. These models that reach beyond organisation on a single scale are usually called *multi-scale models* and they are more close to the historical spirit of understanding system organisation to understand life, as we have seen above. For small micro-organisms, comprehensive models link the metabolic and molecular level with the cellular (Karr et al., 2012) and population growth scales (Weiße et al., 2015). Work on more complex organisms has focused more narrowly (Virtual liver, Holzhütter et al. (2012); Virtual heart; Noble (2002); Bone system, Paoletti et al. (2012)) although there are ambitious collaborative project aiming at whole organisms (virtual rat, virtual human Beard et al., 2012; Kohl and Noble, 2009).

Plant science research is concentrated on the laboratory model species *Arabibopsis thaliana*, which offers an opportunity for broad understanding that includes mechanistic models (Chew, Smith et al., 2014; Voss et al., 2014). This multi-scale view starting from a molecular mechanism and metabolism is realised explicitly in the Framework Model that represents vegetative growth starting from a specific well-understood molecular mechanism, the circadian clock (Chew, Wenden et al., 2014, FMv1). On the other end of the scales spectrum there is modelling work for plants in the crop and ecology world where the organism is usually absent. While this work usually is concerned with more crop or ecologically relevant species, recently Arabidopsis is starting to appear in crop and ecology research in attempts to bring the organism back into the equation

and provide insight into relevant mechanisms that affect ecological or crop phenotypes especially with respect to the interaction with the environment.

The work of this thesis is situated exactly in this space, in the intersection and modelling space between plant science and crop or ecological research. We are particularly interested in linking multi-scale plant models that consider growth through environmental interaction with ecological models that consider ecosystems in the natural environment. Apart from the scientific challenges these multi-scale models present, in plants or elsewhere, there are also some challenges regarding the technical language used for their representation.

1.1 Aims

Therefore, the *aims* of this thesis are:

- Develop a suitable language to address the problem of naturally and succinctly representing multi-scale plant models that go beyond the organism and act in a dynamic natural environment.
- Use the above language to create such multi-scale plant models from the organism to the ecosystem.
- Employ the models for answering questions in ecological models using the mechanistic understanding that comes from including a model of the organism in ecological studies.

In the rest of this chapter I will first give a simplified overview of modelling in the plant domain starting from plant science and going to ecology and crops. My objective here is to locate and motivate the challenges for modelling languages, rather than a comprehensive survey of the state of the art in plant models. I will then overview the language question in multi-scale models that relates to the organisation view of living organisms we noted above. Finally, I give the contributions of this thesis in reference to the outlined aims.

1.2 From plants to crops and ecosystems

Much contemporary work in plant biology research has been focused on understanding molecular mechanisms at the foundation of important plant processes like photosyn-

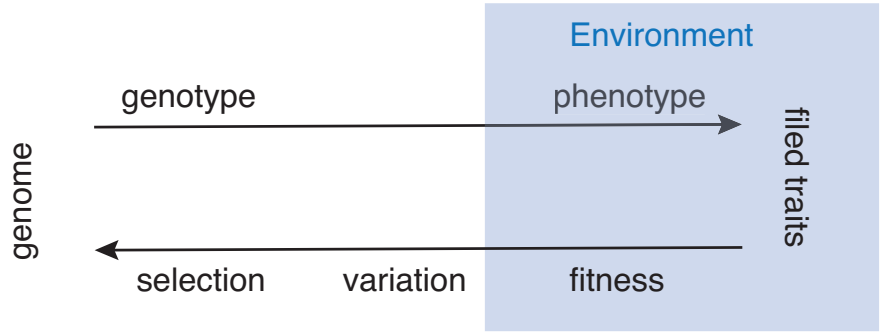


Figure 1.1: Plant biology research is usually concerned with the top part of the cycle and the 'how' of organisms. Evolution and ecology are concerned with the why although they usually use a limited view of the 'how' to make hypotheses about the 'why' (adapted from Andrew J Millar, 2016).

thesis. Unlike mammalian systems, however, the effect of environmental conditions is usually taken into account since plants are highly dependent on those. The importance for crop traits also led to attempts at linking molecular knowledge to organism traits, like growth. We can think of these kinds of questions of how a particular phenotypic trait arises from a molecular mechanism as the *how* questions (top part of cycle in Figure 1.1).

In the fields of ecology and evolution on the other hand, the question can also address more on they *why* organisms evolved the way they have (bottom part of cycle in Figure 1.1). This is usually done by observing genetic variation and with some knowledge of the how this affect organism behaviour make evolutionary hypotheses on the why.

Multi-scale plant models that go beyond the organism can be seen as tools for understanding (evolutionary ecology) and tools for engineering (crop science). Next, we start with plant models with an emphasis on the Framework Model of Chew, Wenden et al. (2014) and go to an overview of crops and evolutionary ecology models.

1.2.1 Multi-scale plant models

Modelling in plant biology is usually concerned with individual mechanisms, for example photosynthesis (Farquhar et al., 1980), leaf stomata conductance (Tardieu and Davies, 1993). What perhaps makes it distinct from modelling work in other species is the importance of the environment and the attempt to link these molecular processes to organism traits directly important for agriculture. The time keeping mechanism of

the circadian clock is particularly important for the synchronisation with the outside environment and has been studied with the use of models both in *Arabidopsis Thaliana* (Locke et al., 2005) and in cyanobacteria (Hertel et al., 2013; Miyoshi et al., 2007). Photosynthesis models have been linked to sucrose synthesis (Zhu, Wang et al., 2013), the circadian clock to flowering time (Salazar et al., 2009), and leaf-stomata conductance to leaf growth (François Tardieu et al., 2015).

Unlike other organisms most of the development of plants happens throughout their lifecycle and a lot of work was based on descriptive models of their structural growth (Mündermann et al., 2005). As the role of molecular mechanisms into development became clearer there is more multi-scale work linking molecular mechanisms to development, for example there are models of the distribution of the hormone auxin along the plant structure, which is known to act as a morphogen (Jönsson et al., 2006; Prusinkiewicz, Crawford et al., 2009). Functional-structural models also try to link molecular mechanisms, usually abstract views of metabolism, to structural growth at the organ level (Christophe et al., 2008).

Plant development models usually ignore most of the underlying biochemistry of the organism except for a very narrow description of directly relevant molecular mechanisms. On the other hand models of molecular mechanisms, while sometimes linked to the organism, they usually ignore the constructive organisation of the plant. The question is can we construct the organism representing its constructive organisation and underlying mechanisms? We next overview one attempt at a digital organism, the Framework Model, which is inspired by the functional-structural models.

Framework Model

The Framework Model represents vegetative growth of Arabidopsis in lab conditions (Chew, Wenden et al., 2014), starting from four independent models that represent photosynthesis and carbon storage (Rasse and Tocquin, 2006), plant structure and carbon partitioning among organs (Christophe et al., 2008), flowering phenology (Chew, Wilczek et al., 2012) and the circadian clock gene circuit and its output to photoperiodic flowering (Salazar et al., 2009). Later updates focussed on plant phenotypes controlled by the clock, such as tissue elongation and starch metabolism (FMv2; Chew, Seaton et al., 2017), or temperature and organ-specific inputs to flowering (Kinmonth-Schultz et al., 2018). The Framework Models align with community efforts to link understanding of crop plant processes at multiple scales, for benefits in agriculture (Wu et al., 2016; Zhu, J. P. Lynch et al., 2016).

Among the limitations of the Framework Models, growth was limited to the vegetative stage, ending upon flower induction. Without reproduction, the models had no seed yield or link to evolutionary fitness. Without seed dormancy, they lacked a major determinant of Arabidopsis life history in the field. Their representation of the circadian clock was also unnecessarily detailed for many studies outside chronobiology. These limitations mean that going beyond the organism to the population level to answer ecological questions is not practically possible without extra effort.

1.2.2 Crop modelling

Crop models use many of the same ideas that we have seen in multi-scale plant models but they have a different more engineering oriented goal in mind instead of the fundamental biology goal of understanding. Usually the motivation is understanding the effect of interventions to either plant genetics or crop management on plant performance, which is usually growth related.

Crop models often regard the entire crop as one big plant. The challenges relate exactly to this assumption. For example, computing light interception of the canopy has to take into account the geometry and assumed configuration and positioning of plants in the field. The constructive organisation of plants is absent and plant processes, like photosynthesis, are scaled to the entire crop (Monteith, 1965). Genetics or molecular mechanisms are represented, if at all, in a phenomenological way. For example, a mutant might have a different set of model parameters and so on (Parent and François Tardieu, 2014).

Since the goal in these models is prediction, as long as they come reasonably close to the behaviour of the crop under different environmental conditions they are considered to be adequate representations. However, because of their nature it becomes difficult to capture the responses of the systems under multiple environmental inputs.

A flows-based view of a plant

Other approaches start from a whole-plant model and then try to extrapolate to the entire crop. The usual representation is of the plant as multiple reservoirs of nutrients with flows between them (France, Thornley et al., 1984). Growth, which is the main goal usually, is calculated as a function of the levels in these nutrient pools (see Figure 1.2).

This view has later found its way into more traditional plant models and it is the metabolism level representation found in the Framework Model, for example, although

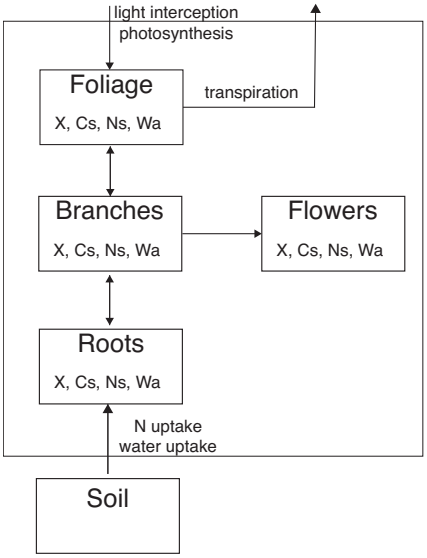


Figure 1.2: A generic plant model with the flows-based view of a plant (adapted from France, Thornley et al., 1984). The plant is represented by several state variables representing the levels of various nutrients in different parts of the plant. The dynamics are then seen as flows between the pools. The pools could also represent entities outside the system, like the soil, for example. The variables represent, structural mass (X), nitrogen (N), carbon (C), and water (Wa). The output is usually the biomass of each part of the plant and especially that of fruits or seeds, which are important in agriculture.

the flows are calculated more mechanistically and the pools are at the organ level reflecting the explicit representation of plant organisation. Again, in this whole plant flow-based models, the constructive organisation of the plant is usually absent preventing fundamental questions. However, as long as the goal is prediction for engineering, then often this kind of representation is adequate.

Multi-scale models

The multi-scale systems modelling approach has also been proposed in crop modelling in attempts to engineer crop traits starting from genetics or from genomes (Chenu et al., 2018; Parent and François Tardieu, 2014; Welch et al., 2005; Wu et al., 2016; Yin and Paul C. Struik, 2008; Yin and Paul C Struik, 2010), where simpler models have demonstrated both the potential of crop modelling in general and the significant demands of detailed models for empirical data that varies in availability (Asseng et al., 2013; G. Hammer et al., 2006).

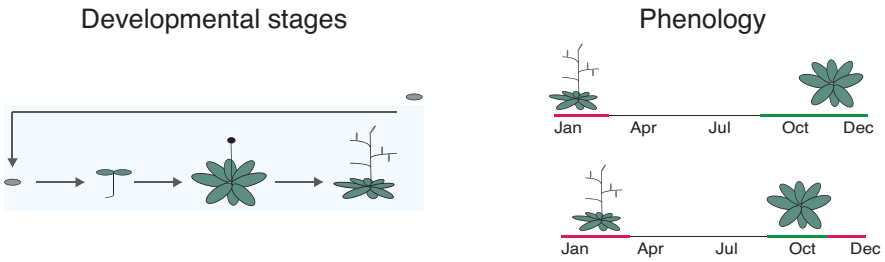


Figure 1.3: Left: Plant developmental stages Right: Two example timings of the growth period in the year (phenology). The vegetative stage is shown in green and the reproductive in red.

1.2.3 Ecology and evolution

The lifecycle of Arabidopis plants can be divided into three major developmental stages where the transition between them is marked by developmental events. Starting out as a seed the plant germinates transitioning to the vegetative stage after which it bolts going to the reproductive stage that ends with the dropping of the new seeds (Figure 1.3).

In natural settings the timing of these events throughout the year (*phenology*) determines the environment that the plant is exposed to during growth. Therefore phenology is a major determinant of plant fitness. If growth coincides, for example, with unfavourable weather conditions the plant might not have enough resources to put into making seeds or might not even survive to reproduction.

Yet species have a natural geographic range that spans a wide range of climatic conditions. For example, Arabidopsis in Europe grows naturally all the way from Spain to Northern Finland. The hypothesis is then that the mechanisms that control these developmental events might be under selective pressure to adapt to different climates, which can then be tested by experimental growth under multiple environments. A lot of work in evolutionary ecology of plants is focused on understanding the molecular mechanisms involved in plant adaptation through the control of developmental events. The assumption is then that these mechanisms reflect adaptation to different environments and therefore hint towards the 'why' questions. Mendez-Vigo et al. (2011) characterise the variation in four flowering related genes across large geographical regions to investigate their involvement in the variation of flowering related traits across the region. Other studies focus on seed dormancy related genes (Chiang et al., 2011) or more broadly on general characterisation of genetic variation for multiple traits across the natural habitats of Arabidopsis (Atwell et al., 2010). The developmental events are correlated since the length of one stage affects the next, which in turn affects the next

generation. Therefore the genetic mechanisms behind two related (and consecutive) developmental events are sometimes studied together (Debieu et al., 2013).

Phenology models

The hypothesis in phenology is that plants sense and integrate environmental signals until they reach the optimal condition for triggering the transition to the next developmental stage. There are experimental evidence of this integration but evidence on the mechanism are still behind although there are is some work pointing to the clock as a possible mechanism.

Phenology models are usually phenomenological and make no attempt to represent the mechanism for sensing and integration of the environmental effects. Instead, they use numerical quantities to represent the progress towards the transition, which is an integral over time of the values of environmental indicators that affect the particular stage under consideration (Chuine, Cortazar-Atauri et al., 2013). For example, if we know that temperature affects a developmental stage, a phenology model might use a temperature sum to track the progress towards the transition:

$$d(t) = \sum_{i}^{t} T(i) - T_b$$

The development at time t, $d(t)$, is the sum of the temperatures at every time unit above a base temperature T_b. The transition time, t_s, is the time such that the development sum reaches a threshold value, $d(t_s) > D_s$. Incorporating genetics is usually represented by adding a sensitivity value to the above, $d(t) = \sum_i^t k \cdot (T(i) - T_b)$ or changing the base temperature, T_b. The idea is that plants adapt by changing, k or T_b, which reflects some molecular mechanism, to change the length of the stage and time their transition at the optimal time depending on the natural environment.

Modelling phenology usually goes in tandem with experimental observations about timing but the models usually stop there and they do not deal with the evolutionary hypotheses that usually follow. These models usually appear in ecological studies and do not consider the genetics while studies of genetic variation of developmental timing adaptation usually do not have a modelling component.

Furthermore, dealing with each developmental stage in isolation means, while being simple and inducing simple 'why' hypotheses, is sometimes not enough as one developmental stage affects the timing and length of the next and the length of the entire generation affects the next generation and so on.

System models

In order to be useful in evolutionary ecology other work has incorporated genetics and fitness into the simple phenology models of the kind we have seen above. Chuine and Beaubien (2001), for example, propose a model that combines fitness with traditional phenology models to predict the distribution of a species. The model is simple, the phenology part predicts the timing and the fitness, given as probabilities of survival under different weather stresses, predicts overall survival rates.

While the simple phenology models consider species mean behaviour, other work has made the links between the genetic studies of timing mechanisms to phenology models to understand the range of expected behaviour both within but also between populations. For example, Wilczek et al. (2009) add genetic variation of the FLC gene (involved in control of flowering time) to a flowering phenology model to predict variation in flowering time and vegetative season length.

Liana T. Burghardt et al. (2015) takes a more systems approach by taking an integrative look at the whole life cycle instead of events in isolation. Variation is also considered explicitly by using an individual-based population model. Fitness, however, is not considered either directly as the reproductive success of the individual in the population or indirectly as survival rates as we have seen before. The focus is more on the timing and the consequences of the genetic environmental interaction.

In representations in both simple and more complex evolutionary ecology models in this domain the organism is absent. While this gives tractable representations of complex processes and allows understanding of experimental results, a more fundamental understanding requires the organism to fill the gaps between the scales from genes to the population. Adding the organism to the population level recovers a more mechanistically founded reproductive success (fitness) as a first step towards explicitly modelling evolutionary dynamics.

1.3 Languages for multi-scale biology

We have already outlined the need for formal models in the description of biological systems. The unique advantage is their formal nature, which means that they can be used as tools for turning hypotheses into predictions using the formal methods that are provided by the technical language used to specify them. Usually this involves simulating them on a computer. Their formal nature also serves another purpose. Since

they are externalisations of our knowledge about a system, the representations can serve as documentation for our knowledge when trying to communicate it to ourselves (for thinking) but also for others.

This view of notation or language as tool for thought and not solely as technical object has been recognised by Kenneth Iverson who succinctly listed the characteristics of a good notation: “ease of expression of common constructs in the domain, suggestivity, ability to subordinate detail, economy, and amenability to formal proofs” (Iverson, 2007).

Modelling in biology becomes necessary when systems are complex. However, the more complex the systems are, the harder models become to read and write. This means that, while they might serve their first purpose as technical objects, they fail on their second purpose as tools for thought. One reason for this might be that the standard technical languages from Physics, like Ordinary Differential Equations, are not adequate for the representation of the constructive nature of the organisation of living organisms. The standard abstraction is to ignore the organisation and describe metric properties of the system captured with their values captured in numeric variables and their dynamics in equations. While this abstraction of systems as numeric variables is sometimes enough, if we really want to fundamentally understand an organism or aspects of it, which is the case in these complex systems, we also need a description of their constructive organisation. This was already noted, as we have seen, by the first ‘structuralists’ and systems theorists like Rashevsky who pointed out the need for the representation of the ’relational aspects of biological systems’ as well as their more metric properties Rashevsky (1954). Later work focused on finding suitable theories of organisation analogous to the equations of motion in Physics. For that reason Rashevsky’s student, Robert Rosen, was called the Newton of biology (Mikulecky, 2001).

While we appreciate the organisation is important, especially at multi-scale representations abstraction through quantifiable properties is also needed especially with more complex systems where we do not have the capacity or the desire to model everything mechanistically.

For complex models like the ones we have seen above from plant, crop, and evolutionary ecology research they are either simple enough that they are captured with ODEs or similar formalisms or they are complex that they cannot be easily mapped to any existing technical language in which case more pragmatic approaches are used where the model becomes a program in a general purpose programming language. This

is problematic because the knowledge is usually obscured in the simulation details.

The more complex models are the ones we are interested in, representing processes like whole-plant models beyond the organism reaching to ecology and evolution. The next chapter (Chapter 2) is dedicated to a more in depth overview of existing work on languages for this domain and goes into more details in some of the issues we highlighted above using illustrative examples.

1.4 Contributions

The work of this thesis is concerned primarily with whole plant models that go beyond the organism, languages for their description, and their applications. In reference to the aims and discussion above, specifically my contributions are:

- A formal language, called *Chromar*, for describing systems like the ones we hinted to above at the intersection of plant biology models in a natural environment. The language follows the structuralist tradition and is therefore object-based, which means that it can capture certain important aspects of the constructive organisation of living systems. At the same time quantifiable properties can also be represented allowing abstraction while further extensions allow an enriched expression language for these properties that combines regular mathematical expressions with (i) a flexible system of state observation based on database operations and (ii) a way of defining deterministically time-dependent values that can be used to describe parts of the systems or the natural environment that we do not wish to model mechanistically (Informal definition: Chapter 3, Formal definition: Chapter 4).

- An implementation of the abstract version of Chromar as an embedded Domain Specific Language (EDSL) in the general-purpose programming language Haskell. This allows the set of expressions and types for quantifiable properties to come from the host language, which increases expressiveness while at the same time maintaining the naturalness of a domain-specific language (Chapter 5).

- An extension of the Framework Model (FM; §1.2.1) to the full-lifecycle of Arabidopsis Thaliana, including reproduction and see dormancy along with modifications that allow operation in realistic weather ranges. This is combined with full-lifecycle phenology models from ecology to provide a population level model

that includes the organism (through the FM). This allows simulation experiments for determining ecological properties in different genotype x environment scenarios that are mechanistically founded and a step towards organism-centred models of evolution (Chapter 6).

- An application of the Framework Model with an engineering perspective, like in case of crop models. Instead of taking the traditional approach to engineer genomes, as we have seen, we take an alternative approach that aims at engineering the environment for achieving specific growth-related organism traits. We pose the problem as an optimal control problem and frame two instances of it, one where the control is direct and another one where the control is indirect through positioning (Chapter 7).

Chapter 2

Background and Related Work

Formal models of physical systems serve two roles (i) documentation and communication of our understanding and (ii) formal analysis (and/or simulation). The most common language for describing the natural world is dynamical systems theory. It has a long tradition in Physics starting from Newton and Poincare. It relies on a conceptual abstraction where instead of describing the objects that form the physical world, it describes quantifiable properties of such objects. For example, when explaining the movement of planets one does not talk about the moon but rather about the position of the moon. Then the evolution of the system in time (or space) is described by (partial) differential (or difference) equations that state the change in the numerical values of the quantifiable properties that form the system as they interact with each other. For example, one might write the equation of position of a particle or a molecule diffusing in a medium. This abstraction of avoiding dealing with the objects directly but rather with some numerical variables representing their properties has been at the core of the success of dynamical systems theory. A big ensemble of methods have been developed for the analysis of such systems, which one gets for free if they choose to describe a physical system of interest in these terms.

Dynamical systems languages have not been as successful in scenarios where the structure of the described system is not static. While one might be able to find a quantifiable property of the system to describe, in cases where the question posed is about the changing structure itself or about how it gives rise to other dynamics, such an abstraction is not adequate. Consider for example plant development and its effect on carbon intake for the entire plant. While in some cases writing an equation for the size of the plant is enough, a truly mechanistic understanding requires treating the developing structure that gives rise to plant size explicitly. How would one write

equations for the size of the leaves, for example, if new ones keep appearing? In those cases one might be better off with a language that allows the description of the objects themselves, their interactions, and organisation. Unsurprisingly such languages have been developed for scientific fields that deal with parts of the natural world where structures are dynamic and self-organising. Languages of increasing complexity exist for the description of biochemical systems starting from simple molecular reactions. The field of plant development also has seen an independent strand of work on languages with explicit description of objects. Both these strands of work have seen a heavy interaction and inspiration from Computer Science where there exist many formalisms in the theory of computation for the description of object behaviour and interaction.

There are languages in both the dynamics and structure space that combine aspects of both worlds. Chromar, the language we describe and that is a big part of this thesis is situated in this space too. It comes from the biochemistry tradition and has discrete objects at its core but at the same time also has attributes to abstract away some internal structure of these objects. It further has features for (i) linking the abstract (attributes) and the concrete (observables) and (ii) explicitly using time, which is inspired, again, by the dynamics world.

Since language design is as much art as it is science it can sometimes be hard to properly situate a new language (Chromar) in the space of already existing languages for the description of the natural world especially if one is only familiar with one part of the world, perhaps the particular formalisms that are standard for their disciplines. We therefore feel the need to include an overview of existing languages (the ones we hinted to above) for biology in order to be able to place our work in the language design space. Since we have particular requirements for structure representation, we will only deal with languages coming from the discrete world or from the dynamics world that include some support for defining discrete organisation as well.

While there are attempts for formal comparisons between languages (Felleisen, 1991), here the design space is so large that such an attempt is probably impossible. We instead will do our comparison informally through the illustrative examples. We finally conclude with a summary of the comparison. In the next chapter we introduce Chromar through the same examples so we can properly place it and its novel characteristics in the same feature space.

A more comprehensive overview of the above dichotomy between dynamics and structure appears in Walter Fontana and Leo W Buss (1996).

2.1 Examples

In this section we will introduce two examples that we will use throughout our language space overview to illustrate their limitations and the motivation for their extensions with new features. We will not attempt a full representation of the models in each formalism.

This will form as an overview of modelling requirements in the field through some simple examples that illustrate general requirements. This is not meant to be a comprehensive overview of structurally complex biological models. In fact, while the below highlight some important aspects, there is an entire range of models in the plant domain that have much more detailed structure and organisation. Examples are biophysical 3-D representations of meristem tissues at the cell level in Arabidopsis (Gruel et al., 2016; Willis et al., 2016). While these have for example the same requirement for the representation of relations (e.g. between cells), the size and complexity of the system might provide some extra practical challenges that we do not consider here.

2.1.1 Root apical meristem and whole-plant effects

Cell production for plant development happens at sites of meristematic activity. In the root this site is behind the root cap and it is called the Root Apical Meristem (RAM). In root development this raises the question about the mechanism through which this gives rise to the root architecture but also how the meristematic cells maintain their position close to the root cap throughout development. The hormone auxin and its distribution of concentration along the root architecture has been pinpointed as a possible mechanism to explain both of these phenomena.

Here we will consider auxin dynamics in a growing 1-D array of root cells proposed by Mironova et al. (2010) as a possible mechanism to explain the observed distribution of auxin concentration throughout the root. We will further consider the effect of the resulting root architecture on above-ground plant growth through its effect on water intake. We will only consider a very abstract version of the shoot. This model is indicative of the kind of comprehensive models that are the focus of this thesis. Here, for example, a more detailed root model is placed in the context of the whole plant (more abstract model) and the surrounding environment.

More precisely we will consider the following dynamics (Figure 2.1):

- Auxin flow from the shoot. Since we are only considering an abstract shoot we can think of auxin flow from the shoot as auxin being produced in the first cell of

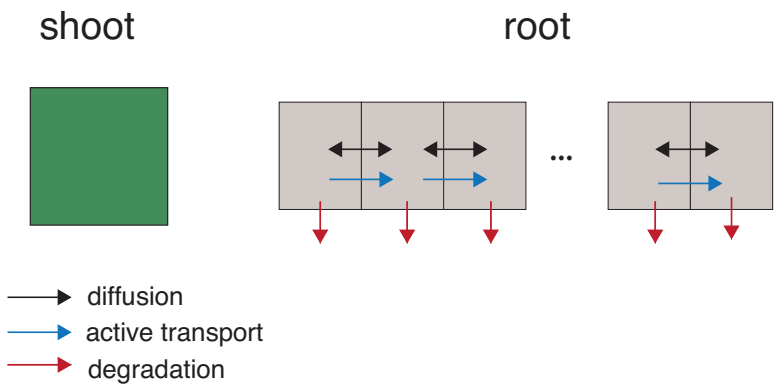


Figure 2.1: Flows of auxin through the root. The root cells are shown as the array of squares with directional auxin flows between them, diffusion is biderectional between cells (black arrows), active transport is towards the root cap (blue arrows) and degradation is flow of auxin outside the system (red arrows).

the root.

- Auxin degradation through cell expansion.
- Auxin diffusion as passive transport between cells driven by concentration differences across the array.
- Auxin active transport by the PIN transporter proteins.
- Cell division.
- Shoot growth, which depends on the water uptake through the root. The uptake depends on the length of the root.

2.1.2 Plant development in a field

The example considers a very abstract view of plant development, but has nevertheless enough details to demonstrate the main features of our notation. Our model is inspired by the Framework Model (FM) of Chew, Wenden et al. (2014), a modular whole-plant model that connects traditional plant biology representations of molecular processes with representations of organ and whole-plant development processes. We will put this model in a more ecologically relevant context (field).

The above-ground part of an Arabidopsis plant architecture before flowering: a collection of leaves arranged in a circle. Each leaf photosynthesises, creating the main currency, carbon; uses some carbon for maintenance and some for growth; and transfers

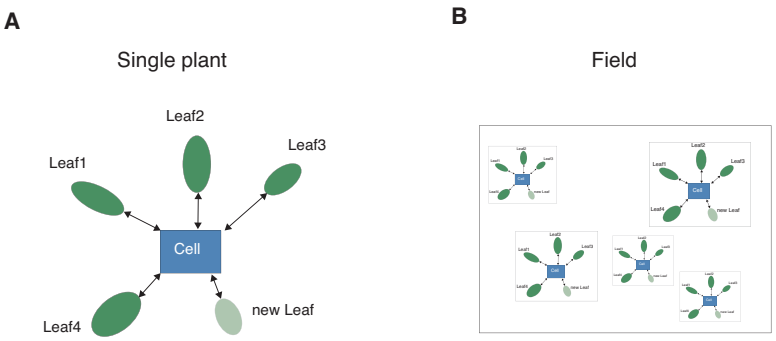


Figure 2.2: A Our simple plant development model. All the interactions, that in this case are transfer of carbon, happen between the central Cell that represents the molecular state of the entire plant and the leaves, which act as carbon sinks. The carbon that goes to the leaves is either used for growth, in which case it is transformed into new material (increase of mass), or to maintain the already existing Leaf by fuelling its life sustaining processes. New leaves are also created, forming new sinks and increasing competition for carbon between the leaves, but also increasing the production of carbon by providing new green areas for photosynthesis. B The simple plant model embedded in a 2-D field.

any remaining carbon to the other leaves. The FM represents the Arabidopsis rosette (collection of leaves) with no preference in the transfer, thus we have an all-to-all communication. Similarly to the FM, in our model all the molecular processes in our model reside in a central plant 'cell' which allows us to keep the leaves as carbon sinks and track their growth, while avoiding the per-leaf molecular processes and their communication (see Figure 2.2).

The processes that affect growth are as follows: we think of *carbon assimilation* per leaf as increasing the carbon concentration of the central Cell depending on the photosynthesis level of a leaf (which will depend on its size); we think of *maintenance respiration* as the central Cell giving some carbon to a leaf; and we think of *growth respiration* as the central Cell giving some carbon to a leaf and the leaf mass increasing. We will also have *new leaf creation*. There are interesting dynamics here such as the interaction between growth and assimilation: the more we grow, the more the leaves can photosynthesise, and the more carbon can go to the central Cell.

We assume that the plants are arranged in a two-dimensional field and that they compete for light, which affects their growth. While we use Arabidopsis here, such models are interesting for the interactions and competition between crops and weeds in the field (Rajcan and Swanton, 2001).

2.2 Object-based languages

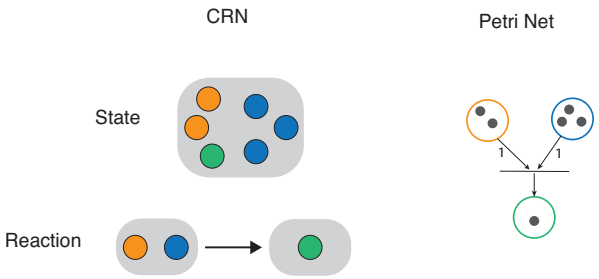
A significant line of work in languages from the discrete (object-based) tradition has stemmed from the world of Biochemistry. A further distinction in this camp is between *rule-based* languages and *process-calculi based* languages that differ in what they consider as their main unit of description – rule-based languages consider the event described as a rule whereas process-based languages consider the individual object/process as the main unit of description. Here we will focus more on languages from the rule-based camp since Chromar follows that tradition. Our focus, however, is not any specific language but rather the features of the languages and the modelling requirements that motivated them. Both camps have undergone similar extensions (starting from simple objects) of features driven by the same modelling requirements so while we focus more on rule-based languages, our observations should apply to the other camp as well, which we will only mention in passing in the following text.

In the following sections we categorise object-based languages based on their features starting from simple objects to structure to dynamics. We will give an overview and focus on languages that combine objects with dynamics since Chromar is situated in this space too. Throughout this text we will use the above examples to illustrate the features and limitations of the languages.

2.2.1 Unstructured: Petri Nets

The simplest language for unstructured collections of simple objects is that of Petri Nets (PNs). At the description level PNs represent discrete objects with types (species) where the dynamics are given as reactions that state how different species interact to change the number of objects of the corresponding species in the state of the system. The state of the system is an unstructured collection (multiset) of copies (molecules) of each species. Qualitatively, applying a reaction to a state (multiset) means removing from the state the elements that appear in the left-hand side (lhs) of the reaction and adding the elements that appear on the right-hand side (rhs) of the reaction.

There is an equivalent graphical representation where the system is given as a bipartite graph representing species as places (round nodes) and reactions as transitions (rectangular nodes). Each place has an associated marking, represented as tokens inside the node, that represents the multiplicity of that species in the state. Applying a transition amounts to moving tokens between places.



A *multiset m* over a set A is a function from A to $\mathbb{N}$ counting the multiplicity of each element $a \in A$ in the multiset. There is submultiset relation on multisets where for two multisets m and m', $m \preceq m'$ if for each element $a \in A$ $m(a) \leq m'(a)$. We write $M[A]$ for the set of all multisets over a set A. Given a set of species Σ, a *reaction* is a structure $\rho = l \xrightarrow{k} r$ where $k \in R$ and both the left-hand and right-hand sides are multisets over the species.

Definition 1. *A* Petri Net *is a pair* (Σ, R) *of sets of species* Σ *and reactions* R.

The state of the Petri Net is a multiset over Σ. A reaction in $\rho \in R$ can be applied to a state s if $l(\rho) \preceq s$ giving a new state $s' = s - l(\rho) + r(\rho)$, which we write as $\rho \bullet s$.

Petri Nets can be given a stochastic interpretation as Continuous Time Markov Chains (CTMCs). A CTMC is a triple (S, Q, I) with S a set of states, $Q : S \times S \to \mathbb{R}$ that gives the transition rate between any two states, and I the initial state. For each reaction apart from the base rate given by k we need to know how many times it can be applied in a give state s (i.e. how many times its left-hand side appears in the state). This is also called a *match*. The multiplicity of a multiset m in another multiset m' (number of matches) is given by:

$$\mu(m, m') = \prod_{a \in m} \binom{m'(a)}{m(a)}$$

Given a PN (Σ, R) and an initial state we can get a CTMC:

$$S = M[\Sigma]$$
$$Q(s, s') = \sum \{k(\rho) \cdot \mu(l(\rho), s) | \rho \in R, r \bullet s = s'\}$$

PNs have been used widely for describing chemical reactions and other simple systems (including many uses in Biology; Hardy and Robillard, 2004). For systems like the ones in our examples they pose some limitations though. Suppose for example that we wanted to write reactions for the diffusion of auxin along the root (see first example §3.2). We could write the following reactions for the auxin molecules in each

cell, where A_1 is an auxin molecule in the first cell, A_2 is an auxin molecule in the second cell, with the diffusion occurring at rate d, and so on:

$$
\begin{aligned}
A_1 &\xrightarrow{d} A_2 \\
A_2 &\xrightarrow{d} A_1 \\
A_2 &\xrightarrow{d} A_3 \\
&\vdots \\
\varnothing &\xrightarrow{\alpha} A_1 \\
&\vdots \\
A_1 &\xrightarrow{\beta} \varnothing \\
&\vdots
\end{aligned}
$$

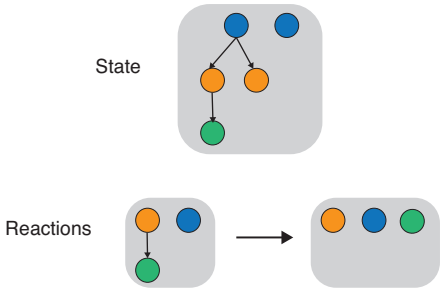
There are two problems with the above description. The first is that it is not very compact. It grows with the number of cells since we have to write the diffusion reaction for every pair of cells in both directions and production/destruction reactions for every cell. The second is that it is hard to see how to describe the creation of new cells because we need to create a new auxin species for the new cell and new reactions, but the notation provides no way to express such a possibility. We could try to make a cell species but then we would have no way of linking the cell objects with the auxin objects and we would have the same problems as above. We also have no way of representing quantifiable properties of these objects, like the size of the cell, for example, needed for an abstract description of growth. Furthermore properties of bigger parts of the state, like the size of the entire root, needed for the water-dependent shoot growth cannot be expressed and neither can parts of the surrounding environment that we do not wish to model in detail like the water in the soil.

2.2.2 Structured

Extensions to simple reactions add support for organisation to the unstructured multisets of objects in PNs. Organisation can be thought of as explicitly adding relations over the collections of objects. There are many relations that could be represented and the ones in this thread are, unsurprisingly, inspired by cell biochemistry.

Hierarchy

The first organisational principle is that of nesting inspired by the compartment organisation inside a cell. Following this cell organisation, languages usually distinguish two kinds of objects – simple ones that cannot be nested and more complex ones that can. P-systems is a language for membrane computing and while it initially considered only simple objects enclosed in a static (potentially multilevel) membrane structure (G. Păun, 2000), in later extensions membranes become first class and are equipped with dynamics (A. Păun, 2001). SMMR (Oury and G. Plotkin, 2013) also makes this distinction between simple objects (species) and complex objects that can be nested (agents). The Calculus of Wrapped Compartments (Coppo, Damiani, Drocco, Grassi and Troina, n.d.) also considers nesting but it also adds notational features for expressing computations that happen on the membrane. A later extension (Coppo, Damiani, Drocco, Grassi, Sciacca et al., 2010) adds names to compartments and they can then be thought of as the agents of SMMR or the membranes of P-systems. The allowed dynamics of these systems reflect the interpretation of the hierarchical relation over collections objects as compartmental organisation inside cells. Therefore sometimes the permitted operations are directly inspired by this view while in other cases they seem more generic. In all cases, however, the additional structure allows expressing more stringent conditions on the left-hand side of rules that select based on structure and type (as possible in PNs).



The nesting relation can also take other interpretations, for example as a ‘part-of’ relation. At a static level this might be reasonable but some allowed dynamics might not be sensible any more as they only make sense for a particular cell-inspired interpretation of the relation (see discussion in Artale et al. (1996)).

Semantics are given, like PNs, with similar interpretations as CTMCs. The states, S, in this case are nested multisets and the transition rates between them are given by appropriate counting of the matches of the nested multisets in rule lhs’s in the state. As states become more complex, however, practically drawing trajectories from the stochastic process becomes harder as finding the number of matches from the lhs of a

rule to the state is more complex than simple multiplication of the multiset multiplicities of the lhs elements.

Going to our examples, it looks possible to represent parts of the root development + auxin dynamics. For example, we could have a Cell_i species per cell to represent root cells and A species to represent auxin molecules. Using nested parentheses to represent the nesting relation we could write the diffusion to the right reaction for the first position of the 1-d array as:

$$\text{Cell}_1(A,x),\text{Cell}_2(y) \rightarrow \text{Cell}_1(x),\text{Cell}_2(y,A)$$

This description has the same problems as the simple PN description in that it is not compact and the creation of new cells requires the creation of new species. If we interpret the nesting relation as the 'next-to' relation we can use a single Cell species and creation of new cells would not require creating new species:

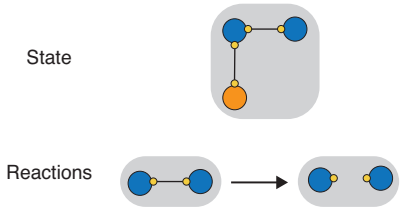
$$\text{State (4 cells): Cell}(\text{Cell}(\text{Cell}(\text{Cell})))$$
$$\text{Division: Cell}(x) \rightarrow \text{Cell}(\text{Cell}(x))$$

Then, however, we would not be able to use the given nesting relation for the actual containment of the auxin molecules. The problem here is that there are two relations, the 'next-to' relation over cells that we need since the communication is only defined for neighbouring cells and a nesting relation between cells and auxin molecules.

The same problems discussed above regarding abstraction of object attributes via quantifiable properties (e.g. cell size), abstractions over parts of the whole state (e.g. size of entire root), and representation of the surrounding environment also appear here.

Links

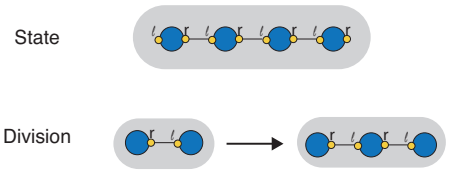
Another organisational principle sometimes considered explicitly is that of complexation inspired by protein complexes. Kappa (Danos and Laneve, 2004) considers the state of the system as a graph, for example:



Reaction (or rules) have the usual qualitative interpretation: anything that 'matches' the lhs in the state can be replaced by the rhs. Kappa also has a stochastic interpretation

as a CTMC where the states are site-graphs. Practically, again, the problem is finding and counting the matches of lhs's of rules into the state, which is even harder in the case of graphs. There are, however, certain properties of Kappa graphs that allow for a more efficient simulation that does not require recomputing the matches after every rule application (Danos, Feret, Walter Fontana and Krivine, 2007).

Using this site-graph approach we can represent the 1-D arrangement of cells from our example:



We, again, have the same problem we noted above regarding properties that cannot easily be represented by discrete objects, like cell properties, observables over states, and time. We note though that the simulator of Kappa, KaSim (Boutillier, Feret et al., 2018), has facilities for observation of states, which can be used inside rules.

Other formalisms allow for more than one relation, for example bigraphs (Milner, 1999) and the more biologically relevant stochastic bigraphs, which have a stochastic interpretation as CTMCs (Krivine et al., 2008). Using this we could represent both the neighbour relation between cells and the nesting of auxin molecules inside cells. The rest of the limitations we noted persist, however, since it is an object-only language.

2.2.3 Combining structure and dynamics

Going beyond the object world, other extensions to these rule-based languages try to combine traditional features of object-based formalisms with dynamical features to represent more abstract quantifiable properties of the objects. It is interesting to note that adding attributes allows one to implicitly represent relations based on attribute selections on the left-hand side of rules (see discussion in §2.2.3 and next chapter).

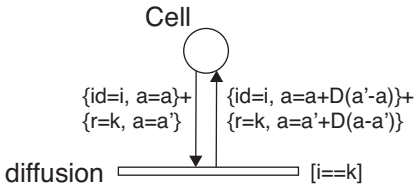
Coloured Petri Nets

As the name implies Coloured Petri Nets are an extension to Petri Nets (§2.2.1) that allows distinctions between objects of each species (colouring of objects) by allowing them to have an associated data value adhering to the type (colour set) of their species (Jensen, 1987). For example, for our root development system instead of having a simple Cell species we can have a Cell species with associated parameters (e.g. position

in the array). Qualitatively CPNs work in the same way as simple PNs. Applying a reaction moves tokens between places. Since we have more complex type of objects the transitions have inscriptions to bind object colour values to variables names so that they can be used in the lhs (out-arrows) of the transition. Note that the same dynamical description is used for the dynamics of objects and their attributes. In the case of object-based dynamics (adding, removing objects) conceptually things are the same as before. For changing attributes we simply remove an object and replace with its attribute changed.

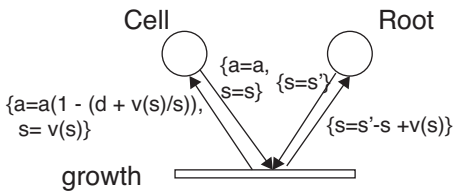
A stochastic version of this CPN formulation has also been used for biological modelling before, for example for describing planar cell polarity in Drosophila wings (Gao et al., 2013) (and see Gilbert et al. (2013) and Runge (2004) for other examples). In these examples where the stochastic version was used, its semantics are just given as a translation to the corresponding simple Petri Net. CPNs are used in their graphical format and most tools therefore use a graphical interface.

Going to our root development example we can use colours to represent the 'next-to' relation between cells, as well as the concentration of auxin and other abstract quantifiable properties like cell size. To represent *diffusion* we employ the Cell colourset $\{\text{id} : \text{int}, \text{l} : \text{int}, \text{a} : \text{int}\}$. Apart from the number of auxin molecules, a, we also have positional information. In order to determine the position of a cell in the array we use its identifier colour id and the identifiers of its left neighbour, l colour, and right neighbour r colour.



This is an example of an implicit representation of a relation between objects where the relation information is stored in the objects themselves. This is in contrast to the explicit representation of relations that we have seen in the languages with representation of structure (hierarchy and links, see previous sections). While it is more flexible, it places more burden on the user to maintain this relation.

To represent *growth* we add another colour, s : real to our Cell colourset and another colourset for the entire root (Root) so that we can store its size (s)



Growth also degrades auxin concentration in the cell. Here we have used an additional colourset to represent global information as the language does not offer the possibility of explicitly defining the functional correspondence between the size colours of the Cell tokens and the size of the entire root. This means that the user has to manually keep track of this function and update the global information.

For both of the transitions we consider we used some colours with real types. Since the semantics of CPNs are given with translations to simple PNs via enumeration of the corresponding simple transitions, the unfolded simple Petri Net has in many cases an infinite number of reactions. This means that in order to do the unfolding at all, the types have to be finite sets, which further means that real values are not allowed. This is also reflected in the implementation of Coloured Petri Net tools where one can define a Stochastic Coloured Petri Net but the definition is unfolded before it is run (Heiner et al., 2012). Petri Net tools further have a graphical interface for defining the models, although there is a hybrid approach that allows mixing the graphical definition with programming language constructs (ML language) Jensen, 1987. While graphical notations are intuitive for smaller models, we find that for larger models they become hard to read.

Finally, following the root development example there is no way to easily represent the water availability in the surrounding soil.

Coloured Stochastic Multilevel Multiset Rewriting (CSMMR)

CSMMR (Oury and G. D. Plotkin, 2011) is an extension to SMMR (Oury and G. Plotkin, 2013) that adds parameters to agents, motivated by the limitations we have noted in the object-only languages that we have considered (§2.2.2 and §2.2.1). Parameters can take part in rules either passively by influencing rates rules, conditions but can also be actively changed. The addition of parameters gives it a very similar flavour to CPNs. However, a CSMMR model, unlike a CPN model, is given stochastic interpretation directly as a CTMC, without unfolding to a simpler form.

Using a CSMMR agent type similar to the CPN Cell colourset for the root development example (§3.2), we can write the following for the diffusion (to the right) of auxin

along the root:

$$\text{Cell}_{\text{id}=i,\text{a}=a}, \text{Cell}_{r=k,a=a'} \rightarrow \text{Cell}_{\text{id}=i,\text{a}=a+D(a'-a)}, \text{Cell}_{r=k,a=a+D(a-a')} \; [i=k]$$

This is very similar to the CPN transition of the same dynamics but written in a completely textual form.

Similar limitations to the CPNs (discussed above) regarding time dependent values (e.g. for representing water in the soil) and observational abstraction over parts of the state (e.g. size of the entire root) remain. The availability of richer types for parameters would also work well with a more expressive expression language to represent their dynamics perhaps utilising a general-purpose programming language, like CPN tools do with ML.

Dynamical Grammars

Dynamical Grammars of Mjolsness and Yosiphon (2006) have a truly hybrid approach where the quantifiable properties of objects can be defined by differential equations as in dynamical systems theory.

$$\{\tau_i[x_{i,1},\ldots,x_{i,k[\tau_i]}]\} \rightarrow \{\tau_i[x_{i,1},\ldots,x_i,k[\tau_i]]\}$$
$$\text{solving } \{\frac{dx_{i,j}}{dt} = f_{i,j}(t)\}$$

Everything else looks similar to other languages in this section, dynamics are given as rules where each side of the rule is a multiset of discrete objects with typed sequence of attributes. The interesting part comes from the solving clause that can be used to express the dynamics of the attributes as differential equations.

Going to our root development example, the diffusion can be expressed in a way that is more familiar to the traditional diffusion models,

$$\{c_i = \text{Cell}(i,a_i), c_{i+1} = \text{Cell}(r,a_{i+1})\} \rightarrow \{c_i, c_{i+1}\}$$
$$\text{solving } \{\frac{da_i}{dt} = D(a_{i+1}-a_i), \frac{da_{i+1}}{dt} = D(a_i - a_{i+1}\}$$

where we use the same Cell agent type as before.

Having the differential equations means we can also express time-dependent values, for example to represent the change of moisture in the soil surrounding the root in an abstract (non-mechanistic) way.

$$\{\text{Shoot(s)}, \text{Env}(w), \text{Root(s)} \rightarrow \{\text{Shoot(g(s))}, \text{Env}(w), \text{Root(s)}\}$$
$$\text{solving } \{\frac{dw}{dt} = f(w,t)\}$$

However, if we are, as it is usually the case, taking the water values from a table then it might be hard to find a mathematical description that fits the differential equation framework. Mixing these two paradigms (object-based and differential equations for the dynamics), while powerful, means that sometimes rules do not correspond naturally to the conceptual view of biological events as instantaneous flows of matter.

