Representing Biological Processes in Modular Action Language $\mathcal{ALM}$

Daniela Inclezan and **Michael Gelfond**
Computer Science Department
Texas Tech University
Lubbock, TX 79409 USA
daniela.inclezan@ttu.edu, michael.gelfond@ttu.edu

Abstract

This paper presents the formalization of a biological process, cell division, in modular action language $\mathcal{ALM}$. We show how the features of $\mathcal{ALM}$—modularity, separation between an uninterpreted theory and its interpretation—lead to a simple and elegant solution that can be used in answering questions from biology textbooks.

Introduction

This paper presents the formalization of a biological process, cell division, in the modular action language $\mathcal{ALM}$ (Gelfond and Inclezan 2009). We show how the features of $\mathcal{ALM}$—modularity, separation between an uninterpreted theory and its interpretation—lead to a simple and elegant representation of cell division. We describe a system that uses our formalization in answering related questions from a biology textbook for the undergraduate level (Campbell and Reece 2001).

This work is part of our collaboration on Project Halo, a research effort by Vulcan Inc. towards the development of a "Digital Aristotle"–*"an application containing large volumes of scientific knowledge and capable of applying sophisticated problem-solving methods to answer novel questions"* (Gunning et al. 2010). Initially, it was not clear how to reason about *dynamic* domains in Digital Aristotle; our goal within the project was to create a methodology that would address this issue. We illustrated our methodology on the domain of biology – one of the several scientific disciplines targeted by Digital Aristotle. In particular, our work addressed the topic of cell division.

Cell division (or *cell cycle*) refers to the phases a cell goes through from its "birth" to its division into two daughter cells. Cells consist of a number of parts, which in turn consist of other parts. We will focus here on *eukaryotic* cells (i.e., cells that contain a visibly evident nucleus), but our approach can be easily extended to *prokaryotic cells* (i.e., cells lacking a nucleus). Eukaryotic cells contain organelles, cytoplasm, and a nucleus; the nucleus contains chromosomes, and the description can continue with more detailed parts. The eukaryotic cell cycle consists of an interphase and a mitotic phase. The interphase is mostly a growth phase, preparing the cell for division. The mitotic phase duplicates the nucleus and then divides the cell and the nuclei within it. This results in two daughter cells that are identical to the original one. The interphase and the mitotic phase are conventionally described as sequences of sub-phases. Depending on the level of detail of the description, these sub-phases may be simple events or sequences of other sub-phases. For example, the more detailed mitotic phase is described as a sequence of two sub-phases: mitosis and cytokinesis. Mitosis, in turn, can be seen as a sequence of five sub-phases, etc. Certain chemicals, if introduced in the cell, can interfere with the ordered succession of events that is the cell cycle.

Answering questions about this domain requires knowing what the structure of the cell is at each stage of the cell cycle, and understanding what motivates the succession of phases. There are two main reasons why this is not trivial. First of all, this calls for a method of representing and reasoning about naturally evolving processes: series of phases and sub-phases that follow one another in a pre-determined order unless the process is interrupted by external actions. Secondly, cell division and the cell structure can be viewed at different levels of granularity. Each question about cell division requires a minimal level of detail in order to be answered. The formalization of the domain should allow for an elaboration tolerant transition from a coarse representation of cell division to a finer one, in such a way that conclusions are consistent across representations.

We chose to formalize the domain in the modular action language $\mathcal{ALM}$, and used this experience as a test case for the design goals of the language: the creation of library modules and the reuse of information. We created a couple of small $\mathcal{ALM}$ libraries concerning cell division. These libraries formed a general theory to be used in describing the domain at every level of granularity. Only the interpretation of the symbols appearing in the general theory depended on granularity. This allowed us to transition from a coarse to a finer representation simply via additions to the interpretation and without any change to the general theory. We represented naturally evolving processes as sequences of actions and treated them as if they were intended by nature. For this purpose, we used a *theory of intentions*[1] (Baral and

[1]For now, the theory of intentions is written as a logic program. In the future, it will be a library module of $\mathcal{ALM}$.

Gelfond 2005; Gelfond 2006) containing the following main tenets: *"Normally intended actions are executed the moment such execution becomes possible"* (non-procrastination) and *"Unfulfilled intentions persist"* (persistence).

