An Executable Formal Model for Specifying and Verifying Clinical Trials

Vivek Nigam[1], Carolyn Talcott[2], Tajana Ban Kirigin[3], Max Kanovich[4], Andre Scedrov[5]

Abstract

Before a novel medical treatment, such as a new drug, can be made available to the general public, their effectiveness has to be experimentally evaluated. Experiments that involve human subjects are called Clinical Trials (CTs). Since human subjects are involved, procedures for CTs are elaborated so that data required for validating the treatment can be collected while ensuring the safety of subjects. Moreover, CTs are heavily regulated by public agencies, such as the Food and Drug Administration (FDA). Violations of regulations or deviations from procedures should be avoided as they may incur heavy penalties and more importantly may compromise the health of subjects. However, CTs are prone to human error, since CTs are carried out by the study team, which might be overloaded with other tasks, such as hospital and/or pharmacy duties, other trials, etc. In order to avoid discrepancies, we propose developing an automated assistant for helping all the parties to correctly carry out CTs as well as to detect and prevent discrepancies as early as possible. This way the proposed automated assistant would minimize error, and therefore increase the safety of the involved subjects. This paper takes the first steps towards that direction. We model a clinical trial by using a partial order among the expected events of an trial and their expected time intervals. This model specifies all possible compliant plans. We then demonstrate how we one can check whether an actual clinical trial execution complies with specification detecting possible deviations. We implemented a prototype in Maude.

1. Introduction

There is little doubt that medical treatments have improved in several ways the quality of life of the population. One can now rely on a huge variety of treatments for different symptoms, from simple pain killers to drugs for diabetes or HIV patients. A

Email addresses: `vivek.nigam@gmail.com` (Vivek Nigam), `clt@csl.sri.comu` (Carolyn Talcott), `bank@math.uniri.hr` (Tajana Ban Kirigin), `mik@eecs.qmul.ac.uk` (Max Kanovich), `scedrov@math.upenn.edu` (Andre Scedrov)

[1]Federal University of Paraíba, Brazil
[2]SRI International, USA
[3]University of Rijeka, HR
[4]Queen Mary, University of London, UK
[5]University of Pennsylvania, USA

great deal of this success is due to the careful understanding of the effectiveness of a treatment through experimentation. Since treatments may affect people's health, before a treatment is made available to the general public, one is required to perform experiments in order to determine its effectiveness. At the final stages of testing, one is often required to test the treatment on human subjects. These procedures are called *Clinical Trials* (CTs) [5]. Only when CTs are successfully carried out and the effectiveness of the treatment is experimentally validated may the treatment be approved by public agencies, *e.g.*, the Food and Drug Administration (FDA), to be made available to the general public.

There are two key concerns while carrying out CTs. The first concern is of assuring that the subjects' health is not compromised by the test. In order to protect the subjects' health, CTs are rigorously regulated and audited by (FDA) inspectors. Violations of FDA regulations may imply heavy penalties, both financial as well as of bad public relations.[6] The second key concern is to collect enough data to determine the effectiveness of the treatment. Without such data, it is most likely that the treatment will not be allowed by public agencies (such as the FDA) to be commercialized.

Normally, a pharmaceutical company (Sponsor) that wants its treatment to be tested hires a Clinical Research Organization (CRO) which is specializing in carrying out CTs. Then a detailed plan, called a *protocol*, is elaborated by specialists explaining how CTs should be carried out in order to obtain the most conclusive results without compromising the health of the subjects involved. For instance, the protocol normally contains the number of subjects that need to be used in the CT, or the type of CT, *e.g.*, blind, double-blinded, comparison with placebo, etc, as well as the duration of the CT.[7] In order to carry out a CT, the Sponsor/CRO collaborate with health institutions, typically hospitals, which have the necessary means, *i.e.*, competent people and the equipment, to perform the clinical research and collect the required data. At each site, a Principal Investigator is assigned and is in charge of the clinical trial carried out at the site.

During a CT it is the responsibility of the Sponsor/CRO and the PI from the health institution to (1) make sure that there are no violations of the regulations imposed by public agencies, such as the FDA, which could lead to penalties, and (2) that there are no deviations from the protocol, which could lead to data that is not conclusive for determining the effectiveness of the treatment. Deviations could also compromise the safety of the subjects, which should be avoided at all costs. Furthermore, health institutions that have a record of deviations in previous CTs may be penalized by the market by not being hired for carrying out future CTs.

Although all participants of a CT, *e.g.*, physicians, study coordinators, pharmacists, and nurses, are informed of and even trained in the protocol and in the regulations, errors do occur. At each site it is common to have more than one ongoing trial, typically 5-10 trials, with different procedures, and they take place along with normal hospital duties. Furthermore, as all subjects are not enrolled in a study at the same time, it is difficult to have a clear overview of the process. Therefore, the process is prone to

[6]FDA maintains a list available online of the name of companies that have in the past violated regulations.
[7]To illustrate the complexity of some CTs, typical protocols are more than 100 pages long.

human error. It is easy to lose track, for example, that some subject has to be called for follow up or for an unscheduled visit. In order to avoid such errors, a monitor from the Sponsor/CRO that has in-depth knowledge of both the regulations and of the protocol monitors the on-going trials on site and *manually* checks the available records in the health institutions to identify situations that could lead to regulation violations or to protocol deviations. A much better approach would be to have an automated assistant that helps to correctly carry out CTs.