We note that there is no possibility for expressing observational abstractions over parts of the state and a similar scheme to the one we used in the CPN formulation, with an extra Root type object will have to be used.

2.2.4 Plant development: L-systems

Another line of work started in the plant development tradition. They followed roughly the same evolution of starting with simple objects and then adding parameters for combining quantifiable properties with objects. Traditional object-only L-systems are a string-rewriting formalism where strings of characters (representing the objects) can be rewritten via rules that specify how characters can be replaced by other character (or sequences of characters). The lhs of the rules can be context-sensitive by specifying the characters surrounding the character intended for replacement:

$$a < b > c \rightarrow a < d > c$$

For example, the above specifies that an object b can be replaced by an object d in the string state of the system in places where it is surrounded by a on the left and c on the right. Unlike the object languages we have seen before that rely on multisets, L-systems rely on strings, which are ordered. Adding parameters work in the same way as the transition from PNs to CPN, for example, and it was motivated by the same modelling requirements.

Unlike the PN family from the Biochemistry tradition, L-systems are not usually interpreted as CTMCs. They instead rely on discrete time and multiple parallel application of rules in cases where more than one is applicable.

The ordered parametric L-systems seem to work well for our root development example since the cells along the root are in an ordered sequence. The 'next-to' relation then needs no further machinery (implicit representation) to be represented. The diffusion rule will look like this, again using a Cell object with similar parameters as before:

$$\text{Cell}(a_i) < \text{Cell}(a_{i+1}) > \rightarrow \text{Cell}(a_i + D(a_{i+1} - a_i) < \text{Cell}(a_{i+1} + D(a_i - a_{i+1}) >$$

L-systems might not be as good in cases where the structure of the system is not linear or cannot (easily) be linearised as a string. Take, for example, the dynamics from our plant development example (§3.1). The leaves in that case and the "cell" are not linearly related. It would therefore be hard to write, for example, the rule for leaf growth that requires information from both the leaves and cell objects.

Similar limitations regarding observational abstraction of parts of the state, time-dependent value representation also apply here.

2.3 Dynamics to structure

In the previous section we focused on languages that are object-based and followed their evolution through adding features like adding extra structure on the objects and then adding features that are more traditionally from the dynamics world. Here, instead, we focus on languages that have dynamics at their heart but have some features for representing (dynamic) structure as well.

Modelica is an object-oriented language (or language standard) combining traditional dynamics representation with structure (Fritzson and Engelson, 1998) with various implementations (W. Li et al., 2007; Otter et al., 1996). Dynamics are given as either non-causal relations between variables or differential equations. While it is object-based, the objects are intended for handling model complexity instead of representation of domain entities. This is also reflected by the fact that the object organisation is static. Later extensions (Nytsch-Geusen et al., 2005) or inspired languages (Zimmer, 2008) add dynamic structure. The module dynamics are given by boolean triggered transitions between model modes representing different versions of the model with possibly different structure. This, again, reflects the fact that modules are intended for structuring a model instead of representation and they seem very far off from conceptual views of biological systems especially if one is talking about the dynamics of populations of objects. If we wanted, for example, to represent the root example with cells as Modelica (+ dynamic structure) modules we would need one mode for each population size. It is also very hard to see how to represent relations between objects that affect the structure changes.

System dynamics is another popular approach to capturing continuous dynamics of variables through the visual notation of stocks and flows. Stocks are state variables and dynamics are given as flows between them. Recent commercial projects integrating a modelling language based on systems dynamics with an integrated development

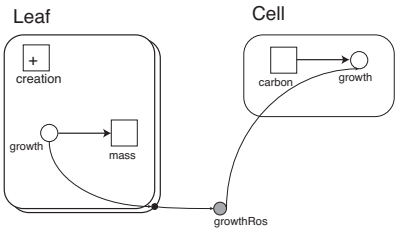
environment have started adding some discrete features as well to combine dynamics with object-based representations (Ventity, AnyLogic; Borshchev and Filippov, 2004; Yeager and Fiddaman, 2014). These have some characteristics in common with the Modelica family of languages but the object dynamics are first-class and are represented through some language abstraction that allows their definition. We next focus in more detail on one of these, Simile, which has been used in biology before.

2.3.1 Simile

Simile is another graphical language that has similarities to our approach (Muetzelfeldt and Massheder, 2003). Simile is used mainly in the domains of ecology and agricultural sciences but has also been used in systems biology (for the whole-plant model (Chew, Wenden et al., 2014) that was the inspiration for our example in §3.1) to exactly solve the kind of limitations we noted in the object-only based languages. External inputs are also supported, which is particularly important for adding weather data to crop models. In Simile there are two levels of definition of a model. At the first level we have continuous variables with rate equations and at the second level we have discrete objects with discrete dynamics – adding/removing. The objects are grouped based on their types, and their behaviour is given at the population level. The dynamics of the two types of entities, continuous variables and objects, are not integrated as is the case in CPNs and CSMMR.

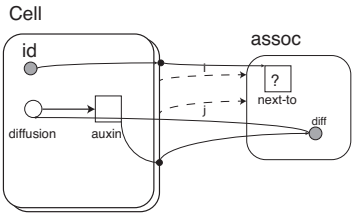
There is an obvious interpretation of the first-level continuous variables as a systems of differential equations but there is no obvious interpretation as a formal mathematical object for the entirety of the Simile features including the object-level dynamics. At the implementation level a model generates a system of Ordinary Differential Equations (ODEs) and then keeps track of the objects. Any time there is a structural change to the model, the system of ODEs is recalculated and integrated numerically until the next point of structural change (e.g. new object in population).

In order to illustrate how Simile features capture the modelling requirements from our examples, let us start with the plant development example (§3.1),



We have a population of Leaf objects and a single Cell object representing what we called Cell in our rules. Each Leaf in the population has a mass that grows as a continuous variable. In order to define the use of carbon for growth from the carbon variable in the Cell object we have to work at the population level by summing the contribution of each Leaf. The population of Leaf objects also grows (see creation box). This concise description is at the level of objects instead of at the level of events, like in rule-based languages, which means that the dynamics are distributed across the model. Having the dynamics at the rule instead of the at the object level sometimes seems more natural as they correspond more intuitively to the conceptual view of biological events. Even with mental models of processes, one usually speaks about the dynamics of the system as interactions of components instead of the behaviour of each component individually.

Object relations can also be encoded through conditions to restrict the dynamics to a subset of the object types, similarly to conditional expressions available in CPNs and CSMMR. For example, the diffusion dynamics that happen only between neighbouring cells can be encoded.



Here the diffusion dynamics are restricted to neighbouring cells through the association submodel, assoc, and a relevant condition (see ‘next-to’ box).

Representing cell division is more challenging as the dynamics of objects can only be represented at the population level. In the case of cell division the object ‘creation’ is not identical for all cells since the position matters for keeping the next-to relation information (identifiers) updated.

Finally, like CPNs, the graphical notation of Simile becomes, in our experience, problematic for models with more than a few variables. While CPNs have graphical elements for species and transitions, Simile represents all variables and interactions graphically further adding to the complexity of the representation. This could, however, be mitigated by model hierarchy. Since the representation of objects in Simile, which could correspond to entire models, encourages model composition, further visual tools that allow hiding unnecessary nested models could be used to handle model complexity.

2.4 Pragmatic approaches

When modellers are confronted with complexity in models like the ones we have seen in the Introduction or the example ones from this chapter, they sometimes turn to more pragmatic approaches when they cannot map their understanding into any suitable technical language.

Most of the pragmatic approaches are based on using an ad-hoc representation of the model as a program in a general-purpose programming language. We next give an overview of these methods with references to the examples as before.

2.4.1 Models as programs

By far the most popular approach when confronted with the limitations of either object-based or dynamics-based languages we noted above, is to use a custom simulation software. In this approach the domain knowledge is represented as a program and it is mixed with the simulation code, especially if one is writing a one-off program. The programs are not usually mapped or interpreted as mathematical objects. Models like the Framework Model (Chew, Wenden et al., 2014) and the whole-cell of Karr et al. (2012), which are examples of the kinds of the comprehensive multi-scale models we are interested in, are written in this way, for example (both given as Matlab programs).

Turning to our examples, in order to write the root development model one could write something like the following in a general-purpose programming language:

```
type Mode = Idle | Growing
type Cell = {s, a: real, m: Mode}
type Shoot = {s : real}
type State = {root : [Cell], shoot : Shoot}

function simulate(s : State, n:Int) -> [State]
  ...
  for i = 1 to n
    s' = step s
    ...

function step(s: State) -> State
  s' = (diff . growth . cellDiv . ...) s
  return s'
```

where we assume we are in a programming language that allows the definition of compound data types used to represent the information about cells and the shoot. We first decide on a representation for the state of the system where we use a list to represent the array of cells in the root since lists are ordered. Then in order to simulate the system we apply the `step` function `n` times to some initial state. The `step` function applies functions that represent all the biological processes that change the systems of the system, diffusion, cell growth and division and so on.

One is faced not only with a decision on representation but also with a decision of a simulation algorithm to follow, since the model is not mapped to a well-defined technical language with existing implementation. Since general-purpose programming languages are inherently sequential, for example, one has to find a reasonable serialisation of the execution of the processes that does not affect the results. Karr et al. (2012) deal with this by using an appropriate time-step and in each time-step randomising the execution of the processes.

This approach gives greater flexibility as one can use any existing programming language constructs, either built-in or through external libraries, for the construction of the model. At the same time, unless one spends a great deal of effort, the resulting representation is hard to read in the first place and hard to maintain or extend in the future. In contrast, the rule-based representations we have seen above are inherently declarative and concurrent and extending a model is just a matter of concatenating more

rules to the existing ones.

2.4.1.1 Multi-model simulation environments

To deal with the problem of extension many models are written in a modular way such that related components are packaged together. Each component is written in the most convenient way. The problem comes in their combination. Sometimes models are written in different programming languages, with different input/output formats, different time-steps, units and other incompatibilities. What happens in practice is that models are usually reimplemented in one language to create a multi-model written in a single language. This is the approach taken, for example, in the Framework Model where all constituent models are reimplemented in Matlab along with other compatibility changes, likes time resolution changes.

There has been some work on multi-model simulators that can deal with heterogeneous models written in possibly different languages or have other incompatibilities. The idea is that the user declares their intentions on how the modules should interact and the rest, like interaction, conversion and simulation, are handled by the framework. The modular integration of processes in the Karr et al. (2012) model inspired the *mois* tool (RBM group, 2015) that uses various techniques, like variable time-steps and backtracking (Bucher et al., 2013), for the simulation of the interacting modules. Another tool (Lang, 2018) was developed as part of the Plants in Silico initiative (Zhu, J. P. Lynch et al., 2016), a collaborative effort on multi-scale plant and crop modelling.

2.4.2 Simulation frameworks

Going a step further, instead of the user having to write custom simulation software, some frameworks provide libraries that expose common functionality for the construction or combination of models.

Agent-based frameworks

In Agent-based modelling (ABM) simulation frameworks, the description of the process/model happens declaratively through the description of the behaviour of classes of agents (see for example Solovyev et al., 2010 used in systems biology).

While initially this looks similar to the object-based formalisms we have seen above, in ABM systems the main unit of description is the entire behaviour of each individual agent, whereas in rule-based formalisms it is the rule, which can both describe

a (possibly partial) behaviour of an individual agent and a synchronised action of two or more agents. This ability to specify synchronisation often leads to more natural descriptions and, more practically, makes the resulting models easier to change and combine. The advantages of being in a programming language remain.

2.4.3 Crop Simulation frameworks

Modelling has historically been used much more extensively in crop science than in plant biology. Several large models incorporating significant environmental components (from soil to ecosystem) have been developed. These have been increasingly developed in a modular fashion providing the user with functionality for common modelling scenarios, like calculation of water uptake or light interception by the canopy (Keating et al., 2003). In order to increase re-usability and decrease user effort, several frameworks have been developed on top of crop simulation software that offer a set of parameterised components that can be used to model different scenarios or even different crops (Brown et al., 2014).

This modular approach of crop models has found its way in complex and comprehensive biology models, like the Karr et al. (2012) whole-cell model and the Framework Model (Chew, Wenden et al., 2014), which were built in a modular fashion with exactly the same motivation – re-usability and extensibility. Writing these models requires familiarity the programming language of the framework but, again, the advantages of a general-purpose programming language remain.

2.4.4 Model interchange formats (SBML)

The growth of the systems biology movement led to the development of many tools for the representation and analysis of models of molecular processes. Most of the models can be represented by all dealt with similar type of processes of genetic networks and were represented similarly as systems of ODE's in the continuous form or Petri Nets in a discrete form. The Systems Biology Markup Language (SBML; Hucka et al., 2003) was developed as an interchange format between the different tools for these kinds of processes and had support for the representation of the most commonly used constructs in these models, reactions, compartments along with model information, like units and so on.

As models started becoming more complex it faced the same limitations we noted in the Introduction and in the Petri Net section regarding dynamic organisation among

others. This led to the development of multiple extensions to the original format (SBML L3; *SBML Level 3 specification* n.d.) to handle the increased complexity. For example, the 'Dynamic Structures' package adds support for defining dynamic structures and the 'Hierarchical Model Composition' package adds support for hierarchy representation. The L3 specification was put to the test for the representation of the Karr et al. (2012) whole-cell model (Waltemath et al., 2016).

SBML models are only used as a description and interchange language and they do not have any interpretation as mathematical objects. Using them directly for modelling is probably impractical although there are some graphical tools that allow the construction of SBML models (Hoops et al., 2006).

Chapter 3

Chromar by example

In this chapter I will introduce Chromar (formally introduced in the next chapter) by means of examples. The example serve two purposes: to introduce the features of the languages but also as a means of comparing Chromar to the related languages in the previous chapter by seeing how Chromar features handle the modelling requirements that the examples present. I finish with an overview of how Chromar fits in the Systems Biology workflow and practical considerations including analysis methods.

Chromar has the following main ideas, which we will explore in this chapter through the examples:

- *basic Chromar* is a rule-based notation with stochastic semantics yielding a Continuous Time Markov Chain (CTMC). It follows the object-based tradition (see previous section) and makes use of discrete objects called *agents*. These have attributes that are defined at the type level, so that every agent of that type has these attributes. For a version of our auxin example a Cell agent type could be, similarly to other parametric languages, $\mathrm{Cell}(\mathrm{pos} : \mathrm{int}, \mathrm{a} : \mathrm{real})$ with attribute pos for positional information and attribute a to keep its auxin concentration. Agents are instantiations of this type with concrete values for the attributes like $\mathrm{Cell}(\mathrm{pos} = 1, \mathrm{a} = a_1)$ for the first cell, $\mathrm{Cell}(\mathrm{pos} = 2, \mathrm{a} = a_2)$ for the second cell and so on. The states of the CTMC are multisets of agents. The rules describe how agents are added to or removed from states at a more abstract level than individual agents. This is done by using patterns on the rule left-hand sides specifying the group of agents for which the dynamics are defined. As we have seen, having parameters help us overcome some of the limitations of object-only languages.

- *extended Chromar* extends basic Chromar with two new features:

(i) *Fluents* — the incorporation of deterministically changing time-dependent values. These are important for modelling dynamics in a changing environment, and

(ii) *Observables* — values calculated from a global view of the system. These are important in cases where a coarse-grained view of the system is needed. This may be because we cannot acquire atomistic data, or because we do not wish to model everything at the same level of detail. Observables also give us a flexible way to observe the state of the system that can be used to report the results of model simulations, as we often need time series of some observable on the state of the system rather than time series of the state itself.

- A concrete realisation of extended Chromar as an embedded Domain Specific Language (DSL) inside Haskell, a functional programming language (Gibbons, 2015). The embedding means that we can use any valid Haskell expression where expressions are expected, for example in the rate expressions and in the right-hand sides of rules. Agent types and the outside environment (defining rate, condition function etc.) are defined directly as Haskell expressions but rules, fluents, and observables are defined using abstract Chromar syntax via quotation (Mainland, 2007).

The examples from the previous section are extensions to the examples that appear in the articles describing Chromar (Honorato-Zimmer, Andrew J Millar et al., 2017; Honorato-Zimmer, Andrew J. Millar et al., 2018). Therefore the following text also in part comes from the same sources. Note that the author order follows the traditional alphabetical order of computer science literature. Ricardo Honorato-Zimmer participated in the meetings/discussions of ideas with Prof Gordon Plotkin who was my primary supervisor for this part of the work while Prof Andrew Millar was secondary. Ricardo Honorato-Zimmer also contributed some code in the implementation of multiset operations in the simulator of Chromar.

3.1 Plant development in a field

3.1.1 Single plant

Since agents are the main entities in our language, let us consider what types of agents we should have to model the above system, given the above discussion. We will need:

- a Leaf type with attributes for appearance: $\text{Leaf}(\text{age} : \text{int}, \text{mass} : \text{real})$,
- a Cell type that represents our main plant 'cell' with an attribute, carbon, to keep the current carbon level: $\text{Cell}(\text{carbon} : \text{real})$ (there will only be one agent of this type at any one point)
- a Ros type that represents the entire Rosette with an attribute, n, to keep the current number of leaves: $\text{Ros}(\text{n} : \text{int})$ (there will also only be one agent pf this type at any one point)

We will use the current number of leaves relative to the index of a particular leaf as a proxy for the leaf's age: the larger the difference between the index of a leaf and the current number of leaves, the older that leaf is.

For the *carbon assimilation* from one particular leaf we need to increase the carbon concentration of the central Cell. The bigger the leaf the faster it contributes to the production of carbon:

$$\text{Leaf}(\text{mass} = m), \text{Cell}(\text{carbon} = c) \xrightarrow{f(m)} \text{Leaf}(\text{mass} = m), \text{Cell}(\text{carbon} = c + 1)$$

We can read this rule as saying that for any two Leaf and Cell agents, the Leaf agent remains the same and the Cell agent increases its carbon content by one. Note that we assign the values of the attributes the left-hand side of the rule to the variables m and c, so that we can refer to them in the right-hand side of the rule and the rate expression. Since the age is not used in the rest of the rule, it can be omitted (see *Missing attributes* syntactic extension, §4.1.1). If we were to write this in a traditional reaction notation we would have to write a reaction for every possible Leaf agent which leads to the compactness problem we have noted earlier (indeed, there would be infinitely many such possibilities). The implicit 'for-all' in the pattern on the left-hand side allows the rule to be applied to new leaves when they are created. For example if the state of the system has two leaves, the central cell, and the rosette agent:

$$\{|\text{Leaf}(\text{age} = 1, \text{mass} = 10), \text{Leaf}(\text{age} = 2, \text{mass} = 5), \text{Cell}(\text{carbon} = 20), \text{Ros}(\text{n} = 2)|\}$$

then the rule is applicable to the two substates

$$\{|\text{Leaf}(\text{age}=1,\text{mass}=10),\text{Cell}(\text{carbon}=20)|\} \text{ and}$$
$$\{|\text{Leaf}(\text{age}=2,\text{mass}=5),\text{Cell}(\text{carbon}=20)|\}$$

For *maintenance respiration*, the central Cell agent gives some carbon to a Leaf agent, with the amount of carbon needed for maintenance depending on the size of the Leaf:

$$\text{Leaf}(\text{mass}=m),\ \text{Cell}(\text{carbon}=c) \xrightarrow{h(m)}$$
$$\text{Leaf}(\text{mass}=m),\ \text{Cell}(\text{carbon}=c-g(m))\ [c \geq g(m)]$$

Note the use of the condition $c \geq g(m)$ to make sure that the carbon levels do not become negative. Also note that here we use both the rate ($h(m)$) and the increment size ($g(m)$) to control the amount by which carbon is updated in a time unit. Since the rate controls the number of times the update will happen, the product of the rate and update functions is the expected amount that will be removed from the carbon pool in a time unit.

Next, *leaf growth* depends on the mass of the leaf, its age (there is a limit on how much a leaf can grow so older leaves stop growing at some point), and the amount of carbon available:

$$\text{Ros}(\text{n}=n),\text{Leaf}(\text{age}=i,\text{mass}=m),\ \text{Cell}(\text{carbon}=c) \xrightarrow{h(n-i,m,c)}$$
$$\text{Ros}(\text{n}=n),\text{Leaf}(\text{age}=i,\text{mass}=m+1),\ \text{Cell}(\text{carbon}=c-1)\ [c \geq 1]$$

The mass of the leaf is also needed in the calculation to make sure that the growth does not exceed observed physical constraints. The growth is capped to a fraction of the current mass of the leaf.

Finally, for *leaf creation* we have:

$$\text{Ros}(\text{n}=n) \xrightarrow{k} \text{Ros}(\text{n}=n+1),\ \text{Leaf}(\text{age}=n+1,\text{mass}=m_0)$$

Fluents and Observables

There are two problems with the above definition:

(i) There is no way to include the environment in the model, and so it is assumed constant. This makes the model very detached from reality. For example, our plants photosynthesise all the time, whereas in reality they only do so during daylight, but we have no direct way to switch between day and night.

(ii) We have two representations of the same process at two different levels of abstraction that have to be kept consistent with each other by the user. Specifically, the n attribute in the Ros agent keeps track of the total number of leaves in the plant. However, there is no way in the language to specify the global information the attribute is tracking, and check that its value is indeed what it should be.

I next introduce two new notational features to solve these problems.

Fluents

To solve the first problem I introduced time-dependent values called *Fluents*. These can be constructed through a combination of a small set of primitives and general expressions, taken from Reactive Programming (FRP) constructs in (Wan and Paul Hudak, 2000) (Fluents are usually called Behaviours in FRP). For example we could write a fluent for light, which is a function from time to the Booleans, and another one for temperature that depends on light:

$$light = \textbf{repeatEvery}\ 24.0\ (\text{True}\ \textbf{when}\ (6 < \textbf{time} < 18)\ \textbf{else}\ \text{False})$$
$$temp = 21.0\ \textbf{when}\ light\ \textbf{else}\ 16.0$$

where the **when**...**else** construct denotes conditional behaviour and **repeatEvery** cycling behaviour. Cycling behaviour is achieved with the modulo operation so the value of **repeatEvery** t_o f at t is the value of f at $t\ mod\ t_0$.

The assimilation rule can then be written as follows:

$$\text{Leaf}(\text{age} = i, \text{mass} = m),\ \text{Cell}(\text{carbon} = c) \xrightarrow{f(m,temp)}$$
$$\text{Leaf}(\text{age} = i, \text{mass} = m),\ \text{Cell}(\text{carbon} = c + 1)\ [light]$$

Very often in biology we have empirical relationships of various quantities with time. Fluents can be used to encode these as well. Such empirical relationships do not provide any mechanistic insight but they can be useful when we either do not have data or do not want to model more mechanistically using rules.

Observables

To solve the second problem, I introduced functions on the state of the system called *observables*. Observables are constructed using a combination of database-inspired **select** and **aggregate** operations: think of our state as a kind of database where we have a collection of agents rather than the more usual (and very similar) collection of records. The **select** operation selects agents of a particular type from part of the state, producing a binding of each such agent's attribute values; the **aggregate** operation then folds the

resulting collection of attribute value bindings into a single value, using a specified combining function and initial value.

For example, to calculate the total mass of the leaves in a state, we could write:

$$lm = \textbf{select}\,\text{Leaf}(\text{age} = i, \text{mass} = m);\ \textbf{aggregate}\ (total : \text{int}.\, total + m)\ 0$$

where we first select every agent that matches a Leaf pattern, producing a binding of its age and mass attributes to i and m, respectively, and then, starting from 0, calculate the result using a combining function that adds the value of m in every such binding to the running total. If we further wished to calculate the average mass of all the leafs in a state, we could use the observable

$$nl = \textbf{select}\,\text{Leaf}(\text{age} = i, \text{mass} = m);\ \textbf{aggregate}\ (count : \text{int}.\, count + 1)\ 0$$

to count the number of leafs in a state, and then divide lm by nl.

Whenever an observable is used, one obtains its 'fresh' value, even when the underlying state has changed (e.g., by creating new leaves, in this case). For example the *leaf growth* rule now becomes:

$$\begin{aligned} &\text{Leaf}(\text{age} = i, \text{mass} = m),\ \text{Cell}(\text{carbon} = c) \xrightarrow{h(nl-i,m,c)} \\ &\text{Leaf}(\text{age} = i, \text{mass} = m+1),\ \text{Cell}(\text{carbon} = c-1)\ [c \geq 1] \end{aligned}$$

where we use our observable nl directly in the rule rate. There is no need to use the Ros agent any more as its only purpose was to keep track of the number of leaves.

Note that observables, like fluents, can be used to model parts of the system in cases where there is either not enough knowledge about a process, or else no desire to model at a more mechanistic level. In this case, for example, the leaves must have some mechanism for knowing their age. However this is not central to our model, so we instead use global knowledge via observables to abstract away from the details that would be involved in modelling that mechanism.

The two features, fluents and observables, can be mixed arbitrarily along with ordinary expressions. We might for example have a fluent inside an observable or a fluent inside an observable and so on. For example we could write:

$$f(m)\ \textbf{when}\ nl > 10\ \textbf{else}\ f'(m)$$

where nl is an observable for the number of leaves. This might be used to introduce the crowding effect on rosette leaves; crowding affects the assimilation rate as it reduces the photosynthetically active area.

3.1.2 Field

Going to the field where we have multiple plants we need some way of knowing first which agents belong to the same plant and second the location of the plant. We do not have an explicit way of representing these relations but we can again use our attributes to encode them. I will use one identifier for the plant and one for the patch of land. Every agent now has a further attribute to store this information. For example, the Leaf agent type becomes: $\text{Leaf}(\text{pos} : (\text{int}, \text{int}), \text{age} : \text{int}, \text{mass} : \text{real})$. The first element of the pos tuple is the plant identifier and the second is the identifier of the patch it belongs to. I further assume that we are in a suitably powerful programming language (e.g. our Haskell embedding) that I have (or can define) the functions pid to select the first element of the tuple (plant id) and cid to select the second element of the tuple (patch id).

The metabolic rules stay the same but add a further condition that the agents on the lhs belong to the same plant. For example, maintenance respiration now becomes:

$$\begin{aligned}&\text{Leaf}(\text{pos} = p, \text{mass} = m),\ \text{Cell}(\text{pos} = p', \text{carbon} = c) \xrightarrow{h(m)} \\ &\text{Leaf}(\text{mass} = m),\ \text{Cell}(\text{carbon} = c - g(m))\ [c \geq g(m) \wedge pid\ p = pid\ p']\end{aligned}$$

We assume that competition between plant affects the amount of light they receive, which in turn affects the photosynthesis rate. The assimilation rule will therefore change so that the light fluent is modulated by a competition index, which will be an observable on the global state of leaves in an entire patch. Here we assume that we are in our Haskell embedding and we have access to the full power of the language. We define the following Haskell functions:

$$\begin{aligned}&\text{comp}\ p = f(\textbf{select}\ \text{Leaf}(\text{pos} = p', \text{mass} = m)\ ;\ \textbf{aggregate}\ (c : \text{real}.\text{sumComp})\ 0 \\ &\text{light}\ p = \textbf{repeatEvery}\ 24.0\ (\text{light}_{\text{max}} \cdot \text{comp}\ p\ \textbf{when}\ (6 < \textbf{time} < 18)\ \textbf{else}\ 0)\end{aligned}$$

that take as input a pos tuple and return a relevant expression for that particular plant in that patch. The combining function sumComp is defined as: $\text{sumComp} = \textbf{if}\ cid\ p' = cid\ p \wedge pid\ p' \neq pid\ p\ \textbf{then}\ c + m;\ \textbf{else}\ c$ that sums the masses of all leaves in a patch except from the given plant. Then with the modulated light definition the assimilation rules will change to:

$$\begin{aligned}&\text{Leaf}(\text{pos} = p, \text{age} = i, \text{mass} = m),\ \text{Cell}(\text{carbon} = c) \xrightarrow{f(m, temp, light\ p)} \\ &\text{Leaf}(\text{age} = i, \text{mass} = m),\ \text{Cell}(\text{carbon} = c + 1)\ [light\ p > 0]\end{aligned}$$

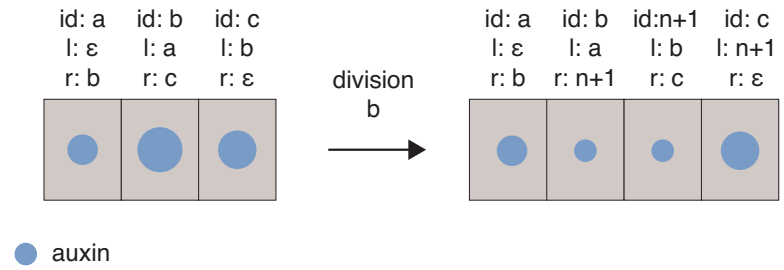


Figure 3.1: Scheme for capturing the 'next-to' relation between cells and the division rule. Each cells is represented by a square and the auxin concentration by the size of the blue circle inside. Positional information is stored using identifiers, each cell stores its own identifer and the identifiers of its left (l attribute) and right (r attribute) neighbours. After a division event a new cell is created and on the left of the mother cell, which is given a new cell. Positional information is changed across the affected cells to reflect this change.

3.2 Root development

For the root example, we again assume we are in our Haskell implementation and we therefore have Haskell types and expressions in our disposal for the definition of the model. We define the following agent types:

- a Cell type to represent the root cells in the array defined as:

$$\begin{aligned}\text{Cell}(&\text{id} : \text{int},\\ &\text{l},\text{r} : \text{link},\\ &\text{s},\text{a},\text{d} : \text{real},\\ &\text{m} : \text{mode})\end{aligned}$$

  The type label link is interpreted as the set $\mathbb{N} \cup \{\varepsilon\}$ and the relevant attributes (l for the left and r for the right neighbour) represent either a bound state (via an identifier) or a non-bound state (ε). The type label mode is interpreted as the set $\{1,2\}$ and it represents the state of the cell, either growing (1) or idle (2). The other attributes represent the size of the cell (s) and its auxin (a) and division-factor (d) concentrations (Figure 3.1).

- a Shoot agent type to represent the entire shoot of the plant defined as $\text{Shoot}(\text{s} : \text{real})$ where the attribute s represents the size of the shoot.

In order to be able to give fresh identifiers to cells created by division we also need to know the current number of cells in the array. We do this using an observable:

$$nc = \textbf{select}\ \text{Cell}(); \textbf{aggregate}\ (count : \text{int}.\, count + 1)\ 0$$

As with our plant example in the previous section, we use a count expression that increases the running count by 1 for each Cell. Note that like rules we can omit attribute bindings when the values are not used in the subsequent definitions of the expression.

Turning to the dynamics of the system, for the auxin flow from the shoot we can have the following rule:

$$\text{Cell}(\text{l}=\varepsilon, a = a) \rightarrow \text{Cell}(\text{a} = a + (a_{init} + \frac{0.17 \cdot \textbf{time}}{t_{cc}}))$$

where t_{cc} is the time duration of the cell cycle. We represent auxin flow from the shoot as auxin being produced in the first cell in the array. We distinguish the first cell by requiring it to be unbound to the left.

For the auxin diffusion we can write the following:

$$\text{Cell}(\text{r}=i, \text{a}=a), \text{Cell}(\text{id}=i, \text{a}=a') \rightarrow \text{Cell}(\text{a}=a + D(a' - a)), \text{Cell}(\text{a}=a' + D(a - a'))$$

For the active transport of auxin in the cell by PIN proteins we have,

$$\begin{aligned} &\text{Cell}(\text{r}=i, \text{a}=a), \text{Cell}(\text{id}=i, \text{a}=a') \rightarrow \\ &\text{Cell}(\text{a}=a - K_0 aPIN(a)), \text{Cell}(\text{a}=a' + K_0 aPIN(a)) \end{aligned}$$

Note that instead of modelling PIN concentration and its effect on auxin directly, we do so implicitly through a function that modulates the amount of auxin being transported. Unlike diffusion, which is bidirectional, active transport takes place in the direction from shoot to root cap.

The production of the division factor is affected by the difference in auxin concentration between neighbouring cells,

$$\text{Cell}(\text{id}=i, \text{a}=a'), \text{Cell}(\text{l}=i, \text{a}=a, \text{d}=d) \rightarrow \text{Cell}(), \text{Cell}(\text{d}=d + \sigma(a' - a))$$

For the growth of the cell and degradation of the concentrations of the division factor, d, and auxin concentration, a, we write:

$$\text{Cell}(\text{s}=s, \text{a}=a, \text{d}=d) \rightarrow \text{Cell}(\text{a}=a(1 - d + \frac{v(s)}{s}), d = d(1 - k(a) + \frac{v(s)}{s}), \text{s}=v(s))$$

The degradation of the division factor, again, depends on the auxin concentration in the cell.

For the cell cycle we need to switch the mode of the cell, m, from growing to idle.

$$\mathrm{Cell}(\mathrm{m}=1,\mathrm{s}=s) \xrightarrow{\rho(s)} \mathrm{Cell}(\mathrm{m}=2)$$

with

$$\rho(s) = \frac{1}{1+e^{-\frac{s-s_{\min}}{\mathbf{time}}}}$$

For cell division, we create a new cell on the right of the dividing cell and split the concentrations of auxin and the dividing factor between the two cells:

$$\begin{aligned}
&\mathrm{Cell}(\mathrm{id}=i,\mathrm{r}=k,\mathrm{a}=a,\mathrm{d}=d,\mathrm{s}=s,\mathrm{m}=2),\mathrm{Cell}(\mathrm{id}=k,\mathrm{l}=i) \xrightarrow{\rho(d)}\\
&\mathrm{Cell}(\mathrm{id}=i,\mathrm{id}=nc,\mathrm{a}=a/2,\mathrm{d}=d/2,\mathrm{s}=s/2,m=1),\\
&\mathrm{Cell}(\mathrm{id}=nc,\mathrm{l}=i,\mathrm{r}=k,\mathrm{a}=a/2,\mathrm{d}=d/2,\mathrm{s}=s/2,m=1),\\
&\mathrm{Cell}(\mathrm{id}=k,\mathrm{l}=nc)
\end{aligned}$$

The cell divides only in the idle phase and the division sets the mode back to growing while the rate of division depends on the concentration of the division factor. The dividing cell gets pushed to the left, keeping its id and changing its right-neighbour identifier to the identifier of the new cell. The new cell gets a fresh identifier from our cell counter observable *nc* and a right-neighbour identifier from the old neighbour of its mother cell (Figure 3.1).

Finally, cells die at the root cap:

$$\mathrm{Cell}(\mathrm{r}=\varepsilon) \xrightarrow{\rho_{\mathrm{death}}} \varnothing$$

We distinguish the end of the root by requiring the cell to be unbound on the right.

Finally, the growth of the shoot follows an abstract functional form:

$$\mathrm{Shoot}(\mathrm{s}=s) \rightarrow \mathrm{Shoot}(\mathrm{s}=f(w,\mathbf{time}))$$

The growth function is a phenomenological function of time and more interestingly water uptake. Assuming that the water uptake depends on the root size, we model it as follows:

$$\begin{aligned}
&rs = \mathbf{select}\ \mathrm{Cell}(\mathrm{s}=s);\ \mathbf{aggregate}\ (total : \mathrm{real}.total+s)\ 0\\
&w = f(\mathbf{time})\ \mathbf{when}\ rs > s_0\ \mathbf{else}\ f(\mathbf{time})\cdot d
\end{aligned}$$

where *rs* is an observable abstracting the size of the entire root from the size of the constituent cells and *w* is a fluent that uses that size to compute the water uptake. The bigger the root the more water the plant takes. We use the fluent here to model a process that while relevant for our process we do not wish to model in detail and mechanistically via rules.

3.3 Applicability of Chromar

In the above section we have explored some examples to showcase the main theoretical ideas of Chromar. All of these examples come from the plant biology world but that is a product of the environment where this thesis was undertaken. A detailed investigation of the applicability is a significant undertaking and beyond the scope of this thesis but I ouline below certain areas where it is not hard to imagine Chromar models being applicable.

In systems theory, as was discussed in Chapter 1, there is a stress on the organisation side of living things. In this view there is the assumption then that we can identify unities, that is objects with well-defined boundaries that form this organisation through their relations (see, for example, Section 9 in Varela et al., 1974). In Chromar discrete entities, agents, are the main entities in the language so writing systems models of many kinds should be quite natural.

Starting from Biology, developmental biology is a field where the systems view was recognised perhaps because there you have cells, which are the most recognised discrete unities in the biological world. In development models there are usually two levels of description, the molecular level creating a field over a possibly changing (e.g. due to growth) topology of cells. We have seen a simple example of this with the root example above but one could easily imagine 'recipes' for writing the most common dynamics at both the molecular and tissue levels. For example, starting with a $\mathrm{Cell}(\mathrm{c} : \mathrm{Real})$ agent for a cell with some concentration of chemical c, diffusion along with other molecular rules (production, degradation, reactions) can be easily written. At the tissue level, cell division should also be possible with some mechanism of defining the neighbour relation. Beyond that when going into tissue mechanics that give rise to shape (morphodynamics, see for example for plants; Formosa-Jordan et al., 2018) then we could have more agent types to represent cell to cell interactions and possible rearrangements (Farhadifar et al., 2007; A. G. Fletcher et al., 2014). The computational requirements are then higher, which might make the current Chromar implementation not suitable, but the description of the model in Chromar even without simulation should still be very useful as a specification of the understanding about a system.

Ecological models is another domain where Chromar will perhaps seem natural to practitioners. It is easy to imagine population models and we see an example of that in Chapter 6. Interaction between population of different species is just a matter of having more than one agent type. Interaction with the environment, which is usually important

in this context, can be encoded with fluents as we do above for water availability for the root and more extensively to encode weather condition in different locations in the whole life Arabidopsis models in Chapter 6. Populations with structure of some kind (e.g. spatial like the topologies of tissues from above) can also be encoded given a suitable representation of relations (as we did above for an array of cells for example). Other quantities (i.e. not discrete objects – agents) can be mixed as well by encoding as agent attributes or fluents. If we consider population models as a special case of evolutionary models (Doebeli et al., 2017) then these could be encoded as well. Again computationally with, for example, performing large population simulations at the individual level might not be preferred, but a description of the rules in Chromar could still be valuable as documentation of our understanding.

Beyond biology, again, we see formal representations with similar characterestics – interacting discrete elements with perhaps some underlying relation between them (e.g. topology) – in fields like the social sciences where they are usually called agent-based models (Abar et al., 2017; Bonabeau, 2002). Some of the most well-studied models of social interaction can be studied with simple relations on the set of agents (e.g. segregation dynamics on a regular lattice Schelling, 1971). As we have noted above and elsewhere in this thesis in Chromar the burden is on the modeller to find and maintain a suitable representation for relations between agents. This might make it harder to encode models on more complex topologies that are sometimes common in social science representations (e.g. social networks; Celli et al., 2010; Rodriguez et al., 2016).

3.4 Comparison to related work

Having seen the properties of Chromar in action in this chapter and of other languages in the same design space in the previous chapter, we can compare their features. Here I will discuss properties relating to general usability features that can be understood without the formal definitions. Comparison and discussion related to more technical parts is done at the point where these are introduced (see next Chapters for formal definition of Chromar and its embedding in Haskell).

3.4.1 Declarative modelling

A distinction exists in programming languages between imperative and declarative styles. In the imperative style a computer program consists of a series of instruction of

how the described process should be computed. On the other hand in more declarative languages the programmer defines what the program should do but not how that should be done. Mainstream programming languages follow the imperative style, which has prevailed, perhaps since computer hardware is inherently imperative (Backus, 2007).

A similar distinction appears in modelling where some models use a declarative style while others are more imperative mixing the what with the how. The usual mathematical languages used in the description of physical processes, like differential equations, for example, are declarative statements about how functions of time (and/or space) on the described quantities relate to the rates of change of these functions. On the other hand, as we have seen, many models are represented as programs in imperative languages, which means that the represented biological knowledge is coupled with how the process should be simulated along with other statements relating to particular choices for the representation of the state of the system (Models as programs §2.4). While libraries or frameworks provide ways to separate to an extent the biology from the implementation of the simulation (see §2.4.2), this is not guaranteed.

If we accept the view that models should serve as both tools for simulation and as tools for thought for our understanding then procedural representations are not adequate. Chromar follows the declarative style. Rules correspond to biological events and the user declares how the state changes during the event. The simulation need not be defined as the interpretation as a CTMC gives that without any extra effort. Here I gave a particular implementation of the language as an embedded language inside Haskell using the Haskell expressions as an expression language. While Haskell follows the declarative style, certain properties of the biology might still be obscured depending on how much of the model lives in the rules and how much in the expressions implementing changes to the agent attributes. Different choices can easily be made though, for example we could restrict the expressions to small set of mathematical expressions (perhaps a subset of MathML Ausbrooks et al., 2003) if the increased power of a general-purpose programming language is not required.

Another common advantage of the declarative style over a more procedural style is model composability. This exists in Chromar too and it is crucial for the more comprehensive multi-models that we are interested in this work. In Chromar composing two models is just a matter of concatenating their rule sets. An example of this is the multi-model lifecycle model of Arabidopsis I present in Chapter 6. This is not as straightforward in procedural model implementations where the whole simulation and representation would have to be altered.

3.4.2 Basic Chromar

The idea of extending simple objects with fields to represent some of their attributes has been used before, as we have seen, for example in Coloured Petri Nets (Jensen, 1987), in a rule-based setting, in CSMMR (Oury and G. D. Plotkin, 2011) and Dynamical Grammars (Mjolsness and Yosiphon, 2006), and in parametric L-systems. Our notation is inspired by these.

The correspondence of CPNs to basic Chromar is straightforward, colour sets are our agent types (records with named attributes), tokens are our agents, and transitions are our rules. CPN transitions also have predicates that are the same as our conditions. One can think of it as a minimal rule-based (and textual) version of stochastic coloured petri nets, where the richer types are first class and not merely a means of translation to a non-coloured version; one can also think of it as a simpler version of CSMMR with only colours left. While graphical notations are intuitive for smaller models, we have found that for larger models they become hard to read whereas text-based approaches like our language produce much more readable representations.

Like the development of other languages we have seen in the previous chapter, basic Chromar provides an extension to the representation of reactions, where the simple named species are upgraded to agents employing rich types, namely records with named typed attributes. Writing rules using these richer types yields a more compact representation than one would get by writing reactions on the simpler types in the traditional reaction setting. Moreover it sometimes helps with writing systems that are difficult to write otherwise as it permits the dynamic creation of agents and therefore their attributes. These are the limitations we noted for the simple reactions in PNs.

Attributes can be used to encode relations like the ‘next-to’ relation in the root development example. However, whenever we use them to encode binding we would probably benefit from using a language that represents these directly such as Kappa (Danos, Feret, Walter Fontana, Harmer et al., 2008) or BioNetGen (Blinov et al., 2004). In general Chromar rule left-hand sides are simple and can only select using types, which means the only relations we can encode directly are products of types (we can think of types as sets containing all the agents of that type). This is often unproblematic, for example in our Plant system (§3.1) *all* Leaf agents interact with *all* Cell (only one in this case) agents so writing the left-hand at the type level works because the rules are then applicable to exactly the pairs of agents we want, $\{(\text{Leaf}_1, \text{Cell}), (\text{Leaf}_2, \text{Cell}), \dots\}$.

In other cases though, the relation we want is some subset of the product of the

types. For example in our array of cells example the diffusion rule is not applicable to all pairs of Cells so writing $\text{Cell}(\dots), \text{Cell}(\dots)$ on the left-hand side gives more pairs than we want. In that case we had to encode the relation through identifiers and the next-to relation pairs were stored inside our agents. The selection was then restricted using attribute conditional expressions. While it works in this simple example, it might become problematic in large models when multiple relations are present, for example.

However, note that in the dividing cell and diffusion model of the previous section we use colours in other ways that cannot be easily represented as binding, or any relation offered by other object-only languages. In particular, the division of the contents of a cell would be hard to express in any of the object-only languages.

3.4.3 Extended Chromar

The use of database inspired operations for the observables is also new and I have found it very useful in model building. The declarative nature of the multiset query primitives makes the definition of observables very intuitive. Similar database-inspired query operations on top of collections are used in LINQ (Budiu et al., 2013) although the collections are usually taken to be lists not multisets. Buneman's comprehension syntax (Buneman et al., 1994), a collection query language similar to practical database query languages, considers other types of collections, including multisets. These are, however, not used in the modelling world.

Observables could be developed further and made into first-class entities in the language, for example by making them attributes of types. For instance in the plant growth example (§3.1 we would keep our initial Ros type and use the observable primitives to define its n attribute instead of defining the observables outside the agents. First-class observables would work particularly well with an extension for a native representation of levels (the 'nested-in' relation we noted earlier), in which case the idea of the agent attributes at a higher level being observables of agents at a lower-level would be both intuitive and powerful.

Fluents allow the convenient expression of changing values, which is important as models become more comprehensive and start reaching outside their domain, for example traditional plant biology models reaching out to the outside world and to crop science.

Fluents can be expressed, albeit in the ODE formulation, in Dynamical grammars. Simulators of some of the languages in the overview in the previous chapter also allow

observables for result reporting. However, the lifting of fluents and observables to first-class entities and their combination with expression coming (potentially) from a general purpose programming language is new and makes for a very powerful extension to the language.

3.4.4 Haskell

My embedding in a programming language increases expressive power, and fits with the availability of rich types. There have been other languages that, while not giving the full power of a programming language, still allow for complex expressions to appear, e.g., inside rate expressions. For example in React(C) (John et al., 2011) rate expressions can be build from a subset of a functional programming language with a reflection option to get access to the full state of the system. This allows conditions to be encoded inside rates by setting the rates to zero if they are not met. Simulators for other widely used languages like KaSim (the simulator for Kappa (Danos, Feret, Walter Fontana, Harmer et al., 2008)) and BioNetGen (Blinov et al., 2004) also allow more sophisticated rate expressions than traditional mass-action kinetic rates.

My embedding in Haskell lifts some constraints of modelling languages and, I think, gets the best of both worlds: it naturally and succinctly captures some elements in our domain of interest but, at the same time, when greater expressive power is needed we can turn to the programming language. This increase in expressive power might come at the expense of the ability to do general analysis of models since we cannot say much about what is happening in the Haskell `exprs` inside the rules. There seems to be a trend, though, in the direction of mixing domain-specificity and general purpose programming languages, for example Pedersen et al. (2015) allow embedded F# scripts inside LBS-κ, and in PySB (Lopez et al., 2013) Kappa models are defined inside Python. Embedding a domain-specific language inside a programming language, as we did, is in some cases better than doing the opposite – embedding a programming language inside a domain-specific language – because we have fewer constraints on our definitions and more generally full access to the language for structuring our model definitions. I also note, as we have seen, that there is a hybrid approach that allows mixing the graphical definition of CPNs with programming language constructs (ML language; Jensen, 1987).

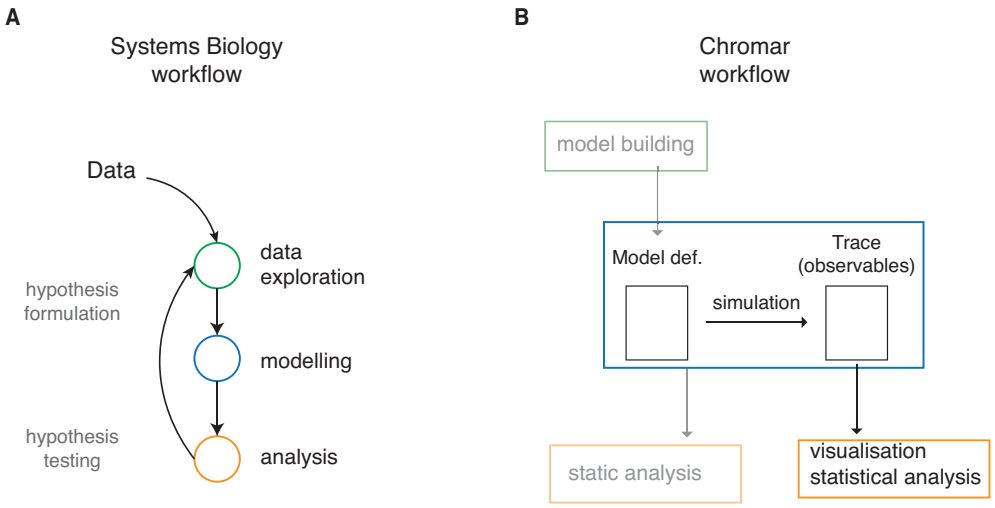


Figure 3.2: A The workflow in systems biology. Hypotheses are formed from data (data exploration) that are then encoded formally into models (modelling). Models can then be used as instruments to test these hypotheses (analysis). The new knowledge can then feed back into more experimental work than can lead to new hypotheses. B The workflow for building models in Chromar. A model definition can be simulated to produce traces, which can then be analysed (e.g. through visualisation – dynamics analysis). Analysis on the model itself (static analysis) could also become available (e.g. using the connection to formal framework of Continuous Time Markov Chains) but it is not currently available in the current realisation of the language (shown in muted colour). Methods for building the model itself, while theoretically possible, are also limited by practical considerations (again shown in muted colour).