We evaluated our work by addressing several textbook questions about cell division. The information given in the question was extracted by hand; relations present in the text of the question but absent from our knowledge base were also defined by hand by way of logic programming rules. The minimal level of detail required to answer each question was detected manually; automatic detection is part of future work. The $\mathcal{ALM}$ system descriptions for the various levels of granularity were translated into logic programs. This allowed us to reduce the problem of question answering to computing answer sets (Gelfond and Lifschitz 1991) of a program consisting of: the encoding of the question, the translation of the corresponding system description, and the theory of intentions. In doing so, we applied a known approach for answering non-trivial questions about dynamic domains using logic programs under the answer set semantics (Baral, Gelfond, and Scherl 2004; Baral et al. 2005; Balduccini, Baral, and Lierler 2008). Based on our evaluation, we concluded that $\mathcal{ALM}$'s features are useful in representing and reasoning about domains like cell division in an elegant manner.

This paper is structured as follows: we begin with a short introduction to $\mathcal{ALM}$ by way of examples. Next, we present our formalization of the domain in $\mathcal{ALM}$. We continue with the representation of several questions from the biology textbook. Finally, we describe the architecture of our system and end with conclusions.

Modular Action Language $\mathcal{ALM}$

$\mathcal{ALM}$ is an extension of action language $\mathcal{AL}$ (Turner 1997; Baral and Gelfond 2000) by means for expressing hierarchies of abstractions. In particular, $\mathcal{ALM}$ has a modular structure and makes a clear distinction between an uninterpreted theory and its interpretation. These features allowed us to address the difficulties of formalizing cell division, as will be shown in the next section. A system description of $\mathcal{ALM}$ is a set of modules forming the uninterpreted theory (the **declarations** part), followed by an interpretation of the symbols (the **structure**):

```
system description name
    declarations of name
        module module_name  {...}
    structure of name  {...}
```

Syntactically, a module can be viewed as a collection of declarations of sort, fluent and action classes of the system. We show the syntax of a module by means of an example:

```
module  move_between_areas
 sort declarations
   things, areas  :  sort
   movers  :  things
 fluent declarations
   loc_in(things, areas)  :  inertial fluent
   axioms
     ¬loc_in(T, A2) if disjoint(A1, A2), loc_in(T, A1).
   end of loc_in
 action declarations
   move  :  action
    attributes
      actor : movers
      origin, dest : areas
    axioms
      move  causes  loc_in(O, A)    if    actor = O,
                                          dest = A.
      impossible move    if    actor = O,
                               origin = A,
                               ¬loc_in(O, A).
   end of move
```

This says that *things*, *areas*, and *movers* are sorts, where *movers* is a sub-sort of *things*; *loc_in* is an inertial fluent with parameters of the sorts *things* and *areas*; and *move* is an action class. The fluent *loc_in* is defined by the static causal law included in its **axioms** part. Actions of the type *move* have three attributes: *actor*, with values from the sort *movers*, and *origin* and *dest*, with values from the sort *areas*. The first *move* axiom says that the occurrence of any action that is an instance of *move* and has actor O and destination A causes O to be located in A. The second axiom says that no instance of action class *move* can occur in a state in which its actor is not located at the origin.

Modules can be combined into *libraries* and imported from there using *import* statements. As modules are organized as tree-like hierarchies, actions and fluents can be defined in terms of other actions and fluents. In the example above, the inertial fluent *loc_in* was defined in terms of the fluent *disjoint*, assumed to be declared in a different module. The next example shows how an action class, *carry*, can be declared in terms of an already declared action class:

```
  carry  :  move
    attributes
      carried_thing : carriables
    axioms
      impossible carry    if    actor = O,
                                carried_thing = T,
                                ¬holding(O, T).
  end of carry
```

This says that *carry* is a special case of the action class *move*, meaning that, in addition to its own attributes and axioms, *carry* inherits the attributes and axioms of *move*.

The next part after the declarations of a system description is its interpretation. The following example illustrates the syntax of an interpretation; we assume that the declaration part of the system description *basic_travel* contains the module *move_between_areas*:

```
structure of  basic_travel
  sorts
    bob, john ∈ movers
    london, paris, rome ∈ areas
  actions
    instance  move(O, A1, A2)  :  move
      actor := O
      origin := A1
      dest := A2
```

```
statics
  $disjoint(london, paris)$.
  $disjoint(paris, rome)$.
  $disjoint(rome, london)$.
```

This structure defines objects of sorts $movers$ and $areas$ from our domain, instances of action $move$ and the value of static fluents.