Although in the past years, there has been improvement with the introduction of Electronic Case Report Forms, there is plenty of room for more. Existing tools perform simple checks, such as checking whether the format of the data entered is correct or whether it falls into some pre-defined range. *There is no formal model that specifies a CT as a process.* Thus, the trial manager still has to check, for example, whether the order of activities performed comply with the trial protocol, or whether a subject is experiencing some unexpected trend, e.g., his blood pressure increased more than is safe during trial execution, or whether Adverse Events have been correctly reported. In summary, due to the complexity of protocols, size of trials and lack of automation, many errors occur which may compromise the health of subjects and the quality of the collected data.

It has been shown that computer tools can be used to support carrying out administrative tasks. For instance, the tool Maude [2, 7], which we use in this work, has been successfully used for planning and workflow formalisms. Other formalisms include the PROforma clinical knowledge representation and workflow language and successors used for modeling health care processes, clinical guidelines, error handling and safe delegation [6]. However, to our knowledge there is no automated assistant on location that helps the medical team or the monitors in carrying out CTs.

For CTs, we identify two possible direct uses for such a tool. The first application is called *plan generation*. Plan generation preemptively avoids violations and deviations from occurring. For instance, once a nurse inputs a new test result in the system, she can be informed of the sequence of actions that need to be performed next, *e.g.*, carry out additional tests. The second application is called *execution monitoring*, that is, to check in timely manner whether there have been violations and/or deviations in the past. When such violations or deviations are detected, the Sponsor/CRO/PI could react by, for instance, informing the corresponding authority, *e.g.*, FDA, and excluding incompetent sites or personnel from the CT or by excluding subjects from the trial.

The main contribution of this paper is threefold. Our first contribution is to set out some minimal requirements of a formal model, detailed in Section 2, obtained by studying protocols and interacting with the community. Our second contribution is to formalize an executable model for compliant CT executions, detailed in Sections 4 and 5. This model can be used to check for deviations and for planning, detailed in Section 6. Our third contribution is the implementation of such a Clinical Trial Assistant (CTA) detailed in 7 and tested it by using case studies provided by the Clinical Data Interchange Standards Consortium (CDISC). A prototype implementation based on the material described here can be obtained from the link:

`http://www.nigam.info/implementations/CTA-v1.0b.zip`

2. Model Requirements

This section justifies some of the design choices for the model we introduce in the following sections. In particular, we want a model that is general enough to specify all compliant executions of a CT without losing the ability to infer deviations. We first detail some model requirements that we learned after studying FDA regulations and CT protocols. Then we detail the types of properties that we are aiming to verify with our tool. We came up with these properties by looking through FDA regulations and by interacting with experts of the field, such as discussing with the CDISC community and talking with CT data managers, such as Principal Investigators (PIs) and Clinical Research Organizations (CROs).

While studying the FDA regulations [5] and existing clinical trial specifications, we identified three requirements needed in order to built a reasonable automated tool for Clinical Trials.

Requirement 1: *Our model should allow to relate dependencies among activities.*

Regulations and protocols often require that to apply some action one must have satisfied some conditions in the past. A typical example of such requirement is that a subject can only participate in a CT if he has signed an informed consent. This is specified in Part 50 – Protection of Human Subjects, Subpart B, Section 50.20, quoted below:

> “ Except as provided in 50.23 and 50.24, no investigator may involve a human being as a subject in research covered by these regulations *unless the investigator has obtained the legally effective informed consent* of the subject or the subject’s legally authorized representative. [· · ·] ”

In particular, a test can only be performed or a treatment administered on a subject if he has signed an informed consent.

Requirement 2: *Our model should allow to mention time explicitly.*

Consider the following clause appearing in the Federal Regulations CFR21, Part 312 [5] for *Investigational New Drug Applications* (INDs):

> “ (c) IND safety reports
>
> (1) Written reports –(i) The sponsor shall notify FDA and all participating investigators in a written IND safety report of:
>
> (A) Any adverse experience associated with the use of the drug that is both serious and unexpected; [· · ·] Each notification shall be made as soon as possible and *in no event later than 15 calendar days* after the sponsor’s initial receipt of the information [· · ·]
>
> (2) Telephone and facsimile transmission safety reports. The sponsor shall also notify FDA by telephone or by facsimile transmission of any unexpected fatal or life-threatening experience associated with the use of the drug as soon as possible but *in no event later than 7 calendar days* after the sponsor’s initial receipt of the information.”

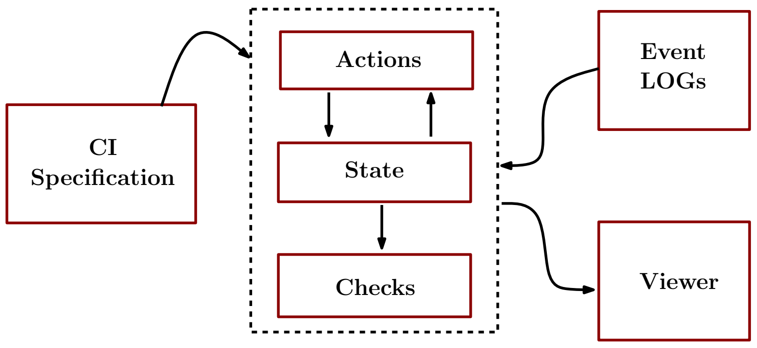


Figure 1: Blocks and modules of our Clinical Trial Assisstant (CTA). The arrows denote the flow of information between the different pieces of CTA.