3.5 Chromar in practise

Models are formalised representations of our hypotheses that can be easily tested to prove or disprove these hypotheses (Gunawardena, 2014). While models are the main instruments in the knowledge acquisition process in systems biology, modelling is not the only step in the process. If models are formalised hypotheses then there must be a step where these hypotheses are formed based on evidence (data exploration) and a later step where they are tested (analysis) with reference to observations. Models are not set in stone and as our hypotheses change they change as well with the availability of new data or from understanding from previous models (Figure 3.2).

The importance of the choice of language becomes even more apparent in this context: since models represent our knowledge and are easily falsifiable they need to be easy to read and write but the choice of language also determines the range of tools

available to the modeller for forming and analysing hypotheses. For this reason many modelling languages are usually packaged in frameworks with language-specific tools for the entire workflow (Boutillier, Maasha et al., 2018; Calzone et al., 2006; Pradal et al., 2008).

Chromar was primarily built with the modelling step in mind and to address specific requirements in the modelling process. However, other language-agnostic (e.g. simulation-based) tools for model building and analysis should be applicable. For example system parameter identification (and more generally model selection) is a much studied problem (inverse problem of going from data to model; Engl et al., 2009). Many of the available techniques rely on simulations to score sets of parameters in the optimisation process (Moles et al., 2003; Toni and Stumpf, 2009).

More specifically since a Chromar model has a CTMC intepretation, static and dynamic analysis methods on CTMCs should also be applicable for Chromar models. We define, for example, the simulation of a Chromar model using the Stochastic Simulation algorithm (SSA) for drawing trajectories from a CTMC (see Chapter 4). Other analyses methods of CTMCs are also available (e.g. approximation; Schnoerr et al., 2017). Probabilistic (or stochastic) model checking is a particularly powerful one (Kwiatkowska et al., 2007). A model checker takes as inputs the description of a CTMC model in some form and a specification of properties to be checked written in an appropriate temporal logic Aziz et al., 2000; Haverkort et al., 2003 and returns the (probability) that a formula is satisfied. In a biological setting, for example, a formula can query the relation between the concentration of different species in relation to time (e.g. what is the probability that the concentration of species A is greater than that of species B between t_1 and t_2?). As most of the biological inspired languages have intepretations as CTMCs, probabilistic model checking has been used in Biology before to query the behaviour of models for analysis (Brim et al., 2013; Heath et al., 2008). It has also been used for model parameter identification using formula probabilities as objectives (Barnat et al., 2012) and for coarsening the state-space of a model (Michaelides et al., 2016).

The only concrete realisation of Chromar is through the Haskell embedding (see Chapter 5). Through the use of Haskell meta-programming machinery, one can use Chromar abstract syntax for rules and enriched expression but the model is in the end a Haskell program. In order to run the model the Haskell program is compiled and then run producing a trace of the defined observables (Figure 3.3). The types of the attributes of the Chromar agents are fixed to be the set of Haskell types. Both the

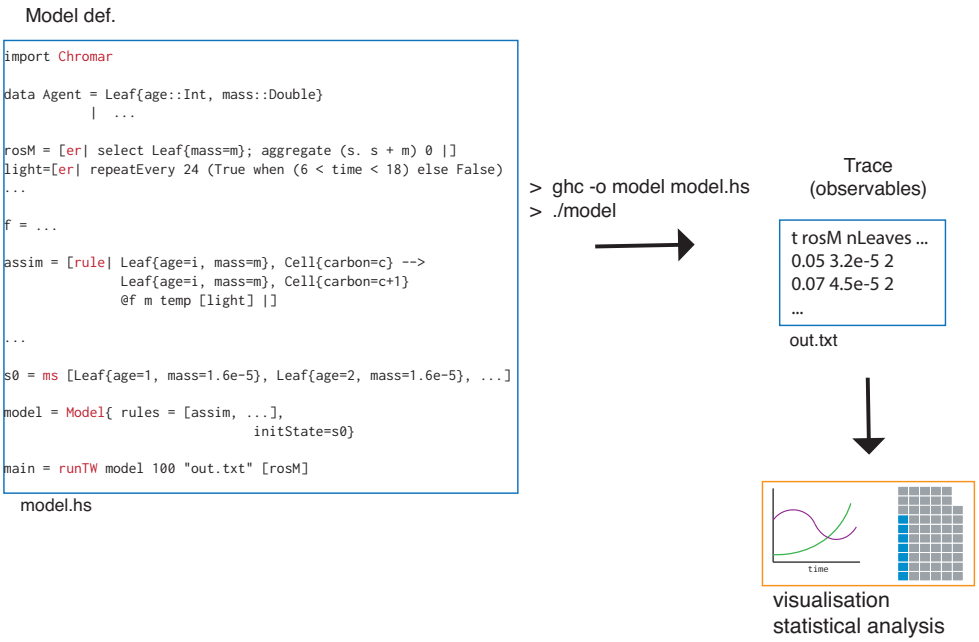


Figure 3.3: The workflow for the current realisation of Chromar as an embedded Domain Specific Language inside Haskell. Haskell's metaprogramming facilities are leveraged to allow domain specific syntax but the model is still a Haskell program. The language features, including the metaprogramming features (quoter functions) are imported as a library and are shown in red in contrast to the rest of the program shown in regular font. To run the model, the program is compiled and then run to produce output traces of defined observables. These can then be analysed perhaps through visualisation or other techniques.

implementation via embedding and the choice of of types for the agent attributes, while practically convenient, preclude some of the theoretically available analysis methods I have outlined above. Practically since any model is a Haskell program there is no way to export or manipulate it into some format that would be suitable for CTMC model checking or other tools (including language-agnostic ones). Furthermore while the choice of Haskell types gives increased expressivity to the modeller, it also means that the spate-space of the model is infinite since Haskell types can (and generally will) have infinite values. This does not affect the SSA since we compute the state rates dynamically but CTMC analysis methods including model checking usually use state-space representations are therefore restricted to finite state-spaces (see for example the PRISM tool; Kwiatkowska et al., 2011).

These restrictions are tied to the current realisation of Chromar. Other implementations of Chromar that are consistent with the formal semantics (Chapter 4) with a

different choice of available types should also be possible. Given a suitable choice of types (e.g. restricted to finite sets) and a realisation where the model itself can be programmatically manipulated could allow an enumeration of concrete reactions from the model rules or the full definition of the Q-matrix of the underlying CTMC. Both of these could serve as an interface to existing tools for the analysis of Chemical Reaction Networks or CTMCs both for querying the model and using identification methods (e.g. for model parameters). It should be noted, however, that even with finite types the state space would generally still be significantly larger than that of a standard Chemical Reaction Network (CRN) due to the combinatorics of different agent types. Unlike CRNs it is not possible to keep a number per agent type in the state.

The existence of fluents further complicates the relation to formal framework of CTMCs since they make the underlying chains time-inhomogeneous. However, this an optional feature of the language and in a particular realisation of the language with the above discussed characteristics (i.e. simpler types, expressions), fluents can be exlcuded as well to ease analysis. The formal definition of Chromar allows flexibility in these choices defining a family of languages. Different choices can made that place each concrete language somewhere in the convenience versus amenability to analysis spectrum.

It should be possible to develop language specific techniques similar to the suite of techniques available for another rule-based language, Kappa, for example (Boutillier, Camporesi et al., 2018; Camporesi et al., 2017). Apart from static analysis, a query language exists for dynamics analysis of the simulation traces that is tailored to the Kappa language primitives (Laurent et al., 2018). An analogous query language for dynamic analysis of Chromar traces would be very powerful and very practically useful. While not providing any static guarantees, it should be easy to use as it is similar to current informal analysis of traces. Furthermore, it should not suffer from restrictions that other formal methods impose on the state-space, for example as we have seen above, which means greater flexibility in realisations of the language.

3.6 Conclusion

In conclusion, extended Chromar and its concrete realisation in Haskell give a highly expressive language that is particularly good for the description of models with a simple structure, such as the Plant example in §3.1 and the root development example in §3.2. The main ideas of Chromar — two levels of dynamics (attribute and agent levels),

a flexible system of observation, and a combination of domain-specific and general programming languages — should carry to other frameworks or languages, for example ones with connection and nesting and perhaps other complex structure.

Chapter 4

Chromar: formal definition

In the previous chapter I gave some examples illustrating how Chromar works. In this chapter I formally introduce the syntax and semantics of Chromar. Formal definition of programming languages is a common practice in the computer science world. It has the many of the same advantages over informal, natural language descriptions as we have noted earlier regarding formal against informal descriptions of biological models. The formal description acts as a tool for the language designer to understand and design the language by making design choices explicit and understanding the relation between different features of the language. This usually also then makes the implementation process easier while the language is not tied to that implementation.

The work in this chapter appears in the articles describing Chromar (Honorato-Zimmer, Andrew J Millar et al., 2017; Honorato-Zimmer, Andrew J. Millar et al., 2018). Note that the author order follows the traditional alphabetical order of computer science literature. Ricardo Honorato-Zimmer participated in the meetings/discussions of ideas with Prof Gordon Plotkin who was my primary supervisor for this part of the work while Prof Andrew Millar was secondary. Ricardo Honorato-Zimmer also contributed some code in the implementation of multisets in the Haskell implementation of Chromar.

4.1 Abstract syntax of basic Chromar

I next define the abstract syntax of basic Chromar.

Regarding notation, we write $\underline{a}$ for a possibly empty sequence of objects, whose typical elements are given by mathematical expressions a. We write $\underline{a}.i$ for the ith element of the sequence and $|\underline{a}|$ for its length, and write the empty sequence as ε and use commas for the composition of sequences. We assume we have a countable set

of names N ranged over by n_a, n_f, id where n_a will be used for agent names, n_f for attribute names, and id for variables.

For generality we assume that basic Chromar is parameterised by a set E of *expressions* (an expression language) ranged over by e, e_r, e_c, and including variables, together with a set of *base types* $B \supseteq \{\text{bool}, \text{real}\}$ ranged over by b; base types are used to specify agent attributes (we use e_c to indicate condition expressions and e_r to indicate rate expressions). We further have *agent types* $\underline{n_f : b}$ that are sequences of distinct typed attribute names.

We may say that basic Chromar is *abstract* in that no choice is made of an expression language and its types; the same holds of extended Chromar (§4.3.1). In contrast, in a *concrete* realisation of the language a particular choice is made. For example, the Haskell embedding of extended Chromar (§4.3) provides such a concrete realisation, where the expressions and types are the Haskell expressions and types.

A *base type environment* $\Gamma = \underline{id : b}$ is a sequence of variable type bindings, and an *agent type environment* $\Delta = \underline{id : at}$ is a sequence of agent type bindings; we require that the variables in environments are distinct. We write $\Gamma[\Gamma']$ for the new base type environment produced by extending or overriding Γ with new type bindings Γ', and define $\Delta[\Delta']$ similarly.

Turning to typing, as regards the expression language we assume we have typing axioms of the form $\Gamma \vdash e : b$ that assign unique base types to expressions in some base type environments, with the following being the only axiom for variables:

$$\Gamma \vdash id : b \quad (id : b \in \Gamma)$$

The syntax and typing rules for the rest of basic Chromar are given in Figure 4.1. I next go through the various syntactic constructs and explain their syntax and typing rules:

Agent declarations

Agent declarations a_d define the sequence of typed attributes that all agents with some name n_a should have (agent decl, Figure 4.1). Using the model of the previous section as an example we had a Leaf type with attributes mass of type real and age of type int, that could be declared with:

$$\textbf{agent}\ \text{Leaf}(\text{mass} : \text{real}, \text{age} : \text{int})$$

We use the sequence of typed attributes $\underline{n_f : b}$ as the *agent type*. The typing rule T-INTRO produces a new such type binding associating the agent name to the agent type.

	Syntax		*Typing rules*
$a_d ::=$	$\textbf{agent}\ n_a(\underline{n_f : b})$	agent decl	$\vdash \textbf{agent}\ n_a(\underline{n_f : b}) : (n_a : \underline{n_f : b})$ (T-INTRO) (all n_fs distinct)
$a_l ::=$	$n_a(\underline{n_f = id})$	left agent expr	$\Delta \vdash n_a(\underline{n_f = id}) : (\underline{id : b})$ (T-L-AGENT) $(n_a : (\underline{n_f : b}) \in \Delta$, all ids distinct)
$a_r ::=$	$n_a(\underline{n_f = e})$	right agent expr	$\dfrac{\Gamma \vdash \underline{e : b}}{\Gamma \mid \Delta \vdash n_a(\underline{n_f = e})}$ (T-R-AGENT) $(n_a : (\underline{n_f : b}) \in \Delta)$
$r ::=$	$\underline{a_l} \xrightarrow{e_r} \underline{a_r}\ [e_c]$	rule expr	$\dfrac{\Delta \vdash \underline{a_l} : \Gamma' \quad \Gamma[\Gamma'] \mid \Delta \vdash \underline{a_r} \quad \Gamma[\Gamma'] \vdash e_r : \text{real} \quad \Gamma[\Gamma'] \vdash e_c : \text{bool}}{\Gamma \mid \Delta \vdash \underline{a_l} \xrightarrow{e_r} \underline{a_r}\ [e_c]}$ (T-RULE)
$m ::=$	$\underline{a_d}; \textbf{init}(\underline{a_r}); \underline{r}$	model def	$\dfrac{\vdash \underline{a_d} : \Delta \quad \Gamma \mid \Delta \vdash \underline{a_r} \quad \Gamma \mid \Delta \vdash \underline{r}}{\Gamma \vdash \underline{a_d}; \textbf{init}(\underline{a_r}); \underline{r}}$ (T-MODEL)

Figure 4.1: Abstract syntax of Chromar with corresponding typing rules for each syntactic construct

The typing for models uses the judgment $\vdash \underline{a_d} : \Delta$ for sequences of agent declarations. This conjoins the agent environments given by the individual agent declarations, provided that no agent name is declared twice. It can be given (inductively on the length of the sequence) by the following axiom and rule:

$$\vdash \varepsilon : \varepsilon \qquad \frac{\vdash a_d : (n_a : \underline{n_f : b}) \quad \vdash \underline{a'_d} : \Delta}{\vdash a_d \underline{a'_d} : (n_a : \underline{n_f : b})\Delta} \quad (n_a \text{ not bound by } \Delta)$$

Agent expressions

Agent expressions have a name and a finite sequence of *attribute expressions* of the form $n = e$. (agent exprs, Figure 4.1). Agent expressions have a dual role: they are

used in the left-hand side of rules as a way of binding values to variables and on the right-hand side of rules as a way of expressing the change in the left-hand side agents. We therefore have two distinct syntactic classes, one for each use:

- left agent expressions, a_l, where all attribute expressions are of the form $n_f = id$, binding the value of an n_f to a variable id, and

- right agent expressions, a_r, where all attribute expressions are of the form $n_f = e$ where the expression e gives the new value of the attribute n_f.

The typing rule T-L-AGENT introduces new type bindings in the environment for the variables in the left agent expressions according to the agent type. Specifically if we have an agent type in the environment $n_a : (n_1 : b_1, \ldots, n_k : b_k)$ and a left agent expression $n_a(n_1 = id_1, \ldots, n_k = id_k)$, each attribute expression $n_i = id_i$ of the the left agent expression produces a type binding, $id_i : b_i$ according to the agent type; this results in bindings for the entire expression $(id_1 : b_1, \ldots, id_k : b_k)$. In Figure 4.1 we use a shorthand notation where we show the typing for a typical element of each sequence in the rule and then underline for the natural extension to the entire sequence.

The typing rule T-R-AGENT makes sure that all expressions assigned to attribute names have the correct type according to the agent type. If we have an agent type binding $n_a : (n_1 : b_1, \ldots, n_k : b_k)$, each attribute expression $n_i = e_i$ of a right agent expression with name n_a should have type b_i in a base type environment Γ. Again we use the shorthand sequence notation, indicating the rule for typical elements of each sequence and then underlining for the extension to the entire sequence.

A further restriction implied by the typing rules is that agent expressions should define all the attributes indicated by their type.

Rule expressions

Rules, r, as we have seen from the examples, have a left-hand side (lhs) consisting of a sequence of left agent expressions, a_l, a right-hand side (rhs) consisting of a sequence of right agent expressions, a_r, a rate expression e_r, and a condition expression e_c (rule expr, Figure 4.1).

The typing rule T-RULE first makes sure via the judgment $\Delta \vdash \underline{a_l} : \Gamma'$ that each left agent expression is well-formed and collects all the basic environments these left agent expressions produce into the environment Γ'. The judgement can be given (inductively

on the length of the sequence) by the following axiom and rule:

$$\Delta \vdash \varepsilon : \varepsilon \qquad \frac{\Delta \vdash a_l : \Gamma \quad \Delta \vdash \underline{a'_l} : \Gamma'}{\Delta \vdash a_l \underline{a'_l} : \Gamma\Gamma'} \quad (\Gamma \text{ and } \Gamma' \text{ do not overlap})$$

where two environments are said to *overlap* if there is a variable they both bind.

Then, in the environment extended with environment produced by the lhs, it is checked that the rest of the rule is well formed; in particular, in $\Gamma[\Gamma'] | \Delta \vdash \underline{a_r}$ it is intended that $\Gamma[\Gamma'] | \Delta \vdash a_r$ holds for each of the a_r. The extended type environment is needed since the variables of the rule lhs may appear in the rhs, rate, and condition expressions.

Model definitions

A full basic Chromar model m is just a sequence of definitions. We have a sequence of agent declarations, $\underline{\mathrm{a_d}}$ followed by the initial state definition, which is just a sequence of right agent expressions, and finally a sequence of rule expressions (model def, Figure 4.1). The typing rule T-MODEL constructs new agent type bindings for all the agent declarations in the model. Then in a type environment extended with the defined agent types, the rule checks the initial state definition and the sequence of rules.

4.1.1 Syntactic extensions

My concrete realisation of Chromar provides two syntactic extensions to the above language which we explain informally below. Chromar with these extensions can be translated to Chromar without them; we omit a formal treatment of the translation.

Missing attributes

We can relax the constraint of having to specify the all the attributes of agents in rules by establishing a correspondence between left and right-hand sides of rules as follows:

- First, bindings to fresh variable are added to left agents for any missing attributes (these attributes are known from its agent type). For example, given an agent type $\mathrm{A}(\mathrm{a} : \mathrm{int}, \mathrm{b} : \mathrm{int})$, the incomplete left agent expression $\mathrm{A}(\mathrm{a} = x)$ becomes $\mathrm{A}(\mathrm{a} = x, \mathrm{b} = y)$, where y is a fresh variable (i.e., one not occurring anywhere else in the lhs).

- Then, for every $\underline{a_l}.i$ in the left-hand side we establish a positional correspondence with an $\underline{a_r}.j$ in the right-hand side, if it exists, and complete the corresponding

right agent expressions with the same fresh name bindings, if needed. Any right agent expressions that do not have a corresponding left agent expression need to be fully defined. Continuing with our example, the rule $\mathrm{A}(\mathrm{a} = x) \to \mathrm{A}(\mathrm{a} = x+1)$ is completed to $\mathrm{A}(\mathrm{a} = x, \mathrm{b} = y) \to \mathrm{A}(\mathrm{a} = x+1, \mathrm{b} = y)$.

Attribute conditional expressions

Condition expressions restrict the applicability of rules by establishing a relation between agents. These relations are often functional dependencies. In such cases, rules can be made easier to read and write by using expressions — not just variables — inside left agent expressions.

For example, consider an agent type $\mathrm{A}(\mathrm{a} : \mathrm{int})$. If we want a rule $\mathrm{A}(\dots), \mathrm{A}(\dots) \to \dots$ to be applicable to only a subset of the $(\mathrm{A}(\dots), \mathrm{A}(\dots))$ pairs in the state, we might use the condition expression $y = f(x)$ to encode a functional dependency of y on x, as in, say, a rule of the form

$$\mathrm{A}(\mathrm{a} = x), \mathrm{A}(\mathrm{a} = y) \to \dots \ [y = f(x)]$$

However, instead of binding to y in the second left agent expression and then putting the condition $y = f(x)$ at the end, we can write the expression $f(x)$ directly in the left agent expression, obtaining the rule

$$\mathrm{A}(\mathrm{a} = x), \mathrm{A}(\mathrm{a} = f(x)) \to \dots$$

This is just syntactic sugar as any rule left-hand side with such expressions can be rewritten as a rule with a binding-only left-hand side. This is done as follows: any non-variable expression in a left agent expression of the rule is rewritten as a binding to a fresh variable, and a corresponding equality condition on this variable is added to the conditional expression of the rule.

4.2 Semantics of basic Chromar

First, regarding notation, we write $\{a_1 \mapsto b_1, a_2 \mapsto b_2, \dots\}$ for finite partial maps, from A to B, with $a_1, a_2, \dots$ distinct elements of A and $b_1, b_2, \dots$ in B. For two such functions σ and σ' we write $\sigma[\sigma']$ for the partial function produced by using σ' to extend or overwrite the values produced by σ. We write $\{|a_1, \dots, a_n|\}$ for the (finite) multiset of elements of a set A whose elements, counting repetitions, are $a_i, \dots, a_n$, and take such

multisets to be functions from A to $\mathbb{N}$ counting multiplicities; note that such a function m is zero for all but finitely many arguments, and that a is an element of m, written $a \in m$, exactly when $m(a) \neq 0$. We write $\mathrm{ms}(\underline{a})$ for the multiset corresponding to the sequence $\underline{a}$. We write $m \preceq m'$ for the submultiset relation; this takes multiplicities into account and is defined pointwise. Finally, we write $\mathrm{M}[A]$ for the set of all finite multisets over a set A and we write $\mathrm{S}[A]$ for the set of all finite sequences over A.

For each basic type $b \in B$ we assume we have an associated set of values V_b, including $\mathrm{V}_{\mathrm{bool}} = \{tt, ff\}$ for bool and $\mathrm{V}_{\mathrm{real}} = \mathbb{R}$ for real. We have $\mathrm{V} = \bigcup_{b \in B} \mathrm{V}_b$ the union of these sets of values ranged over by v. A *value environment* $\sigma = id_1 : v_1, \ldots, id_n : v_n$ with $v_1, \ldots, v_n \in \mathrm{V}$ is a sequence of variable value bindings, where the variables are required to be distinct. We will sometimes treat value environments as functions with finite domain; for example to get the value of id_i in σ we may write $\sigma(id_i)$. We say that $\sigma = id_1 : v_1, \ldots, id_n : v_n$ is a Γ-environment for a base type environment $\Gamma = id_1 : b_1, \ldots, id_n : b_n$ if each $id_i : v_i$ in σ has an entry for that id_i in Γ of the correct type, i.e., $\sigma(id_i) \in \mathrm{V}_{b_i}$.

Regarding the semantics of expressions, we assume that for each Γ-environment σ, base type b, and expression e such that $\Gamma \vdash e : b$, the expression has an evaluation $[\![e]\!]_\Gamma(\sigma) \in \mathrm{V}_b$ (below, we often omit the environment subscripts on evaluation functions). In particular, for typings $\Gamma \vdash id : b$ of variables id, we assume that $[\![id]\!]_\Gamma(\sigma) = \sigma(id)$.

We next give the semantics of the other syntactic constructs of basic Chromar: agent expressions, rules, and models.

Agents

Agents are modelled as structures with a name and a finite partial map from attribute names to base values, $(n_a, \{n_f \mapsto v\})$. We write Av for the set of such agents, ranged over by av. We define the set of *states*, ranged over by s, to be $\mathrm{M}[\mathrm{Av}]$, the set of all finite multisets of agents. When writing agents in examples, we use left agent expressions, but with constants in place of variables. So for example, we write $\mathrm{Leaf}(\mathrm{age} = 1, \mathrm{mass} = 10)$ for the agent $(\mathrm{Leaf}, \{\mathrm{age} \mapsto 1, \mathrm{mass} \mapsto 10\})$.

We evaluate sequences of left and right agent expressions as states. For left agent expressions, suppose we have a typing $\Delta \vdash \overline{n_a(\underline{n_f = id})} : \Gamma$ of a sequence of left agent expressions. Then Γ is determined by Δ, and gives basic types for the variables in the various left agent expressions (and no more). So, for any Γ-environment σ we may

define the evaluation $[\![\underline{n_a(\underline{n_f = id})}]\!]_\Delta(\sigma) \in \mathrm{M}[\mathrm{Av}]$ of the sequence as follows:

$$[\![\underline{n_a(\underline{n_f = id})}]\!]_\Delta(\sigma) = \mathrm{ms}(\underline{n_a(\underline{n_f = \sigma(id)})}$$

For right agent expressions, suppose that we have a typing $\Gamma \mid \Delta \vdash \underline{n_a(\underline{n_f = e})}$ of a sequence of right agent expressions. Then for any Γ-environment σ we define the evaluation $[\![\underline{n_a(\underline{n_f = e})}]\!]_{\Gamma,\Delta}(\sigma) \in \mathrm{M}[\mathrm{Av}]$ of the sequence as follows:

$$[\![\underline{n_a(\underline{n_f = e})}]\!]_{\Gamma,\Delta}(\sigma) = \mathrm{ms}(\underline{n_a(\underline{n_f = [\![e]\!]_\Gamma(\sigma)})}$$

Rules

Rules induce (stochastic) reactions, and I discuss those first. A *reaction* is a structure $\rho = l \xrightarrow{k} r$ with l, r states and k a positive real number. We will assume that these names also act as accessors for the parts of the reaction structure, so for a reaction ρ, $l(\rho)$ returns its left-hand side and so on. We can apply the reaction ρ to a state s if $l(\rho) \preceq s$, when we obtain a new state $s - l(\rho) + r(\rho)$, which we write as $\rho \bullet s$.

A *stochastic matrix* (aka *rate function*) over a set of states S is a function $Q : S \times S \to \mathbb{R}_+$ with Q zero on all but finitely many entries of any row. Note that the sum of two stochastic matrices is also a stochastic matrix, as is the zero matrix. We will use stochastic matrices over $\mathrm{M}[\mathrm{Av}]$ to encode the rate at which reactions occur in different states.

The rate at which a reaction ρ occurs in a state s depends on the number of ways in which $l(\rho)$ occurs in s. For any set A, we define the *multiplicity* of a submultiset $m \in \mathrm{M}[A]$ of another multiset $m' \in \mathrm{M}[A]$ as the number of distinct times m appears as a submultiset of m', defined as:

$$\mu(m, m') = \prod_{a \in m} \binom{m'(a)}{m(a)}$$

Consider for example multisets $m = \{|a, a, b|\}$ and $m' = \{|a_1, a_2, a_3, b|\}$ where we label the a's in m' to distinguish them. The multiplicity of m in m' is three as m appears as a submultiset of m' in three distinct ways:

$$\mu(m, m') = |\{\{|a_1, a_2, b|\}, \{|a_1, a_3, b|\}, \{|a_2, a_3, b|\}\}|$$

With these preliminaries out of the way, we can define the semantics of rules as stochastic matrices on $\mathrm{M}[\mathrm{Av}]$, given suitable environments. Suppose that we are given a rule typing $\Gamma \mid \Delta \vdash \underline{a_l} \xrightarrow{e_r} \underline{a_r}\ [e_c]$. Then we further have $\Delta \vdash \underline{a_l} : \Gamma'$ for a unique Γ'. We also have $\overline{\Gamma} \vdash e_r : \mathrm{real}$, $\overline{\Gamma} \mid \Delta \vdash \underline{a_r}$, and $\overline{\Gamma} \vdash e_c : \mathrm{bool}$, where $\overline{\Gamma} = \Gamma[\Gamma']$.

In order to define the stochastic matrix over M[Av] associated to the rule $\underline{a_l} \xrightarrow{e_r} \underline{a_r}\,[e_c]$, we first need to explain when it matches a state and to define the reaction it denotes. For the first, we say that a Γ'-environment m is a *match* for the rule in a state $s = \{|av_1, \ldots, av_n|\}$ if $[\![a_l]\!]_\Delta(m)$ is a submultiset of s. Note that there are only finitely many possible such rule matches. For the second, suppose we are given a Γ-environment σ and a Γ'-environment σ', then we set[1]

$$\mathcal{R}[\![\underline{a_l} \xrightarrow{e_r} \underline{a_r}\,[e_c]]\!](\sigma,\sigma') \simeq \begin{cases} [\![a_l]\!]_\Delta(\sigma') \xrightarrow{[\![e_r]\!]_{\overline{\Gamma}}(\overline{\sigma})} [\![a_r]\!]_{\overline{\Gamma},\Delta}(\overline{\sigma}) & (\overline{\sigma} = \sigma[\sigma'], [\![e_r]\!]_{\overline{\Gamma}}(\overline{\sigma}) > 0, \\ & \quad [\![e_c]\!]_{\overline{\Gamma}}(\overline{\sigma}) = tt) \\ \text{undefined} & (\text{otherwise}) \end{cases}$$

Note that this reaction only exists if (the denotation of) the rule rate is positive and (the denotation of) the rule condition holds. In case the reaction exists and σ' is a match for the rule in a state s, we say that the match is a *proper* match for the rule in state s, given σ. We can now define the stochastic matrix associated to the rule, given a Γ-environment σ:

$$[\![\underline{a_l} \xrightarrow{e_r} \underline{a_r}\,[e_c]]\!]_{\Gamma,\Delta}(\sigma)(s,s') = \sum\{[\![e_r]\!]_{\overline{\Gamma}}(\overline{\sigma}) \cdot \mu([\![a_l]\!]_\Delta(m), s) \mid m \text{ is a proper match for } a_l \text{ in } s, \overline{\sigma} = \sigma[m], s' = \mathcal{R}[\![\underline{a_l} \xrightarrow{e_r} \underline{a_r}\,[e_c]]\!]_{\Gamma,\Delta}(\sigma,m) \bullet s\}$$

For example, consider the left-hand expression $\mathrm{A}(\mathrm{a} = x), \mathrm{A}(\mathrm{a} = y)$ and state $\{|\mathrm{A}(\mathrm{a} = 1), \mathrm{A}(\mathrm{a} = 2)|\}$. We have two distinct matches $m_1 = \{x \mapsto 1, y \mapsto 2\}$ and $m_2 = \{x \mapsto 2, y \mapsto 1\}$. Next, consider the rule

$$r \;=\; \mathrm{A}(\mathrm{a} = x), \mathrm{A}(\mathrm{a} = y) \xrightarrow{f(x,y)} \mathrm{A}(\mathrm{a} = x + y), \mathrm{A}(\mathrm{a} = y - 1)\,[g(x,y)]$$

assumed well-typed given the agent type environment $A : (a : \mathrm{int})$ and empty base type environment. Then, assuming, for example, that $f(x,y)$ denotes 3 in m_1 and the condition $g(x,y)$ holds there, m_1 yields a reaction from our rule r, namely:

$$\mathcal{R}[\![r]\!](\{\}, m_1) \;=\; \mathrm{A}(\mathrm{a} = 1), \mathrm{A}(\mathrm{a} = 2) \xrightarrow{3} \mathrm{A}(\mathrm{a} = 3), \mathrm{A}(\mathrm{a} = 1)$$

Assuming, further, that the condition does not hold in m_2, for the stochastic matrix of r we will have:

$$[\![r]\!](\{\})(\{|\mathrm{A}(\mathrm{a} = 1), (\mathrm{a} = 2)|\}, \{|\mathrm{A}(\mathrm{a} = 3), \mathrm{A}(\mathrm{a} = 1)|\}) = 3$$

[1] We use Kleene equality in this definition: for any two, possibly undefined, mathematical expressions e and e', $e \simeq e'$ holds if (i) e is defined if, and only if, e' is defined, and (ii) they are equal if they are both defined.

If, on the other hand, the condition does hold in m_2 and $f(x,y)$ denotes 3 there, then the stochastic matrix of r will have value 6. Thus we may count the same reaction several times when applying a rule.

We should remark that there is an alternative way of counting multiplicities. Rather than regarding two occurrences of the same agent as identical and using the binomial coefficient to count up to symmetry, one instead regards them as distinct entities and uses the falling factorial. The former approach is used in the standard biochemical literature and in stochastic multiset rewriting (D. F. Anderson and Kurtz, 2011; Barbuti et al., 2009); the latter approach is used in the rule-based kappa system Danos, Feret, Walter Fontana, Harmer et al., 2008, Section 4.2.3. In the case of the usual chemical reactions, the two approaches are the same up to a multiplicative change in the reaction rate. However this is not the case for rule-based systems. Consider, for example, a rule with left-hand side $\mathrm{A}(\mathrm{a}=x),\mathrm{A}(\mathrm{a}=y)$, and the state $\{|\mathrm{A}(\mathrm{a}=1),\mathrm{A}(\mathrm{a}=1),\mathrm{A}(\mathrm{a}=2)|\}$. The two ways of counting multiplicities of the match $\{x\mapsto 1, y\mapsto 1\}$, give 1 and 2, whereas for the match $\{x\mapsto 1, y\mapsto 2\}$ they both give 2. Nonetheless, a more complex translation between the two approaches is possible, assuming available a sufficiently rich stock of conditional expressions for rate expressions to distinguish the various possible cases.

Stochastic semantics of models

A *continuous time Markov chain (CTMC)* is a tuple (S,Q,I) with S a set of states, $Q : S\times S\to\mathbb{R}$ a stochastic matrix on S, and $I\in S$ the initial state. Basic Chromar models are evaluated as CTMCs with set of states $\mathrm{M}[\mathrm{Av}]$, given a suitable environment. So suppose we have a model typing $\Gamma\vdash \underline{a_d}$; **init** $(\underline{a_r})$; $\underline{r}$. Then, for a unique Δ, we have $\vdash \underline{a_d} : \Delta$, $\Gamma\,|\,\Delta\vdash \underline{a_r}$, and $\Gamma\,|\,\Delta\vdash \underline{r}$. Then, for any Γ-environment σ, the CTMC associated to the model is:

$$[\![\underline{a_d};\ \mathbf{init}\,(\underline{a_r});\ \underline{r}]\!]_{\Gamma,\Delta}(\sigma) = (\mathrm{M}[\mathrm{Av}], \sum_{r\in\underline{r}} [\![r]\!]_{\Gamma,\Delta}(\sigma), [\![\underline{a_r}]\!]_{\Gamma,\Delta}(\sigma))$$

We remark that, in the implementation of Chromar, the Γ-environment needed for the semantics of a model, is supplied by the Haskell context in which the model is defined. We also remark that, with this definition, the same reaction may be counted more than once as it can occur in two different ways, either when applying a given rule (a possibility noted above), or when applying two different rules.

4.2.1 A simulation algorithm

If we try to expand basic Chromar rules to the equivalent reactions for simulation, we may obtain infinitely many reactions (unless we constrain the types of the attributes in our agent types to be finite). However, for a given state only finitely many of these reactions will apply, so we can still use the normal Stochastic Simulation Algorithm (SSA) to get sample paths from the CTMC, without constraining the available attribute types.

Specifically our algorithm is the usual SSA (direct method Gillespie, 1977), but with an extra step (1) that dynamically creates the reactions based on the current state of the system. We assume a model and an environment σ, as above. The algorithm keeps a state s and a time t, starting with state $s_0 = [\![\underline{a_r}]\!]_{\Gamma,\Delta}(\sigma)$ and an initial time 0, then iterates the following sequence of steps as many times as desired.

1. For current state s, generate the multiset of all possible reactions for every rule:

   $$R = \{|\mathcal{R}[\![\underline{r}.i]\!](\sigma, m) \mid i = 1, \ldots, |\underline{r}|, m \text{ is a proper match of rule } \underline{r}.i \text{ in } s, \text{ given } \sigma|\}$$

2. Calculate the total rate $k_T = \sum_{\rho \in R} R(\rho) \cdot (\mu(l(\rho), s) \cdot k(\rho))$. Halt if this is zero.

3. Pick the waiting time t_w for the next reaction event from the exponential distribution with cumulative distribution function $F(t) = 1 - e^{-k_T t}$.

4. Pick exactly one of the reactions, choosing reaction $\rho \in R$ with probability $\frac{R(\rho) \cdot \mu(l(\rho), s) \cdot k(\rho)}{k_T}$.

5. If reaction ρ is picked then update the state to $s' = \rho \bullet s$, and the time to $t + t_w$.

Note that we take account of the multiplicity of reactions here: the same reaction can occur from two different rules, or from one rule using different matches.

4.3 Extended Chromar

I now extend basic Chromar to include the fluent and observable features that were introduced informally in Chapter 3. Observables and fluents extend the given set of (*ordinary*) expressions, $e \in E$, to give what we call *enriched expressions* e_r. These consist of ordinary expressions, fluents and observables, and any combinations of them. Enriched expressions give access to the current state and time, and so we take the value of an enriched expression with type b to be a function from states and times to V_b,

rather than an element of V_b (in the language of intensional logic we work with their *intensions* rather than their *extensions* Fitting, 2004, 2015).

As enriched expressions allow an arbitrary mixing of ordinary expressions, fluents, and observables, we might have an observable inside a fluent or a fluent inside an observable and so on. For example we could write:

$$f(m) \textbf{ when } nl > 10 \textbf{ else } f'(m)$$

where *nl* is an observable for the number of leaves. This might be used to introduce the crowding effect on rosette leaves; crowding affects the assimilation rate as it reduces the photosynthetically active area.

We begin with the syntax of extended Chromar and then proceed to its semantics.

4.3.1 Abstract syntax of extended Chromar

The grammar and typing rules are given in Figure 4.2, and I discuss them next. The discussion begins with enriched expressions and then moves on to agents, rules, and models. These are much as in basic Chromar, but adapted to allow enriched expressions; in particular there is a new syntactic class of declarations that allows (the intensional values of) enriched expressions to be used in models.

Enriched expressions

- Any ordinary expression $e \in E$ is an enriched expression (ord-expr, Figure 4.2).

- Any variable in an enriched expression can refer to (the value of) another enriched expression using the where expression.

  For example we can write $t > 5$ **where** t : real **is** temp to use the temp fluent inside, in this case, an ordinary expression. These expressions are typed inside a type environment extended using the type of the enriched expression in the where clause (T-WHERE, Figure 4.2). In our examples we generally proceed informally, omitting the where clause and instead using the where clause expression directly; for example we might write temp > 5, although that is not officially an enriched expression.

- The time, condition, and repeat expressions are fluents. They are used to provide ways of expressing common motifs of time-dependent behaviour, and appeared in examples given above. The **time** constructor gives access to time. The **when else**

fluent is used to specify conditional behaviour over time, where, if a condition is met, we choose one fluent and, if not, we choose a different one. The typing rule T-COND ensures that the conditional is of type bool and that the two alternatives have the same type; the type of the whole expression is then the common type of the two alternatives. The **repeatEvery** fluent is used to define a repeating behaviour over time, where the first enriched expression is the period of the cycling and the second is the behaviour to be cycled. The T-REPEAT typing rule ensures that the first such expression has type real since it is meant to denote time; the type of the whole expression is then the type of the cycling behaviour.

- The last way to construct enriched expressions is through the database-inspired observables expression (obs, Figure 4.2). The **select** part is followed by a left agent expression. This is used to 'filter out' all the agents that have the agent name of the left agent expression from the current state, while producing bindings of the left agent variables to the values of the corresponding agent fields. For example, to select only Leaf agents from the state, we write **select** Leaf$(\text{age} = i, \text{mass} = m)$, thereby producing a multiset of bindings of i and m to the values of the age and mass fields of the selected agents, with one such binding for each occurrence of a leaf agent in the state multiset. Since multiple agents can have the same attribute values, we need a mutiset of bindings rather than just a set of them.

  The **aggregate** part contains an enriched expression e_r, used to obtain a combining function to 'fold' the state, and a second such expression e'_r, used to obtain the initial value of the folding. The combining function takes the field values from each of the bindings produced by the select part and an initial value, and returns an update to the folding value. To enable this, e_r (but not e'_r) is in the scope of the variables in the left agent expression.

  We remark that it is in typing observables expressions that the need for the agent type environment Δ arises in the typing rules for enriched expressions.

Agent expressions, rules, and models

Enriched expressions can appear in the same places where expressions can be expected and can also be bound to variables in models. In more detail:

- Regarding agent introduction and left agent expressions, we keep the rules T-INTRO and T-L-AGENT of Figure 4.1, but do not repeat them in Figure 4.2.

Regarding right agent expressions and rules, we adapt the rules T-R-AGENT and T-RULE of Figure 4.1 to Figure 4.2 to allow enriched expressions to appear in places where expressions are expected: in right agent expressions, rates, and conditions.

- Variables can be bound to the values of enriched expressions so that they can be referred to later (declare, Figure 4.2). A declaration e_d produces an environment in which the type of the enriched expression is bound to the variable. These environments are used to produce an environment from a sequence of declarations; the rule for this (T-DEC, Figure 4.2) takes account of the facts that declared variables can be used in subsequent expressions and that the same variable may be declared more than once.

- As before, models m have agent declarations, initial states, and rules, but may now use enriched expressions, except in the right agent expression for the initial state, as it would make no sense to have enriched expressions there (for this reason the judgment $\Gamma|\Delta \vdash_{\mathrm{b}} \underline{a_r}$ is used in the T-MODEL rule, with the subscript intended to indicate that the judgement is one of basic, not extended, Chromar).

  Models additionally have a sequence of declarations, $\underline{e_d}$, and the base type environment produced by this sequence is added to the initial base type environment when checking the rules (model def, Figure 4.2). The environment produced by the declaration sequence is given inductively by the following axiom and rule:

$$\Gamma|\Delta \vdash \varepsilon : \varepsilon \qquad \frac{\Gamma|\Delta \vdash e_d : \Gamma' \quad \Gamma[\Gamma']|\Delta \vdash \underline{e_d} : \Gamma''}{\Gamma|\Delta \vdash e_d, \underline{e_d} : \Gamma'[\Gamma'']}$$

  Note that, unlike the cases of agent declarations or rule lhs's, the environments here are 'overlapped' ($\Gamma[\Gamma']$), rather than 'conjoined' (Γ,Γ'), as in the rule for sequences of left agent expressions. This is to enable identifiers bound to enriched expressions declared earlier in a declaration sequence to be used later in it.

4.3.2 Semantics of extended Chromar

We next give the semantics of Chromar extended with fluents and observables, beginning with that of enriched expressions and then moving on to the constructs that use enriched expressions. We need to use functional environments: we say that $\sigma = id_1 : f_1, \ldots, id_n : f_n$ is a *functional* Γ*-environment* for a base type environment $\Gamma = id_1 : b_1, \ldots, id_n : b_n$ if, for

$i = 1,\ldots,n$, f_i is a function from $\mathrm{M}[\mathrm{Av}] \times \mathbb{R}$ to V_{b_i}. We work with these environments as we do with others, in particular treating them as finite functions. Given a functional Γ-environment $\sigma = id_1 : f_1,\ldots,id_n : f_n$ and a state s and a time t we can obtain a Γ-environment by setting $\sigma(s,t) = id_1 : f_1(s,t),\ldots,id_n : f_n(s,t)$. In the other direction we can 'promote' a Γ-environment $\sigma = id_1 : v_1,\ldots,id_n : v_n$ to a functional Γ-environment $\uparrow\sigma = id_1 : f_1,\ldots,id_n : f_n$, by setting $f_i(s,t) = v_i$, for $i = 1,\ldots,n$.

We need some further definitions in order to give the semantics of observables. For any basic environment Γ we write V_Γ for the set of Γ-environments. Then, given any left agent expression $\Delta \vdash a_l : \Gamma$, where $a_l = n_a(\underline{n_f = id})$, we define a 'selection function'

$$\mathrm{sel}_{\Delta,a_l} : \mathrm{M}[\mathrm{Av}] \to \mathrm{M}[\mathrm{V}_\Gamma]$$

that takes a state and returns a multiset of bindings (Γ-environments). This multiset has a binding for each matching agent in the state with multiplicity that of the matching agent. The selection function is defined by:

$$\mathrm{sel}_{\Delta,a_l}(s) = \sigma \in \mathrm{V}_\Gamma \mapsto s(n_a(\underline{n_f = \sigma(id)}))$$

recalling that we treat finite multisets as functions that have value 0 except at finitely many arguments. In this respect, note that, as required, $\mathrm{sel}_{\Delta,a_l}(s)$ is 0 except for finitely many arguments as s is, and the function $\sigma \mapsto n_a(\underline{n_f = \sigma(id)})$ is 1-1.

We recall the list folding function (and see Hutton, 1998). Identifying sequences and lists, for any sets X, Y, combining function $f : X \times Y \to Y$ and $y \in Y$, we define: $\mathrm{fold}(u,f,y) \in Y$ by induction on the length of u as follows:

$$\mathrm{fold}(u,f,y) = \begin{cases} y & (u = \varepsilon) \\ f(x,\mathrm{fold}(u',f,y)) & (u = xu') \end{cases}$$

Under a natural condition on the combining function, we can use list folding to define finite multiset folding. Say that $f : X \times Y \to Y$ has a *commutative (left) action* if, for any $x,x' \in X$ and $y \in Y$, we have:

$$f(x,f(x',y)) = f(x',f(x,y))$$

For such a function f, we can unambiguously define $\mathrm{fold}(m,f,y)$, for a finite multiset $m \in \mathrm{M}[X]$, to be $\mathrm{fold}(u,f,y)$, choosing any $u \in \mathrm{S}[Y]$ such that $\mathrm{ms}(u) = m$.

Finally we assume available a 'linearisation function'

$$\langle\langle - \rangle\rangle : \mathrm{M}[\mathrm{V}_{\underline{id:b}}] \longrightarrow \mathrm{S}[\mathrm{V}_{\underline{id:b}}]$$

making such a choice for multisets of $\underline{id : b}$-environments. It puts the elements of a finite multiset of $\underline{id : b}$-environments in some standard order, repeating them according to their multiplicity, so that $\mathrm{ms}(\langle\langle m \rangle\rangle) = m$ holds for any $m \in \mathrm{M}[\mathrm{V}_{\underline{id:b}}]$.

Semantics of enriched expressions

As discussed above we take the values of enriched expressions to be functions of states and time. So, suppose that we have a typing $\Gamma|\Delta \vdash e_r$ of an enriched expression e_r. Then for any functional Γ-environment σ, we define its evaluation $[\![e_r]\!]_{\Gamma,\Delta}(\sigma) : \mathrm{M}[\mathrm{Av}] \times \mathbb{R} \to \mathrm{V}_b$ as follows, where we divide the definition into cases according to the form of the enriched expression:

$$[\![e]\!]_{\Gamma,\Delta}(\sigma)(s,t) = [\![e]\!]_{\Gamma}(\sigma(s,t))$$

$$[\![e'_r \textbf{ where } id : b \textbf{ is } e_r]\!]_{\Gamma,\Delta}(\sigma) = [\![e'_r]\!]_{\Gamma,\Delta}(\sigma[x \mapsto [\![e_r]\!]_{\Gamma|\Delta}(\sigma)])$$

$$[\![\textbf{time}]\!]_{\Gamma,\Delta}(\sigma)(s,t) = t$$

$$[\![e_r \textbf{ when } e'_r \textbf{ else } e''_r]\!]_{\Gamma,\Delta}(\sigma)(s,t) \;=\; \begin{cases} [\![e_r]\!]_{\Gamma,\Delta}(\sigma)(s,t) & (\text{if } [\![e'_r]\!]_{\Gamma|\Delta}(\sigma)(s,t) = tt) \\ [\![e''_r]\!]_{\Gamma,\Delta}(\sigma)(s,t) & (\text{otherwise}) \end{cases}$$

$$[\![\textbf{repeatEvery } e_r\ e'_r]\!]_{\Gamma,\Delta}(\sigma)(s,t) = \begin{cases} [\![e'_r]\!]_{\Gamma,\Delta}(\sigma)(s,t') & ([\![e_r]\!]_{\Gamma,\Delta}(\sigma)(s,t) > 0) \\ 0 & (\text{otherwise}) \end{cases}$$

where $t' = t \mod [\![e_r]\!]_{\Gamma,\Delta}(\sigma)(s,t)$

$$[\![\textbf{select } a_l; \textbf{ aggregate } (id : b.\, e_r)\ e'_r]\!]_{\Gamma,\Delta}(\sigma)(s,t) = \mathrm{fold}(\langle\langle \mathrm{sel}_{\Delta,a_l}(s) \rangle\rangle, f, [\![e'_r]\!]_{\Gamma,\Delta}(\sigma)(s,t))$$

where $\Delta \vdash a_l : \Gamma'$ and:

$$f(\sigma', v) = [\![e_r]\!]_{\Gamma[\Gamma'[id:b]],\Delta}(\sigma[\sigma'[id : v]])(s,t) \quad (\sigma' : \mathrm{V}_{\Gamma'}, v : \mathrm{V}_b)$$

Here:

- Ordinary expressions, $e \in E$, are evaluated using the provided evaluation function.