Semantically, a collection of modules can be viewed as a mapping of possible interpretations of the symbols of the domain into the transition diagram describing a dynamic system. A system description $\mathcal{D}$ of $\mathcal{ALM}$ is mapped into ground statements of the non-modular action language $\mathcal{AL}$, which uniquely define the transition diagram of $\mathcal{D}$. For example, the action instance $move(bob, london, paris)$ and the $\mathcal{ALM}$ causal law

$move$ **causes** $loc\_in(O, A)$ **if** $actor = O, dest = A$.

are turned into the $\mathcal{AL}$ causal law

$move(bob, london, paris)$ **causes** $loc\_in(bob, paris)$.

Statements of $\mathcal{AL}$ can then be translated into Answer Set Programming (ASP) rules.

System descriptions of $\mathcal{ALM}$ are normally used in conjunction with the description of the system's history, a collection of facts of the form: $happened(a, i)$ (action a happened at time step i); $observed(f, true/false, i)$ (fluent f was observed to be $true/false$ at i); and $intend(v, i)$ (the execution of the action or sequence of actions v was intended at i). Together, the system description and the history define the collection of possible trajectories of the system up to the current step. These trajectories can be extracted from the answer sets of a logic program consisting of the translation of the $\mathcal{ALM}$ system description, the system's history, and axioms for $happened$, $observed$, and $intend$.

Formalizing Cell Division in $\mathcal{ALM}$

We now present our formalization of cell division. We start by modeling the eukaryotic cell. As seen in the introduction, the eukaryotic cell consists of various parts, which in turn consist of other parts. Together, they form a "*part of*" hierarchy, say H. To model this hierarchy, we introduce a sort called $classes\_of\_parts$ (abbreviated here as c_o_p) and the relations: $father(C_1, C_2)$, where C_1 is the father class of class C_2 in H, and $root(C)$, where C is the root of H. We will be interested in the number of different parts present in the environment during different stages of the cell cycle. Therefore, the states of our domain will be described by an inertial fluent, $num(C_1, C_2, N)$, which holds if the number of parts from class C_1 in one part from the class C_2 is N. For instance, $num(nucleus, cell, 2)$ will indicate that at the current stage of the cell cycle, every cell in the environment has two nuclei.

Next, we present the actions of our domain. To describe the cell cycle we will need two action classes: $duplicate$ and $split$. $Duplicate$, which has an attribute $class$ of sort $classes\_of\_parts$, doubles the number of every part from this class present in the environment. $Split$, which also has an attribute $class$, duplicates elements of this class and cuts in half the number of parts from the daughter classes of $class$. For example, if the environment consists of one cell with two nuclei, the action instance $split(cell)$ will increase the number of cells to two, each containing only one nucleus. In addition to these two actions we will have an exogenous action, $prevent$, which will nullify the effects of duplication and splitting for class C. We will make use of this exogenous action in representing external events that interfere with the normal succession of sub-phases of cell division. The description of the domain is given in a module called $basic\_cell\_cycle$, which will eventually become part of a more general $cell\_cycle$ library module.

```
module $basic\_cell\_cycle$
 sort declarations
  $classes\_of\_parts$ : sort   $\%c\_o\_p$
  $numbers$ : sort
  $even\_numbers$ : $numbers$
 fluent declarations
  $father(c\_o\_p, c\_o\_p)$ : static fluent
  axioms
   $\neg father(C_1, C)$   if   $father(C_2, C)$,
                                $C_1 \neq C_2$.
   $\neg father(C_1, C)$   if   $root(C)$.
  end of $father$
  $root(c\_o\_p)$ : static fluent
  axioms
   $\neg root(C_1)$   if   $root(C_2)$,  $C_1 \neq C_2$.
  end of $root$
  $num(c\_o\_p, c\_o\_p, numbers)$ : inertial fluent
  axioms
   $\neg num(C_1, C_2, N_2)$   if   $num(C_1, C_2, N_1)$,
                                    $N_1 \neq N_2$.
   $num(C_3, C_1, N)$          if   $father(C_1, C_2)$,
                                    $num(C_2, C_1, N_1)$,
                                    $num(C_3, C_2, N_2)$,
                                    $C_3 \neq C_2$,
                                    $N = N_1 * N_2$.
  end of $num$
  $prevented\_dupl(classes\_of\_parts)$ :
           inertial fluent
 action declarations
  $duplicate$ : action
   attributes
    $class$ : $classes\_of\_parts$
   axioms
    $duplicate$   causes   $num(C_1, C_2, N_2)$
                  if       $class = C_1$,
                           $father(C_2, C_1)$,
                           $num(C_1, C_2, N_1)$,
                           $N_2 = 2 * N_1$.
    impossible $duplicate$ if $class = C$,
                           $prevented\_dupl(C)$.
  end of $duplicate$
  $split$ : $duplicate$
   axioms
    $split$   causes   $num(C_1, C_2, N_2)$
              if       $class = C_2$,
                       $father(C_2, C_1)$,
                       $num(C_1, C_2, N_1)$,
                       $even\_numbers(N_1)$,
                       $N_1 \neq 0$,  $N_1 = N_2 * 2$.
```