The clause above mentions explicitly two different time intervals. The first is that one must send a detailed safety report to the FDA within 15 days, while the second obligation is that one must notify FDA of such an event within 7 days. These time intervals are used to specify *future obligations*, such as the obligation of sending a safety report.

Protocols mention explicit time intervals as well. For example, a protocol might specify that if some parameter of an urine test is three times above the upper limit, then the same test is repeated with a fresh sample *within 5 days* in order to make sure that the first result is not an isolated result.

Requirement 3: *Our model should be an executable formal model.*

Finally, as detailed in the Introduction, we are interested in a tool that can be used for both monitoring trial events and generating plans for CTs. That is, it should be possible to generate a plan from our model. The best way to achieve this is by ensuring that our model is executable. This also means that we simulate CT executions to not only generate plans but also check for design inconsistencies by using model checking techniques.

3. An Overview of Clinical Investigation Assistant

Figure 1 depicts how our Clinical Investigation Assistant (CTA) works. As we detail in Section 4, this model specifies all the possible compliant executions of the given CT. For example, it contains the relation of activities (Requirement 1) and the timing constraints (Requirement 2) of the CT. The block to the left in Figure 1 represents this model.

The Event Logs describes what actually happened with a subject during the trial. For instance, whenever the subject starts a visit, this information is recorded in the Event Logs, similarly the lab results of some particular activity and the the adverse events that have been reported are recorded. All these events are assumed to have a timestamp associated, which corresponds to the time when the corresponding event happened.

The Viewer is simply an interface where Clinical Administrators and any authorized personnel can check the status of the CT, such as the activities that have been performed and the deviations that have been detected by CTA.

As denoted by the arrows in Figure 1, the CT model and the Event Logs are used by the blocks in the dashed square, namely, by the Actions, State and Checks functions. We enter into the details of how States and Actions are specified in Section 5. Intuitively, the State block contains a description of the current state of a subject in a trial, such as, the activities that he has performed and the adverse events that happened to him as well as the events that are pending. Thus, it can be seen as an abstraction of the information described in the Log Events.

The Action block on the other hand contains the actions (also called rewrite rules) that modify the current state of the system. Given a new event log entry the corresponding action that handles that event is triggered changing the state of the system. For example, if the log event states that the subject has started a visit, then the corresponding rule is fired updating the state of the system to reflect this new event.

Finally, the Checks function takes the specification of the CT, which specifies the set of compliant plans, and the state of the system, which reflects what actually happened, and checks whether the actual execution of the plan complies with the specified one. If deviations are found, then the warning messages are sent to the Viewer.

4. CT Specification: Activities, Activity Pre-Order and Time Intervals

Alphabet. As per Requirement 1, a CT is modeled as a strict pre-order $\prec$ on the set of activities that are expected to be performed. The relation $e_1 \prec e_2$ denotes that the activity e_1 should be performed before the activity e_2. For instance, the informed consent should be signed by a subject before any procedure is carried out on him. Recall that a pre-order is a relation that is transitive and reflexive, but not symmetric. Therefore, it cannot happen that $e_1 \prec e_2$ and at the same time $e_2 \prec e_1$, *i.e.*, there are no dependency cycles on activities.

Types of Activities. During our studies of CT specifications, we identified two different types of activites:

- **In-Visit Activities:** These are activities that should be performed during the visit of a subject. For instance, checking the subjects Vital Signs should be done when the subject visits a health institution;

- **Long-Term Activities:** These are activities that occur over longer time and cannot be completed during a visit. For instance, in some CTs one needs to periodically measure the ECG of a subject for some weeks or take their daily medicines. In these cases, respectively, the subject carries with him during these weeks an equipment that records the ECG information of a subject and takes his lot of medicine pills.

In our implementation, both In-Visit and Long-Term activities are specified in Maude as a pair of strings mapped to the sort `POEle` for Pre-Order Element as follows:

```
sort POEle .
op (_,_) : String String -> POEle .
```

We use the convention that for In-Visit activities the first element of the pair is the name of the task that should be performed while the second is the name of visit when this task should be performed. For example, the pair `("Informed-Consent", "Screening")` denotes the activity of signing the Informed Consent that should be performed during the screening visit. For Long-Term activities, on the other hand, we use the convention that the first and the second pair contain the same string, namely, the name of the long-term activity. For example, `("ECG","ECG")` denotes the Long-Term activity of monitoring the ECG of a subject. Futhermore, we assume that each visit has a start of visit activity, written `"SV"`, and an end of visit activity, written `"EV"`. For example, the following are pre-order elements denoting the start and end of the visit `"Screening"`: `("SV", "Screening")` and `("EV", "Screening")`

In order to construct the set of pre-order elements, one has to specify the name of visits, the tasks that have to be performed during a visit, and the names of long-term activities. These are specified, respectively, by the following operators:

```
op visits : -> ListString .
op tasks : String -> ListString .
op ltAct : -> ListString .
```

where `ListString` is the sort of list of strings.