- Enriched expressions e'_r **where** $id : b$ **is** e_r are evaluated by evaluating the first expression e'_r in a functional environment in which id is bound to the evaluation of the second expression e_r.

- The time expression is evaluated by returning the given time (so ignoring the environment and the state).

- The conditional expression is evaluated by evaluating the first expression if the second expression (the condition) evaluates to true, and otherwise by evaluating the third expression.

- The expression **repeatEvery** e_r e'_r is evaluated at time t by evaluating e'_r at t modulo the evaluation of e_r at t, if the evaluation of e_r is > 0, and returning 0, if the evaluation of e_r at t is ≤ 0. While the semantics always gives a result, it is only sensible if e_r's value is not 0; a more complex semantics for Chromar would produce an error in that case.

- Finally, given a state, the observable expression **select** a_c; **aggregate** $(id : b.\, e_r)$ e'_r is evaluated in two stages. First, the left agent expression $a_l = n_a(\underline{n_f = id})$ is used to filter out all the n_a agents from the state and transform them into environments holding the field values of the agent; this produces a multiset of such environments. Second, the value of the observable expression is produced by a fold on this multiset. The fold uses a combining function obtained from the semantics of e_r, and an initial value given by evaluating e'_r. Note that it employs the linearisation function $\langle\langle - \rangle\rangle$ assumed above. While the semantics gives a result for every such combining function, the result of such a fold using a fixed, but arbitrary, selection function, is only sensible if the combining function has a commutative left action. We leave it to the programmer to make sure their combining function has such an action.

Declarations

Declarations $id = e_r$ bind the values of enriched expressions e_r to variables id. A series of such expressions evaluates to an environment binding the variables to the values of the corresponding enriched expressions.

So, for a single declaration $\Gamma|\Delta \vdash id = e_r : (id : b)$, for every functional Γ-environment σ, we define its evaluation $[\![id = e_r : (id : b)]\!]_{\Gamma,\Delta}(\sigma)$ as a functional $id : b$-environment

as follows:

$$[\![id = e_r]\!]_{\Gamma,\Delta}(\sigma) = (id : [\![e_r]\!]_{\Gamma,\Delta}(\sigma))$$

and for a sequence of declarations $\Gamma|\Delta \vdash \underline{e_d} : \Gamma'$, for every functional Γ-environment σ, we define its evaluation $[\![\underline{e_d}]\!]_{\Gamma,\Delta}(\sigma)$ as a functional Γ'-environment as follows:

$$[\![\underline{e_d}]\!]_{\Gamma,\Delta}(\sigma) = \begin{cases} \varepsilon & (\underline{e_d} = \varepsilon) \\ [\![e'_d]\!]_{\Gamma,\Delta}(\sigma)[[\![\underline{e''_d}]\!]_{\Gamma[\Gamma'],\Delta}(\sigma[[\![e'_d]\!]_{\Gamma,\Delta}(\sigma)])] & (\underline{e_d} = e'_d\underline{e''_d},\ \Gamma|\Delta \vdash e'_d : \Gamma') \end{cases}$$

Note that this definition recurses on $\underline{e_d}$, but, by induction on the length of $\underline{e_d}$, is nonetheless proper.

Agents, Rules, and Models

Enriched expressions can appear in places where simple expressions appeared: right agent expressions, conditions, and rates. This does not change our semantics except that the semantics of all the constructs that involve the enriched expressions have to be parameterised by state and time since their evaluation will depend on those.

The semantics of sequences of left agent expressions is the same as that of basic Chromar. For right agent expressions we make the evident change to accommodate enriched expressions. To this end, to evaluate sequences of right agent expressions as states, suppose that we have a right agent expression typing $\Gamma\,|\,\Delta \vdash \underline{n_a(\underline{n_f = e_r})}$. Then for any functional Γ-environment σ, state s, and time t, we define its evaluation $[\![\underline{n_a(\underline{n_f = e_r})}]\!]_{\Gamma,\Delta}(\sigma,s,t) \in \mathrm{M}[\mathrm{Av}]$ as follows:

$$[\![\underline{n_a(\underline{n_f = e_r})}]\!]_{\Gamma,\Delta}(\sigma,s,t) = \mathrm{ms}(\underline{n_a(\underline{n_f = [\![e_r]\!]_{\Gamma,\Delta}(\sigma)(s,t))}})$$

For the semantics of rules, suppose that we are given a rule typing $\Gamma\,|\,\Delta \vdash \underline{a_l} \xrightarrow{e_r} \underline{a_r}\,[e'_r]$. Then we further have $\Delta \vdash \underline{a_l} : \Gamma'$ for a unique Γ'. We also have $\overline{\Gamma}|\Delta \vdash e_r : \mathrm{real}$, $\overline{\Gamma}|\Delta \vdash \underline{a_r}$, and $\overline{\Gamma}|\Delta \vdash e'_r : \mathrm{bool}$, where $\overline{\Gamma} = \Gamma[\Gamma']$.

For the reaction denoted by the rule, suppose we are given a functional Γ-environment σ, a Γ'-environment σ', and a state s and a time t. Then we set

$$\mathcal{R}[\![\underline{a_l} \xrightarrow{e_r} \underline{a_r}\,[e'_r]]\!](\sigma,\sigma',s,t) \simeq \begin{cases} [\![\underline{a_l}]\!]_\Delta(\sigma') \xrightarrow{[\![e_r]\!]_{\overline{\Gamma},\Delta}(\overline{\sigma})(s,t)} [\![\underline{a_r}]\!]_{\overline{\Gamma},\Delta}(\overline{\sigma},s,t) & \big(\overline{\sigma} = \sigma[\uparrow\sigma'], \\ & \quad [\![e_r]\!]_{\overline{\Gamma},\Delta}(\overline{\sigma})(s,t) > 0, \\ & \quad [\![e'_r]\!]_{\overline{\Gamma},\Delta}(\overline{\sigma})(s,t) = tt\big) \\ \text{undefined} & (\text{otherwise}) \end{cases}$$

Regarding matches, we say that a Γ'-environment m is a *proper match* of the rule in a state s, given a functional Γ-environment σ and a time t, if $[\![a_l]\!]_\Delta(m)$ is a submultiset of s, and $\mathcal{R}[\![\underline{a_l} \xrightarrow{e_r} \underline{a_r}\,[e'_r]]\!](\sigma,m,s,t)$ exists (i.e., $[\![e_r]\!]_{\overline{\Gamma},\Delta}(\overline{\sigma})(s,t) > 0$ and $[\![e'_r]\!]_{\overline{\Gamma},\Delta}(\overline{\sigma})(s,t) = tt$, where $\overline{\sigma} = \sigma[\uparrow m]$).

The stochastic matrix denoted by the rule, given a functional Γ-environment σ and a time t, is:

$$[\![\underline{a_l} \xrightarrow{e_r} \underline{a_r}\,[e'_r]]\!]_{\Gamma,\Delta}(\sigma,t)(s,s') = \sum\{[\![e_r]\!]_{\overline{\Gamma},\Delta}(\overline{\sigma})(s,t)\cdot\mu([\![a_l]\!]_\Delta(m),s) \mid m \text{ is a proper match for } a_l \text{ in } s,\ \overline{\sigma} = \sigma[\uparrow m],\ s' = \mathcal{R}[\![\underline{a_l} \xrightarrow{e_r} \underline{a_r}\,[e'_r]]\!]_{\Gamma,\Delta}(\sigma,m,s,t) \bullet s\}$$

Turning to models, suppose we have a model typing $\Gamma \vdash \underline{a_d};\ \mathbf{init}\,(\underline{a_r});\ \underline{e_d};\ \underline{r}$. Then, for a unique Δ and Γ', we have $\vdash \underline{a_d} : \Delta$ and $\Gamma|\Delta \vdash \underline{e_d} : \Gamma'$, and then also $\Gamma|\Delta \vdash_{\mathsf{b}} \underline{a_r}$ and $\overline{\Gamma}|\Delta \vdash \underline{r}$, where $\overline{\Gamma} = \Gamma[\Gamma']$. Then, for any Γ-environment σ and time t the CTMC associated to the model is:

$$[\![\underline{a_d};\ \mathbf{init}\,(\underline{a_r});\ \underline{e_d};\ \underline{r}]\!]_{\Gamma,\Delta}(\sigma,t) \;=\; (\mathrm{M}[\mathrm{Av}], \sum_{r\in\underline{r}} [\![r]\!]_{\overline{\Gamma},\Delta}(\uparrow\sigma[[\![\underline{e_d}]\!]_{\Gamma,\Delta}(\uparrow\sigma)],t), [\![\underline{a_r}]\!]_{\Gamma,\Delta}(\sigma))$$

where we use the semantics of basic Chromar for the evaluation of the basic Chromar initial state right agent expression a_r.

4.3.3 A simulation algorithm

Using fluents means that the Continuous Time Markov Chain that a Chromar model generates becomes inhomogeneous in time. Therefore the standard Stochastic Simulation algorithm (SSA) does not apply since it assumes constant propensities between reactions. In practice, however, this discrepancy is often not prohibitive and gives very similar results while avoiding the extra computational cost added by non-homogeneity (we discuss this further below).

We therefore use an approximate simulation method where the fluents, and therefore all time-varying expressions in rules, are only evaluated at the times the usual SSA would choose for the next reaction. This is equivalent to keeping their values constant between reactions, at the expense of accuracy. The only change in our algorithm from that for basic Chromar is then in the reaction generation step (1) where the reaction generation happens at the current state and time in an environment extended with the evaluation of defined enriched expressions.

So assume a model and an environment σ, as above, and set σ' to be the functional Γ-environment $\uparrow\sigma[[\![e_d]\!]_{\Gamma,\Delta}(\uparrow\sigma)]$. As before, the algorithm keeps a state s and a time t, starting with state $s_0 = [\![\underline{a_r}]\!]_{\Gamma,\Delta}(\sigma)$ and an initial time 0. It then iterates the same sequence of steps as before, as many times as desired, except that step (1) is changed to

1. For current state s and current time t, generate the multiset of all possible reactions for every rule:

$$R = \{|\mathcal{R}[\![\underline{r}.i]\!](\sigma', m, s, t) \mid i = 1, \ldots, |\underline{r}|, \\ m \text{ is a proper match of rule } \underline{r}.i \text{ in } s, \\ \text{given } \sigma' \text{ and } t|\}$$

Since fluents are only sampled according to the discrete time-jumps followed by the simulation clock that means that in practice we only get a discrete approximation of the continuous functions denoted by the fluent definitions (see for example Figure 4.3). The accuracy of the approximation will depend on the sampling interval and how fast the fluent changes. If, for example, the fluent changes on a faster timescale than that of the model then the approximation will be poor. In practice, however, the fluents are usually on the same or a slower timescale than that of the model since they are usually used to model the physical context of the model, which means we get acceptable approximations. In the case of boolean fluents (like our light fluent from Figure 4.3) similar decisions on how to handle discontinuous changes have to be made in, for example, simulators that support SBML events. A common approach there is to have explicit handling of events in the simulation loop. The stochastic simulation in the iBioSim tool, for example, checks the time of the next event and executes it if it happens before the time of the next reaction (see Watanabe and Chris J. Myers, 2014, Algorithm 7). It should be possible to extend these methods to Fluents of any type and add to our simulation algorithm if increased accuracy is needed. We note too that there are promising new methods for simulating CTMCs with time-varying propensities that alleviate some of the computational cost (Voliotis et al., 2016).

4.3.4 Simulation efficiency

In terms of simulation, our implementation is simple and follows the steps we presented in §4.2. One area of improvement could be in the reaction generation step where we currently generate all the active reactions at every step. In practice we do not need to completely regenerate the reactions at every step since only a subset of them changes

between steps. The performance gains will depend on the efficiency of calculating the change in the reactions set after a change in the state. This idea was already used, to great effect, in the implementation of Kappa Danos, Feret, Walter Fontana and Krivine, 2007.

4.4 Conclusion

Here I defined the formal syntax and semantics of Chromar. This process of definition uncovered and made explicit several design choices that were implicit in the informal version of the language in my head. It furthermore made the extensions and implementation of enriched expression possible by clarifying its relation to the basic parts of the language. The abstract nature of Chromar (parametric to expression and types) was also made easier by this formal definition. In the next chapter I make particular choices regarding these (Haskell expressions and types respectively) but the availability of the formal abstract definition of this chapter means that other choices can be made with less effort.

	Syntax		*Typing rules*
$e_r ::=$	$e \quad (e \in E)$	ord-expr	$\dfrac{\Gamma \vdash e : b}{\Gamma \mid \Delta \vdash e : b}$ (T-OrdExpr)
	e'_r **where** $id : b$ **is** e_r	where	$\dfrac{\Gamma \mid \Delta \vdash e_r : b \qquad \Gamma[id : b] \mid \Delta \vdash e'_r : b'}{\Gamma \mid \Delta \vdash e'_r \textbf{ where } id : b \textbf{ is } e_r : b'}$ (T-Where)
	time	time	$\Gamma \mid \Delta \vdash \textbf{time} : \text{real}$ (T-Time)
	e_r **when** e'_r **else** e''_r	condition	$\dfrac{\Gamma \mid \Delta \vdash e_r : b \qquad \Gamma \mid \Delta \vdash e'_r : \text{bool} \qquad \Gamma \mid \Delta \vdash e''_r : b}{\Gamma \mid \Delta \vdash e_r \textbf{ when } e'_r \textbf{ else } e''_r : b}$ (T-Cond)
	repeatEvery e_r e'_r	repeat	$\dfrac{\Gamma \mid \Delta \vdash e_r : \text{real} \qquad \Gamma \mid \Delta \vdash e'_r : b}{\Gamma \mid \Delta \vdash \textbf{repeatEvery } e_r \; e'_r : b}$ (T-Repeat)
	select a_l; **aggregate** $(id : b.e_r)$ e'_r	obs	$\dfrac{\Delta \vdash a_l : \Gamma' \qquad \Gamma[\Gamma'[id : b]] \vdash e_r : b \qquad \Gamma \vdash e'_r : b}{\Gamma \mid \Delta \vdash \textbf{select } a_l; \textbf{ aggregate } (id : b.e_r) \; e'_r : b}$ (id' not bound in Γ') (T-Obs)
$e_d ::=$	$id = e_r$	declare	$\dfrac{\Gamma \mid \Delta \vdash e_r : b}{\Gamma \mid \Delta \vdash id = e_r : (id : b)}$ (T-Dec)
$a_r ::=$	$n_a(\underline{n_f = e_r})$	right agent expr	$\dfrac{\Gamma \mid \Delta \vdash \underline{e_r : b}}{\Gamma \mid \Delta \vdash n_a(\underline{n_f = e_r})}$ ($n_a : (\underline{n_f : b}) \in \Delta$) (T-R-agent)
$r ::=$	$\underline{a_l} \xrightarrow{e_r} \underline{a_r} \; [e'_r]$	rule expr	$\dfrac{\Delta \vdash \underline{a_l} : \Gamma' \qquad \Gamma[\Gamma'] \mid \Delta \vdash \underline{a_r} \qquad \Gamma[\Gamma'] \mid \Delta \vdash e_r : \text{real} \qquad \Gamma[\Gamma'] \mid \Delta \vdash e'_r : \text{bool}}{\Gamma \mid \Delta \vdash \underline{a_l} \xrightarrow{e_r} \underline{a_r} \; [e'_r]}$ (T-Rule)
$m ::=$	$\underline{a_d}$; **init**$(\underline{a_r})$; $\underline{e_d}$; $\underline{r}$	model def	$\dfrac{\vdash \underline{a_d} : \Delta \qquad \Gamma \mid \Delta \vdash_{\mathrm{b}} \underline{a_r} \qquad \Gamma \mid \Delta \vdash \underline{e_d} : \Gamma' \qquad \Gamma[\Gamma'] \mid \Delta \vdash \underline{r}}{\Gamma \vdash \underline{a_d}; \textbf{ init}(\underline{a_r}); \; \underline{e_d}; \; \underline{r}}$ (T-Model)

Figure 4.2: Abstract syntax of enriched expressions and corresponding typing rules.

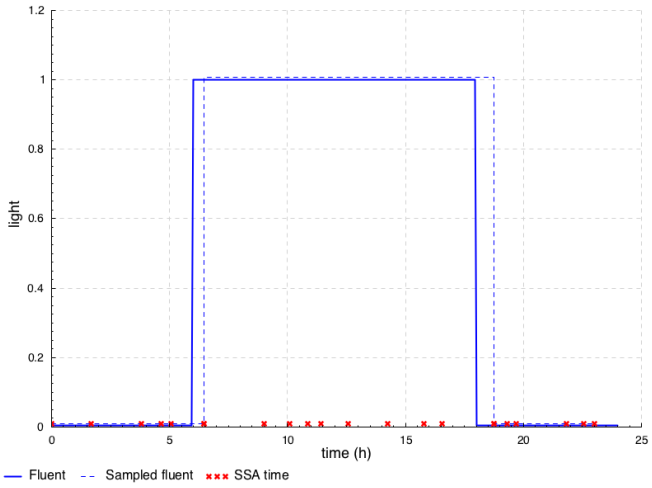


Figure 4.3: Ideal (solid line) versus practical approximation during simulation (dotted line) of the fluent for the light conditions in a day: True **when** $(6 < \textbf{time} < 18)$ **else** False, where we take True to be 1.0 and False to be 0.0. The fluent definition denotes a continuous function of time but in practice the function will only be sampled at the time points the SSA visits (red points). This means that practically we only get a discrete approximation of the fluent.

Chapter 5

Implemented Chromar

As we have seen, extended Chromar is abstract in that no choice is made of the expression language or its types. In this section I describe the implementation via embedding in Haskell of one concrete realisation of Chromar where we fix the expression language E and set of types T to Haskell expressions and types respectively. The implementation is available in a Github repository (`https://github.com/azardilis/Chromar/releases/tag/0.2`).

Embedding a domain-specific language inside a host general purpose programming languages has the advantage that it allows the reuse of the functionality of the host language for the implementation therefore relieving some of the effort involved in creating the infrastructure for a new language. From the programmer's point of view it also allows the possibility of mixing features of the host language with features of the hosted language thus getting a more expressive result. There are two basic ways of embedding a language: (i) shallow embedding where the domain-specific constructs are directly expressed in host language terms (usually through a library) and (ii) deep embedding where embedded language terms are built in some representation in the host language and are then given meaning via interpretation (P. Hudak, 1998). There are advantages and disadvantages to both approaches. Here I choose a middle ground where I use shallow embedding for expressing agent types, initial state, and outside value environment (for the definitions of constants, functions and so on) and deep embedding (via quasiquotations) using domain-specific syntax for rules and enriched expressions. Such a mixed-level embedding in Haskell has been explored before for another modelling language, Hydra, in the context of non-causal, hybrid modelling (Giorgidze and Nilsson, 2010).

I next go through the different entities in extended Chromar and see how they are

represented in Haskell. I will use `typeface` font for Haskell code to distinguish it from the abstract Chromar syntax.

The work in this chapter appears in the articles describing Chromar (Honorato-Zimmer, Andrew J Millar et al., 2017; Honorato-Zimmer, Andrew J. Millar et al., 2018). Note that the author order follows the traditional alphabetical order of computer science literature. Ricardo Honorato-Zimmer participated in the meetings/discussions of ideas with Prof Gordon Plotkin who was my primary supervisor for this part of the work while Prof Andrew Millar was secondary. Ricardo Honorato-Zimmer also contributed some code for the implementation of multisets in the simulator of Chromar.

5.1 Agents

Agent type declarations of the form **agent** $n_a = (\underline{n_f : b})$ become Haskell types with n_a as the name of the type constructor and a sequence of named, typed fields (records). Sequences of agent declarations, $\underline{a_d}$, can be collected in a union type. For example simple versions our Leaf and Cell agent declarations from §3.1 are written thus:

```
data Agent
    = Leaf { age :: Int
           , mass :: Double }
    | Cell { carbon :: Double }
```

where the `data` keyword introduces a new datatype. Agents are values of the `Agent` type, for example `Leaf{age=1, mass=1.5}` and `Cell{carbon=1.2}`. To construct multisets of Agents, we introduce the parametric type `Multiset a = [(a, Int)]`. Then the concrete type `Multiset Agent` maps agents to their counts. Given a function `mset` that constructs a multiset from a list (analogous to our function ms from §4.2), our initial state definition becomes:

```
init = mset [ Leaf{age=1,mass=0.2},
              Leaf{age=2, mass=0.1},
              Cell{carbon=0.5} ]
```

5.2 Rules (via Quasiquotation)

We use a deeper embedding of rules where we allow the expression of rules directly in domain-specific syntax using Quasi-quotes. Quasi-quotes is a Haskell language

extension that allows the use of special syntax (i.e., non standard Haskell syntax) inside `[$quoter| ... |]` brackets along with the name of a quoter function (`$quoter`) that is a function that takes the string inside the brackets and produces Haskell abstract syntax. The produced abstract syntax is injected by the Haskell compiler in place of the quotes.

The implementation here provides the `rule` quoter function that takes the string inside the quotes and produces a reaction generation function (or rather the abstract syntax of such a function) that given a state and time produces a multiset of reactions. We therefore take the rule to be a reaction generating function of type: `Time -> Multiset a -> [Reaction a]` where a reaction is naturally defined as a record type parameterised on an agent type:

```
data Reaction a = Reaction
    { lhs :: Multiset a
    , rhs :: Multiset a
    , rate :: Double
    }
```

Our quoted rules have the following syntax that is almost the same as the one we presented earlier:

$$\mathrm{r_q} ::= \underline{n_a\{\underline{n_f{=}id}\}} \rightarrow \underline{n'_a\{\underline{n'_f{=}e}\}} \mathbin{@} e_r\ [e_c]$$

For example our growth rule from the plant model example becomes:

```
growth = [rule| Leaf{mass=m}, Cell{carbon=c} -->
                Leaf{mass=m}, Cell{carbon=c-1} @g m [c >= 1] |]
```

This looks very close to our abstract syntax, but with some minor syntactic differences such as the placement of the rate expression at the end of the rule preceded by the `@` symbol. Crucially, being inside a programming language means that we can use any valid Haskell expression in the places where expressions are expected, i.e., in the values of fields in the right-hand side of rules, rates, and conditions. A very wide range of expressions are therefore supported, without further effort. We next look at the rule quoter function in more detail.

Rule quoter function

The `rule` quoter function takes a quoted rule expression, r_q, and produces a Haskell function of type `Time -> Multiset a -> [Reaction a]` where `[Reaction a]` is

a list of reactions and `Time` is a synonym for `Double`. The rule function needs to find all matches (according to the rule lhs) and then generate a concrete reaction for each. This leads to the two parts of the rule functions we need to construct: the query part — for finding the matches — and the reaction generation part. For the query part we use list comprehensions and pattern matching for binding the variables in the rule lhs. This structure follows the definition of the reactions denoted by a rule we have seen before (see $\mathcal{R}$, §4.2).

The following example illustrates the translation from quoted rule expression to rule functions. Consider the agent declaration **agent** $\mathrm{A}(\mathrm{a} : \mathrm{int})$ and rule $\mathrm{A}(\mathrm{a} = x), \mathrm{A}(\mathrm{a} = y) \xrightarrow{f(x)} \mathrm{A}(\mathrm{a} = x)\ [x > y]$. For the query part we make one comprehension generator statement for each agent on the left-hand side. The statement for the first agent becomes: `(A{a=x}, _) <- s` for some state s of type `Multiset Agent`. Pattern matching ensures that we only look at A's in the state and that we also bind the `x` variable. Note that we ignore the count of each agent-count pair since we later calculate the multiplicity of the entire lhs inside the state in the reaction generation part of the comprehension. For each such successful match of the first statement we continue to the second statement with `x` bound. The statement for the second element of the rule lhs is similarly `(A{a=y}, _) <- s`. With `x` and `y` bound we have our match m (as defined in §4.2) and we produce a reaction if the condition is true in the environment extended with m. The full comprehension statement is thus:

```
r :: Multiset Agent -> Time -> [Reaction Agent]
r s t =
     [ Reaction
      { lhs = mset [A{a = x}, A{a = y}]
      , rhs = mset [A{a = x}]
      , rate = rr
      }
     | (A {a = x}, _) <- s
     , (A {a = y}, _) <- s
     , x > y
     , let rr = (f x) * m (mset [A{a = x}, A{a = y}]) s
     , rr > 0 ]
```

Each rule generates a multiset of reactions, as noted earlier, since the same reaction can occur from two different matches. The multiset is given as a list where the order is not important. Function `m` is the μ function that calculates the multiplicity of the match

(defined in §4.2). The multiplicity also makes sure that the two patterns do not match the same agent in the state. The comprehension allows that, which means we create some spurious reactions, but the rate of those will be 0. The rate function, `f`, is provided by the outside Haskell environment. This is an example where we leverage the strength of the host language.

5.3 Fluent and Observable features (enriched expressions)

Enriched expressions are also embedded using Quasi-quotes that allows the direct use of their syntax. The implementation provides a quoter function, `er`, for the translation of the surface syntax to its interpretation. The interpretation of an enriched expression is a function from states and time to values in V, exactly like our formal definition in §4.3.2. In Haskell we use the following type:

```
data Er a b = Er { at :: Multiset a -> Time -> b}
```

parameterised by the agent type (`a`) and the return type (`b`).

The surface syntax for enriched expressions is the same as the abstract one with the exception of the where construct. The most common use-case of the where construct is to mix expressions from the provided language E with fluents and observables. For example being able to write $nl + 1$ for some observable nl. In the current implementation we only cover this particular sub-case where variables referring to enriched expressions are embedded inside Haskell expressions. Instead of being 'tagged' with the where clause, we surround them with `$..$` symbols. This resolves the problem of mixing simple and functional values. The functional values (inside `$..$`) will be interpreted to their simple value at current state and time. Other local definitions of enriched expressions can be done using the restricted scope declaration mechanisms of the host language (`let` and `where` in Haskell). All other forms of enriched expressions remain the same as in the grammar provided in §4.3.1.

We omit a formal translation from surface syntax to the Haskell `Er`'s but go through an example to illustrate the `er` quoter function. Consider for instance our enriched expression example from §4.3, giving alternative rate functions depending on the current number of leaves in the plant:

$$f \textbf{ when } nl > 10 \textbf{ else } f'$$

where *nl* is an observable for the number of leaves. This will be written as:

```
rateF = [er| f when $nl$ > 10 else f'|]
  where
    nl = [er| select Leaf{age=i,mass=m}; aggregate (count. count + 1) 0 |]
```

Note that the `$nl$ > 10` to embed an enriched expression inside a normal Haskell expression instead of the n **where** *n* **is** nl construct from our abstract syntax. The number of leaves observable (*nl*) is given using Haskell's `where` clause. The `f` and `f'` functions come from the outside Haskell environment. The above `rateF` quote will be translated to:

```
[er| f when $nl$ > 10 else f'|] = Er {
  at = \s t -> if at [er|$nl$ > 10|] s t
                then at [er|f|] s t
                else at [er|f'|] s t }

[er|$nl$ > 10|] = Er { at = \s t -> at nl s t > 10 }
[er|f|] = Er { at = \s t -> f }
[er|f'|] = Er { at = \s t -> f' }
```

The first equality is an implementation of the semantics of conditional expressions in §4.3.2. The rest follow the semantics of ordinary expressions. Note how we unfold the enriched expression, nl, to its evaluation (`at nl s t`) to achieve the same effect as evaluating the expression in an extended environment (as defined in the semantics of the where construct, §4.3.2). The interpretation of observables will depend on the ordering imposed by our list representation of multisets. While this is not consistent with the semantics, we do not see any problems arising from that as the difference only occurs for non-commutative fold functions, which, in any case, the programmer should not employ (see *Semantics of enriched expressions*, §4.3.2, for a discussion of the validity of the folding function when using linearised multisets). The translation for other types of enriched expressions similarly follows the semantics given in §4.3.2.

5.4 Conclusion

The particular choice I made here for the implementation of Chromar as an embedded DSL in Haskell made it easier to have a first prototype implementation to make

the language practically usable. Several features of Haskell particularly helped with that like pattern matching, which I used in the implementation of rules, record types that are similar to Chromar agent types, and facilities for meta-programming through quasiquotes, which allowed the mix of domain specific and host syntax. The choice of using Haskell expressions and types also followed although a different choice could have been made. However, the availability of Haskell expressions and types as well the fact that we are inside Haskell made model definitions easier, for example when implementing large models like the ones I describe in Chapter 6.

Chapter 6

Framework Models

As we have seen in the Introduction plant models are rarely comprehensive enough to reconstruct the organism and evolutionary ecology models usually abstract the organism away considering only its development as a numerical variable (phenology). The first version of the Framework Model (FMv1; Chew, Wenden et al., 2014) is an attempt at reproducing a 'digital Arabidopsis' inspired by functional-structural models that consider molecular mechanisms as well as development. It only represents vegetative growth, which limits it applicability in ecological studies (§1.2.1). Other models have considered reproductive success through growth, including for Arabidopsis. One simplified approach relates growth and fitness only to the duration of the developmental period and not to its timing in the year, ignoring environmental influences (Prusinkiewicz, Erasmus et al., 2007). On the other hand recent ecology models have taken a more systems view considering the phenology of the entire lifecycle of Arabidopsis plants in a natural setting and even scaled this view to the population level using individual-based models (Liana T. Burghardt et al., 2015). However, the organism is only represented in the usual ecological way as a numerical variable to represent its conceptual development over time ignoring physical aspects of growth and development.

Here I combine the two approaches to present a population level model where the organism is present. In particular in this chapter I do the following:

- Present *FM-lite*, a simplified version of FMv1 (Chew, Wenden et al., 2014) represented in Chromar for studies that do not focus on circadian timing. It represents three of the four constituent models of FMv1 with modifications for real weather condition and without the clock and Flowering Time (FT) genetic circuits (§6.2.2).

- Present *FM-life*, an extension of the FM-lite to the whole Arabidopsis life cycle represented declaratively in Chromar. FM-life includes FM-lite to represent vegetative growth, and a new model of inflorescence growth including reproduction. As well as growth, timing of the developmental stages is given by phenology models, for example for seed dormancy. (§6.2).

- Scale FM-life to the population level by introducing a clustering approximation in order to simulate FM-life tractably at the population scale over decades (§6.3).

- Show simulation results (ecological measure of interest) of the population-level model with contrasting environmental and genetic inputs to highlight that ecological questions can increasingly be informed by mechanistic understanding of growth processes (Doebeli et al., 2017; Andrew J Millar, 2016) (§6.4).

The work in the chapter is under review for the Journal of Experimental Botany and appears as a bioRxiv pre-print (Zardilis et al., 2018).

6.1 Phenology models in Chromar

In many phenology models, the simulated plant accumulates a conceptual development indicator in every time unit as a function of the contributing environmental factors, until a threshold is reached for transition to the next developmental stage. For example, in a seed type $\text{Seed}(\text{dev} : \text{real})$, the dev attribute measures development towards germination. A phenology rule for germination affected by temperature and moisture, starting from dev value d, could be:

$$\text{Seed}(\text{dev} = d) \rightarrow \text{Seed}(\text{dev} = d + f(temp, moist))$$

On average once every time unit the dev attribute of a particular seed will be increased from the present value, d, by a function of the contributing factors $temp$ and $moist$. Further parameters might represent how sensitive the seed is to the environmental factors. At the threshold D_t, the seed germinates to a plant and resets the development measure to 0:

$$\text{Seed}(\text{dev} = d) \rightarrow \text{Plant}(\text{dev} = 0)\,[d > D_t]$$

where the expression inside the square brackets is used to indicate conditional activity of the rule. The rule is active only when the expression evaluates to true.

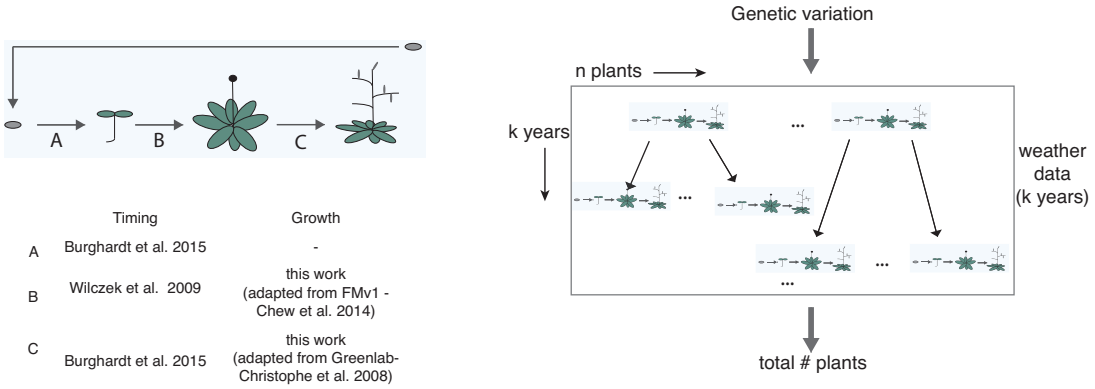


Figure 6.1: Overview of the FM-life and population models used in this study. Left Overview of the models used for the growth and timing components for the three developmental stages: seed dormancy (A), vegetative period (B), and reproductive period (C) and Right Sketch of the population level model. Inputs to the model are the distribution of values of the two genetic parameters (f_i, ψ_i) and weather data from some location for a number of years. The output is some population measure of interest, an example might be the total number of plants after k years.

6.2 FM-life: the component models

The models presented here represent the full life cycle in three stages: seed dormancy (A, left panel, Figure 6.1), vegetative growth up to flowering (B, left panel, Figure 6.1), and the reproductive stage up to seed dispersal (C, left panel, Figure 6.1). Each model (A, B, C) includes a phenology component that represents only timing (Section 6.1). The vegetative and reproductive stage models also represent biomass growth at the organ level, based on the carbon budget of the plant. I varied genetic parameters that affect only the timing components of A (seed dormancy, ψ_i) and B (floral repression during vegetative growth, f_i), for comparison to Liana T. Burghardt et al. (2015). Each parameter value for an individual plant can be fixed or selected probabilistically from a distribution as described (Liana T. Burghardt et al., 2015). The three models were integrated in a whole life-cycle model of one plant (FM-life), and then extended to a population of such plants.

6.2.1 Seed dormancy model (A)

The seed dormancy model is the Chromar version of the model of Liana T. Burghardt et al. (2015), which is based in turn on Alvarado and Bradford (2002). It represents the

development of a newly-dispersed seed from $dev = 0$ to a threshold value, D_g, where the seed germinates. Above baseline levels of temperature T_b and of moisture (see below), increasing moisture and temperature speed the progress towards germination. The additional developmental units added (hydrothermal units, htu) at every time unit are described by:

$$\text{htu}(t) = \begin{cases} (\Psi(t) - \Psi_b(t)) \cdot (T(t) - T_b) & \text{if } T_b < T(t) \leq T_o \text{ and } \Psi_b(t) < \Psi(t) \\ (\Psi(t) - \Psi_b(t)) \cdot (T_o - T_b) & \text{if } \mathrm{T}(t) > T_o \text{ and } \Psi_b(t) < \Psi(t) \\ 0 & \text{otherwise} \end{cases}$$

where $\Psi(t)$ (in MPa) and $T(t)$ (in °C) give the moisture (water potential) and temperature levels at time t respectively. The definition distinguishes between operating in suboptimal and supraoptimal temperatures (below or above T_o respectively). The baseline moisture is used to represent the dormancy level of the seed. If Ψ_b is high, the seed accumulates htu slowly for a given set of environment conditions, whereas if Ψ_b is low, development is faster in the same conditions. From an initial dormancy level, ψ_i, seeds lose dormancy (Ψ_b becomes smaller) over time at a rate r that is also a function of the environmental conditions, moisture and temperature, and represents the observed process of after-ripening. ψ_i is also used to represent the genetic effect on dormancy, where high ψ_i represents stronger dormancy.

In Chromar, the Seed type captures information about the seed development process: $\text{Seed}(\text{gntp} : (\text{real}, \text{real}), \text{dev} : \text{real}, r : \text{real})$. The gntp attribute stores the genotype of the organism, ψ_i (seed dormancy level) and f_i (floral repression level), which is passed on to the agents representing the later stages of development and transmitted unchanged to the next generation. dev stores the cumulative development indicator (sum of htu up to the current timepoint), and r stores the after-ripening up to the current timepoint. The development rule is the following:

$$\text{Seed}(\text{gntp} = a, \text{dev} = d, \text{r} = r) \xrightarrow{1.0}$$
$$\text{Seed}(\text{dev} = d + htu(temp, moist, f(r, temp, moist).\psi_i), \text{r} = f(r, temp, moist))$$

where temp and moist are fluents describing temperature and moisture. We use the 'dot' (.) operator for accessing the two genetic parameters of the gntp attribute. The following rule represents germination, starting the vegetative stage:

$$\text{Seed}(\text{gntp} = a, \text{dev} = d, \text{r} = r) \xrightarrow{1.0}$$
$$\text{Plant}(\text{gntp} = a), \text{Root}(\ldots), \text{Leaf}(\ldots), \text{Leaf}(\ldots)\ [d > D_g]$$

The abstract Plant agent represents the plant at the vegetative stage, along with agents for the root and the two cotyledon leaves. The initial configuration of the organs at germination is as introduced by Chew, Wenden et al. (2014). Note that the genotype attribute is passed from seed to emerged plant unchanged.

In order to illustrate the functioning of the model I show some illustrative simulations of the htu accumulation over a year for real weather data in two locations in Europe, Valencia in Spain and Oulu in Finland, and for two values of the initial base moisture level, ψ_i. In the case of Valencia we can see that there are two periods during the year that are favourable for development towards germination, Spring and Autumn (Figure 6.2 C). Development progresses faster in the low dormancy case ($\psi_i = 0$) than in the high dormancy case ($\psi_i = 2.5$) as expected. The constraints of development to two periods of the year and the binary threshold to germination mean that sometimes if a seed misses the window for germination it has to wait until the next favourable season (Figure 6.2 A).

In the case of Oulu only summer provides favourable conditions for germination (Figure 6.2 D). In the low dormancy case there is progress to germination during the summer while in the high dormancy case there is no development (Figure 6.2 B).

6.2.2 Vegetative growth model (FM-lite) (B)

For the vegetative stage I introduce a simplified version of FMv1 (Chew, Wenden et al., 2014) for use in studies that do not focus on circadian timing. FM-lite has three constituent models represented in Chromar with modifications to environmental responses (see below), and without the fourth, circadian clock and FT gene expression model of FMv1.

Timing

The timing component is the simpler flowering phenology model of Wilczek et al. (2009) rather than the augmented version in FMv1 (combination of Chew, Wilczek et al. (2012) with Salazar et al. (2009)). Vegetative development extends from $dev = 0$ to a threshold value, D_f, where the plant flowers. The main contributing environmental factors are photoperiod, ambient temperature and vernalisation, giving the modified photothermal units, mptu, at a time t as:

$$\text{mptu}(t) = \text{photoperiod}(t) \cdot \text{thermal}(t) \cdot \text{vernalisation}(t)$$

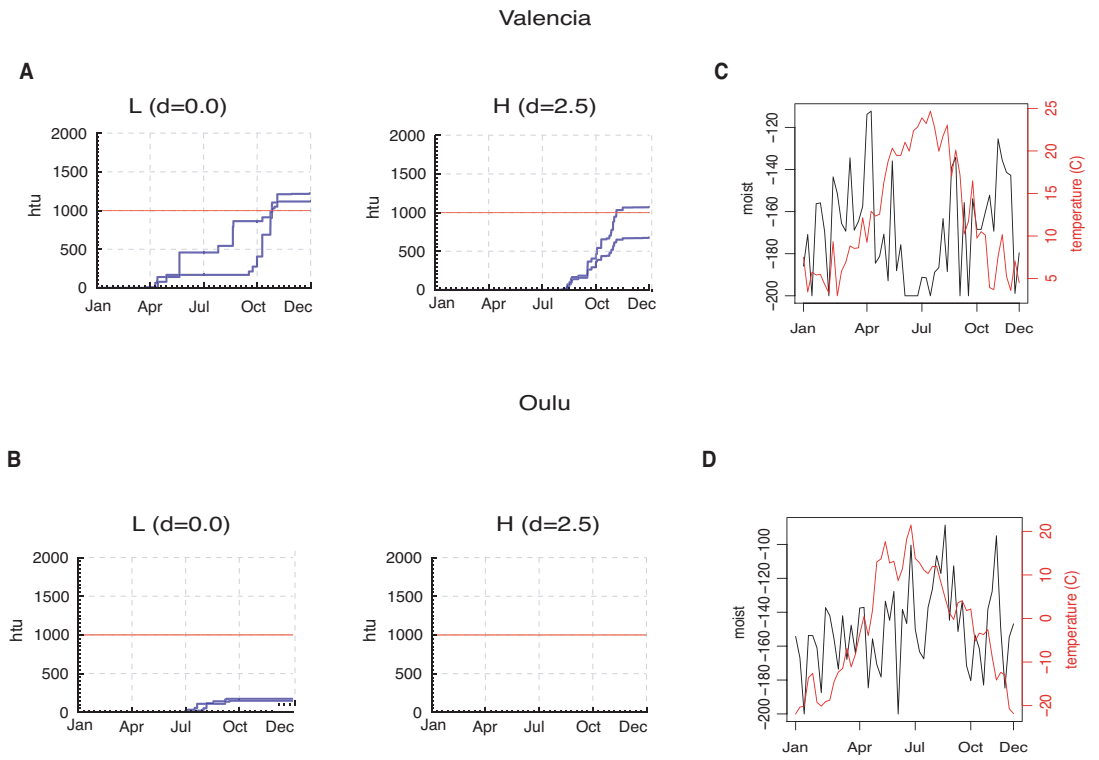


Figure 6.2: Indicative runs of the seed dormancy model for two location and two genetic backgrounds (two values of the initial seed dormancy level) run with a daily timestep. A Two runs of the seed dormancy model for yearly weather in Valencia (C) for two different value of ψ_i. The red line is the germination threshold. Stochasticity comes from the model representation and can sometimes be amplified by the binary threshold and weather constraints to lead to different germination behaviour of otherwise identical seeds (H case) B Same as A for weather data from Oulu (D) C Yearly weekly averages of moisture and temperature levels in Valencia D Weather data as in (C) for Oulu.

The vernalisation term accounts for both the observed requirement for a specific duration of exposure to cold and is also used to represent the genetic effect on the progress towards flowering, modelled as $\text{vernalisation}(t) = f(wc, f_i)$, where wc is the exposure to cold accumulated up to t and f_i is the genetic parameter for the initial floral repression, as in Wilczek et al. (2009).

In Chromar, the plant type: $\text{Plant}(\text{gntp} : (\text{real}, \text{real}), \text{dev} : \text{real}, \text{wc} : \text{real})$ includes the genotype attributes as noted above, the development so far (dev), and finally the accumulated winter chilling (wc). The development rule is then:

$$\text{Plant}(\text{gntp} = a, \text{dev} = d, \text{wc} = w) \xrightarrow{1.0}$$
$$\text{Plant}(\text{dev} = d + mptu(temp, dl, a.f_i, w), \text{wc} = f(w))$$

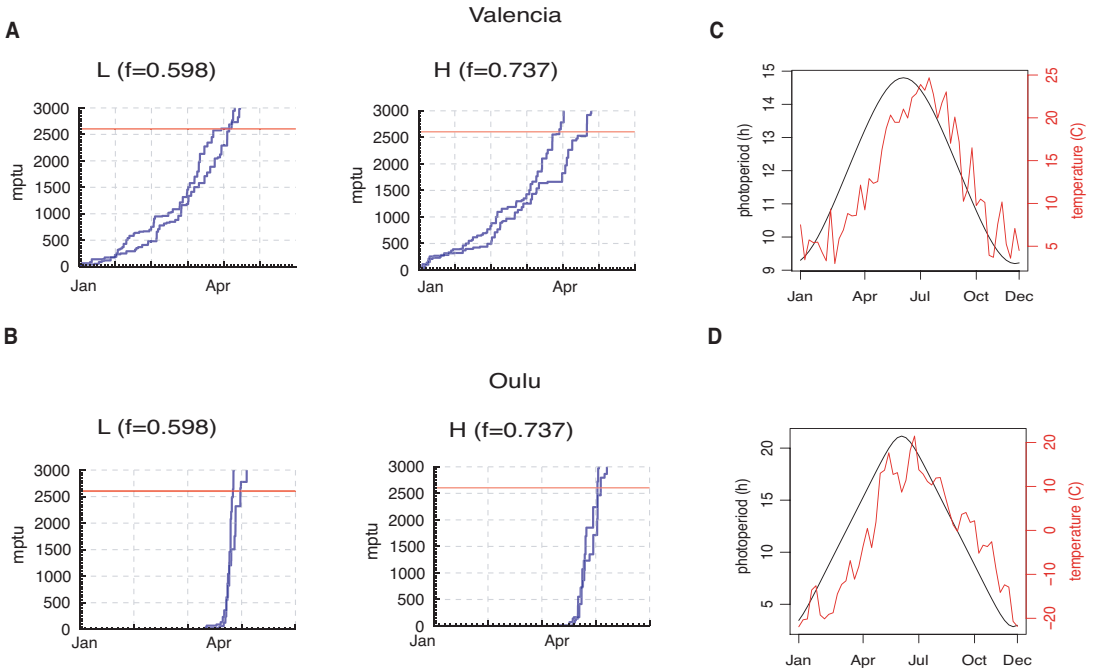


Figure 6.3: Indicative runs of the flowering model for two locations (Valencia, Oulu) and two genetic backgrounds ($f_i = 0.598$, $f_i = 0.737$) run with a daily time-step. A Two runs of the flowering phenology model with yearly weather for Valencia for two values of the initial floral repression level. The red line indicates the flowering threshold B Similar to (A) for Oulu C Weekly averages of photoperiod and temperature for a year in Valencia D Weekly averages of photoperiod and temperature for a year in Oulu.

where $temp$ and dl are fluents for temperature and day length respectively, and w is the present value of wc. The transition to a flowering plant, FPlant, follows:

$$\text{Plant}(\text{gntp}=a, \text{dev}=d, \text{wc}=w) \xrightarrow{1.0} \text{FPlant}(\text{gntp}=a)\ [d > D_f]$$

I again plot some illustrative runs of this model in two different locations in Europe, Valencia and Oulu, and for two values of the initial floral repression, f_i (Figure 6.3). In Valencia there are favourable conditions throughout the year for progress towards flowering even in winter and the floral repression level does not affect the development much (Figure 6.3 A, C). In Oulu progress happens rapidly in spring when temperatures rise above 0 °C coinciding with longer photoperiods (Figure 6.3 B, D).

Growth

As in FMv1 (Chew, Wenden et al., 2014), the growth component includes a carbon budget for the plant from Rasse and Tocquin (2006), which in turn includes photosyn-

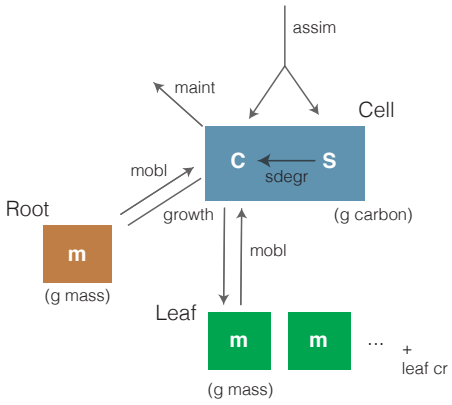
thesis rate equations based on the Farquhar et al. (1980). Growth at the organ level (rosette leaves and root) is represented based on the Greenlab model (Christophe et al., 2008). We will consider a sucrose carbon pool (c), a starch carbon pool (s), and one pool for the biomass of the root and each of the rosette leaves (left panel, Figure 6.4). In Chromar we have the following agents to store the state (amount of carbon, or total biomass) of these pools:

- Cell$(c, s : \text{real})$ An agent that stores the amount of carbon in the sucrose (c attribute) and starch pools (s attribute). The amounts are carbon totals at the whole plant level.
- Leaf$(\text{m} : \text{real}, \text{i} : \text{int})$ An agent that represents a rosette leaf. It has attributes for its mass (m) and its index of appearance (i).
- Root$(\text{m} : \text{real})$ An agent that represents the root with an attribute for its mass (m).