end of $split$
$prevent\_duplication$: **action**
attributes
$class : classes\_of\_parts$
axioms
$prevent\_duplication$ **causes** $prevented\_dupl(C)$ **if** $class = C$.
end of $prevent\_duplication$

As mentioned previously, the cell cycle is conventionally described as a sequence of two consecutive phases, which in turn consist of other sub-phases. The following module represents knowledge about sequences of actions.

module $sequence$
sort declarations
$elements,\ sequences,\ numbers$: **sort**
fluent declarations
$component(elements, numbers, sequences)$: **static fluent**
$length(numbers, sequences)$: **static fluent**
axioms
$\neg length(N_1, S)$ **if** $length(N_2, S)$, $N_1 \neq N_2$.
end of $length$

Various system descriptions of $\mathcal{ALM}$ specifying this process at different levels of granularity will contain the $basic\_cell\_cycle$ and $sequence$ modules and will differ from each other only by their structure. This shows that $\mathcal{ALM}$'s feature of separating the interpreted theory from its interpretation is useful in representing domains at different levels of abstraction.

We first consider a model in which cell cycle is viewed as a sequence consisting of the interphase and the mitotic phase. Interphase is considered to be an elementary action, whereas the mitotic phase is a sequence of two elementary actions: mitosis and cytokinesis. We also limit our domain to cells contained in an experimental environment that is usually called *sample*. This refinement of the cell cycle includes the modules *basic_cell_cycle* and *sequence*, and the structure:

structure of $cell\_cycle(1)$
sorts
$sample, cell, nucleus \in classes\_of\_parts$
$cell\_cycle, mitotic\_phase \in sequences$
$interphase, mitotic\_phase, mitosis, cytokinesis \in elements$
actions
instance $interphase$: **action**
instance $mitosis$: $duplicate$
$class := nucleus$
instance $cytokinesis$: $split$
$class := cell$
statics
$father(sample, cell)$.
$father(cell, nucleus)$.
$root(sample)$.
$component(interphase, 1, cell\_cycle)$.
$component(mitotic\_phase, 2, cell\_cycle)$.
$length(2, cell\_cycle)$.
$component(mitosis, 1, mitotic\_phase)$.
$component(cytokinesis, 2, mitotic\_phase)$.
$length(2, mitotic\_phase)$.

We can use this initial model to determine the number of cells and nuclei in the sample at the end of the cell cycle, assuming that our initial sample consists of one cell with one nucleus. The solution can be obtained from the answer set of a program, Π_1, consisting of the ASP translation of the $cell\_cycle(1)$ system description, the theory of intentions, and the domain history. The history, $\mathcal{H}_1$, is written as:

$observed(num(cell, sample, 1), true, 0)$.
$observed(num(nucleus, cell, 1), true, 0)$.
$intend(cell\_cycle, 0)$.

The last statement of $\mathcal{H}_1$ captures our intuition that the purpose of a cell is to divide, expressed as an intention. The answer set of Π_1 will contain the last step 3 and the facts:

$holds(num(cell, sample, 2), 3)$
$holds(num(nucleus, sample, 2), 3)$
$holds(num(nucleus, cell, 1), 3)$

Hence, at the end of the cycle, the sample contains two cells with one nucleus each.