Pre-Order of Activities and Time Intervals. The pre-order of activities is specified in our model over activities by using the following operator:

```
op _<ev_ : POEle POEle -> Bool .
```

For example, the following equation specifies that before the vital signs of a subject is performed during the Screening visit, he must have signed the Informed Consent:

```
eq ("Informed Consent", "Screening") <ev
                    ("Vital Signs", "Screening") = true .
```

As we will see below, it is possible to specify more general conditions for when $act1$ `<ev` $act2$ where $act1$ and $act2$ are activities.

For satisfying Requirement 2 described in Section 2, we need another operator describing the explicit time intervals between activities. We use the following operators:

```
sorts Interval SetIntervals .
subsort Interval < SetIntervals .
op (_,_,_) : POEle POEle Nat -> Interval .
```

The first two lines specify two sorts, one for one Interval, which is a subsort of the second, for Intervals. An Interval is then constructed by the operator defined in the third line. An Interval of the form (act_1, act_2, n) specifies that the activity act_2 should be performed exactly n time units (usually days) after activity act_2. Thus the following specifies that visit `Visit 1` should be performed 10 days after the end of the visit `Screening`

```
(("EV", "Screening"), ("SV", "Visit 1"), 10)
```

These intervals vary from trial to trial and have to be provided using the following operator which returns a set of intervals:

```
op intervals : -> SetIntervals .
```

It is also possible to incorporate more refined specification, such as tolerance days to the interval specifications. We refrain to do so here for simplicity.

Finally, to connect the pre-order with the intervals we use a collection of definitions similar to the the following one:

```
ceq ele1 <ev ele1 = true if after(ele1, ele2) .
```

where the operator `after(ele1, ele2)` checks in the list of intervals whether activity `ele2` should be performed after activity `ele1`. The equation above then maps `ele1 <ev ele1` to `true` if this the case.

5. State and Actions

Given the details of the trial have been specified in the form of visit names, the partial order of events, the timing constraints, etc, we construct an executable model in the form of actions or rewrite rules and states.

5.1. State

The state of the system is constructed using the following sort and operators in Maude:

```
sort State .
op none : -> State [ctor] .
op __ : State State -> State [ctor assoc comm id: none] .
```

The first operator (`none`) denotes an empty state, while the second operator (__) specifies that the justaposition of two states yields an state. The keywords `ctor assoc comm id: none` specify respectively that an operator is a constructor of a sort, that the operator is associative and commutative, and the identiy of this sort is the operator `none`. Given such a specification, Maude is able to use its equational theory to identify for example the following states:

`A B C none` and `B A C`

where `A B C` are some arbitrary elements of a state.

A state should be interpreted as the current state of a subject in the trial; for instance, it mentions among other things all the activities that have been performed and those that have to be performed. For this purpose we use the following basic State operators:

```
sort StateEle .
subsort StateEle < State .
op st : POEle Date? -> StateEle .
op cr : String Date? -> StateEle .
op time : Date? -> StateEle .
```

As one can notice, these operators rely on the sorts `Date` and `Date?`. These are sorts represent timestamps which include year, month, day, hour and minute specified by the operator `date` shown below. Furthermore, the latter sort is a supersort of the former by including the non-defined date `noDate` below:

```
op date : Nat Nat Nat Nat Nat -> Date .
op noDate : -> Date? [ctor] .
```

The operator `st` takes an activity, specified as a `POEle` as described in Section 4, and an element of sort `Date?`, and it denotes that the given activity has been performed at some time or not yet performed. For example, the following state

```
st(("Informed Consent", "Screening"),
                    date(2013, 12, 20, 16, 20))
```

specifies that the subject being monitored has signed the Informed Consent at 16:20 of 20/12/2013.

The second State operator, `cr`, is used to keep track of which visit the subject is currently in, if he already started the visit, or the following scheduled visit if he is not in a visit. For instance,

```
cr("Visit-1", noDate)
```

specifies that the subject's next visit is `Visit-1` and that he has not yet started it (`noDate`).

The third state operator, `time`, is used to specify the global time of a state. We assume that there is only one such occurrence of `time` in a state and as we will see when we specify the rewrite rules, this will be preserved during execution. For instance, the following

```
time(date(2014, 2, 28, 10, 25)
```

specifies that the current time is 10:25 of 28/02/2014. Time is updated by using actions and tick events as we will see next.

Initial State. The initial state of for a subject is constructed using the list of visits, specified by the operator `visits`, the activities expected to be carried out in each visit, specified by the operator `tasks`, and the long term activities, specified by the operator `ltAct`. Each State Operator is timestamped with `noDate`, denoting that the corresponding activity has not yet been performed. Thus, the initial configuration has the following shape:

```
time(noDate) st(("SV", "Screening 1"), noDate)
             st(("Informed Consent", "Screening 1"), noDate)
             ...
             cr("Screening1" noDate)
```

It specifies that no activities have yet been performed. The global time has not been initialized and that the expected next visit is Screening 1.