For each organ we have a growth flow from the sucrose carbon pool to the mass of the organ (growth rule, Figure 6.4. The growth amount depends on the demand function of the organ ($d(i,t)$ rule rate function) and its 'sink strength' ($g(m)$), which varies among organs. The value of the demand function varies over time between 1 (maximum demand) and 0 (no demand) at the end of the expansion period of the organ. The amount of carbon requested by an organ at every time unit is $g(m) \cdot d(i,t)$. Depending on the metabolic status of the whole plant (level of c pool) and the requests from other organs, an organ will receive either the full expected amount or a portion of it.

A flow in the opposite direction (mobl rule, Figure 6.4) represents carbon mobilisation from the organs if the central sucrose pool (Cell(c)) is reduced to a critical level. Thus each organ can be either a net sink or source of carbon. For each organ, we also have a flow leaving the system from the c pool for the cost of the maintenance respiration and other processes of the organ (maint rule, Figure 6.4). Photosynthetic carbon fixation is represented by the assimilation process (assim rule, Figure 6.4). The amount of assimilate at every time unit is the product of the photosynthesis rate, which is a function of environmental conditions at that time step, and the projected area of the rosette. Here I use an observable, a_{ros}, for the effective rosette area, which is a function of the global state of the rosette at the current time (derived from the masses of all the current leaves) and takes into account the effect of shading, as in Christophe et al. (2008). The carbon partitioning function includes a baseline partitioning to starch, then support of a target sucrose level, with excess sucrose supporting growth and a final

Flows



Rules

growth

$$\mathrm{Leaf}(\mathrm{m}=m,\mathrm{i}=i),\mathrm{Cell}(\mathrm{c}=c) \xrightarrow{d(i,t)}$$
$$\mathrm{Leaf}(\mathrm{m}=m+g(m)),\mathrm{Cell}(\mathrm{c}=c-g(m))$$
$$[c-g(m)>c_b]$$
similarly for other organs

maint

$$\mathrm{Leaf}(\mathrm{m}=m),\mathrm{Cell}(\mathrm{c}=c) \rightarrow$$
$$\mathrm{Leaf}(\mathrm{m}=m),\mathrm{Cell}(\mathrm{c}=c-f_m(m))$$
$$[c-f_m(m)>0]$$
similarly for other organs

mobl

$$\mathrm{Leaf}(\mathrm{m}=m),\mathrm{Cell}(\mathrm{c}=c) \rightarrow$$
$$\mathrm{Leaf}(\mathrm{m}=m-f_t(m)),\mathrm{Cell}(\mathrm{c}=c+f_t(m))$$
$$[c\leq c_b]$$
similarly for other organs

assim

$$\mathrm{Cell}(\mathrm{c}=c,\mathrm{s}=s) \xrightarrow{\phi(e(t))}$$
$$\mathrm{Cell}(\mathrm{c}=c+k_c a_{\mathrm{ros}},\mathrm{s}=s+(1-k_c)a_{\mathrm{ros}})$$
$$[c\leq c_b]$$

sdegr

$$\mathrm{Cell}(\mathrm{c}=c,\mathrm{s}=s) \rightarrow$$
$$\mathrm{Cell}(\mathrm{c}=c+f_d(e(t)),\mathrm{s}=s-f_d(e(t)))$$
$$[\mathrm{night}\wedge s-f_d(e(t))>0]$$

leaf cr

$$\mathrm{Plant}(),\mathrm{VAxis}(\mathrm{n}=n) \xrightarrow{f_o(e(t))}$$
$$\mathrm{Plant}(),\mathrm{VAxis}(\mathrm{n}=n+1),\mathrm{LAxis}(\mathrm{i}=n+1),$$
$$\mathrm{Leaf}(\ldots)$$

Figure 6.4: An overview of the dynamics on FM-lite (growth component of vegetative stage). The dynamics take the form of flows between different reservoirs of carbon, here shown in a graphical way in the left panel with the corresponding Chromar rules on the right.

overflow to additional starch production, as in Rasse and Tocquin (2006). At night, no photosynthesis occurs and carbon from the starch pools flows to the sucrose pool (sdegr rule, Figure 6.4). Finally, we have the creation of new leaves, which impacts the above processes indirectly by creating more demand for growth and adding maintenance costs (leaf cr rule, Figure 6.4). Leaves are created by the main apical meristem (VAxis agent) along with an LAxis agent that can give rise to lateral branches after flowering (see next section).

It is interesting to note that in FMv1 carbon partitioning between processes and organs is done deterministically whereas in our Chromar representation, partitioning is an emergent, stochastic effect of competition for the finite amount of sucrose carbon in the main reservoir. For example, partitioning of carbon among organs for growth in FMv1 is done explicitly by dividing the demand of each organ by the sum of the demands of all other organs $g(m) \cdot \frac{d(i,\,t)}{\sum d(i,t\,)}$. In the Chromar representation we do not have this explicit division by the global demand, which means that the amount of carbon that an organ gets is higher at each growth event but growth events are rarer because not all growth request are successful (competition). In order to avoid 'starvation' of organs due to the increased carbon amount I decreased the growth request size to $g(m)/10$ and increased the rate of the growth rule to $10 \cdot d(i,t)$. If $c - g(m)/10 > c_b$ then the request is successful and the organ gets the whole amount for growth otherwise the request is unsuccessful and the organ gets nothing. The competition recovers the explicit partitioning of FMv1 albeit with greater variability even with the smaller request size (see Figure 6.5).

Modifications for natural conditions

FMv1 was developed for lab conditions. As an initial approach to reflect plant responses to the broader range of relevant conditions in nature, we made the following changes:

- The rate of photosynthesis is set to 0 below 0 °C
- The maintenance cost for an organ is also 0 below 0 °C
- The rate of photosynthesis is affected by soil moisture through stomatal closure. The photosynthesis rate is affected by a stomata term $f_{\text{stom}}(moist)$, which is a simple phenomenological function that relates soil moisture and stomatal closure (France, Thornley et al., 1984).

These conservative changes give a lower bound on the effects of natural weather conditions.

Comparison of FM-lite with FMv1

In addition to the weather responses, Wilczek flowering model and emergent carbon partitioning among organs, our model representation uses the stochastic rule-based Chromar as opposed to the deterministic Matlab program of FMv1. In order to compare the model representations, I simulated growth in the two models for a fixed number of hours in lab conditions, where the modifications to weather responses have no effect. The two models were simulated in lab conditions (22 °C, 12/12 light/dark cycles) for 800 growth hours and showed comparable results (Figure 6.5). FMv1 was simulated in Matlab while FM-lite was simulated in the Haskell implementation of Chromar and the results were averaged over five runs. The rosette mass results are the closest since they represent the development of multiple Leaf agents, masking the stochastic effects on each Leaf. The difference between the final rosette mass of the FMv1 and FM-lite (averaged over 5 runs) simulations is within 10% of the final rosette mass in FMv1. The stochasticity is more apparent for the root where the growth curves are further apart. The difference between final root mass in FMv1 and FM-lite (averaged over 5 runs) is ~20% of the final root mass in FMv1. Sucrose carbon levels are also more variable in FM-lite, since the growth rule (removing sucrose carbon from the central pool) provides organs with a larger amount but less frequently than the small fixed amount at every time step in FMv1 (see previous section).

6.2.3 Reproductive stage model (C)

Timing

The timing component is a thermal time model from Liana T. Burghardt et al. (2015), representing the development of the inflorescence and seed from $dev = 0$ at flowering, to a threshold value, D_s, where the plant disperses its seeds. Here there is no genetic input. Moreover, there is no input from the vegetative or growth model from the same stage so for example a plant with a small rosette that makes very little fruit will mature its seed at the same time as a very branchy plant with a lot of fruit. This is a simplification and in fact throughout the FM-life model we assume independence of timing with growth although the duration of the period will indirectly affect growth (see indicative simulations below, Figure 6.8). The thermal units that accumulate at time t are simply the value of the temperature at t above a base temperature T_b:

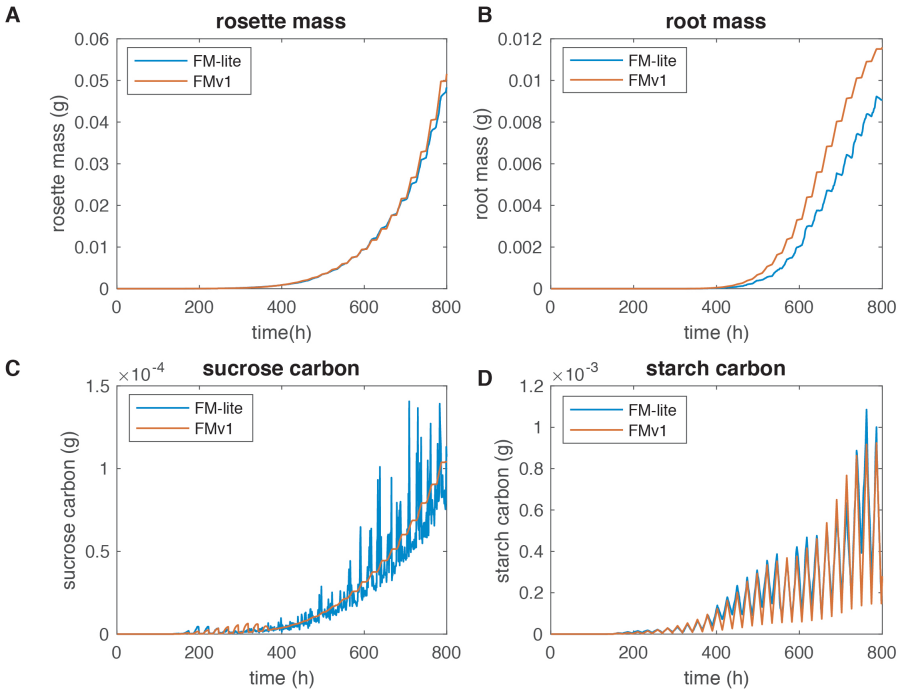


Figure 6.5: A comparison of the original FM implementation (FMv1) with the adaptation used in this work (FM-lite) for 800 hours of growth. FM-lite simulations were performed in the Haskell implementation of Chromar and results were averaged over five runs. FMv1 simulations were carried out in the Matlab environment. A Comparison of simulated rosette mass trajectories between FMv1 and FM-lite B Comparison of simulated root mass trajectories between FMv1 and FM-lite C Comparison of simulated sucrose carbon between FMv1 and FM-lite and D Comparison of simulated starch carbon between FMv1 and FM-lite.

$$tu(t) = \begin{cases} T(t) - T_b & \text{if } T(t) > T_b \\ 0 & \text{otherwise} \end{cases}$$

Writing into Chromar we have an FPlant(dev : real) type for a flowered plant and the following rule for its development that follows from the tu definition above:

$$\text{FPlant}(\text{dev} = d) \rightarrow \text{FPlant}(\text{dev} = d + tu(temp))$$

Finally, the transition to seed happens when the accumulated development reaches D_s:

$$\text{FPlant}(\text{attr} = a,\ \text{dev} = d) \rightarrow \text{Seed}(\text{attr} = a,\ \text{dev} = 0,\ r = 0)\ \ [d > D_s]$$

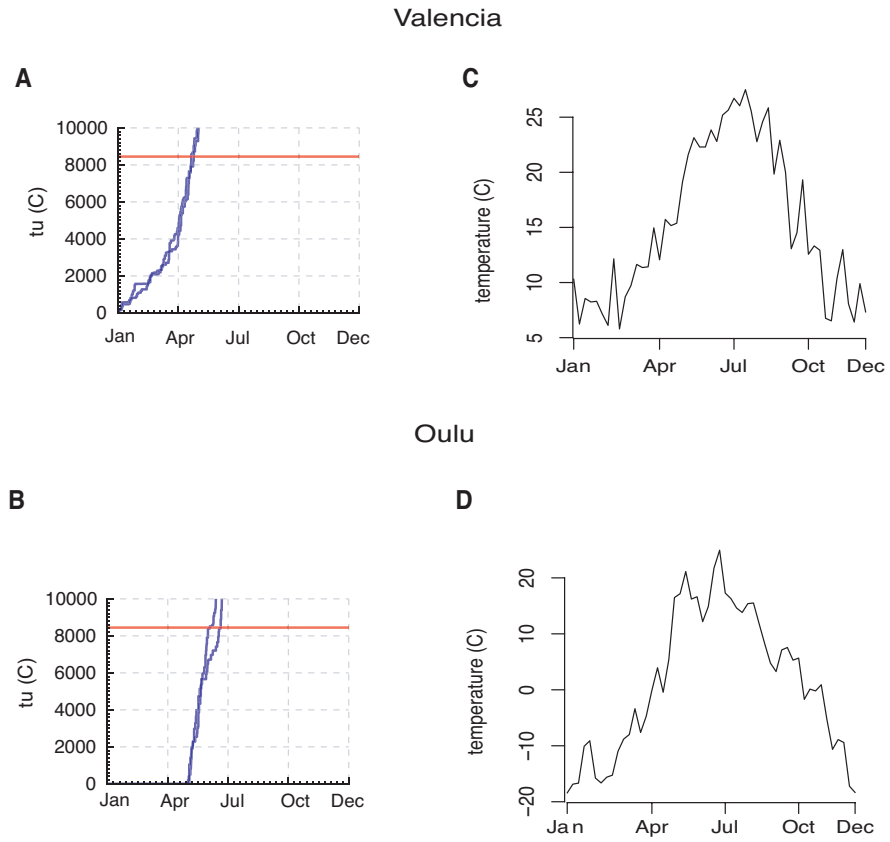


Figure 6.6: Indicative runs of the timing to seed dispersal model run with daily timestep. A Since temperature is favourable in Valencia throughout the year development takes place continuously even in the winter. B In Oulu development towards seed dispersal can happen only in late spring, summer, and early autumn when temperatures are favourable. Development then starts accumulating rapidly in late spring. C Weekly temperature averages for year in Valencia D Weekly temperature averages for a year in Oulu.

Note that the genotype attribute of the parent plant is transferred to the seeds unchanged.

As in the other phenology models I show some illustrative simulations of this model in the two standard locations, Valencia and Oulu (Figure 6.6). In Valencia there is development throughout the year whereas in Oulu the conditions are only favourable for only part of the year from spring to autumn where the temperatures are above freezing.

Growth

I developed the growth component of the reproductive stage model, which is loosely related to the Greenlab model (Christophe et al., 2008). The metabolic processes affecting the carbon budget of the plant are the same as in vegetative growth but with

additional organ types to represent the Arabidopsis inflorescences. Organs appear in units (metamers) with a metamer identifier. Each growth unit on the main axis consists of an internode (stem between leaves), a leaf, and a lateral meristem that can give rise to a lateral axis. I consider only the primary and secondary axes (lateral branches), thus metamers on the lateral axis lack a further lateral meristem. All fruits on an axis are represented on its last metamer, replacing the leaf; this metamer also lacks a meristem. Two indices represent metamer position: the index of the metamer along its axis and the index of the parent metamer along the primary axis (left panel, Figure 6.7). We define the following new agent types to represent this structure:

- INode$(i, pi : \text{int}, m : \text{real})$ to represent the internode (stem between successive leaves). Attribute i is the temporal index of appearance in its axis (primary or lateral) and attribute pi is the parent primary metamer. The cotyledons have indices 1 and 2 on the primary axis, for example.

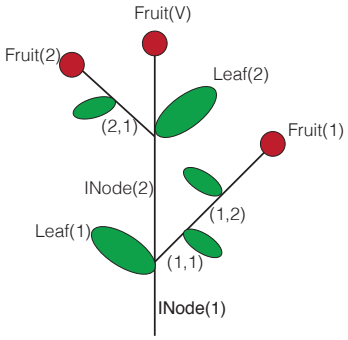
- LLeaf$(i, pi : \text{int}, m : \text{real})$ to represent a leaf on the lateral axes.

- Fruit$(i, pi : \text{int}, m : \text{real})$ to represent a fruit on the axis.

The maximum number of inflorescence metamers on the main axis is taken to be 20% of the number of rosette leaves at flowering time (n_f) and given by $v_{\max}(n_f)$ (Pouteau and Albertini, 2009). The maximum number of growth units on each lateral axis is given by $l_{\max}(i)$, a decreasing function of the index of the lateral axis starting from a maximum of 6 at the axis after the cotyledons (index 3) and going to a minimum of 1 at the topmost lateral branch (Mündermann et al., 2005). The topmost lateral axis can only appear with a delay after the apical fruit has appeared on the primary axis. Each successive lateral branch going down can only start developing with a delay after the fruit of the axis above it has appeared. The delay associated with lateral axis growth, given in the rules by t_{del}, is a function of the metabolic state of the plant, as described (Christophe et al., 2008).

The new organ types have associated sink strengths and demand functions. The cauline leaves on the main axis contribute to the photosynthetically active area and can shade the rosette leaves underneath them. The lateral leaves contribute to photosynthesis without shading. Internodes and fruits do not contribute to photosynthesis. Seeds are not directly represented, so a birth function $b(m)$ is required to calculate the number of seeds for a given fruit mass m at seed dispersal time, as described below.

Structure



Rules

vgrowth
$$\mathrm{FPlant}, \mathrm{VAxis}(\mathrm{n}=n) \xrightarrow{f(e(t))}$$
$$\mathrm{FPlant}, \mathrm{VAxis}(\mathrm{n}=n+1)$$
$$\mathrm{INode}(\mathrm{i}=n+1, \mathrm{pid}=V\,n+1),$$
$$\mathrm{Leaf}(\mathrm{i}=n+1, \ldots)$$
$$[n < v_{\max}(n_f)]$$

vfruit
$$\mathrm{FPlant}, \mathrm{VAxis}(\mathrm{n}=n) \xrightarrow{f(e(t))}$$
$$\mathrm{FPlant}, \mathrm{VAxis}(\mathrm{n}=n+1), \mathrm{Fruit}(\mathrm{pi}=V\,n+1)$$
$$[n = v_{\max}(n_f)]$$

lgrowth
$$\mathrm{FPlant}, \mathrm{LAxis}(\mathrm{i}=i, \mathrm{n}=n), \mathrm{Fruit}(\mathrm{pi}=V\,i+1) \xrightarrow{f(e(t))}$$
$$\mathrm{FPlant}, \mathrm{LAxis}(\mathrm{n}=n+1), \mathrm{Fruit},$$
$$\mathrm{INode}(\mathrm{i}=n+1, \mathrm{pi}=L\,i), \mathrm{LLeaf}(\mathrm{i}=n+1, \mathrm{pi}=L\,i)$$
$$[n < l_{\max}(n_f, i) \wedge t_{\mathrm{del}}]$$

lfruit
$$\mathrm{FPlant}, \mathrm{LAxis}(\mathrm{i}=i, \mathrm{n}=n) \xrightarrow{f(e(t))}$$
$$\mathrm{FPlant}, \mathrm{LAxis}(\mathrm{n}=n+1), \mathrm{Fruit}(\mathrm{pi}=L\,i)$$
$$[n = l_{\max}(n_f, i)]$$

+ metabolic processes (see Figure 6.4)

Figure 6.7: Overview of the structural part of the growth component of the reproductive stage model. The numbering scheme used to keep track of the positions of the organs in the inflorescence architecture is shown on the left. On the right the Chromar rules used to grow a structure like the one on the left panel (see main text for details).

I plot two indicative runs of the vegetative + reproductive stage models at two different temperatures to show the interplay between growth and seed dispersal timing (Figure 6.8). At 22 °C we get two fruit agents and one lateral branch at seed dispersal (1379 hours after sowing) with a total of 14 out of 25 non-senesced leaves on the main axis (Figure 6.8A, B). While the photosynthesis rate is higher at 25°C, the growth period is shorter, which means less growth time in the exponential growth period resulting in a smaller plant and less fruit mass (Figure 6.8 C, D).

6.3 From the lifecycle to the population

The Chromar framework allows us simply to concatenate the rules of timing and growth components of the three models above, to represent the whole life cycle (Figure 6.9). Then given an initial state with the genetic attributes of the plant (gntp attribute of agents) and the environmental conditions for a particular location, $e(t)$, we can simulate an entire life cycle from seed to seed. The timing components of the model give us the timing within the year of the growth period (vegetative + reproductive stages) and therefore the environmental conditions that the plant is exposed to during growth. The growth components predict growth at the individual organ level with these environmental conditions and therefore give us the environmentally determined seed number given by the $b(m)$ function, which can depend on growth (mass m of fruits).

6.3.1 Population level model and plotting conventions

Since FM-life estimates the number of seeds at the end of the life cycle, these can initiate multiple independent copies of the model in the next generation. We then have a classical evolutionary birth process, sometimes called a branching process since it unfolds in tree-like way. The potential number of individuals in generation i, n_i, is equal to the sum of the number of seeds produced by the individuals in the previous generation (see Discussion). Dormant seed never die in the model and may germinate after several years (Liana T. Burghardt et al., 2015).

Since we are using an individual-based model, n_i becomes computationally prohibitive to simulate over decades of population growth. In order to overcome this limitation, I simulated the timing (phenology) and growth components sequentially and used conservative birth functions $b(m)$ where m is the final fruit mass at seed dispersal. Figure 6.10 introduces the plotting conventions for these results. The timing components

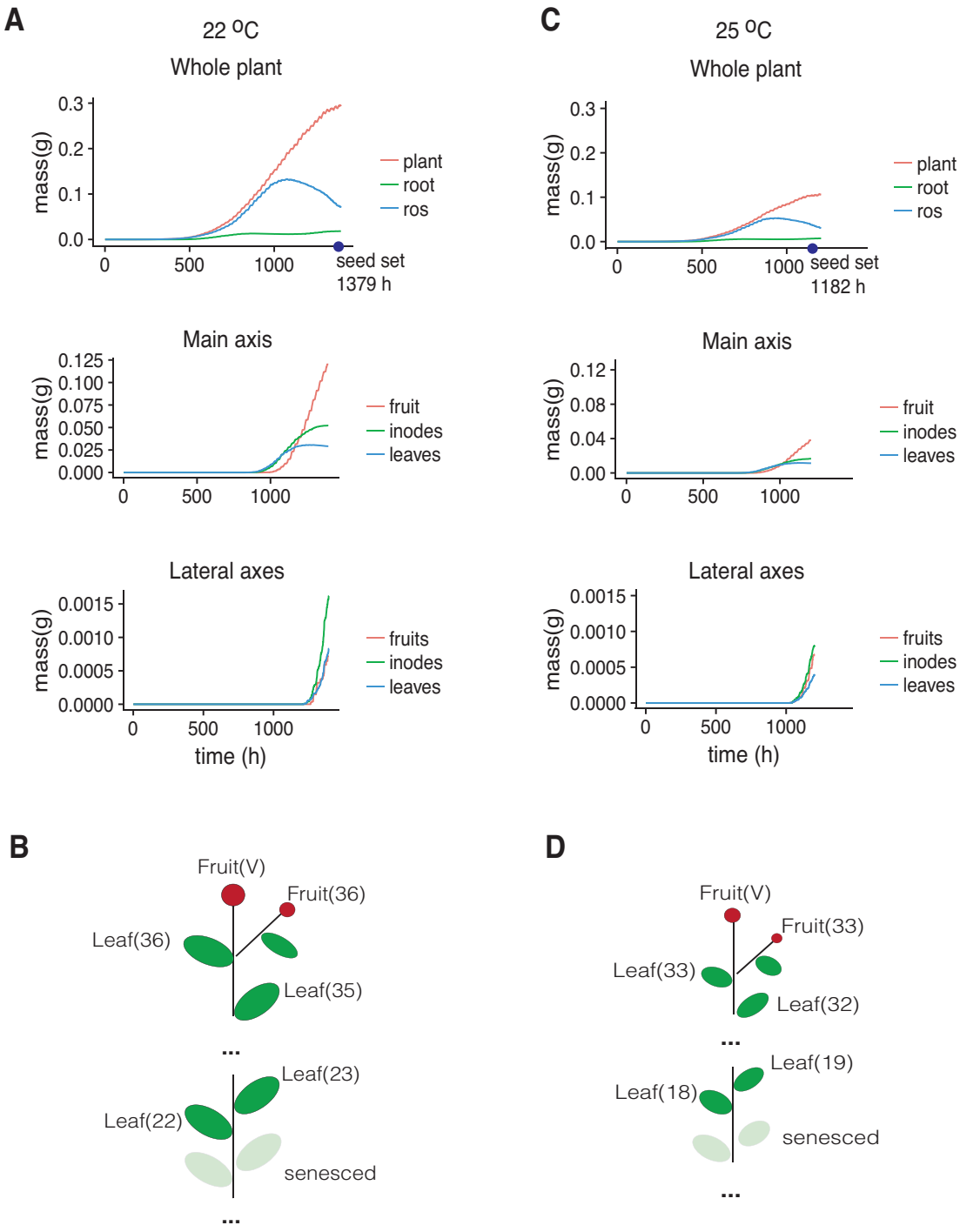


Figure 6.8: Indicative runs of the vegetative + reproductive stage models (germination to seed dispersal). A 5-run averages of growth period indicative biomass traces (whole plant, main axis, lateral axes) for growth at 22 ℃ (and lab conditions, light intensity=120μmol $\cdot$ $m^{-2} \cdot s^{-1}$, CO2=420ppm). B Sketch of the structure of the inflorescence at seed dispersal (1379 h) C Growth period simulations with same conditions as in (A) but at 25 ℃. D Same as in (B) for growth at 25 ℃.

Phenology

$\text{Seed}(\text{gntp}=a,\text{dev}=d,\text{r}=r) \xrightarrow{1.0}$
$\text{Seed}(\text{dev}=d+htu(temp,moist,f(r,temp,moist),a.\psi_i),\text{r}=d_r(r,temp,moist))$

$\text{Seed}(\text{gntp}=a,\text{dev}=d,\text{r}=r) \xrightarrow{1.0} \text{Plant}(\text{gntp}=a),\text{Root}(\ldots),\text{Leaf}(\ldots),\text{Leaf}(\ldots)\ [d>D_g]$

$\text{Plant}(\text{gntp}=a,\text{dev}=d,\text{wc}=w) \xrightarrow{1.0} \text{Plant}(\text{dev}=d+mptu(temp,dl,a.f_i,w),\text{wc}=d_w(w))$

$\text{Plant}(\text{gntp}=a,\text{dev}=d,\text{wc}=w) \xrightarrow{1.0} \text{FPlant}(\text{gntp}=a)\ [d>D_f]$

$\text{FPlant}(\text{dev}=d) \xrightarrow{1.0} \text{FPlant}(\text{dev}=d+tu(temp))$

$\text{FPlant}(\text{attr}=a,\ \text{dev}=d) \xrightarrow{1.0} \text{Seed}(\text{attr}=a,\ \text{dev}=0,\ r=0)\ \ [d>D_s]$

Metabolism

growth (per organ type)

$\text{Leaf}(\text{m}=m,\text{i}=i),\text{Cell}(\text{c}=c) \xrightarrow{d_l(i,t)} \text{Leaf}(\text{m}=m+g_l(m)),\text{Cell}(\text{c}=c-g_l(m))[c-g_l(m)>c_b]$

$\text{Root}(\text{m}=m),\text{Cell}(\text{c}=c) \xrightarrow{d_r(t)} \text{Root}(\text{m}=m+g_r(m)),\text{Cell}(\text{c}=c-g_r(m))[c-g_r(m)>c_b]$

$\text{LLeaf}(\text{m}=m),\text{Cell}(\text{c}=c) \xrightarrow{d_{ll}(t)} \text{LLeaf}(\text{m}=m+g_{ll}(m)),\text{Cell}(\text{c}=c-g_{ll}(m))[c-g_{ll}(m)>c_b]$

$\text{INode}(\text{m}=m),\text{Cell}(\text{c}=c) \xrightarrow{d_i(t)} \text{INode}(\text{m}=m+g_i(m)),\text{Cell}(\text{c}=c-g_i(m))[c-g_i(m)>c_b]$

$\text{Fruit}(\text{m}=m),\text{Cell}(\text{c}=c) \xrightarrow{d_f(t)} \text{Fruit}(\text{m}=m+g_f(m)),\text{Cell}(\text{c}=c-g_f(m))[c-g_f(m)>c_b]$

maintenance (per organ type)

$\text{Leaf}(\text{m}=m),\text{Cell}(\text{c}=c) \xrightarrow{1.0} \text{Leaf}(\text{m}=m),\text{Cell}(\text{c}=c-r_l(m))$

$\text{Root}(\text{m}=m),\text{Cell}(\text{c}=c) \xrightarrow{1.0} \text{Root}(\text{m}=m),\text{Cell}(\text{c}=c-r_r(m))$

$\text{LLeaf}(\text{m}=m),\text{Cell}(\text{c}=c) \xrightarrow{1.0} \text{LLeaf}(\text{m}=m),\text{Cell}(\text{c}=c-r_{ll}(m))$

$\text{INode}(\text{m}=m),\text{Cell}(\text{c}=c) \xrightarrow{1.0} \text{INode}(\text{m}=m),\text{Cell}(\text{c}=c-r_i(m))$

$\text{Fruit}(\text{m}=m),\text{Cell}(\text{c}=c) \xrightarrow{1.0} \text{Fruit}(\text{m}=m),\text{Cell}(\text{c}=c-r_f(m))$

mobilisation (per organ type)

$\text{Leaf}(\text{m}=m),\text{Cell}(\text{c}=c) \xrightarrow{1.0} \text{Leaf}(\text{m}=m-t_l(m)),\text{Cell}(\text{c}=c+t_l(m))\ [c\le c_b]$

$\text{Root}(\text{m}=m),\text{Cell}(\text{c}=c) \xrightarrow{1.0} \text{Root}(\text{m}=m-t_r(m)),\text{Cell}(\text{c}=c+t_r(m))\ [c\le c_b]$

$\text{LLeaf}(\text{m}=m),\text{Cell}(\text{c}=c) \xrightarrow{1.0} \text{LLeaf}(\text{m}=m-t_{ll}(m)),\text{Cell}(\text{c}=c+t_{ll}(m))\ [c\le c_b]$

$\text{INode}(\text{m}=m),\text{Cell}(\text{c}=c) \xrightarrow{1.0} \text{INode}(\text{m}=m-t_i(m)),\text{Cell}(\text{c}=c+t_i(m))\ [c\le c_b]$

$\text{Fruit}(\text{m}=m),\text{Cell}(\text{c}=c) \xrightarrow{1.0} \text{Fruit}(\text{m}=m-t_f(m)),\text{Cell}(\text{c}=c+t_f(m))\ [c\le c_b]$

assimilation

$\text{Cell}(\text{c}=c,\text{s}=s) \xrightarrow{\phi(e(t))} \text{Cell}(\text{c}=c+k_c a_{\text{plant}},\text{s}=s+(1-k_c)a_{\text{plant}})[c\le c_b]$

starch

$\text{Cell}(\text{c}=c,\text{s}=s) \rightarrow \text{Cell}(\text{c}=c+f_d(e(t)),\text{s}=s-f_d(e(t)))\ [\text{night}\wedge s-f_d(e(t))>0]$

Development

$\text{Plant}(),\text{VAxis}(\text{n}=n) \xrightarrow{f_l(e(t))} \text{Plant}(),\text{VAxis}(\text{n}=n+1),\text{LAxis}(\text{i}=n+1),\text{Leaf}(\ldots)$

$\text{FPlant},\text{VAxis}(\text{n}=n) \xrightarrow{f_l'(e(t))}$
$\text{FPlant},\text{VAxis}(\text{n}=n+1),\text{INode}(\text{i}=n+1,\text{pid}=V\,n+1),\text{Leaf}(\text{i}=n+1,\ldots)\ [n<v_{\max}(n_f)]$

$\text{FPlant},\text{VAxis}(\text{n}=n) \xrightarrow{f_r(e(t))} \text{FPlant},\text{VAxis}(\text{n}=n+1),\text{Fruit}(\text{pi}=V\,n+1)\ [n=v_{\max}(n_f)]$

$\text{FPlant},\text{LAxis}(\text{i}=i,\text{n}=n),\text{Fruit}(\text{pi}=V\,i+1) \xrightarrow{f_{ll}(e(t))}$
$\text{FPlant},\text{LAxis}(\text{n}=n+1),\text{Fruit},\text{INode}(\text{i}=n+1,\text{pi}=L\,i),\text{LLeaf}(\text{i}=n+1,\text{pi}=L\,i)\ [n<l_{\max}(n_f,i)\wedge t_{\text{del}}]$

$\text{FPlant},\text{LAxis}(\text{i}=i,\text{n}=n) \xrightarrow{f_r'(e(t))} \text{FPlant},\text{LAxis}(\text{n}=n+1),\text{Fruit}(\text{pi}=L\,i)\ [n=l_{\max}(n_f,i)]$

Figure 6.9: A list of all the rules for the dynamics of FM-life. Phenology rules determine the timing of the life-cycle transitions. Metabolism rules encode the flow of carbon and starch between the main pools (Cell agent) and the organs. Development rules encode the creation of new organs – in the vegetative stage the creation of new leaves and in the reproductive stage the organs representing the full infloresence structure described in §6.2.3.

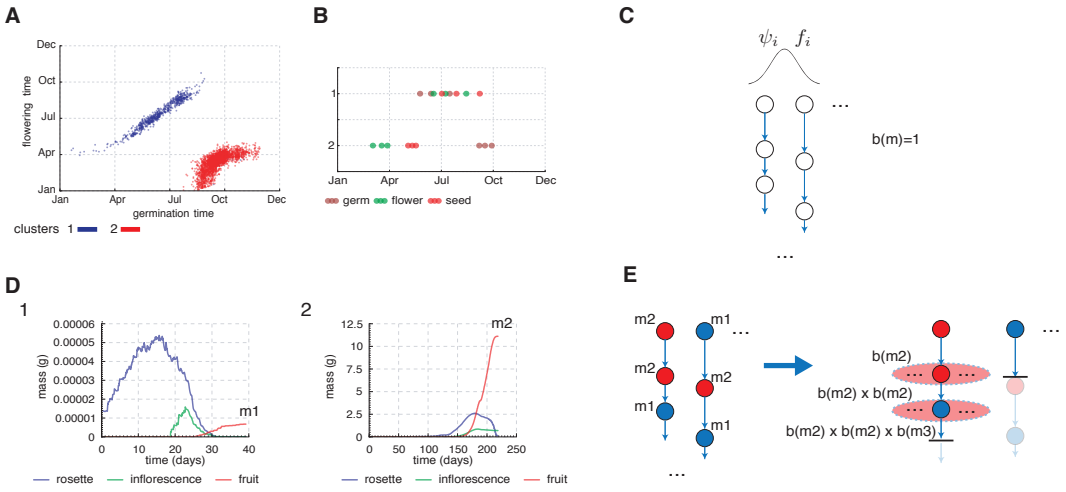


Figure 6.10: The two-stage simulation of the population level model. A The distributions of developmental events (germination, flowering) from the phenology-only simulation of the population model (C) with the identification of two clusters representing two distinct strategies (blue, red) B The 25-th, 50-th, and 75-th percentiles of the distributions of developmental events of the two clusters from A (cluster labels in y-axis). Germination time distributions are in brown, flowering time distributions are in green and seed dispersal distributions in red. C Illustration of the phenology only simulation with growth-independent birth function b(m)=1 (each plant makes a single seed). The choice of, ψ_i (seed dormancy level) and f_i (initial floral repression) is done probabilistically through a normal distribution D Results of simulations of the growth models for the median dates of developmental events (B) for the clusters identified in (A). m_1, m_2 are the final fruit masses for a plant with median developmental event timings for clusters 1 and 2 respectively (as identified and shown in A, B) E Illustration of the assignment of fruit masses to the lifecycles of the phenology-only simulation (C) according to their clusters. m_1, m_2 refer to final fruit masses for a median plant in clusters 1 and 2 (from growth simulations in D and same 2 clusters in A, B). This recovers the full branching population process where the horizontal bars indicate a lineage dying out ($b(m) = 0$ at that generation – no reproduction) and the potential rest of the lineages is shaded. The breadth of the ellipses indicates the reproductive success when there is reproduction ($b(m) > 0$).

were first simulated with $b(m) = 1$, such that each plant makes one seed regardless of fruit mass (since it is not available), as in Liana T. Burghardt et al. (2015) but with a daily time-step. The phenological simulation results in an unbranched sequence of developmental stage timings for each lineage (Figure 6.10 C). The simulation results for several decades typically revealed a small number of life cycle growth strategies, from clusters of individual life cycles. The clusters were generated using k-means clustering, where k is chosen by visual inspection of the life cycle plots (Figure 6.10A). Alternative clustering approaches might be an area for future work. Figure 6.10A shows the distribution over a year of all individual life cycles that conformed to two contrasting life cycle strategies under environmental conditions for Valencia (see Results). Cluster membership depends on the dates and durations of multiple developmental stages. This is hard to visualise, because the timing of any single developmental stage partially overlaps among different strategies. Figure 6.10B therefore summarises the median dates of all three developmental transitions in each strategy, here illustrated by 1. a summer growth strategy and 2. a winter growth strategy. In the next stage, the growth models were simulated once per cluster, with the environmental conditions associated with the typical timing of that cluster (median vegetative and reproductive stages). This returns the typical biomass of organs over time, including the fruit mass at seed dispersal (m1 for cluster 1, m2 for cluster 2; Figure 6.10D). Finally, each life cycle is assigned the fruit mass m associated with its cluster, and thereby a growth-based, birth function $b(m)$ that evaluates to 0 in some cases (no reproduction). Thus, the second stage recovers a version of the branching lineage tree, where some lineages die out (Figure 6.10E).

The output population measure I will use for the results in the next section and to compare scenarios is the total population of plants over all lineages over all generations. I define the population measure for one lineage as follows:

Definition 2 (single-lineage population measure)**.** *The* single-lineage population measure*, p_i, for a lineage i is the total number of plants over all generations.*

$$p_i = 1 + \sum_{k=2}^{n} \left(\prod_{j=1}^{k-1} b(m_{ij})\right)$$

where n is the total number of generations, m_{ij} is the final fruit mass of the plant (given by its cluster membership) in generation j of lineage i, and $b(m)$ is a birth function returning the number of offspring given a fruit mass.

For example, consider lineage 1 with three generations starting with a plant with final fruit mass m_{11}. For the next generation we have $b(m_{11})$ plants and then $b(m_{11}) \times b(m_{21})$.

The population measure for that lineage is $1 + b(m_{11}) + b(m_{11}) \times b(m_{21})$.

The population measure for multiple lineages starting from multiple plants in the initial population is the sum of the population measures of all the lineages.

Definition 3 (multiple-lineage population measure)**.** *The* multiple-lineage population measure*, P, for N lineages is the sum of the single-lineage population measure for all of them.*

$$P = \sum_{i=1}^{N} p_i$$

For the results in the next section and throughout the rest of this chapter I report, unless otherwise stated, the multiple-lineage population measure. The above definitions require a birth function, which I use in a very simple form, as follows:

$$b(m) = \begin{cases} 1 & \text{if } m > m_0 \\ 0 & \text{otherwise} \end{cases}$$

A plant produces one seed or none, the latter in life cycles with fruit mass at seed dispersal m less than a threshold m_0. Below, I make some conservative choices for the value of the reproductive threshold value, m_0, to explore the effect on the output population measure.

To illustrate the resulting process consider the populations plots over time in Figure 6.11 for two locations, Valencia and Oulu, and two combinations of genetic parameters for dormancy level and initial floral repression. I use the above binary birth function. This does not show the cumulative population measure from above but rather how many plants are present at any instance for a period from year 15 to year 60 of a 60 year simulation. Since the output of the birth function is less than 1 the population either stays constant or decreases over time as lineages die out when a plant in a generation does not have fruit mass above the threshold value. For any other function that at least distinguishes the reproductive success of individuals (e.g. b(m) = {0, 1, 2}) I could have used the population at the end of the simulation as the population measure. With the current birth function, however, it is impossible to distinguish between strategies so I use the cumulative population measure (see above) that carries, even if only conservatively, the reproductive success over time.

Finally, I distinguish three sources of variability in the population model: (i) weather varies between years, (ii) genetic parameters can vary among the initial population if their values are chosen probabilistically, and (iii) simulation results vary due to stochasticity in the model representation.

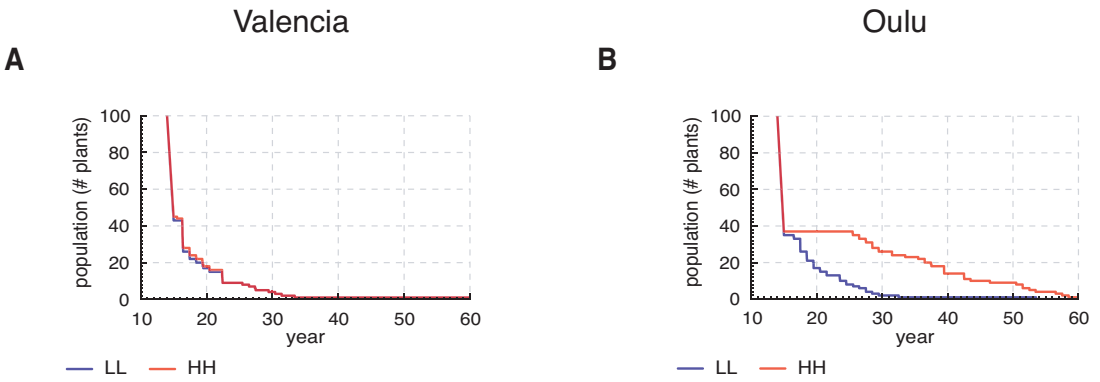


Figure 6.11: Populations plots over time for two locations, Valencia and Oulu, and two genetic backgrounds (two values of seed dormancy and initial floral represssion levels). A Number of plants present at any instance from year 15 to year 60 of a 60-year simulation in Valencia. B Similar to A but in Oulu.

6.3.2 Weather data

For the phenology model simulations I used the weather data that accompanied the Liana T. Burghardt et al. (2015), available from a Dryad repository (L. Burghardt et al., 2014). In this dataset weather inputs over 60 years were generated stochastically for four locations in Europe: Halle, Valencia, Norwich, and Oulu. The weather inputs include values for temperature (in °C), moisture (water potential in MPa), and daylength (in hours).

For the growth simulations I used weather data from the ECMWF ERA-Interim dataset over the years 2010-2011 (Dee et al., 2011). A program provided by Mathew Williams and Luke Smallman (School of GeoSciences, University of Edinburgh) that uses methods from Williams et al. (2001) was used to generate hourly inputs given daily averages from the dataset for temperature and radiation. For the soil moisture input used in the photosynthesis rate calculation I used a daily average of soil moisture values from the dataset and assumed that is constant throughout the day (swvl parameters in the ERA dataset). The soil moisture parameter here is a number in arbitrary units from 0 to 1 that represents the 'wetness' of the soil while the soil moisture used above measure water potential and is given in MPa.

6.4 Results

The population of FM-life models (see previous section) allows us to test how growth processes that alter reproductive success affect the life history strategies of Arabidopsis growing in different environmental conditions (location) and with different genetic parameters in the initial population. We can therefore explore the genotype x environment interaction, using a population measure. To illustrate this potential, I compare simulation results for two previously-studied locations, Valencia (Spain) and Oulu (Finland), and two opposing combinations of genetic parameters, high seed dormancy /high floral repression (HH) and low seed dormancy /low floral repression (LL). Within an initial population of 100 seeds, the seed dormancy levels, ψ_i, were assigned probabilistically, sampling from a normal distribution with mean 0.0 and standard deviation, 1, for the Low dormancy level (L) and mean 2.5 with the same standard deviation for the High dormancy case (H) (Figure 6.14 A, C). Floral repression was fixed at either 0.598 for the Low level (L) and 0.737 for the High level (H), values that were chosen to reflect the behaviour of natural populations of Arabidopsis in Wilczek et al. (2009). Both parameter choices follow Liana T. Burghardt et al. (2015). The simulation time period was 60 years and, as in Liana T. Burghardt et al. (2015), we discarded the first 15 years of the simulation to focus on stable life history strategies. A key difference from the earlier work is that even our conservative choice of birth function (see Methods) allows some lineages to die out.

6.4.1 Valencia

Figure 6.12 shows the results of the two-stage simulation for a population of the LL genotype in Valencia (Figure 6.12A). We identify four possible life history strategies based on the timing (phenology) components of the FM-life model:

1. *summer-only strategy* where the entire growth is in the summer. The growth period is quite short and the conditions unfavourably hot and dry. In the growth simulation, the rosette leaves senesce before the reproductive stage (blue curve). The drought effect on photosynthesis severely limits the carbon available for fruit mass (red curve).

2. *spring strategy* where the entire growth period is in the spring. The growth period is only slightly longer than the *summer-only* strategy but it falls in more favourable

weather conditions. The rosette lifetime extends beyond flowering to support fruit growth, which combined with favourable weather gives high fruit mass.

3. *winter-repr strategy* spans the winter/early spring period. A short vegetative period in the end of summer/early Autumn ends with flowering and a long reproductive stage over the winter/early spring. The rosette is senescing when favourable conditions return in early spring, seriously limiting fruit development.

4. *winter-veg strategy* again spans the winter/early spring period. The life cycle duration is similar to strategy 3 but slightly later germination delays flowering until Spring. The rosette grows all winter, overlapping with a short reproductive stage and supporting high fruit mass.

Plants with life cycle strategies 2 and 4 predicted orders of magnitude more fruit mass than plants with life cycle strategy 3 or the least successful strategy 1 (Figure 6.12C). This result clearly ranked the strategies available to plants of the LL genotype, although the absolute values of the predicted biomass are less certain (see Discussion). The 100 lineages amassed 4905 potential lifecycles over 45 years of phenological simulation (Figure 6.12A). Without a minimum mass threshold (m_0) for reproduction, 66% of potential life cycles followed the more successful *spring* and *winter-veg* strategies (2 & 4; Figure 6.12C). Figure 6.13A shows the sequential transitions between strategies. For example, 60% of potential plants following the successful *spring* strategy (2) disperse their seeds early enough for the next generation to adopt the *winter-veg* strategy (4), achieving two generations per year. These transitions underlie the bimodal distribution of life cycle times reported by Liana T. Burghardt et al. (2015) for this simulation.

Simulation of the HH genotype (Figure 6.12D, E) identified similar strategies. Since the seed have longer dormancy, the population amassed fewer potential life cycles (2954 as opposed to 4905 in the LL case; Figure 6.12A). The growth and final fruit masses are different because of slight variation in timing of the growth period but strategies 2 and 4 are again more successful than strategies 1 and 3 (Figure 6.12F). A higher fraction of potential life cycles followed the successful strategies (78% as opposed to 66% in the LL case; Figure 6.12D). Higher seed dormancy reduced the germination in the summer and early autumn that led to the less successful strategies 1 and 3, so any strategy was likely to be followed by either strategy 2 or 4 in the next generation (Figure 6.13 C).

In order to calculate the population success we make two choices for the reproduction mass threshold, m_0, which eliminate one or both of the least successful strategies.