Representing Textbook Questions

We now present our approach in answering several end-of-the-chapter review questions about cell division; the questions are taken from a biology textbook (Campbell and Reece 2001). We start with question 12.9: *"In some organisms mitosis occurs without cytokinesis occurring. This will result in ___"*. To encode the information given in the text of this question, we simply expand the history $\mathcal{H}_1$ by

$\neg happened(cytokinesis, I)$

for every step I. The expression *"will result in"* (which appears in the text) is encoded via the predicate $result$ defined by the rule:

$$result(F) \leftarrow fluent(\mathbf{inertial}, F),\ holds(F, I),\ last\_step(I).$$

The corresponding answer set will now contain:

$result(num(cell, sample, 1))$
$result(num(nucleus, sample, 2))$
$result(num(nucleus, cell, 2))$

which indicates that the result is one cell with two nuclei.

Next, we consider question 12.15: *"A researcher treats cells with a chemical that prevents DNA synthesis. This treatment traps the cells in which part of the cell cycle?"* To answer this question the system will need to know more about the structure of the cell and that of the interphase and mitosis. The second refinement of the cell cycle provides this additional knowledge. The following cell components will be added: *the chromosomes inside the nucleus, the chromatids that are part of the chromosomes, and the DNA inside the chromatids*. The interphase is a sequence $[g_1, s, g_2]$ where g_1 and g_2 are elementary actions and s is a sequence of two elementary actions: *DNA synthesis*, and *the creation of sister chromatids*. Mitosis is a sequence of five actions: *prophase, prometaphase, metaphase, anaphase, and telophase*. The treatment of the cells with the chemical is represented by an exogenous action that prevents the duplication of the DNA.

structure of $cell\_cycle(2)$
sorts
$sample, cell, nucleus, chromosome, chromatid,$
$dna \in classes\_of\_parts$
$cell\_cycle, mitotic\_phase, interphase, s, mitosis$
$\in sequences$
$interphase, mitotic\_phase, mitosis, cytokinesis,$
$g1, s, g2, dna\_synthesis, sister\_chromatids,$
$prophase, prometaphase, metaphase, anaphase,$
$telophase \in elements$
actions
instance $g1$: **action**
instance $dna\_synthesis$: $duplicate$
$class := dna$
instance $sister\_chromatids$: $split$
$class := chromatid$
instance $g2$: **action**
instance $prophase$: **action**
instance $prometaphase$: **action**
instance $metaphase$: **action**
instance $anaphase$: $split$
$class := chromosome$
instance $telophase$: $split$
$class := nucleus$
instance $cytokinesis$: $split$
$class := cell$
instance $treatment$: $prevent\_duplication$
$class := dna$
statics
$father(sample, cell).$
$father(cell, nucleus).$
$father(nucleus, chromosome).$
$father(chromosome, chromatid).$
$father(chromatid, dna).$
$root(sample).$
$component(interphase, 1, cell\_cycle).$
$component(mitotic\_phase, 2, cell\_cycle).$
$length(2, cell\_cycle).$
$component(g1, 1, interphase).$
$component(s, 2, interphase).$
$component(g2, 3, interphase).$
$length(3, interphase).$
$component(dna\_synthesis, 1, s).$
$component(sister\_chromatids, 2, s).$
$length(2, s).$
$component(mitosis, 1, mitotic\_phase).$
$component(cytokinesis, 2, mitotic\_phase).$
$length(2, mitotic\_phase).$
$component(prophase, 1, mitosis).$
$component(prometaphase, 2, mitosis).$
$component(metaphase, 3, mitosis).$
$component(anaphase, 4, mitosis).$
$component(telophase, 5, mitosis).$
$length(5, mitosis).$

We can capture the scenario in question 12.15 via the following history denoted by $\mathcal{H}_2$:
$observed(num(cell, sample, 1), true, 0).$
$observed(num(nucleus, cell, 1), true, 0).$
$observed(num(chromosome, nucleus, 1), true, 0).$[2]
$observed(num(chromatid, chromosome, 1), true, 0).$
$observed(num(dna, chromatid, 1), true, 0).$
$intend(cell\_cycle, 0).$
$happened(treatment, 0).$

The occurrence of action *"treat cells"* from the story is represented via the observation that the exogenous action $treatment$ happened at 0, included in $\mathcal{H}_2$. To answer our question, we define a relation $trapped\_in$: the cell is trapped in phase V if V precedes an intended phase that never started

$$\begin{array}{lcl} trapped\_in(V) & \leftarrow & component(V, K, S), \\ & & component(V_1, K_1, S), \\ & & intended(V_1), \\ & & \text{not } started(V_1), \\ & & K_1 = K + 1. \end{array}$$

We add the rule above to the ASP encoding. The answer set of the resulting program will contain $trapped\_in(g1)$, where $g1$ is the answer to question 12.15.