5.2. Actions

Formally, an action in its most general form is as follows:

$$F_1 \cdots F_n \texttt{ => } G_1 \cdots G_m \texttt{ if } C$$

where $F_1, \ldots, F_n, G_1, \ldots, G_m$ are state elements and C is a term of type `Bool`. Such a rule is applicable to a state $\mathcal{S}$ if there is grounding substitution σ, such that $F_1\sigma \cdots F_n\sigma \in \mathcal{S}$ and the conditions $C\sigma$ are satisfied. The resulting state of applying this action with σ is $\mathcal{S} \cup \{G_1\sigma, \ldots, G_m\sigma\} \setminus \{F_1\sigma, \ldots, F_m\sigma\}$. Finally, we omit the condition C if it is always true, that is, there are no side conditions.

As described above, states describe a fixed point in time. Actions on the other hand specify how and when a state can change to another state. In Maude, actions are specified using rewrite rules. For example, the following action initializes the global time of the initial state shown above:

```
rl[TIME.INIT]:
 (initTimeEv(dateTime)) (time(noDate)) =>  (time(dateTime)) .
```

Here `dateTime` is a date variable with the actual time, while `noDate` is the constant specifying that the global time has not yet been initialized.

Thus when the rule above is applied to a state, the global time is initialized with the date matched with `dateTime`. The fact `(initTimeEv(dateTime))` is used intuitively as a trigger for this action. The concrete value for `dateTime` is obtained by the log of events that actually happened in the trial. In particular, the first event in the log of events is a time initialization event that sets the global time of the state. Our tool processes this event and includes this fact to the state of the system with a concrete date. (This operation corresponds to the arrow in Figure 1 from Event Logs to the dashed square.) Once this fact is added to the state, and assuming that time has not yet been initialized, *i.e.*, its argument is `noDate`, the rewrite rule above is applicable and once applied removes the fact `(initTimeEv(dateTime))`σ from the state and initializes the global time by updating the parameter of `time` with the concrete date `dateTime`σ.

There are other such operators that are used to trigger actions and rewrite states, namely, the following:

```
op initTimeEv : Date -> StateEle .
op nextEv : String Nat Date -> StateEle .
op nextEvAE : String Nat Date String -> StateEle .
```

Besides the operator used to initialize time, we also have an operator, `nextEv`, that triggers normal events, such as when an activitiy is performed, and another operator, `nextEvAE`, for Adverse Events. This distinction is needed because Adverse Events are treated differently from the normal events, as the former are not scheduled and they lead to future obligations as detailed in Section 3, namely, to inform the FDA.

Figure 2 contains some actions for handling In-Visit activities. The first and third actions handle, respectively, the start and the end of a visit. These are triggered when the parameter of `nextEv` is, respectively, the string `"SV"` and `"EV"`. Notice that in

```
 rl[VISIT.START]:
  (time(dateTime)) (nextEv(strCur, "SV", n:Nat, dateTime))
  (cr(strCur, noDate)) (st(("SV", strCur), noDate))
=>
  (time(dateTime)) (cr(strCur, dateTime)) (st(("SV", strCur), dateTime)) .

crl[ACT.InVISIT]:
  (time(dateTime)) (nextEv(strCur, strAct, n1, dateTime))
  (cr(strCur, vsDate)) (st((strAct, strCur), noDate))
=>
  (time(dateTime)) (cr(strCur, vsDate)) (st((strAct, strCur),dateTime))
if mem(strAct, tasks(strCur)) .

crl[VISIT.END]:
  (time(dateTime)) (nextEv(strCur, "EV", n1, dateTime))
  (cr(strCur, vsDate)) (st(("EV", strCur), noDate))
=>
  (time(dateTime)) (st(("EV", strCur), vsDate)) (cr(strNx,noDate))
if strNx := next(strCur, visits) .
```

Figure 2: Sample of Actions for Planned In-Visit Activities. Here `next` takes a string str and a list of strings $lstrs$ and returns the string in $lstr$ that follows the string str if it exists, otherwise it returns an error. Moreover, `mem` takes a string str and a list of strings $lstr$ and returns true str belongs to $lstr$ and false otherwise.

```
 crl[LONGTERM]:
  (time(dateTime)) (nextEv(strVisit, strAct, n1, dateTime))
  (st((strAct, strAct) ,noDate))
  =>
  (time(dateTime)) (st((strAct, strAct),dateTime))
if mem(strAct, ltAct) .
```

Figure 3: Sample of Actions for Long Term Activities. Here `mem` takes a string str and a list of strings $lstr$ and returns true str belongs to $lstr$ and false otherwise.

the third rule one also updates the state element `cr` with the following visit. This is done by by the function `next`.

The second action in Figure 2 labelled `Act.InVISIT`, specifies the action of performing an activity in a visit. The activity will be matched with the variable `strAct` which should be a member of the task of the current visit, specified by the action's conditional statement. Once this action is applied a date is assigned to the corresponding activity.

Similar actions also exist for long term activities. Figure 3 depicts one such action. It basically updates the date of the corresponding activity as done in Action labelled `Act.InVISIT` in Figure 2 with the difference that long term actions do not need to happen during a visit.