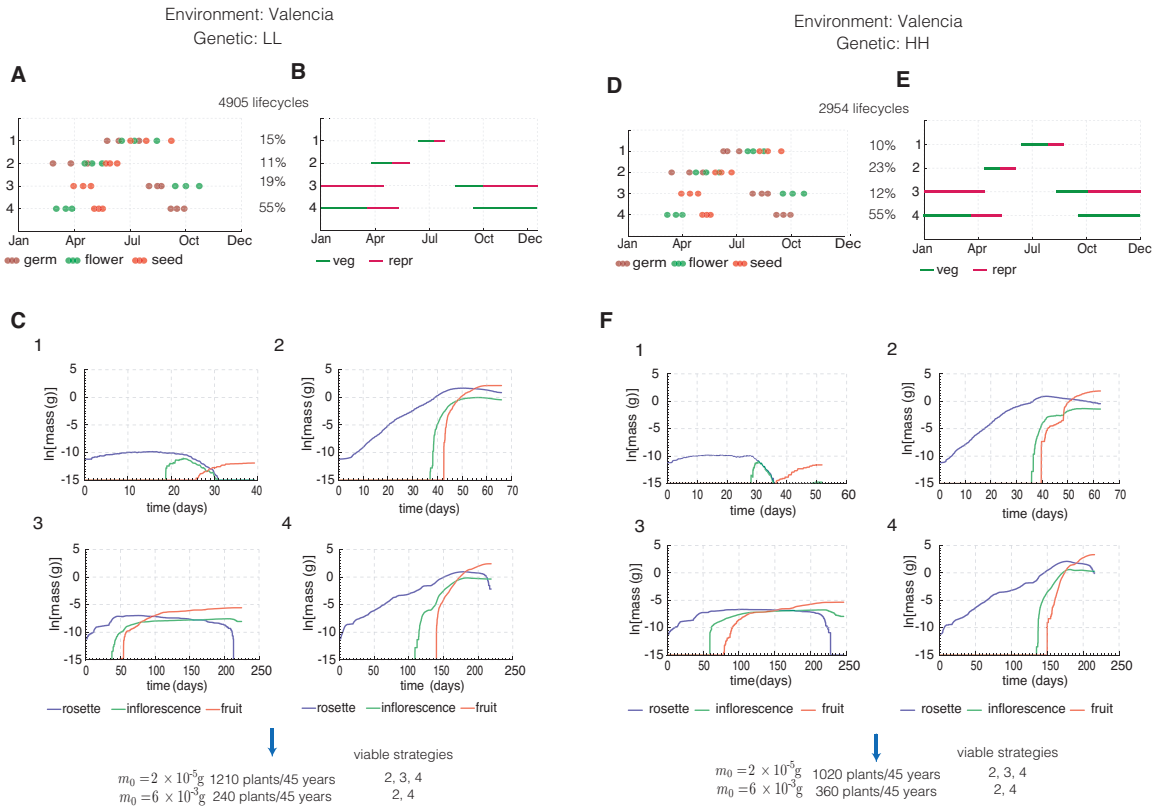


Figure 6.12: Population experiments in Valencia in two different genetic backgrounds (LL:low dormancy, Low floral repression and HH: High dormancy, High floral repression). A Results of the phenology-only simulation results for the LL genotype. The 25-th, 50-th, and 75-th percentile of the distribution of developmental events are shown for each identified cluster B Illustration of growth stages over a year for each cluster from A according to the median time of the distribution of developmental events for each cluster. C Growth simulations over the growth period shown in panel B for each cluster. D, E, F Equivalently for the HH genotype in the same location.

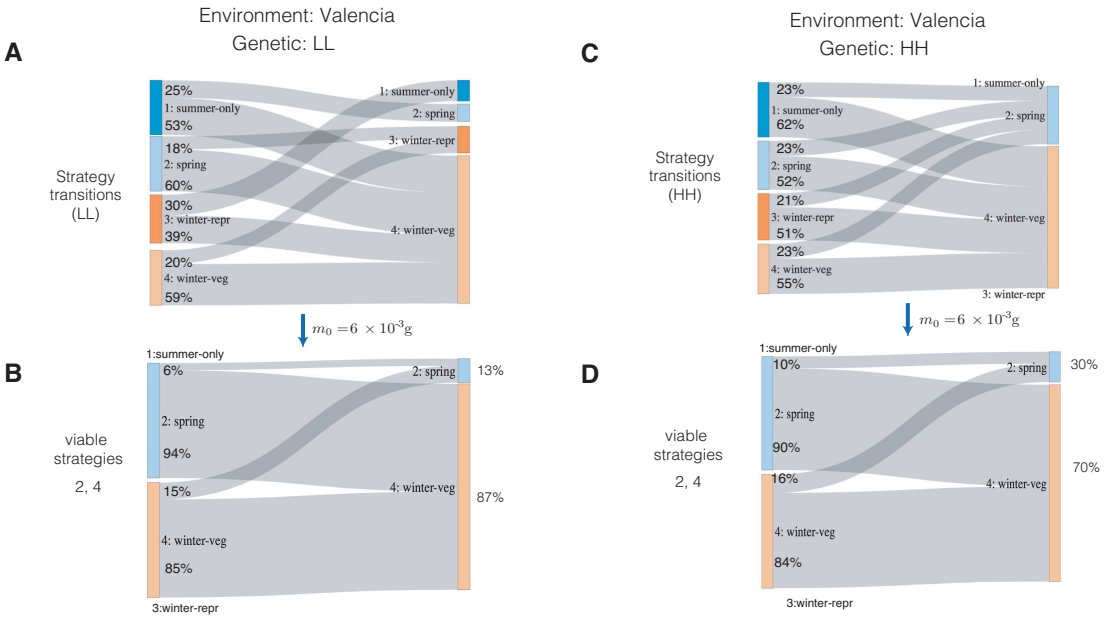


Figure 6.13: Transition probabilities between life history strategies from Figure 6.12 in Valencia. A Probabilities of successive strategies for the LL genotype B Probabilities of successive strategies for the LL genotype after eliminating life cycles with strategies 1 and 3 using a reproductive threshold. C, D Similarly for the HH genotype.

Choosing a value $m_0 = 2 \times 10^{-5}$ g (the mass of a single seed) eliminated the *summer-only* strategy from both genotypes, which gives a population of 1210 plants over 45 years in the LL case (Figure 6.12C). The HH genotype allows a larger percentage of viable life cycles but we predict fewer plants (1020) since the number of potential life cycles was lower (Figure 6.12D). Choosing a value $m_0 = 6 \times 10^{-3}$ g left only two viable strategies, 2 and 4, for both genotypes. Reciprocal transitions between the strategies were still possible but *winter-veg* was strongly favoured (Figure 6.13 B, D). The LL genotype predicted 240 plants in total over 45 years, compared to 360 plants for the HH genotype: G x E interaction favoured the HH genotype despite its smaller number of potential life cycles. Thus, modelling the growth processes not only distinguished among the potential life cycle strategies within a genotype but also distinguished between the genotypes.

The effect of seed dormancy on lineage length can be seen in Figure 6.14. For the low dormancy distribution (mean $\psi_i = 0$) taking into account only phenology, the distribution of dormancy values is negatively correlated with lineage length, as expected (more dormant lineages have fewer generations, Figure 6.14 B – timing only). Applying a growth-based birth function, which can sometimes be 0, gives a different picture

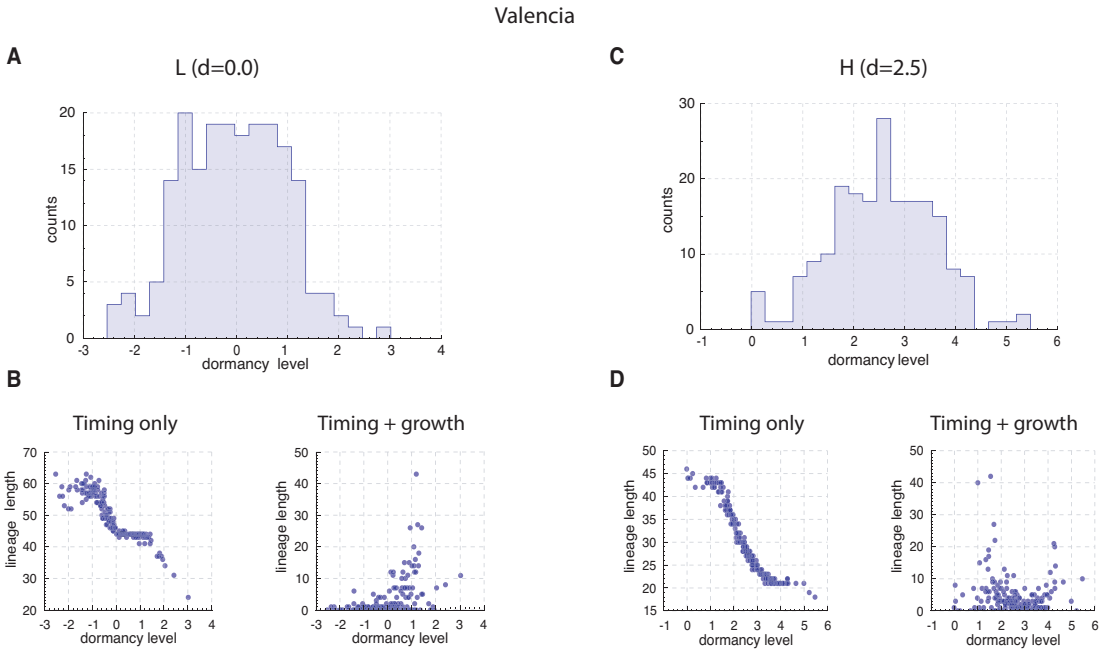


Figure 6.14: Dormancy level correlation with lineage length in the Valencia experiments A Distribution of dormancy levels for the Low (L) dormancy (ψ mean=0, sd=1) case. B The correlation of dormancy level values with lineage length first when considering only timing (phenology model alone) or when considering timing and growth (applying growth-based birth function to population). C Distribution of dormancy levels for the High (H) dormancy case (ψ mean=2.5, sd=1) D The correlation of dormancy level values with lineage length first when considering only timing (phenology model alone) or when considering timing and growth (applying growth-based birth function to population).

as higher dormancy is positively correlated with lineage length, perhaps reinforcing the genotype distinction we observed above (Figure 6.14 B – timing+growth). In the high dormancy case (mean $\psi_i = 2.5$), again, there is a negative correlation between dormancy and lineage length. Applying the birth function changes the correlation where lineages at the two ends of the dormancy distribution do better than the ones close to the mean (Figure 6.14 D). This requires further investigation as it is not clearly interpretable like the result in the low dormancy case.

6.4.2 Oulu

The equivalent simulations were performed for conditions in Oulu, Finland in the same LL and HH genetic backgrounds (Figure 6.15). The results indicated 3 potential life cycle strategies (Figure 6.15A, B, E, F):

1. *summer-only strategy* where the entire life cycle occurs in the summer. The vegetative period is short, the rosette is very small and supports negligible fruit growth (Figure 6.15C).

2. *winter-repr strategy* where a life cycle of almost a year has a very short vegetative stage, followed by a long reproductive stage over the winter. Again, the very small rosette supports little fruit growth in the following Spring.

3. *winter-veg strategy* where the plant over-winters in the vegetative stage. Unlike in Valencia, the rosette grows little over the winter. Rapid rosette growth in the following spring supports a substantial inflorescence and fruit development, though the predicted fruit mass is smaller than in Valencia.

The severe winter conditions limited the number of potential life cycles to 2361 for the LL genotype or 363 for HH. A higher proportion of HH life cycles followed the successful *winter-veg* strategy (3; 32% against 24% in LL; Figure 6.15B, F). Surprisingly, a majority of life cycles for both genotypes followed the *winter-repr* strategy (2). Applying the reproductive threshold mass, m_0, eliminated one or both of strategies 1 and 2 (Figures 6.15 C, G), suggesting a strong selective pressure for greater floral repression to reduce the number of *winter-repr* life cycles. With $m_0 = 2 \times 10^{-3}$ g, the LL genotype yielded 159 plants over 45 years compared to 53 plants for HH. All G x E combinations had actively-growing plants at the end of the simulation. Interestingly, plants of the HH genotype had higher average reproductive success per plant in Oulu yet the LL plants were more successful by our population measure. The faster development of LL plants allowed more, short lifecycles within the simulated interval (consistent with the phenology model alone).

The distinction between genotypes can perhaps also be appreciated by looking at the correlation between dormancy values in the distributions and lineage length (Figure 6.16). In both low and high dormancy case, with 0 and 2.5 mean ψ_i respectively, the correlation does not change when considering timing and growth as opposed to timing alone (Figure 6.16 B, D). This can be interpreted by the higher reproductive success of LL plants we have identified above.

6.5 Discussion

I present a whole-life-cycle multi-model for growth and reproduction of *Arabidopsis thaliana*, FM-life, combining phenology models that time the developmental stages

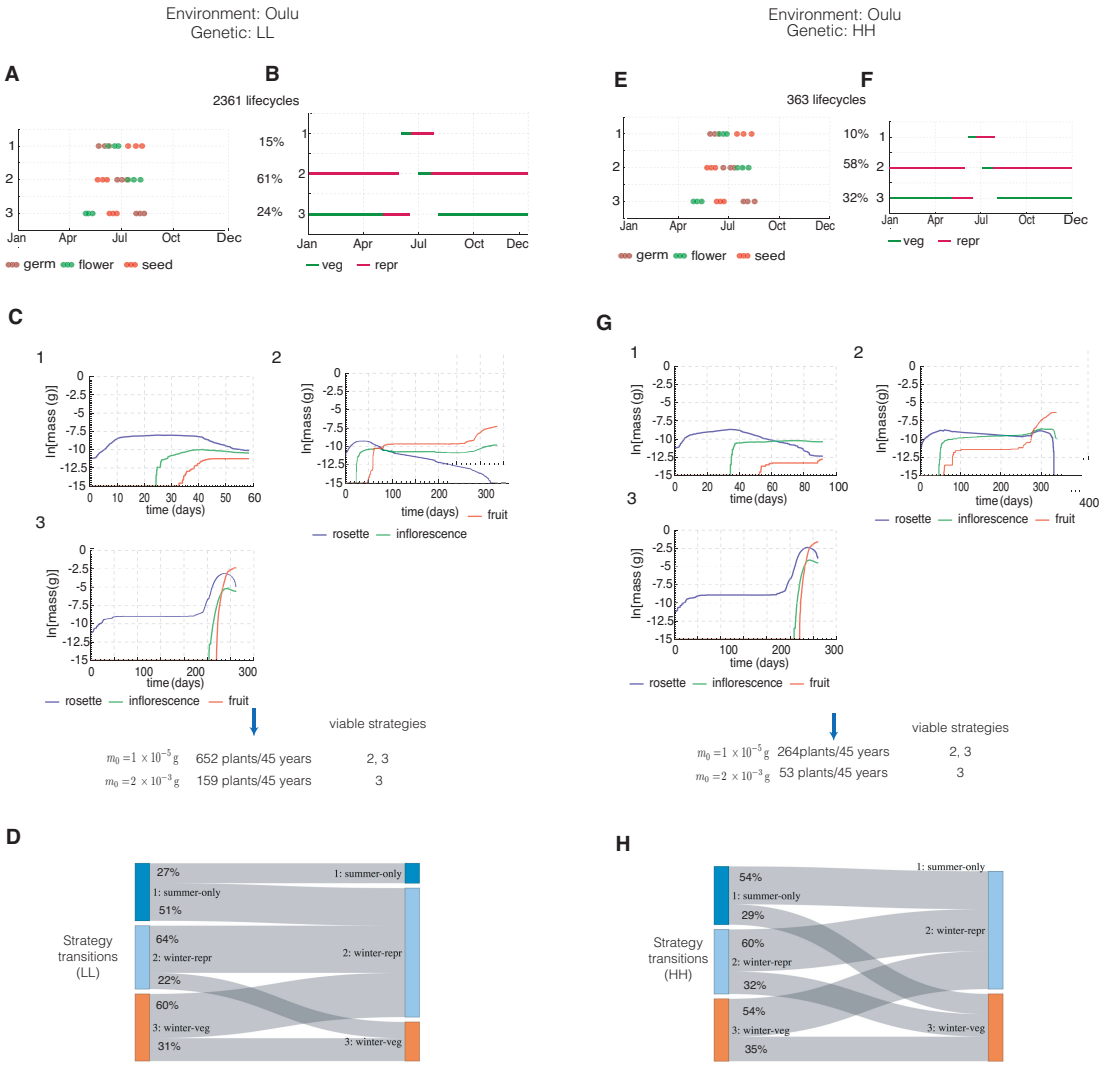


Figure 6.15: Population experiments in Oulu in two different genetic backgrounds (LL ? low dormancy, Low floral repression and HH-High dormancy, High floral repression). A Results of the phenology-only simulation results for the LL genotype. The 25-th, 50-th, and 75-th percentile of the distribution of developmental events are shown for each identified cluster B Illustration of growth stages over a year for each cluster according to the median time of the distribution of developmental events for each cluster. C Growth simulations over the growth period shown in panel B for each cluster. D Probabilities of successive strategies. E, F, G, H, J Equivalently for the HH genotype in the same location.

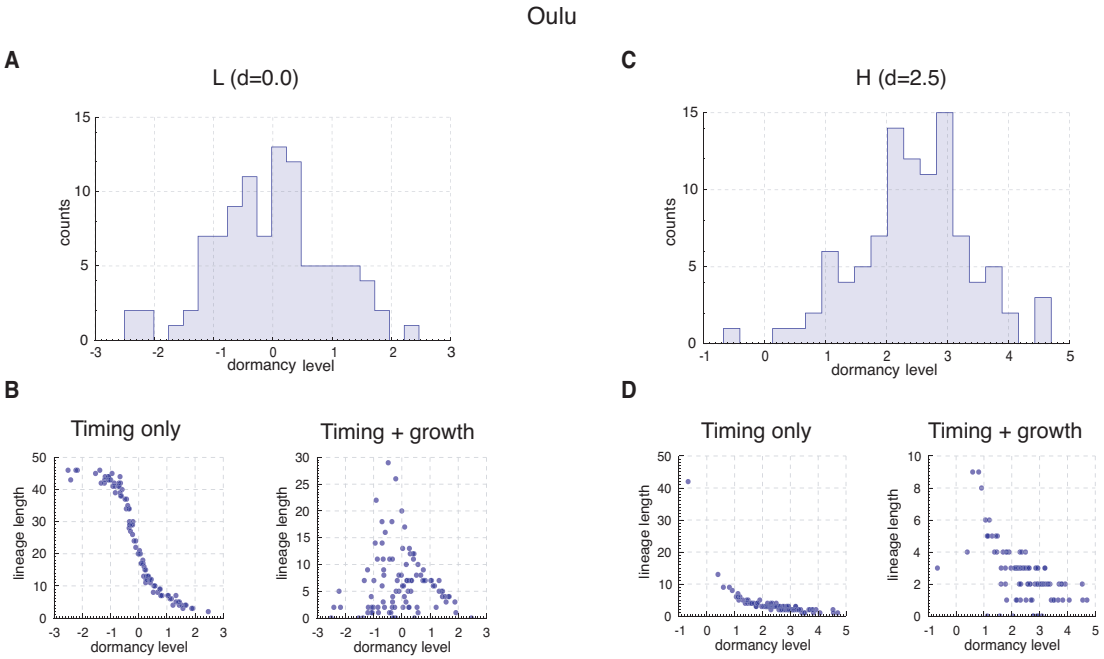


Figure 6.16: Dormancy level correlation with lineage length in the Oulu experiments A Distribution of dormancy levels for the Low (L) dormancy (ψ mean=0, sd=1) case. B The correlation of dormancy level values with lineage length first when considering only timing (phenology model alone) or when considering timing and growth (applying growth-based birth function to population). C Distribution of dormancy levels for the High (H) dormancy case (ψ mean=2.5, sd=1) D The correlation of dormancy level values with lineage length first when considering only timing (phenology model alone) or when considering timing and growth (applying growth-based birth function to population).

and growth models to predict organ biomass. The simple, FM-lite model of vegetative growth, and its extension to the reproductive stage in FM-life, simulate broader, mechanistically-founded components of fitness at the individual plant level compared to the phenology models alone. Most insights from the component models naturally remain (Liana T. Burghardt et al., 2015; Christophe et al., 2008; Rasse and Tocquin, 2006; Wilczek et al., 2009). Multi-models are helpful in emphasising interactions. The cauline leaves in the inflorescence model, for example, extend the duration of photosynthetic competence. As cauline leaves can be produced 6 months later than early rosette leaves in the *winter-veg* strategy (Figure 6.15), they remain active photosynthetic sources (Earley et al., 2009; Leonardos et al., 2014) when the rosette leaves are senescing. The growth models provided the fruit mass that I used as an indicator of reproductive success, such that metabolic and developmental processes of growth

informed a more mechanistic understanding of ecological, population dynamics over multiple generations.

The growth model allowed us to discriminate among alternative life cycle strategies in each G x E combination, by selecting against strategies that were compatible with the phenology models alone but had qualitatively worse growth. In previous work, strategies with high seed dormancy in southern Valencia and low dormancy in northern Oulu were noted to align with the behaviour of the cognate wild populations (Atwell et al., 2010; L. Burghardt et al., 2014; Chiang et al., 2011; Mendez-Vigo et al., 2011). In each G x E combination, individual plants in our simulations might adopt alternative life cycle strategies. The less-successful strategies were lethal in my model, eliminating >95% of potential lifecycles (simulated by the phenology model alone) for the LL genotype in Valencia, for example (Figure 6.12A, C). Thus, my results supported the observed genotypic distinction between Valencia and Oulu, because the requirement for a minimum fruit mass eliminated more lineages of the less-successful genotype in each case (Figures 6.12C,F and 6.15C,G).

My approach might appear conservative, as the binary birth function (one seed/no seed) ignored variation in seed mass among life cycle strategies, which might otherwise reinforce the advantage of successful strategies. The successful genotype LL in Oulu, however, had lower reproductive success per plant than HH, suggesting a more subtle balance of advantage. Genotypes with Low dormancy and High floral repression (LH) are observed in far northern locations (Atwell et al., 2010). I therefore simulated the LH genotype (Figure 6.17) . LH plants delayed flowering time enough to reduce the frequency of potential *summer-only* life cycles to 9% compared to 15% in LL (Figure 6.17B) and increased the fruit mass of the successful *winter-veg* life cycle close to the HH genotype (Figure 6.17C). The LH model predicted slightly higher reproductive success overall, returning 171 life cycles (Figure 6.17C) compared to 159 for the LL variant (Figure 6.15C), consistent with the observation of LH genotypes at this location.

I note also that while I focused on two locations on the two extremes of the native range of Arabidopsis in Europe, similar analyses can be performed for more moderate climates (see results for Halle, Germany and Norwich, UK; Appendix A). These seem to suggest similar genotype distinctions to the ones we observed for Valencia and Oulu (at various degrees).

Variability in timing results, as we have noted, appears as a result of variable weather conditions each year, difference in genetic parameter values, and the stochasticity in-

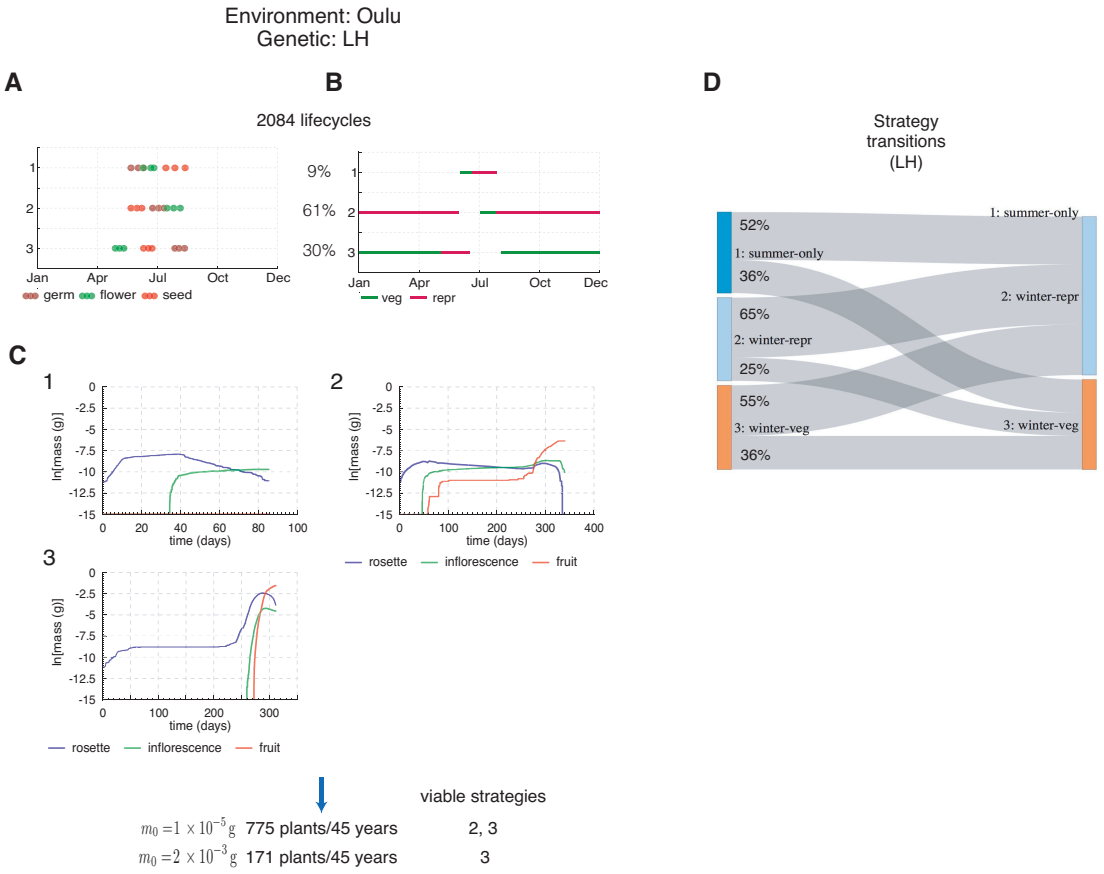


Figure 6.17: Simulation results for a combined variant in Oulu (LH - low dormancy, high floral repression). A Results of the phenology-only simulation results. The 25-th, 50-th, and 75-th percentile of the distribution of developmental events are shown for each identified cluster B Illustration of growth stages over a year for each cluster according to the median time of the distribution of developmental events for each cluster. C Growth simulations over the growth period shown in panel B for each cluster. D Probabilities of successive strategies.

herent in the model representation (Figure 6.18 A, B). Both parameters and variable weather inevitably have some effect on the timing results and in turn on the growth results and overall population measures we considered (Figure 6.18 B, C). As for the stochasticity in the model representation, we can get a hint on the effect from the indicative simulations of the seed dormancy model. There the stochasticity might, for example, drastically change the germination behaviour of otherwise identical seeds because of the hard transition thresholds and weather constraints during the year (Figure 6.2 A). It would be interesting to quantify this effect perhaps by changing the phenology simulations from a daily to an hourly timestep. Perhaps this also suggests that this type of

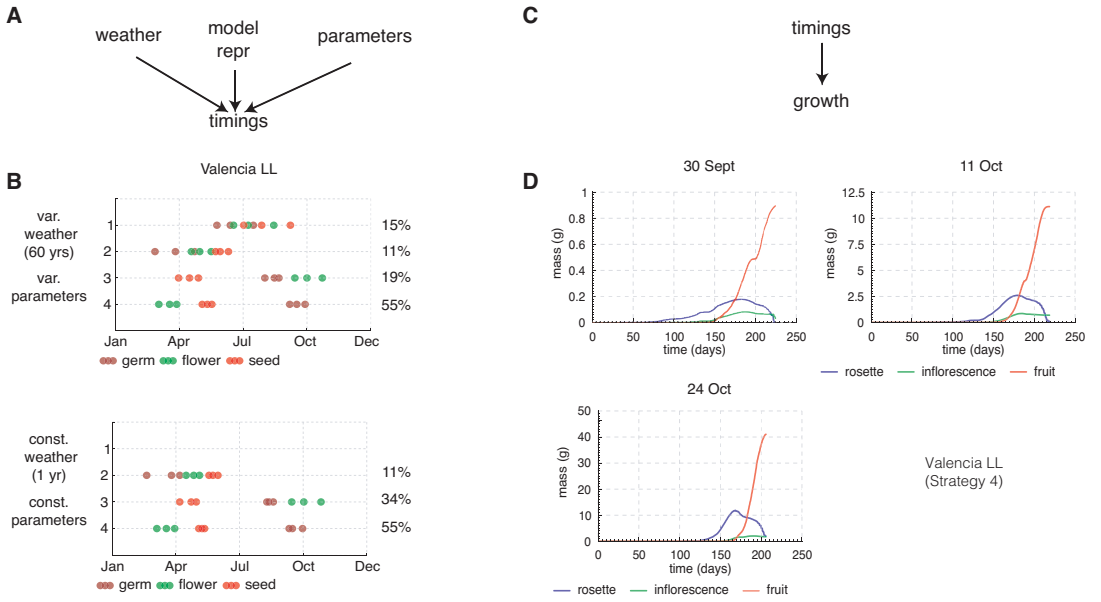


Figure 6.18: Robustness of timing and growth simulation results. A Sources of variability for the timing results. B Timing results for variable weather (60 years) and variable genetic parameters chosen from a distribution and timing results for simulations with constant weather (same 1-year weather over 60 years) and constant parameters. The timings of the strategies are very similar but strategy 1 life cycles have moved to strategy 3. This could be because of the particular year of weather data I used for the simulations. C Sources of variability for growth results D Growth simulation results starting from three different germination dates (corresponding to 25-th, 50-th, and 75-th percentiles of the distribution of germination times) of strategy 4 in Valencia LL (Figure 6.12)

phenomenological models that are meant to reproduce field experimental data are best suited to deterministic representations. I could have used the fluents of Chromar, for example, to represent the timing parts of the models (enriched expressions of Chromar §4.3.1).

A limitation of my work arises from the fact that the phenology component models of FM-life have been validated against field data (Liana T. Burghardt et al., 2015; Wilczek et al., 2009) whereas the growth component models have not (Christophe et al., 2008; Rasse and Tocquin, 2006). Biomass simulations are inevitably sensitive to the timing of the growth period, because a longer interval of exponential growth in good conditions rapidly changes absolute biomass, as illustrated in Figure 6.18D. I therefore modified the growth models conservatively to account for severe winter conditions and to limit photosynthesis in dry conditions (§6.2.2). The moisture effect on seed maturation (through photosynthesis) in particular is important and extends the

effects of water availability, an important field environmental condition, to the whole lifecycle as opposed to only the seed dormancy stage in (Liana T. Burghardt et al., 2015), for example. Nonetheless, the FM-life model predicted unreasonably high fruit mass in some cases. The binary birth function ensured that this had no effect on our population measure. Among possible gaps in understanding of the environmental effects on growth in natural settings or in my representation, I repeat previous caution (Chew, Seaton et al., 2017; Chew, Wenden et al., 2014) that models of nutrient balance for Arabidopsis will be helpful. Rosette biomass in the Framework Model is understandably sensitive to photosynthetic parameters (Chew, Wenden et al., 2014) yet these have not been validated in Arabidopsis across the wide range of photoperiods and temperatures simulated here (B. Walker et al., 2013). FM-life predicts a discretised fruit mass and hence reproductive success for a typical representative of each life cycle strategy, approximating an underlying, continuous distribution of fruit mass. The accuracy of this approximation will depend on the variation within clusters. The benefit lies in computational tractability, allowing us to simulate differential reproductive success that is informed by understanding of growth processes.

My approach here builds upon previous models that predict fitness and population processes in Arabidopsis, which have focussed on developmental components of fitness or on phenology (Prusinkiewicz, Erasmus et al., 2007; Satake et al., 2013; Springthorpe and Penfield, 2015). Linking these components sharpens ecological insight, by understanding the performance of genetic variants in the environment that underlies differences in fitness (see discussions in Liana T. Burghardt et al., 2015; Doebeli et al., 2017) and can thus inform evolutionary hypotheses. This is perhaps one of the first instances of a model going from the molecular mechanism to the whole plant and to the population, which points in the direction of closing the circle of understanding allowing one to ask 'why' as well as 'how' questions (see §1.2 and more general discussion in §8.3). Beyond the mechanisms investigated here, it further demonstrates that making these links is possible for a complex organism and provides a blueprint or perhaps even a scaffold for linking other mechanisms and understanding from other fields. These integrated stories through models are inevitably incomplete at first but they can act as a measure of the progress of our understanding and shape future research. This was also a signficant test for Chromar and this model shows the breadth of applicability since we have several different types of dynamics at different levels of organisation.

Adding genetic variation between generations will in future model Arabidopsis evolution explicitly, perhaps after competing genetic variants *in silico* using adaptive

dynamics approaches (Brännström et al., 2013; Weiße et al., 2015). Adding different genetic variants in the populations of FM-life models introduces competition implicitly as the different variatns will have different reproductive rates, which means that inspecting the population structure at any timepoint can give a ranking of the variants. For example, if we had two populations with two different dormancy or floral repression levels they would have different birth dates at each location depending on the period in the year that they grow as we have seen. Competition can also be added more explicitly by, for example, considering death of the growing plants at different developmental stages and competition for common resources like space and soil nutrients. This allows to explicitly test fitness hypotheses for different mechanisms going to the molecular level. Thus, the FM-life model offers a unique tool (and opportunity) to bridge among disciplines in plant biology, ecology and evolution.

Chapter 7

Optimal control of plant traits

Biological dynamics, from single cell behaviours to emergent properties of populations, are the result of the complex interplay between genomes and the environment. Efforts to tightly regulate them have so far largely been focused on (re)engineering genomes. In this chapter I instead address the problem of controlling the plant biomass manipulating the surroudning environment. While the automatic climate control problem has been studied before for greenhouses, the focus of such efforts was on the minimisation of energy consumption and/or achieving assigned climate regimes. Seldom those works attempted to optimise crop traits (Aaslyng et al., 2003; Chalabi et al., 1996; Challa, 1990; Udink ten Cate and Challa, 1983).

More insulated and sophisticated growth environments with more precise control are becoming available as part of the urban and controlled-environment agriculture movement (Despommier, 2013; Mok et al., 2014). There are many commercial examples (Aerofarms, Motorleaf) and others like Intelligent Growth Solutions (IGS) that are based on novel technologies, for example, for reducing energy consumption and increasing control over the growth space (lighting patent Aykroyd, 2016, automated growth tower patent Aykroyd et al., 2018). These more sophisticated growth environment allow us to focus more on crop traits and even more precisely at specific quality standards (size, uniformity) that crops need to meet (see for example EU marketing standards on fruit and vegetables; European Commission, 2011).

In this chapter, I present a formulation of the climate control problem for *Arabidopsis thaliana* plants as an optimal control problem (Kirk, 2012) where the control variables are climate conditions (temperature) and the performance criterion is a crop trait. It has already been proposed that more mechanistic models are needed when linking between multiple scales instead of the usually empirical models used in crop modelling (Yin and

Paul C Struik, 2010; Yin, Paul C Struik and Kropff, 2004). Here I use the Framework Model (FMv1; Chew, Wenden et al., 2014) as a mathematical description of vegetative plant growth of *Arabidopsis thaliana* that provides mechanistic links from molecular regulation all the way to whole-plant traits, like biomass. In particular, in this chapter I do the following:

- Formulation and solution of a *direct problem* of offline climate control where the control variable is the temperature of a growth chamber in order to achieve a particular plant biomass at a particular time (predicted from the FM). The solution to the problem uses standard gradient-based optimisation techniques by discretising the growth period and assuming constant temperature within each time interval.

- Formulation and solution of an *indirect problem* of offline climate control. Here I assume that we cannot control temperature precisely in the growth space, which leads to temperature inhomogeneities. I further assume that the position of plants can be controlled (e.g. switch their positions) in a linear array that has a temperature gradient. The control variable is the position of the plants and the performance criterion is homogeneity in final plant biomass (as predicted from the FM) for the population of plants in the array. For the solution of the problem I use techniques from combinatorial optimisation since the control variables are discrete.

7.1 The model and main idea

For this work I use the Framework Model (FM) as described in Chew, Wenden et al. (2014). Unlike the rest of this thesis where I use stochastic plant models in Chromar, in this chapter I use the deterministic version of the FM in Matlab as provided in Chew, Wenden et al. (2014). This switch was motivated by practical reasons. Since I use simulation-based optimisation in this chapter the speed of the simulation is crucial and the deterministic realisation of the FM provides a signigificantly faster simulation time. Furthermore, Matlab has a much greater choice of available optimisation libraries compared to Haskell, the language where the stochastic version of Chromar is implemented and embedded (other choices of languages are possible but the Haskell is the only currently available one).

The FM takes environmental inputs (CO2 level, temperature, light intensity, and temperature) and outputs several growth-related quantities over its simulation time. In

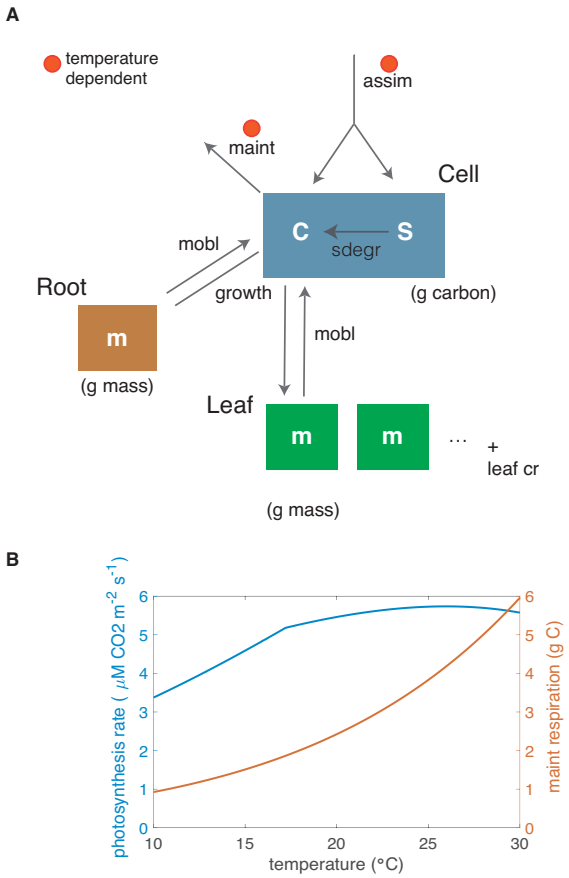


Figure 7.1: Metabolic processes as flows in the Framework Model (FM). A The FM keeps track of carbon movement in different pools, one for each organ (leaves and root) and two whole-plant reserves of sucrose carbon (c) and starch carbon (s). Maintenance respiration and assimilation (intake of carbon) are explicitly temperature dependent (indicated by red dots) B Photosynthesis and maintenance rate trends for a range of temperature values.

the following work, however, I will treat the FM model a Single Input Single Output system accepting temperature as input and outputting biomass by regarding all the other inputs as constant and ignoring the rest of the output signals. Despite the 'black box' approach we take to model plant growth, it is instructive to look at two particular metabolic processes that are highly temperature dependent.

All the metabolic processes keep track of the main building block of new mass, carbon, and are represented as flows between various pools representing the organs (root and leaves only at the vegetative stage) and whole plant reserves of carbon as sucrose (C) and starch (S) (Figure 7.1A). The two metabolic processes that are temperature dependent are (i) assimilation, the intake of carbon from the environment represented

as an in-flow into the central reservoir of sucrose carbon (C) and (ii) maintenance respiration, the use of carbon for maintaining the life-sustaining processes inside the plant represented as an out-flow from the main sucrose carbon reservoir. Figure 7.1B shows the rates of these flows for a range of temperature while keeping the other environmental inputs constant (light intensity,120 μmol $\cdot m^{-2} \cdot s^{-1}$, CO2=420ppm).

While temperature directly affects only two of the processes it also indirectly affects the other since all of them compete for the sucrose carbon in the main reservoir. Therefore and not surprisingly temperature is a major determinant of growth rate and final biomass. The *main idea* of this work is to control temperature and therefore indirectly control growth to achieve a particular growth-related objective of interest using the FM as our ground truth to evaluate the growth of an *Arabidopsis thaliana* plant at different temperature inputs.

In the FM the growing plants starts as seeds and wait for a period of time (emergence period), which is a function of temperature, before they emerge and start vegetative growth. In the following we only control temperature in the vegetative growth phase and not in the emergence period. Therefore we assume that seeds are kept in a constant temperature of 22°C during the emergence period.

7.2 Direct problem

Here I seek to design a control scheme that modulates temperature to reach a specific biomass after a particular period of time. This is relevant to growers, for example, that have specific requirements both in terms of time and crop attributes.

Given the characteristic of the problem at hand I formulate is as an optimal control problem, i.e. find a time-profile of temperature function, $T^*(t)$, over a period $[t_0, t_f]$ that minimises the difference between the final biomass of the plant (as predicted by the FM) and a target biomass m_0:

$$J = m_{T^*}(t_f) - m_0$$

where $m_{T*}(t)$ is the temperature dependent biomass trajectory predicted from the FM. To reflect physical constraints, we impose upper and lower bounds on the temperature $T_l < T(t) < T_u$. Additionally, here I consider the case of a 'single plant control'. Notably, this approach can be extended, with appropriate modifications, to the control of crops.

As conventional in optimal control, we can convert the optimal control into a standard non-linear optimisation problem by discretising the time domain $[t_0, t_f]$ into k

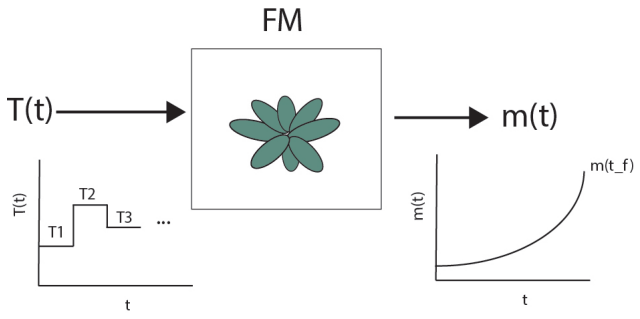


Figure 7.2: Direct problem of climate control. For the *direct problem* we try to find a temperature function or sequence of temperature values in the optimisation formulation in order for the final biomass value as predicted by the FM (using this temperature function as input) to reach a specific predefined value, m_0.

intervals and assuming a constant temperature value inside each interval (Kraft, 1985).

Definition 4 (biomass-only problem)**.** *The* biomass-only problem *is the problem of finding a sequence of temperature values, $T_{1,k} = T_1, \ldots, T_k$, that minimises the square of the difference between the final biomass value of the plant at t_f, $m_{T_{1,k}}(t_f)$, as predicted by the FM, for the sequence $T_{1,k}$.*

$$\underset{T_1,\ldots T_k}{\operatorname{argmin}} \left(m_{T_{1,k}}(t_f) - m_0\right)^2 \ \textit{subject to}$$
$$T_l < T_i < T_u; \quad i = 1 \ldots k$$

Note that I use $T_{i,k}$ and $T_1, \ldots, T_k$ interchangeably for a sequence of k temperature values.

Since repeated, large switchings of the temperature introduce wear and tear of the actuators and use energy, we extend our control problem to take into account the control effort as well.

Definition 5 (biomass+control effort problem)**.** *The* biomass+control effort *problem is an extension to the* biomass-only problem *where the sought sequence of temperatures $T_{1,k} = T_1, \ldots, T_k$ minimises both the distance to the target biomass, m_0, and the control effort defined as the average jump between successive temperatures in the sequence, $\underline{T}$.*

$$\underset{T_1,\ldots T_k}{\operatorname{argmin}} \left(m_{T_{1,k}}(t_f) - m_0\right)^2 + \frac{1}{k}\sum_{i=2}^{k} T_i - T_{(i-1)}$$

The bound constraints are the simple lower/upper bounds as before.

7.2.1 Results

In this section I solve the two formulations of the optimisation problems (direct problem) using the `fmincon` function for constrained non-linear optimisation from the Matlab optimisation toolbox for some choices of m_0, t_f, and k. While I vary m_0 to explore different instances of the problem, I generally leave t_f and k constant ($t_f = 512h$, $k = 4$). There are methods that adapt k that we could have used that are implemented in more sophisticated tools (mesh refinement, AMIGO tool; Balsa-Canto et al., 2016)). The final time is dictated by the bolting time of the plant under the range of temperatures that we consider since the FM only considers vegetative growth (before bolting). Bolting time is more than $512h$ for all the temperature values in the range that I consider ($[10, 30]$).

I use a performance metric, $\log(\sqrt{L(T_{1,k}, t_f, m_0)}/m_0)$, to assess particular optimisations for different problem instances (specific k, t_f, and m_0) that states the final value of the biomass objective function $L(T_{1,k}, t_f, m_0)) = (m_{T_{1,k}}(t_f) - m_0)^2$ as a percentage of the target biomass.

Definition 6 (ε-reachable)**.** *A target biomass value,m_0, defining a particular instance of the biomass-only or biomass+control effort problems is* ε-reachable *if there exist a k and a particular sequence of temperature $T_{1,k} = T_1, \ldots, T_k$ such that the ratio of the performance metric to the target biomass, m_0, is less than a tolerance value,* ε*.*

$$\frac{\sqrt{L(T_{1,k}, m_0, t_f)}}{m_0} < \varepsilon$$

In the following I assume that a specific sequence of temperatures that satisfies our tolerance, ε, exists if the particular optimisation algorithm I use can converge to that sequence.

Figure 7.3 shows results of the optimisation procedure for both the biomass-only and biomass+control effort problems defined in the previous section for a range of target biomasses. For the biomass-only problem there is a set of reachable target biomasses $[0.03, 0.25]$ (Figure 7.3A) for a tolerance $\varepsilon = 0.1$, $k = 4$, and $t_f = 512$h. The upper and lower bound constraints for the temperature values are set to 10 °C and 30 °C respectively. The optimisation converges to different solutions on different runs (Figure 7.3B and C, D, E for details of particular solutions for $m_0 = 0.05$, $m_0 = 0.1$, and $m_0 = 0.15$ respectively). It is also interesting to explore if the optimisation increases the reachable set compared to a naive exhaustive simulation of the FM with constant temperature inputs in the constrained space [10 °C, 30 °C]. Exhaustive simulation of the FM in the space [10 °C, 30 °C] with a 0.5 °C step gives final (at $t_f = 512$h) biomasses

between 0.0093g at 10 °C and 0.1806g at 17.5 °C while more than one temperature input can give the same (or very close) final biomass (orange line, Figure 7.3B).

For the biomass+control effort problem I find a smaller reachable set $[0.03, 0.21]$ compared to the biomass-only problem (Figure 7.3F) with the same parameters. The optimisation procedure though in this case mostly converges to the same solution at different runs and the solutions mostly keep the temperature constant (Figure 7.3F and H, I, J for details of particular solutions for $m_0 = 0.05$, $m_0 = 0.1$, and $m_0 = 0.15$ respectively). Comparing the solutions to the naive exhaustive search as before we can see the optimisation converges to constant temperature solutions that are very close to the ones given by the simulation with a similar reachable set. In cases where two constant temperature inputs give the same final biomass the optimisation cannot distinguish between the two and can converge to either (for example, Figure 7.3H and temperature distributions in G).

Finally, we can compare the results of the optimal strategy with a random strategy (as a control) where the sequence of temperatures, $T_1, \ldots, T_k$ is picked at random (Figure 7.4). For 100 simulation runs for both optimal and random strategies, the optimal strategy performs significantly better for target biomasses $m_0 = 0.05$ and $m_0 = 0.1$.

7.3 Growth space inhomogeneities experiment

I have so far assumed that we can perfectly control the environmental conditions inside the growth chamber such that all the plants are exposed to the same temperature. In actual growth spaces, this is rarely possible. To quantify the effect of temperature inhomogeneity on plant growth we conducted an experiment designed to test the difference in growth (biomass) of two sets of *Arabidopsis thaliana* plants grown in two different locations on a shelf (Figure 7.5E). The first group of plants is grown in the middle of the shelf and the second on the side of the shelf.