Now, let us address question 12.4: *"A particular cell has half as much DNA as some of the other cells in a mitotically active tissue. The cell in question is most likely in:*

a. G_1	*c. prophase*	*e. anaphase"*
b. G_2	*d. metaphase.*	

This question can be rephrased as: *"In which of the following sub-phases does a mitotically active cell have half as much DNA as in other sub-phases."* We use the second refinement of cell division, $cell\_cycle(2)$. The domain history, $\mathcal{H}_3$, obtained from the text of the question is identical to $\mathcal{H}_2$, except for the statement $happened(treatment, 0)$.

We add to our encoding the following axioms defining the relation *"half as much DNA"* in the question:

$$\begin{array}{lcl} half(A) & \leftarrow & action(A), \\ & & occurs(A, I), \\ & & holds(num(dna, cell, N), I), \\ & & holds(num(dna, cell, N_1), I_1), \\ & & N_1 = N * 2. \\ half(S) & \leftarrow & sequences(S), \\ & & constant\_num\_dna\_cell\_in(N, S), \\ & & holds(num(dna, cell, N_1), I_1), \\ & & N_1 = N * 2. \end{array}$$

The relation $half(V)$ holds if V is a phase during which the amount of DNA per cell has a constant value and there is *some* state of the cell cycle that has double that amount. We encode the possible answers as:

$$\begin{array}{lcl} answer(a) & \leftarrow & half(g1). \\ answer(b) & \leftarrow & half(g2). \\ answer(c) & \leftarrow & half(prophase). \\ answer(d) & \leftarrow & half(metaphase). \\ answer(e) & \leftarrow & half(anaphase). \end{array}$$

The answer set of the program that is obtained will contain $half(g1)$ and $answer(a)$, as expected.

Finally, we consider question 12.12: *"Starting with a fertilized egg (zygote), a series of five cell divisions will produce an early embryo with how many cells?"* In order to answer this question, our refinement of cell division only needs to specify two elements of the sort $classes\_of\_part$:

[2]The number of chromosomes per nucleus depends on the species. When it is not given, we can assume it is 1.

the sample and the cell. The refinement does not need to mention the sub-phases of the cell cycle; rather, the cell cycle can be viewed as a simple action of the type $duplicate$. Additionally, the refinement will include a sequence called $five\_divisions$, consisting of five cell cycle actions:

```
structure of  $cell\_cycle(3)$
  sorts
    $sample, cell \in classes\_of\_parts$
    $five\_divisions \in sequences$
    $cell\_cycle \in elements$
  actions
    instance $cell\_cycle : duplicate$
      $class := cell$
  statics
    $father(sample, cell).$
    $root(sample).$
    $component(cell\_cycle, 1, five\_divisions).$
    $component(cell\_cycle, 2, five\_divisions).$
    $component(cell\_cycle, 3, five\_divisions).$
    $component(cell\_cycle, 4, five\_divisions).$
    $component(cell\_cycle, 5, five\_divisions).$
    $length(5, five\_divisions).$
```

Our history, $\mathcal{H}_4$, will be:

$observed(num(cell, sample, 1), true, 0).$
$intend(five\_divisions, 0).$

The answer set will contain $ends(five\_divisions, 5)$[3] and $holds(num(cell, sample, 32), 5)$, which say that the five rounds of cell division are completed at time step 5 and that there are 32 cells in the sample at step 5, as expected.

The Systems

We created a small system that is able to answer the textbook questions above, together with other sample questions. We abstracted away from the natural language processing task. We assumed that the information given in the question (the *recorded history*) and the possible answers, when they exist as in question 12.4, are already encoded in ASP.

We had several system descriptions for cell division, representing the domain at different levels of granularity: the three system descriptions presented above ($cell\_cycle(1)$, $cell\_cycle(2)$, and $cell\_cycle(3)$) plus an additional description, $cell\_cycle(4)$, not mentioned here. We wrote a procedural program that automatically translates these system descriptions into ASP programs. Then, we created a script file that would assemble the logic programs needed to answer each question by putting together the encoding of a question, the translation of the corresponding system description and the theory of intentions. For now, question encodings are associated with one of the four system descriptions by hand. In the future, this may be done automatically based on some keywords in the question. For example, if the word "DNA" is used, then the system description $cell\_cycle(2)$ containing the class of parts dna should be selected. Note that selecting the most appropriate system description is not an essential task from the point of view of the inferences that are made, as the most detailed representation of cell division can be used to reason correctly in all cases. However, this selection has an impact on efficiency.[4]

We created a second script file that ran the CLASP[5] (Gebser et al. 2007) answer set solver for the resulting program of every question, and then output the answers. We had a total of ten test questions. They were all answered correctly by our system. The results were returned by the CLASP solver in less than 0.4 seconds on a machine with 1.70 GHz CPU and 1GB RAM running 32-bit Windows. We used the above system as a conceptual validation of our methodology.