More interesting are the actions for handling AE events depicted in Figure 4. The

```
 rl[AE.NOTRESOLVED]:
  (time(dateTime)) (nextEvAE("ACT.AE", tag, dateTime, "NOTRESOLVED"))
=>
  (time(dateTime)) (stAE("AE-NOTRESOLVED", dateTime, tag))
  (stAE("FDA-PENDING", noDate, tag)).

crl[AE.FDA]:
  (time(dateTime)) (nextEvAE("ACT.FDA", tag, dateTime, "INFORMED"))
  (stAE("AE-NOTRESOLVED", dateTime2:Date, tag))
  (stAE("FDA-PENDING", noDate, tag))
=>
  (time(dateTime)) (stAE("FDA-INFORMED", dateTime, tag)) .
if not (grDate(dateTime, incDate(7, dateTime2:Date))) .

crl[AE.WARNING]:
  (time(dateTime)) (stAE("AE-NOTRESOLVED", dateTime2:Date, tag))
  (stAE("FDA-PENDING", noDate, tag))
=>
  (time(dateTime)) (stAE("AE-NOTRESOLVED", dateTime2:Date, tag))
  (stAE("FDA-LATE", noDate, tag))
if grDate(dateTime, incDate(7, dateTime2:Date)) .

rl[AE.FDA-LATE]:
  (time(dateTime)) (nextEvAE("ACT.FDA", tag, dateTime, "INFORMED"))
  (stAE("AE-NOTRESOLVED", dateTime2:Date, tag))
  (stAE("FDA-LATE", noDate, tag))
=>
  (time(dateTime)) (stAE("FDA-INFORMED", dateTime, tag))
  (stAE("FDA-LATE", noDate, tag)) .
```

Figure 4: Sample of Actions for AE Events. Here the function `incDate` takes two parameters, a natural number n and a date d and returns the date n days after d, while `grDate` takes two dates d_1 and d_2 and returns true if d_1 is after d_2 and false otherwise.

first action `AE.NOTRESOLVED` is triggered by Adverse Events. This is specified by the term `nextEvAE`, which not only has a date but also a unique identifier `tag` for this event. This identifier is used to uniquely identify an Adverse Event. Once this action is performed, two state elements with head `stAE` are added to the state, namely, one with the date of the event and the string `"AE-NOTRESOLVED"`, and the other one with an undefined date, `noDate`, and the string `"FDA-PENDING"`. The former specifies that an AE event happened, while the latter specifies the future obligation of informing the FDA as described in Section 2. The actions `AE.FDA` and `AE.WARNING` specify respectively when the FDA has been informed within 7 days, while the latter is an action issuing a warning whenever FDA is not informed within 7 days. If FDA is informed in time, then a state element with string `"FDA-INFORMED"` is added to the state, while a state element with string `"FDA-LATE"` denotes a deviation. Finally, the last action `AE.FDA-LATE` specifies the case when FDA is informed even though the time to inform FDA has elapsed. Notice that the deviation expressed by the state element with string `"FDA-LATE"` is not removed by the action `AE.FDA-LATE`, as the deviation occurred.

Finally, our tool also has actions that handle unscheduled visits and tasks. We do not show these actions here as they are very similar to the actions used to handled AE events.

6. Verifying Properties

We now describe how to verify properties, such as check whether a deviation has occurred. Here we consider the following four types of deviations, which are all intra-subject properties:

- *Missing Activity Deviations:* If an activity that was expected to be performed during a visit is not performed, then it characterizes a Task Deviation, *e.g.*, if a Vital Signs check was expected during the Screening visit, but it was not performed.
- *Activity Ordering Deviations:* If an activity A was expected to happen before an activity B, but this was not the case in practice, *i.e.*, in the Log of Events, we classify this situation as an Activity Ordering Deviation, *e.g.*, whenever the signing the informed consent is performed after a medical procedure.
- *Time Interval Deviation* If an activity A was expected to occur exactly n time units after activity B, but this was not the case, then we classify this situation as a Time Interval Deviation, *e.g.*, if the Visit 1 should have happened 2 weeks after the Screening visit, but it happened 3 weeks after the Screening visit.
- *AE Deviations:* If an AE event is not reported in timely fashion to the FDA, *i.e.*, at least 7 days after the event occurred as described in Section 2, then this characterizes an AE Deviation.

From the machinery introduced in the previous sections, we can easily check whether the state of a clinical trial has any one of the deviations described above. We use the following operator to check the three deviations given a state:

```
op check : State -> [CheckSetPairs] .
eq check(St) = checkTasks(St) checkIntervals(St) checkAE(St) .
```

where `[CheckSetPairs]` is a sort for a set of warning messages, which include a description of the warning and pointers to which POEle this warning is related to. Intuitively, these warning messages are used by the Viewer as denoted by the arrow in Figure 1 from the dashed square to the Viewer.

Finally, we use the three auxiliary operators `checkTasks checkIntervals checkAE` to check for, respectively, the first, the second and third, and the fourth types of deviations. In the following paragraph we describe informally how these operators are implemented. As we describe, the design choices used to describe states contains all the information needed to check for the three types of deviations.

Missing Activity Deviations. Given the specification of a CT in the form of definitions for `visits` and `tasks`, we can check whether for a given state $\mathcal{S}$, which activities, if any, that were expected to be performed up to the current visit have not been performed.