The environmental conditions in terms of light and temperature are very different between the two sets of plants (Figure 7.5C, D). We took two samples at two different stages of vegetative development and before bolting, one at 23 days after germination (plants M1/1[1-6], M2/2[1-6] for the middle group and plants S1/1[1-6], S2/2[1-6] for the side group – Figure 7.5) and one 27 days after germination (plants M1/2[1-6], M2/1[1-6] for the middle group and plants S1/2[1-6], S2/1[1-6] for the side group – Figure 7.5). The differences in the environmental conditions affected the growth and resulted in differences in biomass of the two groups (Figure 7.5A, B). While

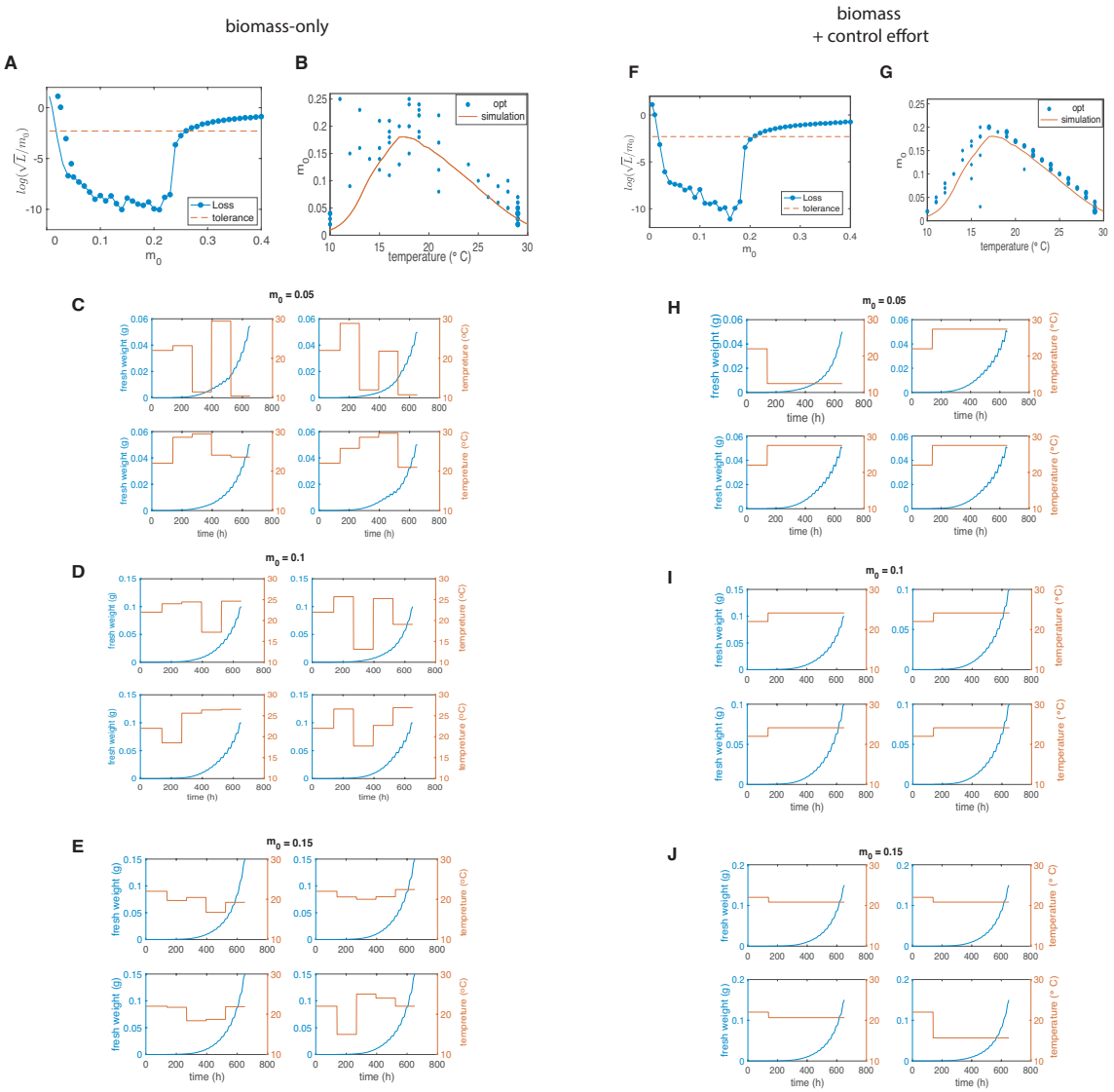


Figure 7.3: Results for both biomass-only and biomass+control effort formulations of the direct problems over a range of target biomasses with $k = 4$ and $t_f = 512$. A The performance metric for the biomass-only problem formulation over a range of target biomasses from 0.01 to 0.4. The other environmental inputs are constant: CO2, 420ppm; light intensity, 120μmol $\cdot\, m^{-2} \cdot s^{-1}$; 12:12-h light/dark cycle. For the first 139 hours of simulations the plants are still seeds and are assumed to be kept in a constant temperature of 22 ℃. The reachable set is under the dotted line. B Comparison between the solutions obtained with the optimisation and a naive exhaustive simulation with constant temperatures over the interval [10 ℃, 30 ℃]. For the optimisation solutions per m_0 I give the two most commonly occurring temperatures in the solution space (10 runs of the optimisation per m_0). C, D, E Four example solutions and corresponding biomass time series (as obtained from FM simulation) for target biomasses $m_0 = 0.05$ (C), $m_0 = 0.1$ (D), and $m_0 = 0.15$ (E). F Performance metric for the biomass+control effort problem formulation over a range of target biomasses. G Same as B but for the biomass+control effort problem formulation H, I, J Similar to C, D, E but for the biomass+control effort formulation.

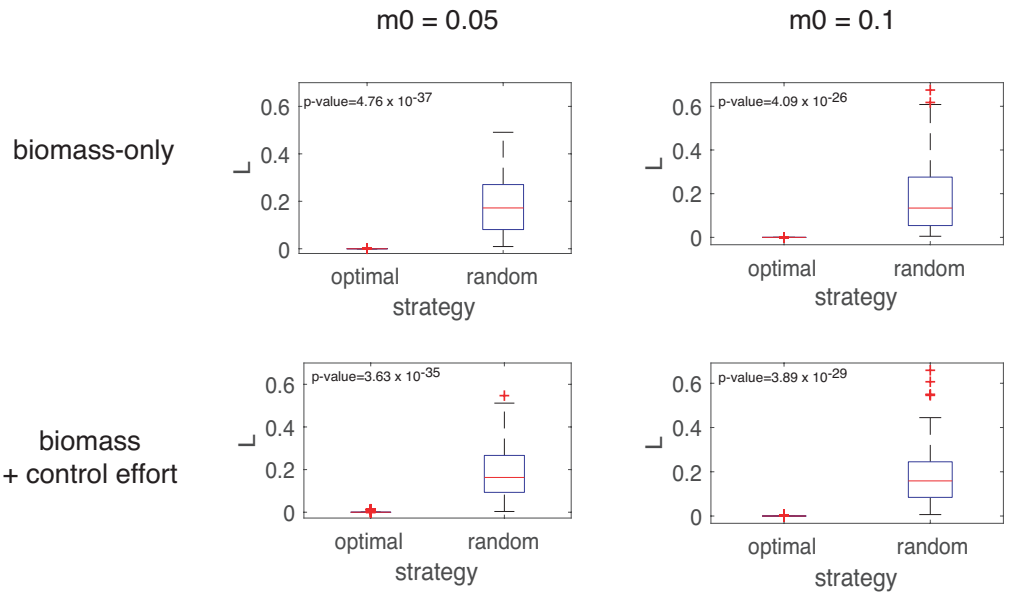


Figure 7.4: Comparison between optimal and random strategies for both formulations of the direct problem. The comparison is for two target biomasses in the controllable range $m_0 = 0.05$ and $m_0 = 0.1$ over 100 runs of the optimisation 100 runs of the random strategy where temperatures are chosen at random for the 4 intervals.

the differences in biomass is not statistically significant in the 23-day sample, in the 27-day sample the difference in biomass between the two groups is significant. The environmental differences are amplified during the exponential growth of the plants.

7.4 Indirect problem

In the direct problem formulation I assumed perfect control inside the growth space of a crop such that the temperature is homogeneous and all the plants are exposed to the same conditions. This is rarely true even in academic growth environments (see previous section, for example). Here I address the problem of controlling plant growth acting on the position of plants in a linear array: a linear temperature gradient is imposed on such an array. As in this case we can only change the temperature the plant is growing indirectly, i.e. by changing its position in the linear array, we call this problem "indirect". This formulation more closely resembles industrially relevant problems.

I assume that have n plants and the growth space is a linear array of n positions such that each plant occupies one position. I further assume that I am given a global temperature perhaps coming from an optimisation procedure (for example, see the direct

problem before) but due to non-precise control this gives rise to a temperature gradient along the array where the plants are positioned. The optimal control problem becomes then to find an optimal position function $P^*(i,t)$ that gives the position of plant i at time t over a period $[t_0,t_f]$ such that some non-uniformity index over the final biomasses of the n plants is minimised:

$$J = G(m_1(t_f),\ldots,m_n(t_f))$$

Here I write $m_i(t_f)$ for the biomass of i-th plant at the final time t_f and we use the Gini index (denoted by G) as our non-uniformity metric.

As in the case of the direct problem we can turn the above into an optimisation problem by discretising the time domain $[t_0,t_f]$ into k intervals and assuming that the positions of the plants are constant inside each interval. The objective is to obtain by t_f a set of plants with similar (satisfactory) biomass. The problem then becomes to find the a sequence of $n \cdot k$ values,

$$P = \begin{bmatrix} p_{11} & p_{12} & \cdots \\ \vdots & \ddots & \\ p_{n1} & & p_{nk} \end{bmatrix}$$

,where $p_{i,j}$, is the position of plant j at interval i:

$$\begin{aligned} &\min_P \; G(m_1(t_f),\ldots,m_n(t_f)) \text{ subject to} \\ &1 < p_{ij} < n \\ &p_{ij} \text{ integers} \end{aligned}$$

The positioning of the plant indirectly determines the temperature function, $T(t)$, over $[t_0,t_f]$ for the plants. Unlike the direct problem the decision variables of the optimisation problem are discrete. We therefore cannot use standard gradient-based optimisation techniques. In the results in the following sections we use a technique from combinatorial optimisation (see next section) that uses some heuristics to search the space of all solutions while following our objective function from above.

A constraint that I have overlooked in the formulation above is the need to have one plant, and only one, in each position at every single time point. We can formalise this as: for any interval j all the values in the sequence $p_{1j},\ldots,p_{nj}$ should be unique. In order to not have to deal with this extra constraint we make a change to the formulation of the problem such that the solutions of the combinatorial optimisation procedure

are guaranteed to satisfy this 'non-overlap constraint' without having the constraint explicitly in the optimisation procedure. In particular suppose that we have a permutation matrix, $\Pi(n)$, that lists all the possible permutation of n numbers:

$$\Pi = \begin{bmatrix} 1 & 2 & 3 & \dots n \\ 2 & 1 & 3 & \dots n \\ \vdots & & \vdots & \\ n & n-1 & \dots & 1 \end{bmatrix}$$

I redefine the optimisation with the decision variables being indices into the permutation matrix so for example if at interval j one we get π_j then we have the positions of each plant, i, at that interval $P_{ij} = \Pi_{(\pi_j,i)}$

Definition 7 (permutation-index problem)**.** *The* permutation-index problem *is a variant of the indirect problem of climate control where we seek a sequence of values* $\pi_1, \dots \pi_k$ *that minimise the inhomogeneity in the final biomasses (after* t_f *hours, as predicted by the FM) of n plants.*

$$\min_{\pi_1,\dots\pi_k} G(m_1(t_f), \dots, m_n(t_f)) \text{ subject to}$$
$$1 < \pi_i < n!$$
$$\pi_i \text{ integers}$$

The number of all possible permutations for n numbers is given by $n!$. The order of permutations in Π is important here as we need to make sure that solutions that are close together in the permutation index space are also close together in the positions space. Here we assume that every permutation listed in Π is one flip away from the permutation above it. Therefore consecutive points in permutation space are adjacent in position space as well. For the instances of the problems we dealt with this solution was adequate. However, for larger instances the search space might become prohibitively large ($n!$) in which case we can go back to the previous formulation of the problem and introduce an overlap penalty in the objective function.

7.4.1 Results

In this section we explore solutions to the permutation-index problem defined in the previous section using combinatorial optimisation techniques. In particular, we use a

Matlab implementation (MEIGO too; Egea et al., 2014) of the Variable Neighbourhood Search algorithm (VNS; Mladenović and Hansen, 1997).

For the following experiment described in this section we assumed a growth space of 5 position with 5 growing plants, one for each position, $k = 4$ intervals and $t_f = 512$ hours. The temperature gradient goes from 20 °C at position 1 on one end of the growth array to 24 °C at position 5 at the other end of the array. The positions in between are assumed have a temperature linearly interpolated between the values at the two ends of the array. We use a normalised performance metric $\tilde{G} = G/G_s$ where G_s is the value of the Gini-index for a position matrix where the positions of the plants are unchanged over the growth period.

Unsurprisingly the optimisation procedure returns results where the plant positions are shuffled over the time intervals (example solutions: Figure 7.7A). The optimal strategy is significantly better though than a naive strategy where the plants are randomly assigned positions at each interval (Figure 7.7B). Both strategies are better over 100 runs than the static strategies where the plant positions are unchanged in the growing period (static G value indicated by dotted line, Figure 7.7B).

7.5 Discussion

I present a formulation of the climate control problem for achieving particular growth-related plant (or population of plants) attributes as an optimal control problem and solution after transforming it to an optimisation problem. Optimal control of climate in greenhouses has been studied for a long time but since greenhouses are not insulated to weather conditions, greater effort (and therefore energy) is needed for climate control. Optimal control studies have therefore mainly been focused on reducing energy consumption (Fisher et al., 1997; Ramírez-Arias et al., 2012; Sagrado et al., 2016) with very few also taking into account plant processes like photosynthesis for optimising growth as well as energy consumption(Aaslyng et al., 2003; Harun et al., 2015). Here I assume that our growth space is more insulated so that more precise control is available at less effort. This allows us to focus on crop traits and even at more precise quality control standards like size and uniformity. I do, however, consider a control effort quantity in the biomass+control effort problem formulation of the direct problem that should be related to energy consumption. In the indirect problem we do not have that even though switching the plant positions will require significant energy even at sophisticated growth environments.

In order to address the climate control problem I start with a *direct problem* where I try to find an optimal input temperature signal to the FM so that the output biomass signal has a particular value at some final time, t_f. The formulation of the problem with an added penalty for control effort along with exhaustive simulations with constant temperature inputs suggest that a single constant temperature input is almost enough to achieve the same results as the optimal strategy (Figure 7.3A, B, F, G). There is very little increase in the reachable set using the optimal strategy as opposed to naive enumeration of final biomasses using single-temperature simulations of the FM (Figure 7.3B, G). temperature, find paper As we have seen from experimental results though even if I set a global optimal temperature (for example as given by a solution of an optimisation problem representing the direct problem) we are not guaranteed that all the plants will have the same conditions. Distances as little as 1m (same shelf), even in spaces designed specifically to reduce climate inhomogeneities, can lead to significant environmental and therefore growth differences among a population of plants. I therefore turned to a formulation of an *indirect problem* of climate control where we accept the inhomogeneities and try to minimise them (and therefore also growth inhomogeneities) by switching the positions of plants along a space with a temperature gradient.

For the solutions of both problem statements, direct and indirect, I use the FM as our ground truth to predict plant behaviour at different temperature input profiles. The FM starts with biochemical models that have been validated only in a narrow range of experimentally relevant temperatures so even in our seemingly conservative temperature range ([10°C, 30 °C]) there might be gaps in our understanding (B. Walker et al., 2013). This limits the predictive power of the model across the range that we consider. Moreover there might be other temperature effects on plant physiology that I do not consider. The FM is deterministic but often there is variability in growth (Abley et al., 2016), which might suggest an adaptive online strategy in a possible implementation with a continuous monitoring of the growing plants. Instead of a priori assigning the temperature signal like I do here, in an adaptive online setting the input signal could be recomputed during growth taking into account the development of the plant up to that point.

Our approach builds on ideas from traditional climate control in greenhouses and the availability of more sophisticated growth spaces to suggest using optimal control of climate for control of crop traits and quality standards, like size and uniformity. Of course while the application of such methods is most interesting for crop species that

are practically relevant, here we chose the model species *Arabidopsis thaliana*. It will be interesting first to test our offline in-silico optimisation experiments in an actual prototype implementation for Arabidopsis plants given the limitations of the model-only approach outlined above. Application of such techniques for more practically relevant crops will also be very attractive and there is a number of models to back such an application for commercial crops (potato models, Fleisher et al., 2017; tomato model, Heuvelink, 1999; wheat models, Martre et al., 2015). These are plant growth models that are used for understanding but similar (although simpler) crop models have been used more practically for optimal growth in greenhouses, for example with cucumber crops (Challa, 1990). Finally, another possible application of, for example, the indirect problem formulation might be in academic settings where technical variability is expected to be minimal and there is an increase in the use of automated phenotyping and growth platforms (PHENOPSIS Granier et al., 2006; platforms in the European Plant Phenotyping Network, `https://eppn2020.plant-phenotyping.eu/`).

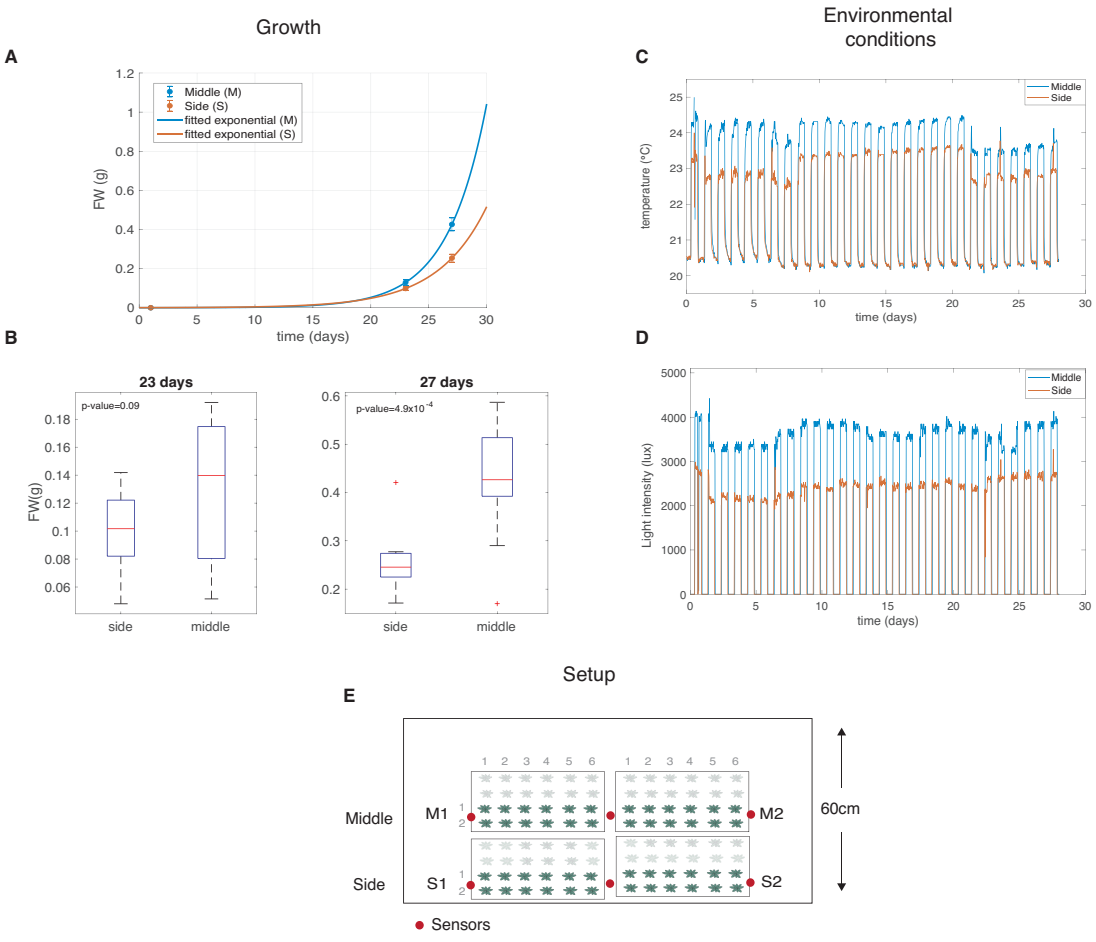


Figure 7.5: Results of experiment designed to investigate growth inhomogeneities resulting from environmental inhomogeneities in a controlled growth room. A Mean biomass for the two groups (middle, side) for the two samples (23 and 27 days) and fitted exponential curves. B Full distribution for the two samples (23 and 27 days) C Average temperature time series over the three sensors placed along the positions of the middle and side groups. D Average light time series averaged over the three sensors places along the positions of the middle and side groups. Note that light is given in lux instead of the μmol that the models use since the readouts from the sensors are given in lux. E Experimental setup showing the positions of the plants and sensors used to get the environmental data in C, D.

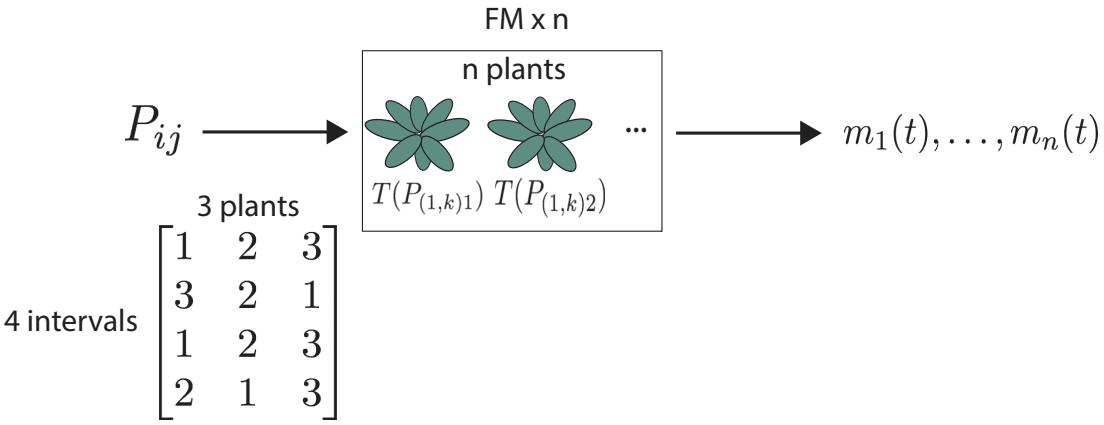


Figure 7.6: Indirect problem of climate control. In the *indirect problem* we try to find a position function (map from time and plant id to position in a linear array) such that the final biomasses of the plants as predicted by the FM (using the position dependent temperature of the plants as input) to be as close to each other as possible (minimum Gini-index).

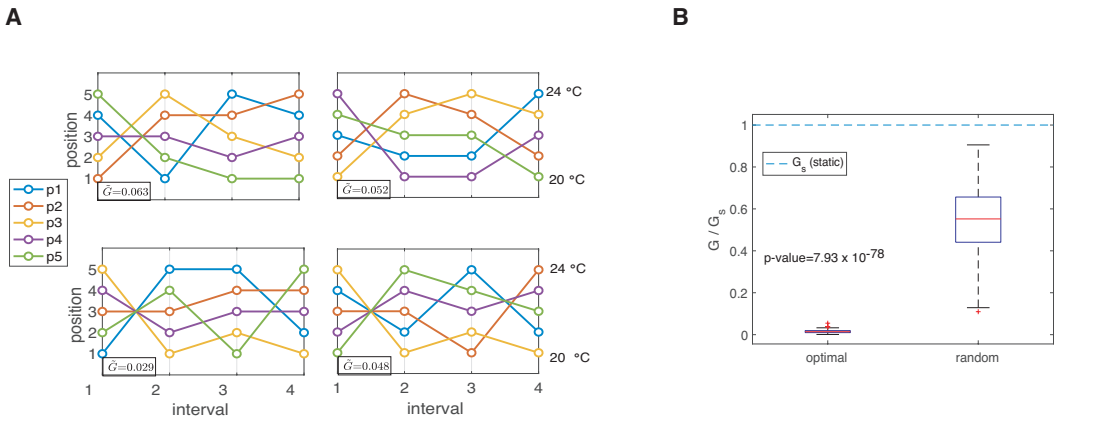


Figure 7.7: Comparison between optimal and random strategies for the indirect problem. A Four example solutions returned by the optimisation procedure for four intervals, five plants, and a growing space of five positions arranged linearly. Each colour represents one plant and a line the movement of a plant across the growing space over the four intervals. The temperature gradient starts from 20 ℃ in one end of the array and goes up to 24 ℃ on the other end. B Comparison of the objective function values (normalised Gini-index of final biomasses of plants) over 100 runs of the optimal and random strategies where the plants take random positions at each intervals. The Gini-index for the static strategy is shown with a dotted line.

Chapter 8

Conclusions

After a brief summary of the work presented in this thesis, I next go through and discuss aspects of my work in order to understand its limitations, implications that follow, and possible future work that it suggests.

8.1 Summary

No complex system can be understood by extrapolation of the properties of its elementary components. In order to understand a complex system, like an organism, one needs to consider explanations at various levels of detail. This is especially true in biological systems where entities (processes and so on) and their organisation give rise to properties of processes at the next level of detail and they in turn to the next until you get to the level of a living entity, the organism. In order to understand the organism (and consequently life) we therefore need to understand the links between the explanations at the various levels of detail.

The importance of organisation between components has been stressed before and it is in fact the fundamental concept of systems theory. The stress on organisation and its constructive nature also leads to questions about representation. Abstraction to quantifiable properties alone is not adequate to represent the rich organisation of biological systems. I presented *Chromar* a language that combines organisational aspects through the explicit representation of discrete entities, agents, and non-organisational aspects through abstract properties of these entities, attributes. The concept of abstraction is there in extensions to the expression language used for the dynamics of the attributes. Observables recognise the need for explanations at different levels by allowing an explicit functional link between abstract quantifiable properties and more explicit rep-

resentations. Fluents, again, allow abstraction via the description of system properties using time-dependent deterministic value for describing parts of the system we do not wish to model mechanistically.

In contemporary plant research the focus has been on the model species Arabidopsis. Functional-structural models have focused on the constructive organisation of plant organs (development) as well as functional aspects like metabolism. The Framework Model (Chew, Wenden et al., 2014) is inspired by these but adds more breadth by explicitly including a genetic circuit and phenology models to time the conceptual development towards flowering. The FM covers one part of plant development but does not extend to reproduction, which means that it has limited applicability in ecological studies. Ecological studies on the other hand consider the full lifecycle but of an abstract version of plants where its construction and physical development are absent. I presented *FM-life* an extension to the FM to the full lifecycle including reproduction and its scale-up to the population level via a clustering approach to tractably simulate population of plants over multiple decades in different genotype x environment scenarios.

Finally, I presented another more engineering oriented use of multi-scale plant model as devices for optimising plant traits. Instead of the more traditional genome engineering approach we focused on engineering the environment during their development to achieve specific growth-related traits.

8.2 Chromar and representations of multi-scale biology

The ‘structuralists’ that we have overviewed in Chapter 1 and were the proponents of organisation and the systems view regarded biology (and life) as inherently modular with this being a defining (if not the most defining) characteristic of living systems. Whether biology is inherently modular is debated (Hartwell et al., 1999). Nevertheless, treating as such is a justifiable choice to advance understanding. Modules includes ‘visibly’ differentiated substructures like the cell or organs, for example, given in spatial terms but also other entities that form units in a non-spatial way that could be identified by the strength of the interactions between their components or their functional goal. For example, one might identify functional units in reaction networks (Ederer et al., 2003) and hierarchical modules in metabolic networks (Ravasz et al., 2002). These modules are organised either by interaction – more so the case in non-spatial modules – or by physical forces – more the case in spatially distinguished modules like cells (Simon, 1962, ‘Biological and Physical systems’ section).

As we have pointed out, in order to understand complex biological systems, like entire organisms, we need explanations at multiple levels of detail. At each level of detail we can identify different modules, which can either be treated as undivided wholes or as organisations of their elementary components/modules (Varela et al., 1974). The links between the functioning of modules at different levels of detail are crucial for fundamental questions in biology. We will call these aspects of biology that deal with modules, their organisation (interaction), and hierarchy as the *organisational* aspects of systems. Importantly this organisation changes dynamically.

Explicitly representing the functioning of modules in organisational only terms in detail, however, is probably impractical in many cases. Even when considering the elementary components of a module and their organisation, certain aspects of the functioning of a module are still more practically captured through some abstraction to pick out properties of interest that perhaps are important for the links to the next level of detail. The abstraction that one uses to understand the functioning of a module will depend on the level of understanding and practical considerations. We will call these the *non-organisational* or *abstract* aspects of a system (or its model).

Given that in multi-scale explanations of systems require representation of both non-organisational and organisational aspects, the question of the technical language used for this representation is interesting. For the organisational parts since we are considering discrete entities, we need collections of objects (Walter Fontana and Leo W Buss, 1996). Other aspects of the organisation possibly also then need the description of relations on these collections of objects (Rashevsky, 1954; Rosen, 1958). The constructive nature of this organisation means that we need a way to describe the creation/destruction of objects and the creation or deletion of pairings of objects in the defined relations. In Chromar we have simple representations of organisational aspects where objects, agents, can be defined along with their creation/destruction. Explicit definition of relations is not possible, but we have seen that different kinds of relations on objects exist in object-based languages (§2.2). Implicit representation is possible through agent attributes. While possible, this puts an extra burden on the user to first decide on a representation of the relation through agent attributes and then to maintain this relation consistent when the state changes when rules are applied. This burden increases if one needs to keep track of more than one dynamic relation. This might suggest ways our notation could be extended. Ideally the language would track such relations and have a special notation for the most common types of relations – for example in biology the ‘connects-to’ and ‘nested-in’ relations seem natural, as we have seen (§2.2). We could then write rule

left-hand sides that say ‘this rule is applicable to any two leaves that are connected’ or ‘this rule is applicable to any two cells inside the same leaf’. A combination of Chromar with Kappa features resulting in a version of Kappa with richer types, for example, would be very powerful.

For the non-organisational aspect the question of representation is more difficult since the point of abstraction is to use the most convenient or practical explanation for a particular module. Chromar allows agents (modules) to have attributes, which are lists of typed values. While attributed objects (colours and so on) have been used in languages before, in Chromar these take an ‘elevated’ status where they could stand for entire models. Their dynamics come with discrete transitions, which is the same way the dynamics of objects (deletion/creation) are defined. This is very general and could satisfy a range of applications, and we have seen that the Dynamical Grammars of Mjolsness and Yosiphon (2006) also allow the description in terms of the more familiar ODEs.

What if we wanted to use the most convenient abstraction for the abstract parts of each module though? Pragmatic approaches like multi-model simulators (Lang, 2018; RBM group, 2015) allow that through concurrent running of executables representing the simulation of each module. The organisation of the modules, for example communicating the level of variables (interaction), is usually represented externally in a configuration file. While this achieves the goal of simulation, the resulting multi-model does not achieve the second goal of models that we have identified regarding understanding since the organisation is opaque. The connections are merely identification of variables and it is not clear how these affect each other. Furthermore, these connections are expressed in a static way, which encourages larger modules that are guaranteed to be static.

What the above suggest for possible future work is a system that handles both the organisational aspects in an object-based language and the non-organisational aspects possibly in many languages. The combination of these in Chromar through enriched expressions suggests a way that both organisational and abstract aspects could co-exist naturally in the same system. Consider for example the system shown in Figure 8.1. Like Chromar, the organisational aspects are represented by discrete objects that have types (colours in the diagram). Non-organisational aspects exist as part of these modules to describe aspects of their functioning that we want to capture in a convenient abstraction. This is like the attributes of agents in Chromar, which can represent entire models (rectangles attached to objects, Figure 8.1). Different levels of explanation can be

combined with observables, for example a variable in one level might be a function of the state at another level (dotted line between objects and variables in abstract models, Figure 8.1). Other dynamics can be defined independently of any module, like Chromar does with fluents (f, Figure 8.1). Both of these can coexist with the dynamics of state variables inside each module, in a similar way that attribute dynamics are given using enriched expressions, which combine regular expressions, observables, and fluents.

Both the organisational and non-organisational dynamics can be given with rules, like we have seen with Chromar. The object rules can add/delete or relate/unrelate objects. Rules can also be used for the interaction of the internal dynamics of each module. For example, if a state variable from one module is needed to compute the state variable for another, this interaction can be captured by synchronising them through a rule. Internal dynamics are captured through rules referring only to single modules.

Unlike Chromar, however, the abstractions in each module or the independent ones should be written in any language and explicit relations should also be representable. In order to achieve this we will either need formal mappings between the mathematical interpetations of the abstractions used in each module as suggested by Mjolsness (2018) or a more pragmatic system using simulation synchronisation, for example through message passing, which is the approach taken by Lang (2018). This does not answer the question of explicit representation of organisation (through relations) but this could come from any of the object-based languages we have overviewed that do have these features (§2.2).

The system of organisation of processes reminds us of the graph of process interactions of Rashevsky (1954), who was one of the first to point to the importance of considering the 'relational' aspects of Biology (what we called organisational aspects here). It is also comparable to the multiscale topological structures of Godin and Caraglio (1998), which is the main representation structure used in another tool for simulation of plant systems based on data-flows on graphs (OpenAlea; Pradal et al., 2008).

8.2.1 Practical considerations

While there is a lot of work on theoretical aspects of modelling languages, practical aspects related to the practical use of these languages can be overlooked especially in more theoretical (modelling) languages from computer science. This might be because these are usually tied to short-lived research projects or because there is too much focus

on theoretical aspects of these languages. Some exceptions exist, for example Kappa (§2.2.2) has seen a continuous development of a suite of practical tools for its use for a number of years (Kappa platform; Boutillier, Maasha et al., 2018).

Chromar could similarly benefit from practical tools to overcome practical hurdles to its use. For example, in the current implementation one would have to have familiarity with Haskell to write Chromar rules and have the Haskell ecosystem (compiler and so on) installed to compile models or access the simulator. Other languages have used more mainstream and familiar languages, like Python (Kappa or L-systems in Python; Boudon et al., 2012; Lopez et al., 2013) but that still leaves the cost of having a working installation, which is sometimes prohibitive.

Having these practical tools will be especially important for systems like the one we outlined in the previous section that allows the combination of multiple modules. In comprehensive models it is becoming rare that a single person writes the entire model. A practical modelling tool that combines model definition with a library of existing modules would be very powerful. This is the approach taken by simulation frameworks (see §2.4.2) in general-purpose programming languages like CHASTE (suite of software for heart simulation Mirams et al., 2013) or APSIM (crop modelling Keating et al., 2003). Drawing from the ideas in the previous section and the above framework examples, one could imagine a (possibly visual) interface where existing or new modules can be connected like jigsaw puzzle pieces similar to the way programs are defined in visual programming languages like Scratch (Maloney et al., 2010). This perhaps could work on the web, like a version of the Kappa platform, to lower the adoption curve.

8.2.2 Database ideas

For the observables we use an analogy of the state of the system (multiset of agent values) to a database. The first part of the observables is defined with a ‘select’ statement that picks out parts of the state similarly to the select statement in database query languages. Similar techniques appear already in trace-query languages where we have observables over state traces (sequences of states) instead of single states (Laurent et al., 2018). The left-hand sides of rules can similarly be thought of as select statements as they pick out subsets of the state where the transformation represented by the rule can be applied.

This connection can be exploited to allow the adoption of techniques from database research for rule-based systems. One area could be to use techniques from database

research to increase the performance of language implementations. We highlighted some possible ways our implementation could be improved by avoiding recomputing the matches of each rule at every step of the simulation (§4.3.4). This is similar to the *view maintenance problem*, a well studied problem in database theory. Views are query results over a database that need to be kept consistent under changes to the underlying database. There are efficient solutions to the problem for databases with duplicates (database is taken as a multiset) that could be exploited in the calculation of our matches to avoid the current naive recomputation at each step (Griffin and Libkin, 1995).

Apart from practical results a large amount of theory work also becomes available, for example for query languages on multisets, which are ubiquitous in the object-based modelling languages (Buneman et al., 1994; Libkin and Wong, 1997).

8.3 Organism-centred evolutionary ecology

It is not necessary to model everything at the mechanism level ('how' level). If the intension is to understand at an abstract level what a particular process does as a computational unit and especially if there is lack of data (or knowledge) then a more phenomenological description at some abstract level is adequate and probably even preferable (see levels of description of Marr, 1982, Chapter 1). For example the theory of syntax of Chomsky (2014) is a purely computational description of language and does not describe how this is physically realised in neurons in the brain.

Similarly, many models in plant ecology and evolution abstract away the organism and have an abstract view of the population as a whole. Understanding the mechanism though is also important. This has always been true but recent advances in understanding, the availability of detailed datasets across scales along with developments in computational tools (e.g. high-performance computing) make this practically achievable (Marshall-Colon et al., 2017; Zhu, J. P. Lynch et al., 2016).

The connections between the abstract computational view of a process and its physical realisation is vital in answering the 'how' questions of life (see Introduction, Figure 1.1). As more knowledge of the mechanism becomes available for plants, it becomes attractive to replace the abstract with the mechanistic to get a more mechanistically-founded view of evolutionary processes, as we have seen in Chapter 6.

On the opposite side of this however, connecting a low-level mechanism up the scales to the population level might help understand and answer 'why' questions about the evolution of the mechanism too. If evolution is a population process (Doebeli et al.,

2017) then modelling a mechanism in detail in the context of the organism and the population might explain its evolution ('Nothing in biology makes sense except in light of population genetics'; M. Lynch and Walsh, 2007).

As we go from individual detailed mechanisms to the population level it is always important to experimentally test our understanding and hypotheses that we formalise in models. While population experiments in natural conditions are common for crop species, they usually only consider one generation. The short length of the Arabidopsis lifecycle along with detailed molecular and genetic understanding might make it possible to perform experiments to see the links between genomes and the population over multiple generations in natural conditions. Rubin et al. (2018), for example, do field experiments (natural weather conditions) with populations of circadian clock mutant Arabidopsis plants to observe the effect on developmental timing and shoot architecture. This study is again for a single generation but interestingly includes traits like total fruit mass that could be used as proxies for reproductive success therefore allowing validation of multi-scale models and the evolutionary hypotheses they suggest.

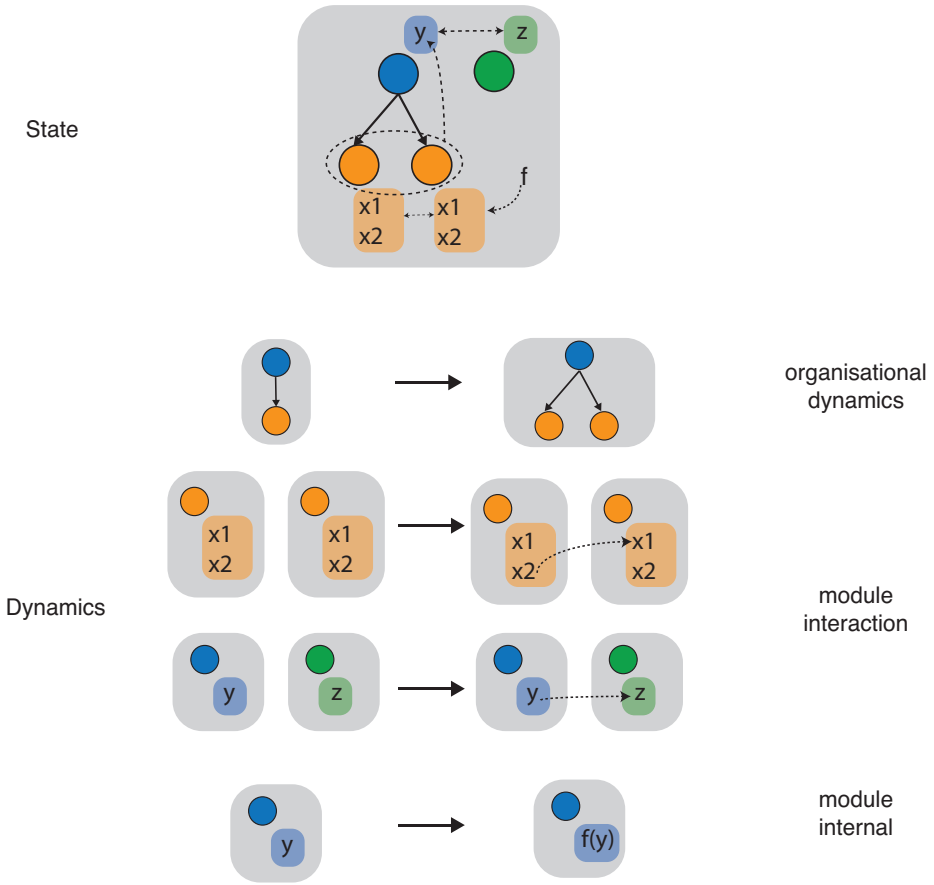


Figure 8.1: A hypothetical system inspired by Chromar that combines non-organisational and organisational descriptions. The state of the system is given by discrete objects with types (coloured circles) organised in a relationfor the organisational parts and with abstract dynamics attaches to these objects (rectangles next to circles) for the non-organisational parts. Dotted lines show interactions between parts of the state. The dynamics are given as rules that can change the organisation, synchronise variables between the non-organisational parts of modules or simply represent the internal dynamics of modules.

Bibliography

Aaslyng, Jesper Mazanti, Jens B Lund, Niels Ehler and Eva Rosenqvist (2003). 'IntelliGrow: a greenhouse component-based climate control system'. In: *Environmental Modelling Software* 18.7, 657?666.

Abar, Sameera, Georgios K Theodoropoulos, Pierre Lemarinier and Gregory MP O'Hare (2017). 'Agent based modelling and simulation tools: a review of the state-of-art software'. In: *Computer Science Review* 24, pp. 13–33.

Abley, Katie, James CW Locke and HM Ottoline Leyser (2016). 'Developmental mechanisms underlying variable, invariant and plastic phenotypes'. In: *Annals of botany* 117.5, pp. 733–748.

Allen, Garland E. (1977). 'The Evolution of an Evolutionist . C. H. Waddington'. eng. In: *Isis* 68.4, pp. 669–670. ISSN: 0021-1753.

Alvarado, V and KJ Bradford (2002). 'A hydrothermal time model explains the cardinal temperatures for seed germination'. In: *Plant, Cell & Environment* 25.8, pp. 1061–1069.

Anderson, David F. and Thomas G. Kurtz (2011). 'Continuous Time Markov Chain Models for Chemical Reaction Networks'. en. In: *Design and Analysis of Biomolecular Circuits*. Springer, New York, NY, pp. 3–42. ISBN: 978-1-4419-6765-7 978-1-4419-6766-4. DOI: 10.1007/978-1-4419-6766-4_1.

Artale, Alessandro, Enrico Franconi, Nicola Guarino and Luca Pazzi (1996). 'Part-whole relations in object-centered systems: An overview'. In: *Data & Knowledge Engineering* 20.3, pp. 347–383.

Asseng, Senthold et al. (2013). 'Uncertainty in simulating wheat yields under climate change'. In: *Nature Climate Change* 3.9, p. 827.

Atwell, Susanna et al. (2010). 'Genome-wide association study of 107 phenotypes in Arabidopsis thaliana inbred lines'. In: *Nature* 465.7298, p. 627.

Ausbrooks, Ron et al. (2003). 'Mathematical markup language (MathML) version 2.0 . W3C Recommendation'. In: *World Wide Web Consortium* 2003.

Aykroyd, Henry (2016). 'Novel power and lighting arrangement'. US20160057836A1. URL: https://patents.google.com/patent/US20160057836A1/en (visited on 06/07/2018).

Aykroyd, Henry, Peter Tyrell, Lewis Marriott and David Scott (2018). 'Automated tower with many novel applications'. en. US9974252B2. URL: https://patents.google.com/patent/US9974252B2/en (visited on 06/07/2018).

Aziz, Adnan, Kumud Sanwal, Vigyan Singhal and Robert Brayton (2000). 'Model-checking Continuous-time Markov Chains'. In: *ACM Trans. Comput. Logic* 1.1, pp. 162–170. ISSN: 1529-3785. DOI: 10.1145/343369.343402.

Backus, John (2007). *Can programming be liberated from the von Neumann style?: a functional style and its algebra of programs*. ACM.

Balsa-Canto, Eva, David Henriques, Attila Gábor and Julio R. Banga (2016). 'AMIGO2, a toolbox for dynamic modeling, optimization and control in systems biology'. en. In: *Bioinformatics* 32.21, pp. 3357–3359. ISSN: 1367-4803. DOI: 10.1093/bioinformatics/btw411.

Barbuti, Roberto, Giulio Caravagna, Andrea Maggiolo-Schettini and Paolo Milazzo (2009). 'An intermediate language for the stochastic simulation of biological systems'. In: *Theoretical Computer Science*. Concurrent Systems Biology: To Nadia Busi (1968–2007) 410.33, pp. 3085–3109. ISSN: 0304-3975. DOI: 10.1016/j.tcs.2008.09.032.

Barnat, Jiri et al. (2012). 'On Parameter Synthesis by Parallel Model Checking'. In: *IEEE/ACM Trans. Comput. Biol. Bioinformatics* 9.3, pp. 693–705. ISSN: 1545-5963. DOI: 10.1109/TCBB.2011.110.

Beard, Daniel A et al. (2012). 'Multiscale modeling and data integration in the virtual physiological rat project'. In: *Annals of biomedical engineering* 40.11, pp. 2365–2378.

Blinov, Michael L, James R Faeder, Byron Goldstein and William S Hlavacek (2004). 'BioNetGen: software for rule-based modeling of signal transduction based on the interactions of molecular domains'. In: *Bioinformatics* 20.17, pp. 3289–3291.

Bonabeau, Eric (2002). 'Agent-based modeling: Methods and techniques for simulating human systems'. In: *Proceedings of the national academy of sciences* 99.suppl 3, pp. 7280–7287.

Borshchev, Andrei and Alexei Filippov (2004). 'From system dynamics and discrete event to practical agent based modeling: reasons, techniques, tools'. In: *Proceedings of the 22nd international conference of the system dynamics society*. Vol. 22. Citeseer.

Boudon, Frédéric, Christophe Pradal, Thomas Cokelaer, Przemyslaw Prusinkiewicz and Christophe Godin (2012). 'L-Py: an L-system simulation framework for modeling plant architecture development based on a dynamic language'. In: *Frontiers in Plant Science* 3, p. 76.

Boutillier, Pierre, Ferdinanda Camporesi et al. (2018). 'KaSa: A Static Analyzer for Kappa'. In: *Computational Methods in Systems Biology*. Ed. by Milan Češka and DavidEditors Šafránek. Lecture Notes in Computer Science. Springer International Publishing, pp. 285–291. ISBN: 978-3-319-99429-1.

Boutillier, Pierre, Jérôme Feret, Jean Krivine and Lý Kim Quyên (2018). *KaSim and KaSa Reference Manual*. URL: https://tools.kappalanguage.org/docs/KaSim-manual-master/KaSim_manual.htm.

Boutillier, Pierre, Mutaamba Maasha et al. (2018). 'The Kappa platform for rule-based modeling'. In: *Bioinformatics* 34.13, pp. i583–i592.

Brännström, Åke, Jacob Johansson and Niels von Festenberg (2013). 'The Hitchhiker's Guide to Adaptive Dynamics'. en. In: *Games* 4.3, pp. 304–328. DOI: 10.3390/g4030304.

Brim, Luboš, Milan Češka and David Šafránek (2013). 'Model Checking of Biological Systems'. In: *Formal Methods for Dynamical Systems: 13th International School on Formal Methods for the Design of Computer, Communication, and Software*

Systems, SFM 2013, Bertinoro, Italy, June 17-22, 2013. Advanced Lectures. Ed. by Marco Bernardo, Erik de Vink, Alessandra Di Pierro and HerbertEditors Wiklicky. Lecture Notes in Computer Science. Springer Berlin Heidelberg, pp. 63–112. ISBN: 978-3-642-38874-3. DOI: 10.1007/978-3-642-38874-3_3. URL: https://doi.org/10.1007/978-3-642-38874-3_3.

Brown, Hamish E. et al. (2014). 'Plant Modelling Framework: Software for building and running crop models on the APSIM platform'. In: *Environmental Modelling & Software* 62, pp. 385–398. ISSN: 1364-8152. DOI: 10.1016/j.envsoft.2014.09.005.

Bucher, Dominik, Ilias Garnier, Ricardo Honorato and Vincent Danos (2013). 'Decomposition of strongly coupled systems'. In: *Young Researchers Workshop on Concurrency Theory*. URL: http://www.dominikbucher.com/assets/documents/Bucher%5C%202013%5C%20DecompositionOfStronglyCoupledSystems.pdf.

Budiu, Mihai, Joel Galenson and Gordon D Plotkin (2013). 'The compiler forest'. In: *European Symposium on Programming*. Springer, pp. 21–40.

Buneman, Peter, Leonid Libkin, Dan Suciu, Val Tannen and Limsoon Wong (1994). 'Comprehension syntax'. In: *ACM Sigmod Record* 23.1, pp. 87–96.

Burghardt, Liana T., C. Jessica E. Metcalf, Amity M. Wilczek, Johanna Schmitt and Kathleen Donohue (2015). 'Modeling the Influence of Genetic and Environmental Variation on the Expression of Plant Life Cycles across Landscapes.' In: *The American Naturalist* 185.2, pp. 212–227. ISSN: 0003-0147. DOI: 10.1086/679439.