Next, we worked on making our methodology compatible with the current Digital Aristotle, which uses the language $\mathcal{F}$LORA-*2* (Kifer 2005), a variant of logic programming based on the well-founded semantics (Van Gelder, Ross, and Schlipf 1991). We created a second system that translated $\mathcal{ALM}$ system descriptions into $\mathcal{F}$LORA-*2*, generated $\mathcal{F}$LORA-*2* logic programs for every question to be answered, and ran the $\mathcal{F}$LORA-*2* inference engine[6] on each of these programs. Our translation from $\mathcal{ALM}$ into $\mathcal{F}$LORA-*2* and the general relation between our approach and the $\mathcal{F}$LORA-*2* axiomatization are discussed in (Inclezan 2010). We showed that the two methods are not equivalent, as the ASP method is sound and complete with respect to the $\mathcal{ALM}$ specification while the $\mathcal{F}$LORA-*2* method is only sound in the general case. However, the $\mathcal{F}$LORA-*2* approach worked correctly on our test questions; there were no significant differences in efficiency between the two methods.

Conclusions and Future Work

In this paper we have described a formalization of a biological process, cell division, in modular action language $\mathcal{ALM}$. We have presented two algorithms that use this formalization to answer textbook review questions: an ASP and a $\mathcal{F}$LORA-*2* algorithm. We have indicated that $\mathcal{ALM}$ allows for comparisons between different formalisms, due to its simple and elegant design tailored for the specification of a distinct type of problems: dynamic domains.

Our interest in the biological domain came from our involvement in Project Halo whose goal is to create a digital tutor for a wide range of disciplines, including biology. Our $\mathcal{F}$LORA-*2* algorithm was used in the Digital Aristotle system to answer questions about cell division. The representation of cell division is not trivial. First of all, it requires addressing the classical problem of commonsense reasoning, i.e., the ability to predict direct and indirect effects of actions. We have demonstrated that writing general axioms about the effects of actions in $\mathcal{ALM}$ allows us to express and reason about such effects in an efficient and, most importantly, elaborance tolerant way. Secondly, the representation of cell division requires reasoning about naturally evolving processes, and the ability to switch smoothly between coarse and fine representations of the domain. We have shown that $\mathcal{ALM}$ is a suitable language for this task. The modularity of

[3]The predicate $ends(v, i)$, saying that the action or sequence of actions v ends at time step i, is part of the theory of intentions.

[4]Such impact was minimal for the domain and sample questions we addressed.

[5]http://potassco.sourceforge.net/

[6]http://flora.sourceforge.net/

the language allows for the reuse of information. The separation between the uninterpreted theory and its interpretation enabled us to represent the domain at different levels of granularity by reusing the same theory and changing only its interpretation.

Action languages and ASP were previously used to describe other biological processes. For example, representing and reasoning about biological signaling networks was discussed in (Baral et al. 2004; Tran and Baral 2007; Tran, Baral, and Shankland 2005). Their approach adds a new notion to action languages, that of *triggered actions*, i.e., actions that normally occur in any state satisfying certain properties. Our formalization of cell division did not require this addition. However, the theory of intentions we used contains a statement similar to a triggering rule, to express the non-procrastination principle.

Our work can be extended in several ways. First of all, a general methodology of how to represent domains at different levels of granularity in $\mathcal{ALM}$ can be established. Secondly, the task of processing questions written in natural language can be investigated. Work done in this direction for biomedical questions appeared in (Erdem and Yeniterzi 2009), where the questions were encoded into ASP. Two special topics within the natural language processing task are: detecting the required level of granularity based on the text of the question, and generating ASP definitions for relations not present in our theory about biology, by extracting information from lexical databases.

Acknowledgments This work was partially funded by Vulcan Inc. We thank Benjamin Grosof and Michael Kifer for discussions on the topic of this paper.