A state has all the information needed to determine Missing Activity Deviations. In particular, we check which is the current visit, cur, by looking at the parameter the state element `cr`. By looking at the definition of `visits` and cur, we determine the visits, $vis_1, \ldots, vis_n$ that should precede cur. For each vis_i, we use the definition of `task` to determine the activities that are expected for each visit, and construct the POEles:

$$(act_1^1, vis_1), \ldots, (act_{m_1}^1, vis_1), \ldots, (act_1^n, vis_n), \ldots, (act_{m_n}^n, vis_n)$$

Now we simply check whether the state elements `st` with each one of the these POEles has a timestamp, denoting that these activities have been performed. This can be easily computed.

Activity Ordering and Time Intervals Deviations. To check that the activities not only have been performed but have been performed in the expected order and under specified timing intervals, we proceed in a similar way for checking Missing Activity Deviations. The main difference is that we use the pre-order `<ev` and the specification of `intervals` described in Section 3.

Given a state $\mathcal{S}$, we check that the relation of two elements ele_1 `<ev` ele_2 is satisfied, that is, the timestamps dt_1 and dt_2 of the corresponding `st` is such that dt_1 is before dt_2. Moreover, if these elements appear in an interval relation, then we additionally check whether the interval is satisfied by dt_1 and dt_2. It it is not, then a warning is created. For example, given the following interval

```
(("EV", "Screening"), ("SV", "Visit 1"), 10)
```

and assuming that the facts below are in the state $\mathcal{S}$:

```
 st(("EV", "Screening"), date(2013, 12, 20, 16, 20))
 st(("SV", "Visit 1"), date(2013, 12, 25, 16, 20))
```

then we would generate a warning as the start of Visit 1 was 5 days after the end of the Screening Visit.

AE Deviations:. We generate two warnings. The first one is to inform that FDA should be informed, while the second warning informs the deviation that FDA was not informed in timely manner. In particular, the former warning is issued when a fact of the form below is in the given state

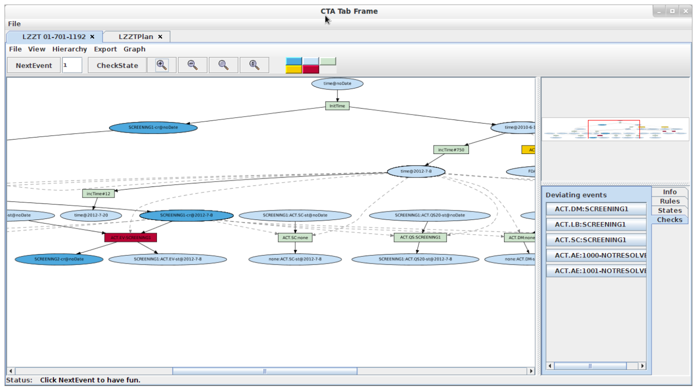
```
 (stAE("FDA-PENDING", noDate, tag))
```

denoting that the obligation that FDA should be informed. On the other hand, we generate a warning of the latter type when a fact of form below is in the given state:

```
(stAE("FDA-LATE", noDate, tag))
```

denoting that one did not comply with the requirement of informing FDA of an AE event.

t



Deviating events

ACT.DM:SCREENING1
ACT.LB:SCREENING1
ACT.SC:SCREENING1
ACT.AE:1000-NOTRESOLVE
ACT.AE:1001-NOTRESOLVE

Status: Click NextEvent to have fun.

Figure 5: Warnings and deviations.

7. Tool Description and Experimentation

The CTA model structure was based on concepts defined in the Clinical Data Interchange Standards Consortium (CDISC)[8]. In particular, we used three different CDISC standards when implementing and testing our tool. The first standard Study Data Tabulation Model (SDTM) defines the format of the LOG of events, such as the names used for activities as well as the measurement units used in for example the lab results. The second standard was the Protocol Representation Model (PRM), which specifies the study objectives, methodology and the statistical analysis that are to be carried out as well as the organization of the study. Finally, we based our design of the trial on the Study Design Model (SDM) also developed by CDISC.

All three standards, SDTM, SDM and PRM, provide standards ways of defining the terminology that should will be used in the specification of a trial, specified by SDM, on the data that is to be collected, specified by SDTM, and the analyses that are to be made, specified by PRM. (More information on these standards can be obtained from the CDISC homepage.) Our model complements these standards by viewing a clinical trial as a process, allowing the construction of an assistant.

In order to test the ideas, we used an example provided in the CDISC SDM-XML specification package [1]. The example included an SDM specification of a trial titled *Xanomeline (LY246708) Safety and Efficacy of the Xanomeline Transdermal Therapeutic System (TTS) in Patients with Mild to Moderate Alzheimer's Disease*, along with files representing data collected from several subjects. We derived a CTA model of this trial from the SDM documents by hand, including the activities, and partial order and timing constraints order, as detailed in the previous sections.