Burghardt, LT, CJE Metcalf, AM Wilczek, J Schmitt and K Donohue (2014). *Data from: Modeling the influence of genetic and environmental variation on the expression of plant life cycles across landscapes*. Dryad Digital Repository. DOI: 10.5061/dryad.nv0p1.

Calzone, Laurence, François Fages and Sylvain Soliman (2006). 'BIOCHAM: an environment for modeling biological systems and formalizing experimental knowledge'. In: *Bioinformatics* 22.14, pp. 1805–1807. ISSN: 1367-4803. DOI: 10.1093/bioinformatics/btl172.

Camporesi, Ferdinanda, Jérôme Feret and Kim Quyên Lý (2017). 'KaDE: A Tool to Compile Kappa Rules into (Reduced) ODE Models'. In: *Computational Methods in Systems Biology*. Ed. by Jérôme Feret and HeinzEditors Koeppl. Lecture Notes in Computer Science. Springer International Publishing, pp. 291–299. ISBN: 978-3-319-67471-1.

Celli, Fabio, F Marta L Di Lascio, Matteo Magnani, Barbara Pacelli and Luca Rossi (2010). 'Social network data and practices: the case of friendfeed'. In: *International Conference on Social Computing, Behavioral Modeling, and Prediction*. Springer, pp. 346–353.

Chalabi, ZS, BJ Bailey and DJ Wilkinson (1996). 'A real-time optimal control algorithm for greenhouse heating'. In: *Computers and electronics in agriculture* 15.1, 1?13.

Challa, H (1990). 'Crop growth models for greenhouse climate control.' In: *Theoretical production ecology: reflections and prospects*. Pudoc, 125?145.

Chenu, K et al. (2018). 'Integrating modelling and phenotyping approaches to identify and screen complex traits: transpiration efficiency in cereals'. In: *Journal of experimental botany* 69.13, pp. 3181–3194.

Chew, Yin Hoon, Daniel D Seaton et al. (2017). 'Linking circadian time to growth rate quantitatively via carbon metabolism'. In: *biorxiv*, p. 105437.

Chew, Yin Hoon, Robert W Smith, Harriet J Jones, Daniel D Seaton, Ramon Grima and Karen J Halliday (2014). 'Mathematical models light up plant signaling'. In: *The Plant Cell*, tpc–113.

Chew, Yin Hoon, Bénédicte Wenden et al. (2014). 'Multiscale digital Arabidopsis predicts individual organ and whole-organism growth'. en. In: *Proceedings of the National Academy of Sciences* 111.39, E4127–E4136. ISSN: 0027-8424, 1091-6490. DOI: 10.1073/pnas.1410238111.

Chew, Yin Hoon, Amity M Wilczek, Mathew Williams, Stephen M Welch, Johanna Schmitt and Karen J Halliday (2012). 'An augmented Arabidopsis phenology model reveals seasonal temperature control of flowering time'. In: *New Phytologist* 194.3, pp. 654–665.

Chiang, George CK et al. (2011). 'DOG1 expression is predicted by the seed-maturation environment and contributes to geographical variation in germination in Arabidopsis thaliana'. In: *Molecular Ecology* 20.16, pp. 3336–3349.

Chomsky, Noam (2014). *Aspects of the Theory of Syntax*. Vol. 11. MIT press.

Christophe, Angélique, Véronique Letort, Irène Hummel, Paul-Henry Cournède, Philippe de Reffye and Jérémie Lecœur (2008). 'A model-based analysis of the dynamics of carbon balance at the whole-plant level in Arabidopsis thaliana'. en. In: *Functional Plant Biology* 35.11, pp. 1147–1162. ISSN: 1445-4416. DOI: 10.1071/FP08099.

Chuine, Isabelle and Elisabeth G Beaubien (2001). 'Phenology is a major determinant of tree species range'. In: *Ecology Letters* 4.5, pp. 500–510.

Chuine, Isabelle, Iñaki Garcia de Cortazar-Atauri, Koen Kramer and Heikki Hänninen (2013). 'Plant development models'. In: *Phenology: an integrative environmental science*. Springer, pp. 275–293.

Coppo, Mario, Ferruccio Damiani, Maurizio Drocco, Elena Grassi, Eva Sciacca et al. (2010). 'Hybrid calculus of wrapped compartments'. In: *Proceedings Compendium of the Fourth Workshop on Membrane Computing and Biologically Inspired Process Calculi (MeCBIC'10)*. Vol. 40, pp. 103–121.

Coppo, Mario, Ferruccio Damiani, Maurizio Drocco, Elena Grassi and Angelo Troina (n.d.). 'Stochastic Calculus of Wrapped Compartments'. In: *EPTCS* 28, pp. 82–98.

Danos, Vincent, Jérôme Feret, Walter Fontana, Russell Harmer and Jean Krivine (2008). 'Rule-Based Modelling, Symmetries, Refinements'. en. In: *Formal Methods in Systems Biology*. Lecture Notes in Computer Science. Springer, Berlin, Heidelberg, pp. 103–122. ISBN: 978-3-540-68410-7 978-3-540-68413-8. DOI: 10.1007/978-3-540-68413-8_8.

Danos, Vincent, Jérôme Feret, Walter Fontana and Jean Krivine (2007). 'Scalable simulation of cellular signaling networks'. In: *Asian Symposium on Programming Languages and Systems*. Springer, pp. 139–157.

Danos, Vincent and Cosimo Laneve (2004). 'Formal molecular biology'. In: *Theoretical Computer Science* 325.1, pp. 69–110.

Debieu, Marilyne et al. (2013). 'Co-Variation between Seed Dormancy, Growth Rate and Flowering Time Changes with Latitude in Arabidopsis thaliana'. In: *PLoS ONE*

8.5. Ed. by Justin O. Borevitz, e61075. ISSN: 1932-6203. DOI: 10.1371/journal.pone.0061075.

Dee, Dick P et al. (2011). 'The ERA-Interim reanalysis: Configuration and performance of the data assimilation system'. In: *Quarterly Journal of the royal meteorological society* 137.656, pp. 553–597.

Despommier, Dickson (2013). 'Farming up the city: the rise of urban vertical farms'. In: *Trends in biotechnology* 31.7, pp. 388–389.

Dobzhansky, Theodosius (1964). *Genetics and the origin of species*. 3 edition, revised.. New York: Columbia University Press.

Doebeli, Michael, Yaroslav Ispolatov and Burt Simon (2017). 'Towards a mechanistic foundation of evolutionary theory'. In: *eLife* 6.

Earley, Eric J, Bronson Ingland, Jacob Winkler and Stephen J Tonsor (2009). 'Inflorescences contribute more than rosettes to lifetime carbon gain in Arabidopsis thaliana (Brassicaceae)'. In: *American Journal of Botany* 96.4, pp. 786–792.

Ederer, Michael, Thomas Sauter, Eric Bullinger, Ernst-Dieter Gilles and Frank Allgöwer (2003). 'An approach for dividing models of biological reaction networks into functional units'. In: *Simulation* 79.12, pp. 703–716.

Egea, Jose A et al. (2014). 'MEIGO: an open-source software suite based on metaheuristics for global optimization in systems biology and bioinformatics'. In: *BMC bioinformatics* 15.1, p. 136.

Eigen, Manfred (1971). 'Selforganization of matter and the evolution of biological macromolecules'. In: *Naturwissenschaften* 58.10, pp. 465–523.

Engl, Heinz W, Christoph Flamm, Philipp Kügler, James Lu, Stefan Müller and Peter Schuster (2009). 'Inverse problems in systems biology'. In: *Inverse Problems* 25.12, p. 123014.

European Commission (2011). *Commission implementing regulation (EU) no 543/2011 laying down detailed rules for the application of Council Regulation (EC) No 1234/2007 in respect of the fruit and vegetables and processed fruit and vegetables sectors*.
https://eur-lex.europa.eu/legal-content/en/ALL/?uri=CELEX:32011R0543.

Farhadifar, Reza, Jens-Christian Röper, Benoit Aigouy, Suzanne Eaton and Frank Jülicher (2007). 'The influence of cell mechanics, cell-cell interactions, and proliferation on epithelial packing'. In: *Current Biology* 17.24, pp. 2095–2104.

Farquhar, GD v, S von von Caemmerer and JA Berry (1980). 'A biochemical model of photosynthetic CO2 assimilation in leaves of C3 species'. In: *Planta* 149.1, pp. 78–90.

Felleisen, Matthias (1991). 'On the expressive power of programming languages'. In: *Science of computer programming* 17.1-3, pp. 35–75.

Fisher, Paul R, Royal D Heins, Niels Ehler and J Heinrich Lieth (1997). 'A decision-support system for real-time management of Easter lily (Lilium longiflorum Thunb.) scheduling and height?I. System description'. In: *Agricultural Systems* 54.1, 23?37.

Fitting, Melvin (2004). 'First-order intensional logic'. In: *Annals of pure and applied logic* 127.1-3, pp. 171–193.

— (2015). 'Intensional Logic'. In: *The Stanford Encyclopedia of Philosophy*. Ed. by Edward N. Zalta. Summer 2015. Metaphysics Research Lab, Stanford University.

Fleisher, David H et al. (2017). 'A potato model intercomparison across varying climates and productivity levels'. In: *Global change biology* 23.3, pp. 1258–1281.

Fletcher, Alexander G, Miriam Osterfield, Ruth E Baker and Stanislav Y Shvartsman (2014). 'Vertex models of epithelial morphogenesis'. In: *Biophysical journal* 106.11, pp. 2291–2304.

Fontana, W. and L. W. Buss (1994). 'What would be conserved if "the tape were played twice"?' en. In: *Proceedings of the National Academy of Sciences* 91.2, pp. 757–761. ISSN: 0027-8424, 1091-6490. DOI: 10.1073/pnas.91.2.757.

Fontana, Walter and Leo W Buss (1996). *The Barrier of Objects: From Dynamical Systems to Bounded Organizations*. URL: %5Curl%7Bhttp://pure.iiasa.ac.at/4999%7D.

Formosa-Jordan, Pau, José Teles and Henrik Jönsson (2018). 'Single-Cell Approaches for Understanding Morphogenesis Using Computational Morphodynamics'. In: *Mathematical Modelling in Plant Biology*. Ed. by Richard J.Editor Morris. Springer International Publishing, pp. 87–106. ISBN: 978-3-319-99070-5. DOI: 10.1007/978-3-319-99070-5_6. URL: https://doi.org/10.1007/978-3-319-99070-5_6.

France, James, John HM Thornley et al. (1984). *Mathematical models in agriculture.* Butterworths.

Fritzson, Peter and Vadim Engelson (1998). 'Modelica—A unified object-oriented language for system modeling and simulation'. In: *European Conference on Object-Oriented Programming*. Springer, pp. 67–90.

Gao, Qian, David Gilbert, Monika Heiner, Fei Liu, Daniele Maccagnola and David Tree (2013). 'Multiscale modeling and analysis of planar cell polarity in the Drosophila wing'. In: *IEEE/ACM Transactions on Computational Biology and Bioinformatics (TCBB)* 10.2, pp. 337–351.

Gibbons, Jeremy (2015). 'Functional Programming for Domain-Specific Languages'. en. In: *Central European Functional Programming School*. Ed. by Viktória Zsók, Zoltán Horváth and Lehel Csató. Lecture Notes in Computer Science 8606. Springer International Publishing, pp. 1–28. ISBN: 978-3-319-15939-3 978-3-319-15940-9. DOI: 10.1007/978-3-319-15940-9_1.

Gilbert, David, Monika Heiner, Fei Liu and Nigel Saunders (2013). 'Colouring space-a coloured framework for spatial modelling in systems biology'. In: *International Conference on Applications and Theory of Petri Nets and Concurrency*. Springer, pp. 230–249.

Gillespie, Daniel T (1977). 'Exact stochastic simulation of coupled chemical reactions'. In: *The journal of physical chemistry* 81.25, pp. 2340–2361.

Giorgidze, George and Henrik Nilsson (2010). 'Mixed-Level Embedding and JIT Compilation for an Iteratively Staged DSL'. en. In: *Functional and Constraint Logic Programming*. Lecture Notes in Computer Science. Springer, Berlin, Heidelberg, pp. 48–65. ISBN: 978-3-642-20774-7 978-3-642-20775-4. DOI: 10.1007/978-3-642-20775-4_3.

Godin, C. and Y. Caraglio (1998). 'A Multiscale Model of Plant Topological Structures'. In: *Journal of Theoretical Biology* 191.1, pp. 1–46. ISSN: 0022-5193. DOI: 10.1006/jtbi.1997.0561.

Granier, Christine et al. (2006). 'PHENOPSIS, an automated platform for reproducible phenotyping of plant responses to soil water deficit in Arabidopsis thaliana permitted the identification of an accession with low sensitivity to soil water deficit'. In: *New Phytologist* 169.3, pp. 623–635.

Griffin, Timothy and Leonid Libkin (1995). 'Incremental Maintenance of Views with Duplicates'. In: *Proceedings of the 1995 ACM SIGMOD International Conference on Management of Data*. SIGMOD '95. New York, NY, USA: ACM, pp. 328–339. ISBN: 978-0-89791-731-5. DOI: 10.1145/223784.223849.

Gruel, Jérémy et al. (2016). 'An epidermis-driven mechanism positions and scales stem cell niches in plants'. In: *Science advances* 2.1, e1500989.

Gunawardena, Jeremy (2012). 'Some lessons about models from Michaelis and Menten'. In: *Molecular biology of the cell* 23.4, pp. 517–519.

— (2013). 'Biology is more theoretical than physics'. In: *Molecular biology of the cell* 24.12, pp. 1827–1829.

— (2014). 'Models in biology: 'accurate descriptions of our pathetic thinking''. In: *BMC Biology* 12.1, p. 29. ISSN: 1741-7007. DOI: 10.1186/1741-7007-12-29.

Hammer, Graeme et al. (2006). 'Models for navigating biological complexity in breeding improved crop plants'. In: *Trends in plant science* 11.12, pp. 587–593.

Hardy, Simon and Pierre N Robillard (2004). 'Modeling and simulation of molecular biology systems using petri nets: modeling goals of various approaches'. In: *Journal of bioinformatics and computational biology* 2.04, pp. 619–637.

Hartwell, Leland H, John J Hopfield, Stanislas Leibler and Andrew W Murray (1999). 'From molecular to modular cell biology'. In: *Nature* 402.6761supp, p. C47.

Harun, Ahmad Nizar, Robiah Ahmad and Norliza Mohamed (2015). 'Plant growth optimization using variable intensity and Far Red LED treatment in indoor farming'. In: *Smart Sensors and Application (ICSSA), 2015 International Conference on*. IEEE, 92?97.

Haverkort, B., H. Hermanns, J. Katoen and C. Baier (2003). 'Model-Checking Algorithms for Continuous-Time Markov Chains'. In: *IEEE Transactions on Software Engineering* 29.6, pp. 524–541. ISSN: 0098-5589. DOI: 10.1109/TSE.2003.1205180.

Heath, John, Marta Kwiatkowska, Gethin Norman, David Parker and Oksana Tymchyshyn (2008). 'Probabilistic model checking of complex biological pathways'. In: *Theoretical Computer Science*. Converging Sciences: Informatics and Biology 391.3, pp. 239–257. ISSN: 0304-3975. DOI: 10.1016/j.tcs.2007.11.013.

Heiner, Monika, Mostafa Herajy, Fei Liu, Christian Rohr and Martin Schwarick (2012). 'Snoopy–a unifying Petri net tool'. In: *International Conference on Application and Theory of Petri Nets and Concurrency*. Springer, pp. 398–407.

Hertel, Stefanie, Christian Brettschneider and Ilka M Axmann (2013). 'Revealing a two-loop transcriptional feedback mechanism in the cyanobacterial circadian clock'. In: *PLoS computational biology* 9.3, e1002966.

Heuvelink, E (1999). 'Evaluation of a dynamic simulation model for tomato crop growth and development'. In: *Annals of Botany* 83.4, pp. 413–422.

Hodgkin, Alan L and Andrew F Huxley (1952). 'A quantitative description of membrane current and its application to conduction and excitation in nerve'. In: *The Journal of physiology* 117.4, pp. 500–544.

Holzhütter, Hermann-Georg, Dirk Drasdo, Tobias Preusser, Jörg Lippert and Adriano M Henney (2012). 'The virtual liver: a multidisciplinary, multilevel challenge for systems biology'. In: *Wiley Interdisciplinary Reviews: Systems Biology and Medicine* 4.3, pp. 221–235.

Honorato-Zimmer, Ricardo, Andrew J Millar, Gordon D Plotkin and Argyris Zardilis (2017). 'Chromar, a language of parameterised agents'. In: *Theoretical Computer Science*.

Honorato-Zimmer, Ricardo, Andrew J. Millar, Gordon D. Plotkin and Argyris Zardilis (2018). 'Chromar, a Rule-based Language of Parameterised Objects'. In: *Electronic Notes in Theoretical Computer Science*. 7th International Workshop on Static Analysis and Systems Biology (SASB 2016) 335, pp. 49–66. ISSN: 1571-0661. DOI: 10.1016/j.entcs.2018.03.008.

Hoops, Stefan et al. (2006). 'COPASI—a complex pathway simulator'. In: *Bioinformatics* 22.24, pp. 3067–3074.

Hucka, Michael et al. (2003). 'The systems biology markup language (SBML): a medium for representation and exchange of biochemical network models'. In: *Bioinformatics* 19.4, pp. 524–531.

Hudak, P. (1998). 'Modular domain specific languages and tools'. In: *Proceedings. Fifth International Conference on Software Reuse (Cat. No.98TB100203)*, pp. 134–142. DOI: 10.1109/ICSR.1998.685738.

Hutton, Graham (1998). 'Fold and Unfold for Program Semantics'. In: *Proceedings of the Third ACM SIGPLAN International Conference on Functional Programming*. ICFP '98. New York, NY, USA: ACM, pp. 280–288. ISBN: 978-1-58113-024-9. DOI: 10.1145/289423.289457.

Iverson, Kenneth E (2007). 'Notation as a tool of thought'. In: *ACM SIGAPL APL Quote Quad* 35.1-2, pp. 2–31.

Jensen, Kurt (1987). 'Coloured petri nets'. In: *Petri nets: central models and their properties*. Springer, pp. 248–299.

John, Mathias, Cédric Lhoussaine, Joachim Niehren and Cristian Versari (2011). 'Biochemical reaction rules with constraints'. In: *European Symposium on Programming*. Springer, pp. 338–357.

Jönsson, Henrik, Marcus G Heisler, Bruce E Shapiro, Elliot M Meyerowitz and Eric Mjolsness (2006). 'An auxin-driven polarized transport model for phyllotaxis'. In: *Proceedings of the National Academy of Sciences* 103.5, pp. 1633–1638.

Karr, Jonathan R et al. (2012). 'A whole-cell computational model predicts phenotype from genotype'. In: *Cell* 150.2, pp. 389–401.

Kauffman, Stuart A (1992). 'The origins of order: Self-organization and selection in evolution'. In: *Spin glasses and biology*. World Scientific, pp. 61–100.

Keating, Brian A et al. (2003). 'An overview of APSIM, a model designed for farming systems simulation'. In: *European journal of agronomy* 18.3-4, pp. 267–288.

Kinmonth-Schultz, Hannah A, Melissa J MacEwen, Daniel D Seaton, Andrew J Millar, Takato Imaizumi and Soo-Hyung Kim (2018). 'Mechanistic model of temperature influence on flowering through whole-plant accumulation of FT'. In: *bioRxiv*, p. 267104.

Kirk, Donald E (2012). *Optimal control theory: an introduction*. Courier Corporation.

Kitano, Hiroaki (2002). 'Systems biology: a brief overview'. In: *Science* 295.5560, pp. 1662–1664.

Kohl, Peter and Denis Noble (2009). 'Systems biology and the virtual physiological human'. In: *Molecular Systems Biology* 5.1, p. 292.

Kraft, Dieter (1985). 'On Converting Optimal Control Problems into Nonlinear Programming Problems'. en. In: *Computational Mathematical Programming*. NATO ASI Series. Springer, Berlin, Heidelberg, pp. 261–280. ISBN: 978-3-642-82452-4 978-3-642-82450-0. DOI: 10.1007/978-3-642-82450-0_9.

Krivine, Jean, Robin Milner and Angelo Troina (2008). 'Stochastic bigraphs'. In: *Electronic Notes in Theoretical Computer Science* 218, pp. 73–96.

Kwiatkowska, Marta, Gethin Norman and David Parker (2007). 'Stochastic Model Checking'. In: *Formal Methods for Performance Evaluation: 7th International School on Formal Methods for the Design of Computer, Communication, and Software Systems, SFM 2007, Bertinoro, Italy, May 28-June 2, 2007, Advanced Lectures*. Ed. by Marco Bernardo and JaneEditors Hillston. Lecture Notes in Computer Science. Springer Berlin Heidelberg, pp. 220–270. ISBN: 978-3-540-72522-0. DOI: 10.1007/978-3-540-72522-0_6. URL: https://doi.org/10.1007/978-3-540-72522-0_6.

— (2011). 'PRISM 4.0: Verification of Probabilistic Real-Time Systems'. In: *Computer Aided Verification*. Ed. by Ganesh Gopalakrishnan and ShazEditors Qadeer. Lecture Notes in Computer Science. Springer Berlin Heidelberg, pp. 585–591. ISBN: 978-3-642-22110-1.

Lang, Meagan (2018). *cis_interface*. URL: https://github.com/cropsinsilico/cis_interface.

Laurent, Jonathan, Hector F Medina-Abarca, Pierre Boutillier, Jean Yang and Walter Fontana (2018). 'A Trace Query Language for Rule-based Models'. en. In: p. 19. URL: https://fontana.hms.harvard.edu/sites/fontana.hms.harvard.edu/files/documents/traceQL.pdf.

Leonardos, Evangelos D et al. (2014). 'Photosynthetic capacity of the inflorescence is a major contributor to daily-C-gain and the responsiveness of growth to elevated CO2 in Arabidopsis thaliana with repressed expression of mitochondrial-pyruvate-dehydrogenase-kinase'. In: *Environmental and experimental botany* 107, pp. 84–97.

Li, Wenyong, A Abel, K Todtermuschke and Tong Zhang (2007). 'Hybrid vehicle power transmission modeling and simulation with simulationX'. In: *Mechatronics and Automation, 2007. ICMA 2007. International Conference on*. IEEE, pp. 1710–1717.

Libkin, Leonid and Limsoon Wong (1997). 'Query languages for bags and aggregate functions'. In: *Journal of Computer and System sciences* 55.2, pp. 241–272.

Locke, James CW et al. (2005). 'Extension of a genetic network model by iterative experimentation and mathematical analysis'. In: *Molecular systems biology* 1.1.

Lopez, Carlos F, Jeremy L Muhlich, John A Bachman and Peter K Sorger (2013). 'Programming biological models in Python using PySB'. In: *Molecular systems biology* 9.1, p. 646.

Lynch, Michael and Bruce Walsh (2007). *The origins of genome architecture*. Vol. 98. Sinauer Associates Sunderland (MA).

Mainland, Geoffrey (2007). 'Why It's Nice to Be Quoted: Quasiquoting for Haskell'. In: *Proceedings of the ACM SIGPLAN Workshop on Haskell Workshop*. Haskell '07. New York, NY, USA: ACM, pp. 73–82. ISBN: 978-1-59593-674-5. DOI: 10.1145/1291201.1291211. URL: http://doi.acm.org/10.1145/1291201.1291211.

Maloney, John, Mitchel Resnick, Natalie Rusk, Brian Silverman and Evelyn Eastmond (2010). 'The scratch programming language and environment'. In: *ACM Transactions on Computing Education (TOCE)* 10.4, p. 16.

Marr, David (1982). 'Vision: A computational investigation into the human representation and processing of visual information. MIT Press'. In: *Cambridge, Massachusetts*.

Marshall-Colon, Amy et al. (2017). 'Crops in silico: Generating virtual crops using an integrative and multi-scale modeling platform'. In: *Frontiers in plant science* 8, p. 786.

Martre, Pierre et al. (2015). 'Multimodel ensembles of wheat growth: many models are better than one'. In: *Global change biology* 21.2, pp. 911–925.

Mendez-Vigo, Belen, Xavier F. Picó, Mercedes Ramiro, José M. Martínez-Zapater and Carlos Alonso-Blanco (2011). 'Altitudinal and climatic adaptation is mediated by flowering traits and FRI, FLC and PhyC genes in Arabidopsis thaliana'. en. In: *Plant Physiology*, pp.111.183426. ISSN: 0032-0889, 1532-2548. DOI: 10.1104/pp.111.183426.

Michaelides, Michalis, Dimitrios Milios, Jane Hillston and Guido Sanguinetti (2016). 'Property-driven state-space coarsening for continuous time Markov chains'. In: *International Conference on Quantitative Evaluation of Systems*. Springer, pp. 3–18.

Michaelis, Leonor and Maude L Menten (1913). 'The kinetics of the inversion effect'. In: *Biochem. Z* 49, pp. 333–369.

Mikulecky, Donald C (2001). 'Robert Rosen (1934-1998): a snapshot of biology's Newton'. In: *Computers and Chemistry* 4.25, pp. 317–327.

Millar, Andrew J (2016). 'The intracellular dynamics of circadian clocks reach for the light of ecology and evolution'. In: *Annual Review of Plant Biology* 67, pp. 595–618.

Milner, Robin (1999). *Communicating and mobile systems: the pi calculus*. Cambridge university press.

Mirams, Gary R et al. (2013). 'Chaste: an open source C++ library for computational physiology and biology'. In: *PLoS computational biology* 9.3, e1002970.

Mironova, Victoria V et al. (2010). 'A plausible mechanism for auxin patterning along the developing root'. In: *BMC systems biology* 4.1, p. 98.

Miyoshi, Fumihiko, Yoichi Nakayama, Kazunari Kaizu, Hideo Iwasaki and Masaru Tomita (2007). 'A mathematical model for the Kai-protein–based chemical oscillator and clock gene expression rhythms in Cyanobacteria'. In: *Journal of Biological Rhythms* 22.1, pp. 69–80.

Mjolsness, Eric (2018). 'Prospects for Declarative Mathematical Modeling of Complex Biological Systems'. In:

Mjolsness, Eric and Guy Yosiphon (2006). 'Stochastic process semantics for dynamical grammars'. In: *Annals of Mathematics and Artificial Intelligence* 47.3-4, pp. 329–395.

Mladenović, Nenad and Pierre Hansen (1997). 'Variable neighborhood search'. In: *Computers & operations research* 24.11, pp. 1097–1100.

Mok, Hoi-Fei, Virginia G. Williamson, James R. Grove, Kristal Burry, S. Fiona Barker and Andrew J. Hamilton (2014). 'Strawberry fields forever? Urban agriculture in developed countries: a review'. en. In: *Agronomy for Sustainable Development* 34.1, pp. 21–43. ISSN: 1774-0746, 1773-0155. DOI: 10.1007/s13593-013-0156-7.

Moles, Carmen G, Pedro Mendes and Julio R Banga (2003). 'Parameter estimation in biochemical pathways: a comparison of global optimization methods'. In: *Genome research* 13.11, pp. 2467–2474.

Monteith, JL (1965). 'Light distribution and photosynthesis in field crops'. In: *Annals of Botany* 29.1, pp. 17–37.

Muetzelfeldt, Robert and Jon Massheder (2003). 'The Simile visual modelling environment'. In: *European Journal of Agronomy* 18.3, pp. 345–358.

Mündermann, Lars, Yvette Erasmus, Brendan Lane, Enrico Coen and Przemyslaw Prusinkiewicz (2005). 'Quantitative Modeling of Arabidopsis Development'. en. In: *Plant Physiology* 139.2, pp. 960–968. ISSN: 0032-0889, 1532-2548. DOI: 10.1104/pp.105.060483.

Noble, Denis (2002). 'Modeling the heart–from genes to cells to the whole organ'. In: *Science* 295.5560, pp. 1678–1682.

Nytsch-Geusen, Christoph et al. (2005). 'MOSILAB: Development of a Modelica based generic simulation tool supporting model structural dynamics'. In:

Otter, Martin, Hilding Elmqvist and François E Cellier (1996). 'Modeling of multibody systems with the object-oriented modeling language Dymola'. In: *Nonlinear Dynamics* 9.1-2, pp. 91–112.

Oury, Nicolas and Gordon Plotkin (2013). 'Multi-level modelling via stochastic multi-level multiset rewriting'. In: *Mathematical Structures in Computer Science* 23.02, pp. 471–503.

Oury, Nicolas and Gordon D Plotkin (2011). 'Coloured stochastic multilevel multiset rewriting'. In: *Proceedings of the 9th International Conference on Computational Methods in Systems Biology*. ACM, pp. 171–181.

Paoletti, Nicola, Pietro Lio, Emanuela Merelli and Marco Viceconti (2012). 'Multi-level computational modeling and quantitative analysis of bone remodeling'. In: *IEEE/ACM Transactions on Computational Biology and Bioinformatics (TCBB)* 9.5, pp. 1366–1378.

Parent, Boris and François Tardieu (2014). 'Can current crop models be used in the phenotyping era for predicting the genetic variability of yield of plants subjected to drought or high temperature?' In: *Journal of experimental botany* 65.21, pp. 6179–6189.

Păun, Andrei (2001). 'On P systems with active membranes'. In: *Unconventional Models of Computation, UMC'2K*. Springer, pp. 187–201.

Păun, Gheorghe (2000). 'Computing with membranes'. In: *Journal of Computer and System Sciences* 61.1, pp. 108–143.

Pedersen, Michael, Andrew Phillips and Gordon D Plotkin (2015). 'A high-level language for rule-based modelling'. In: *PloS one* 10.6, e0114296.

Pouteau, Sylvie and Catherine Albertini (2009). 'The significance of bolting and floral transitions as indicators of reproductive phase change in Arabidopsis'. en. In: *Journal of Experimental Botany* 60.12, pp. 3367–3377. ISSN: 0022-0957. DOI: 10.1093/jxb/erp173.

Pradal, Christophe, Samuel Dufour-Kowalski, Frédéric Boudon, Christian Fournier and Christophe Godin (2008). 'OpenAlea: a visual programming and component-based software platform for plant modelling'. en. In: *Functional Plant Biology* 35.10, pp. 751–760. ISSN: 1445-4416. DOI: 10.1071/FP08084.

Prusinkiewicz, Przemyslaw, Scott Crawford et al. (2009). 'Control of bud activation by an auxin transport switch'. In: *Proceedings of the National Academy of Sciences* 106.41, pp. 17431–17436.

Prusinkiewicz, Przemyslaw, Yvette Erasmus, Brendan Lane, Lawrence D. Harder and Enrico Coen (2007). 'Evolution and Development of Inflorescence Architectures'. en. In: *Science* 316.5830, pp. 1452–1456. ISSN: 0036-8075, 1095-9203. DOI: 10.1126/science.1140429.

Rajcan, Irena and Clarence J Swanton (2001). 'Understanding maize–weed competition: resource competition, light quality and the whole plant'. In: *Field Crops Research* 71.2, pp. 139–150.

Ramírez-Arias, A, Francisco Rodríguez, José Luis Guzmán and Manuel Berenguel (2012). 'Multiobjective hierarchical control architecture for greenhouse crop growth'. In: *Automatica* 48.3, 490?498.

Rashevsky, N. (1954). 'Topology and life: In search of general mathematical principles in biology and sociology'. en. In: *The bulletin of mathematical biophysics* 16.4, pp. 317–348. ISSN: 0007-4985, 1522-9602. DOI: 10.1007/BF02484495.

Rasse, Daniel P. and Pierre Tocquin (2006). 'Leaf carbohydrate controls over Arabidopsis growth and response to elevated CO2: an experimentally based model'. en. In: *New Phytologist* 172.3, pp. 500–513. ISSN: 1469-8137. DOI: 10.1111/j.1469-8137.2006.01848.x.

Ravasz, Erzsébet, Anna Lisa Somera, Dale A Mongru, Zoltán N Oltvai and A-L Barabási (2002). 'Hierarchical organization of modularity in metabolic networks'. In: *science* 297.5586, pp. 1551–1555.

RBM group, Edinburgh (2015). *mois: Module Integration Simulator v2*. URL: https://github.com/edinburgh-rbm/mois.

Rodriguez, Nathaniel, Johan Bollen and Yong-Yeol Ahn (2016). 'Collective dynamics of belief evolution under cognitive coherence and social conformity'. In: *PLoS one* 11.11, e0165910.

Rosen, Robert (1958). 'A relational theory of biological systems'. en. In: *The bulletin of mathematical biophysics* 20.3, pp. 245–260. ISSN: 0007-4985, 1522-9602. DOI: 10.1007/BF02478302.

— (1991). *Life itself: a comprehensive inquiry into the nature, origin, and fabrication of life*. Columbia University Press.

Rubin, Matthew J et al. (2018). 'Circadian rhythms are associated with shoot architecture in natural settings'. In: *New Phytologist* 219.1, pp. 246–258.

Runge, Thomas (2004). 'Application of coloured Petri nets in systems biology'. In: *Proceedings of the Fifth Workshop and Tutorial on Practical Use of Coloured Petri Nets and the CPN Tools*. Citeseer, pp. 77–96.

Sagrado, José del, JA Sanchez, Francisco Rodríguez and Manuel Berenguel (2016). 'Bayesian networks for greenhouse temperature control'. In: *Journal of Applied Logic* 17, 25?35.

Salazar, José Domingo et al. (2009). 'Prediction of photoperiodic regulators from quantitative gene circuit models'. In: *Cell* 139.6, pp. 1170–1179.

Satake, Akiko, Tetsuhiro Kawagoe, Yukari Saburi, Yukako Chiba, Gen Sakurai and Hiroshi Kudoh (2013). 'Forecasting flowering phenology under climate warming by modelling the regulatory dynamics of flowering-time genes'. In: *Nature communications* 4, p. 2303.

SBML Level 3 specification (n.d.). URL: http://sbml.org/Documents/Specifications/SBML_Level_3.

Schelling, Thomas C (1971). 'Dynamic models of segregation'. In: *Journal of mathematical sociology* 1.2, pp. 143–186.

Schnoerr, David, Guido Sanguinetti and Ramon Grima (2017). 'Approximation and inference methods for stochastic biochemical kinetics—a tutorial review'. In: *Journal of Physics A: Mathematical and Theoretical* 50.9, p. 093001.

Simon, Herbert A (1962). 'The Architecture of Complexity'. In: *Proceedings of the American Philosophical Society* 106.6, pp. 467–482.

Solovyev, Alexey et al. (2010). 'SPARK: a framework for multi-scale agent-based biomedical modeling'. In: *Proceedings of the 2010 Spring Simulation Multiconference*. Society for Computer Simulation International, p. 3.

Springthorpe, Vicki and Steven Penfield (2015). 'Flowering time and seed dormancy control use external coincidence to generate life history strategy'. In: *Elife* 4.

Tardieu, F and WJ Davies (1993). 'Integration of hydraulic and chemical signalling in the control of stomatal conductance and water status of droughted plants'. In: *Plant, Cell & Environment* 16.4, pp. 341–349.

Tardieu, François, Thierry Simonneau and Boris Parent (2015). 'Modelling the coordination of the controls of stomatal aperture, transpiration, leaf growth, and abscisic acid: update and extension of the Tardieu–Davies model'. In: *Journal of Experimental Botany* 66.8, pp. 2227–2237.

Thompson, Darcy Wentworth et al. (1942). 'On growth and form.' In: *On growth and form.*

Toni, Tina and Michael PH Stumpf (2009). 'Simulation-based model selection for dynamical systems in systems and population biology'. In: *Bioinformatics* 26.1, pp. 104–110.

Turing, Alan Mathison (1952). 'The chemical basis of morphogenesis'. en. In: *Phil. Trans. R. Soc. Lond. B* 237.641, pp. 37–72. ISSN: 0080-4622, 2054-0280. DOI: 10.1098/rstb.1952.0012.

Udink ten Cate, AJ and H Challa (1983). 'On optimal computer control of the crop growth system'. In: *III International Symposium on Energy in Protected Cultivation 148*, 267?276.

Varela, F. G., H. R. Maturana and R. Uribe (1974). 'Autopoiesis: The organization of living systems, its characterization and a model'. In: *Biosystems* 5.4, pp. 187–196. ISSN: 0303-2647. DOI: 10.1016/0303-2647(74)90031-8.

Voliotis, Margaritis, Philipp Thomas, Ramon Grima and Clive G. Bowsher (2016). 'Stochastic Simulation of Biomolecular Networks in Dynamic Environments'. en. In: *PLOS Computational Biology* 12.6, e1004923. ISSN: 1553-7358. DOI: 10.1371/journal.pcbi.1004923.

Voss, Ute, Anthony Bishopp, Etienne Farcot and Malcolm J Bennett (2014). 'Modelling hormonal response and development'. In: *Trends in plant science* 19.5, pp. 311–319.

Waddington, C. H. (Conrad Hal) (1961). *The nature of life*. eng. London: Allen & Unwin.

Walker, Berkley, Loren S Ariza, Sarah Kaines, Murray R Badger and Asaph B Cousins (2013). 'Temperature response of in vivo Rubisco kinetics and mesophyll conductance in Arabidopsis thaliana: comparisons to Nicotiana tabacum'. In: *Plant, Cell & Environment* 36.12, pp. 2108–2119.

Waltemath, Dagmar et al. (2016). 'Toward community standards and software for whole-cell modeling'. In: *IEEE Transactions on Biomedical Engineering* 63.10, pp. 2007–2014.

Wan, Zhanyong and Paul Hudak (2000). 'Functional Reactive Programming from First Principles'. In: *Proceedings of the ACM SIGPLAN 2000 Conference on Programming Language Design and Implementation*. PLDI '00. New York, NY, USA: ACM, pp. 242–252. ISBN: 978-1-58113-199-4. DOI: 10.1145/349299.349331. URL: http://doi.acm.org/10.1145/349299.349331.

Watanabe, Leandro H. and Chris J. Myers (2014). 'Hierarchical Stochastic Simulation Algorithm for SBML Models of Genetic Circuits'. English. In: *Frontiers in Bioengineering and Biotechnology* 2. ISSN: 2296-4185. DOI: 10.3389/fbioe.2014.00055.

Weiße, Andrea Y., Diego A. Oyarzún, Vincent Danos and Peter S. Swain (2015). 'Mechanistic links between cellular trade-offs, gene expression, and growth'. en. In: *Proceedings of the National Academy of Sciences* 112.9, E1038–E1047. ISSN: 0027-8424, 1091-6490. DOI: 10.1073/pnas.1416533112.

Welch, SM, JL Roe, S Das, Z Dong, R He and MB Kirkham (2005). 'Merging genomic control networks and soil-plant-atmosphere-continuum models'. In: *Agricultural Systems* 86.3, pp. 243–274.

Wilczek, Amity M et al. (2009). 'Effects of genetic perturbation on seasonal life history plasticity'. In: *Science* 323.5916, pp. 930–934.

Williams, Mathew, Edward B Rastetter, Gaius R Shaver, John E Hobbie, Elizabeth Carpino and Bonnie L Kwiatkowski (2001). 'Primary production of an arctic watershed: an uncertainty analysis'. In: *Ecological applications* 11.6, pp. 1800–1816.

Willis, Lisa et al. (2016). 'Cell size and growth regulation in the Arabidopsis thaliana apical stem cell niche'. en. In: *Proceedings of the National Academy of Sciences* 113.51, E8238–E8246. ISSN: 0027-8424, 1091-6490. DOI: 10.1073/pnas.1616768113.

Wu, Alex, Youhong Song, Erik J Van Oosterom and Graeme L Hammer (2016). 'Connecting biochemical photosynthesis models with crop models to support crop improvement'. In: *Frontiers in plant science* 7, p. 1518.

Yeager, Larry and Thomas Fiddaman (2014). *Entity-Based System Dynamics*. en.

Yin, Xinyou and Paul C. Struik (2008). 'Applying modelling experiences from the past to shape crop systems biology: the need to converge crop physiology and functional genomics'. en. In: *New Phytologist* 179.3, pp. 629–642. ISSN: 1469-8137. DOI: 10.1111/j.1469-8137.2008.02424.x.

Yin, Xinyou and Paul C Struik (2010). 'Modelling the crop: from system dynamics to systems biology'. In: *Journal of Experimental Botany* 61.8, pp. 2171–2183.

Yin, Xinyou, Paul C Struik and Martin J Kropff (2004). 'Role of crop physiology in predicting gene-to-phenotype relationships'. In: *Trends in Plant Science* 9.9, pp. 426–432.

Zardilis, Argyris, Alastair Hume and Andrew J Millar (2018). 'A multi-model Framework for the Arabidopsis life cycle'. In: *bioRxiv*, p. 358408.

Zhu, Xin-Guang, Jonathan P. Lynch, David S. LeBauer, Andrew J. Millar, Mark Stitt and Stephen P. Long (2016). 'Plants in silico: why, why now and what?—an integrative platform for plant systems biology research'. en. In: *Plant, Cell & Environment* 39.5, pp. 1049–1057. ISSN: 1365-3040. DOI: 10.1111/pce.12673.

Zhu, Xin-Guang, Yu Wang, Donald R Ort and Stephen P Long (2013). 'e-photosynthesis: a comprehensive dynamic mechanistic model of C3 photosynthesis: from light capture to sucrose synthesis'. In: *Plant, cell & environment* 36.9, pp. 1711–1727.

Zimmer, Dirk (2008). 'Introducing Sol: A General Methodology for Equation-Based Modeling of Variable-Structure Systems'. In: *Modelica 2008: proceedings of the 6th International Modelica Conference; March 3rd-4th, 2008, University of Applied Sciences Bielefeld*. Modelica Association, pp. 47–56.

Appendix A

Additional FM-life population simulations

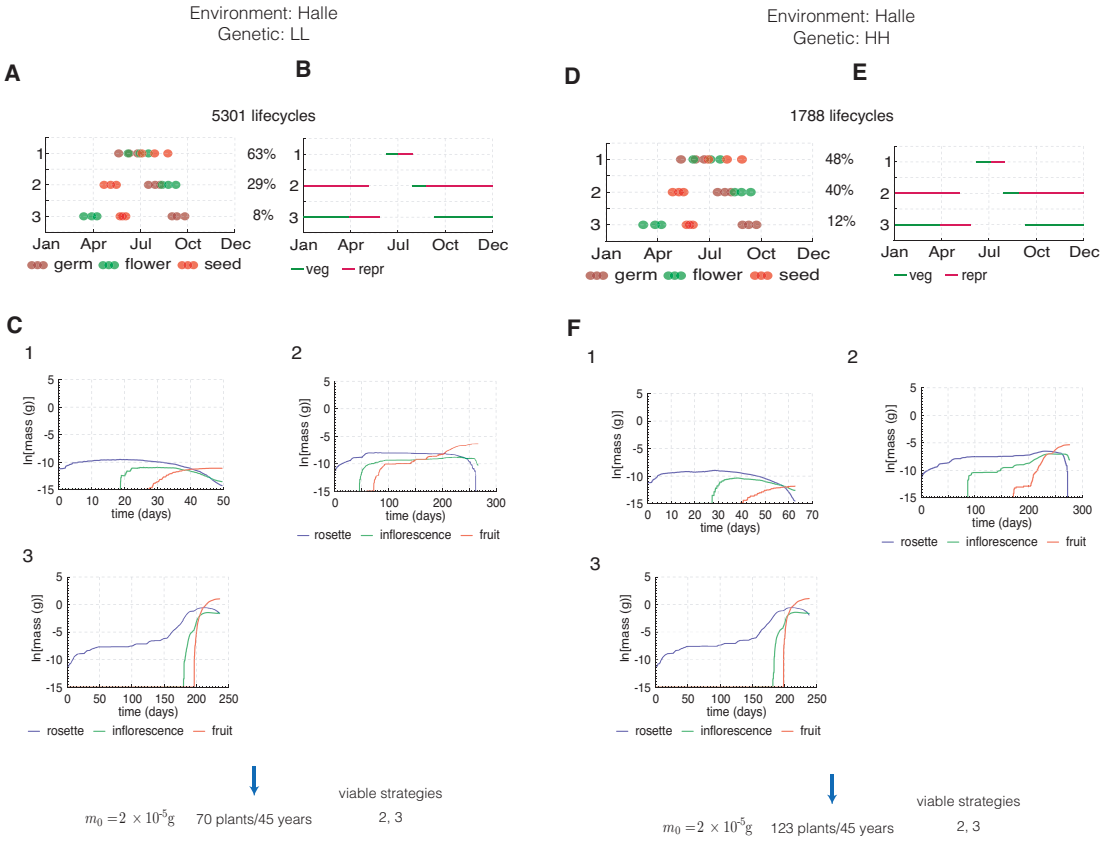


Figure A.1: Population experiments in Halle (Germany) in two different genetic backgrounds (LL:low dormancy, Low floral repression and HH: High dormancy, High floral repression). A Results of the phenology-only simulation results for the LL genotype. The 25-th, 50-th, and 75-th percentile of the distribution of developmental events are shown for each identified cluster B Illustration of growth stages over a year for each cluster from A according to the median time of the distribution of developmental events for each cluster. C Growth simulations over the growth period shown in panel B for each cluster. D, E, F Equivalently for the HH genotype in the same location.

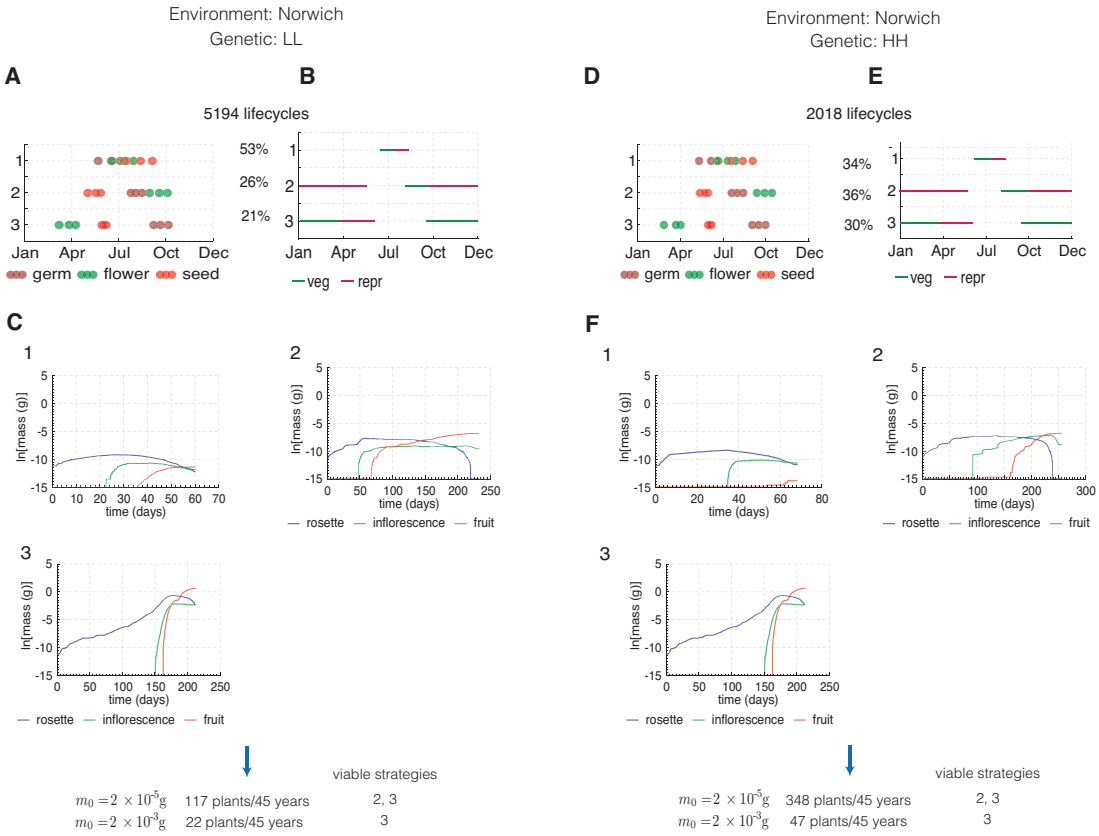


Figure A.2: Population experiments in Norwich (England) in two different genetic backgrounds (LL:low dormancy, Low floral repression and HH: High dormancy, High floral repression). A Results of the phenology-only simulation results for the LL genotype. The 25-th, 50-th, and 75-th percentile of the distribution of developmental events are shown for each identified cluster B Illustration of growth stages over a year for each cluster from A according to the median time of the distribution of developmental events for each cluster. C Growth simulations over the growth period shown in panel B for each cluster. D, E, F Equivalently for the HH genotype in the same location.