References

Balduccini, M.; Baral, C.; and Lierler, Y. 2008. Knowledge Representation and Question Answering. In van Harmelen, F.; Lifschitz, V.; and Porter, B., eds., *Handbook of Knowledge Representation*. Elsevier. 779–820.

Baral, C., and Gelfond, M. 2000. *Reasoning Agents in Dynamic Domains*. Norwell, MA: Kluwer Academic Publishers. 257–279.

Baral, C., and Gelfond, M. 2005. Reasoning about Intended Actions. In *AAAI-05: Proceedings of the 20th National Conference on Artificial Intelligence*, 689–694. AAAI Press.

Baral, C.; Chancellor, K.; Tran, N.; Tran, N. L.; Joy, A.; and Berens, M. 2004. A Knowledge Based Approach for Representing and Reasoning about Cell Signalling Networks. volume 20, 15–22. Oxford, UK: Oxford University Press.

Baral, C.; Gelfond, G.; Gelfond, M.; and Scherl, R. B. 2005. Textual Inference by Combining Multiple Logic Programming Paradigms. In *Proceedings of the AAAI-05 Workshop on Inference for Textual Question Answering*, 1–5. Pittsburgh, PA: AAAI Press.

Baral, C.; Gelfond, M.; and Scherl, R. 2004. Using Answer Set Programming to Answer Complex Queries. In *Workshop on Pragmatics of Question Answering at HLT-NAAC2004*.

Campbell, N. A., and Reece, J. B. 2001. *Biology*. Benjamin Cummings, 6th edition.

Erdem, E., and Yeniterzi, R. 2009. Transforming Controlled Natural Language Biomedical Queries into Answer Set Programs. In *BioNLP-09: Proceedings of the Workshop on BioNLP*, 117–124. Morristown, NJ: Association for Computational Linguistics.

Gebser, M.; Kaufmann, B.; Neumann, A.; and Schaub, T. 2007. Conflict-Driven Answer Set Solving. In *IJCAI-07: Proceedings of the 20th International Joint Conference on Artifical Intelligence*, 386–392. San Francisco, CA: Morgan Kaufmann Publishers Inc.

Gelfond, M., and Inclezan, D. 2009. Yet Another Modular Action Language. In *Proceedings of SEA-09*, 64–78. University of Bath Opus: Online Publications Store.

Gelfond, M., and Lifschitz, V. 1991. Classical Negation in Logic Programs and Disjunctive Databases. *New Generation Computing* 9(3/4):365–386.

Gelfond, M. 2006. Going Places – Notes on a Modular Development of Knowledge about Travel. AAAI 2006 Spring Symposium Series, 56–66. AAAI Press.

Gunning, D.; Chaudhri, V. K.; Clark, P.; Barker, K.; Chaw, S.-Y.; Greaves, M.; Grosof, B.; Leung, A.; McDonald, D.; Mishra, S.; Pacheco, J.; Porter, B.; Spaulding, A.; Tecuci, D.; and Tien, J. 2010. Project Halo–Progress Toward Digital Aristotle. *AI Magazine* 31(3):33–58.

Inclezan, D. 2010. Computing Trajectories of Dynamic Systems Using ASP and Flora-2. Paper presented at NonMon@30: Thirty Years of Nonmonotonic Reasoning Conference, Lexington, Kentucky, 22-25 October.

Kifer, M. 2005. Nonmonotonic Reasoning in FLORA-2. In Baral, C.; Greco, G.; Leone, N.; and Terracina, G., eds., *Logic Programming and Nonmonotonic Reasoning*, volume 3662 of *Lecture Notes in Computer Science*. Springer Berlin / Heidelberg. 1–12.

Tran, N., and Baral, C. 2007. Reasoning about Non-immediate Triggers in Biological Networks. *Annals of Mathematics and Artificial Intelligence* 51(2-4):267–293.

Tran, N.; Baral, C.; and Shankland, C. 2005. Issues in Reasoning about Interaction Networks in Cells: Necessity of Event Ordering Knowledge. In *AAAI-05: Proceedings of the 20th national conference on Artificial Intelligence*, 676–681. AAAI Press.

Turner, H. 1997. Representing Actions in Logic Programs and Default Theories: A Situation Calculus Approach. *Journal of Logic Programming* 31(1-3):245–298.

Van Gelder, A.; Ross, K. A.; and Schlipf, J. S. 1991. The Well-Founded Semantics for General Logic Programs. *Journal of the ACM* 38:619–649.