The prototype CTA also comes with an graphical interface (written in Java) to provide a visual representation of the design, as well as individual subjects' progress (the Viewer in Figure 1). Figure 5 shows the partial order of trial events. The events that have been performed for the subject whose progress is being monitored are colored

[8]http://www.cdisc.org/

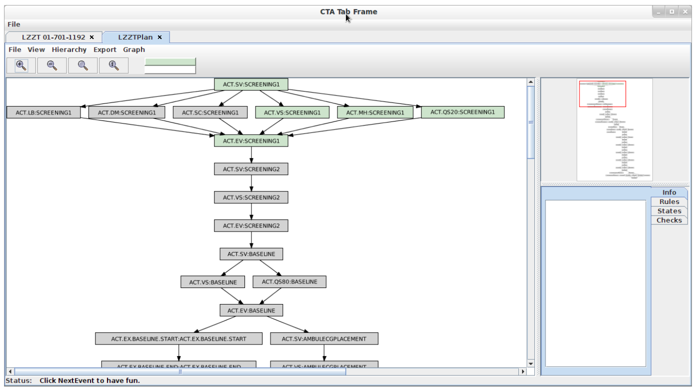


Figure 6: Partial order of activities.

green. Figure 6 shows how a monitor can review a subjects progress. The panel on the left shows only events that have been read from the event log (rectangles). Events are connected by elements of the model state that are changed by the event (ovals). In the top-left corner of the figure there are two buttons “Next Event” and “Check State”. The first button reads the next chronological event from the event log for this subject and updates the internal state of CTA. The graphical representations of the state in the two screens is also updated. The second button is used to check the current subject trial state and lists warnings about deviations and other problems in the bottom-right text box. In the displayed example, CTA found that three activities, DM (Demographics), LB (Lab Tests), and SC (Subject Characterization) were not performed during Screening. Moreover, two AE events were not correctly handled. Events associated with the warnings are color coded (a red and yellow coded event are visible) using the color key in the tool bar.

8. Related Work

A key innovation with respect to existing models is that CTA provides an *executable model of a trial with formal semantics that can be understood by both humans and machines*. A subject’s progress through a trial is viewed as execution of a process, described by the logged events. Temporal properties can be checked using a variety of logical reasoning tools. Progress of multiple subjects becomes a concurrent distributed process, and relations between events within or across subjects can be specified and checked. An executable trial model can also be used as the basis of a trial simulator to explore properties of a trial design for example given assumptions about distributions of subject characteristics.

This contrasts with existing Clinical trials models and analysis methods which are mainly focused on data collection. CDISC’s SDTM and PRM models are used to standardize names of concepts and their relationships, the information needed to describe trial elements, and unambiguous description of data representation and when it is collected. Available analyses include edit checks, that can ensure case report forms are correctly filled in and data values a within given ranges; and statistical analyses on collected data to determine the quality of the data and the confidence in the outcomes.

The Medidata Rave electronic data capture (EDC) system [8] supports creation and management of documents spanning multiple subjects, sites, and studies. Rave is certified on CDISC standards and provides edit-check, query, audit and report generation. Rave does not provide longitudinal or cross subject/study analysis. CluePoints™ [4] uses statistical modeling techniques to identify anomalous data and site errors and detect fraud. Blueprint Clinical COMPASS™ [3] provides risk-based monitoring technology reduce the amount of monitoring by identifying crucial data elements and risk factors.

These standards, models and analyses are crucial to help ensure quality results from clinical trials and simplify trial management. What is missing is support for real-time monitoring for early detection of problems and support for longitudinal and cross-subject analysis.

9. Conclusion and Future Work

This paper introduced a novel model for specifying and verifying Clinical Trials, which was used in the implementation of a Clinical Trial Assistant in Maude. The is a number of future directions to follow from this work. Up to now we have verified very simple properties that involved a single subject in a given time. We expect to be able to extend our verification analysis to include:

- Inter-Subject verification. We should be able to specify properties that depend not only on the execution of a single subject, but of multiple subjects. For example, there are CT designs where the start of a CT of one subject depends on the correct execution of some previous subjects.
- Thresholds Soft Intervals: It should also be possible to specify properties that not only depend on the time intervals of activities, but also on the results of experiments, such as the Blood Pressure of subjects.
- We do not handle periodic events, such as when the subject should take a medicine every two days.
- More refined Adverse Events: In practice not all Adverse Events are treated equally. Some are classified as serious other as not that serious. It should be possible to incorporate such a distinction in our model.
- Spatial and Epistemic modalities: Other more complicated properties involving spatial and epistemic modalities should be enforced. For example, in many trials, it should not be possible that medicine bottles leave the hospital dependencies.

Acknowledgments: Nigam was supported by the Alexander von Humboldt Foundation, the CNPq and Capes.

[1] CDISC. CDISC Study Design Model in XML (SDM-XML), Version 1.0, 2011.

[2] M. Clavel, F. Durán, S. Eker, P. Lincoln, N. Martí-Oliet, J. Meseguer, and C. Talcott. *All About Maude: A High-Performance Logical Framework*. Springer, 2007.

[3] B. Clinical. last accessed 2014 February 2.

[4] Cluepoints. last accessed 2014 February 2.

[5] FDA. Code of federal regulations, Title 21, Chapter 1, Subchapter D, Part 312: Investigational new drug application. Available at `http://www.accessdata.fda.gov/scripts/cdrh/cfdocs/cfcfr/CFRSearch.cfm?CFRPart=312`.

[6] M. A. Grando, M. Peleg, and D. Glasspool. A goal-oriented framework for specifying clinical guidelines and handling medical errors. *Journal of Biomedical Informatics*, 2009.

[7] S. Iida, G. Denker, and C. Talcott. Document logic: Risk analysis of business processes through document authenticity. *Journal of Research and Practice in Information Technology*, 2011.

[8] Medidata. last accessed 2014 February 2.