# An Executable Specification of Oncology Dose-Escalation Protocols with Prolog

David C. Norris[1] and Markus Triska[2]

[1] Precision Methodologies, LLC, Wayland MA 01778, USA
`david@precisionmethods.guru`
`https://precisionmethods.guru`
[2] `triska@metalevel.at`
`https://www.metalevel.at`

**Abstract.** We present, as a pure Prolog program, the first executable specification of the $3 + 3$ dose-escalation protocol commonly used in early-phase oncology drug development. In this program, the imperative operations of the protocol emerge as consequences of clinically meaningful anticipatory-regret scenarios that are declared as CLP($\mathbb{Z}$) constraints. This 'regret-constrained' (RC) specification yields a robust formulation which can be used to prove clinically meaningful safety and liveness properties of the protocol before incorporating it into a trial, and then as an on-line decision support system while the trial is underway. Our RC specification also readily accommodates certain pragmatic modifications to trial enrollment which severely strain traditionally imperative formulations. The features of modern Prolog systems let us describe the $3 + 3$ protocol with a short and general program that has desirable algebraic properties and can therefore be used, tested and reasoned about in several different ways.

**Keywords:** System safety · Formal system verification · Clinical trials · Dose finding.

## 1 Introduction

Clinical development of a new cancer drug for human use typically follows a 3-phase process. Phase 1 trials, which concern us here, try a drug for the first time in humans, exploring relations between *dosing* and *toxicity* which preclinical animal experimentation cannot adequately characterize. Subsequent phase 2 and 3 trials address *efficacy*, and are premised on dosing recommendations yielded by phase 1. Because cancer drugs' toxicity usually becomes evident on a shorter time-scale than their efficacy, phase 1 cancer trials often adopt a *maximum tolerated dose* (MTD) heuristic which calibrates dosing to produce mild-to-moderate toxicity. While it has long been recognized that MTD varies from one patient to another [1], it remains an almost universal practice in phase 1 oncology trials to define 'tolerability' in *population* terms, according to what proportion of patients experience a severe, 'dose-limiting' toxicity (DLT).

## 2 Prolog prerequisites and compatibility considerations

Our key contribution, described in the following sections, is the specification of a dose-escalation protocol in the form of an executable Prolog program. We have chosen Prolog as implementation language for several reasons: First, Prolog is highly suited for this application, allowing us to achieve a general and efficient program with a small code-base. Second, Prolog remains the best-known logic programming language, continuing to be taught at universities around the world. This makes our specification immediately accessible to a substantial number of programmers. Third, a number of different Prolog implementations exist, and yet are mutually compatible to varying degrees thanks to the existence of an ISO standard that defines their common syntactic and semantic features.

Standards are of great importance in the medical sector and play a significant role in procurement decisions, resolution of legal disputes, warranty questions, and the preparation of teaching material. It is to be expected that the use of an ISO-standardized programming language will enable the broadest possible adoption of our approach in such a safety-critical application area. For these reasons, we are using Scryer Prolog for our application. Scryer Prolog is a modern Prolog system written in Rust that aims for strict conformance to the Prolog ISO standard and satisfies all syntactic conformity tests given in *https://www.complang.tuwien.ac.at/ulrich/iso-prolog/conformity_ testing*. It is freely available from: *https://www.scryer.pl*. The queries we show are executed with version 0.9.3.

A moderate knowledge of Prolog is required to fully understand the code presented here. We are using only basic Prolog predicates, and features from the following additional libraries that are available in Scryer Prolog and also for several other Prolog systems:

```
:- use_module(library(lists)).
```

Commonly known relations over lists, such as `length/2` and `maplist/N`.

```
:- use_module(library(clpz)).
```

Declarative integer arithmetic, with a *monotonic* execution mode requiring that all logic variables standing for concrete integers be wrapped with the dedicated functor `(#)/1`. We use CLP($\mathbb{Z}$) constraints such as `(#=)/2` and `(#>)/2` to denote the respective relations between integer expressions, allowing for very general definitions.

```
:- use_module(library(reif)).
```

This library provides the meta-predicates `if_/3` and `tfilter/3` as described in [5].

```
:- use_module(library(dcgs)).
```

Definite clause grammars (DCGs) which are available in virtually all Prolog systems. We are using this built-in grammar mechanism to describe *sequences* of events. The nonterminal `... //0` means *any sequence*.

```
:- use_module(library(error)).
```

This library provides sound type tests. If a term is not sufficiently instantiated to decide its type, an instantiation error is raised to prevent incorrect failure in cases where a more specific instantiation would yield a solution.

```
:- use_module(library(si)).
```

"si" stands for *sufficiently instantiated*, and also for *sound inference*. This library provides additional sound type tests.

```
:- use_module(library(lambda)).
```

We use this higher-order construct to conveniently *project away* uninteresting answer substitutions in toplevel interactions, and also to define anonymous auxiliary predicates that do not warrant a standalone definition or dedicated name.

```
:- initialization(assertz(clpz:monotonic)).
```

We run our application in the monotonic execution mode to ensure that all admissible cases are taken into account, and consequently use for example the term `#X` to denote a logic variable `X` that stands for a concrete integer.

We provide a collection of texts that explain these language features in more detail at *https://www.metalevel.at/prolog*. These language features are explained in more detail at *https://www.scryer.pl* and the resources it links to.

The Prolog code we present in the following sections is designed to run with all conforming Prolog systems where the above provisions are present.

## 3 Dose-escalation trials

Participants in phase 1 oncology studies are typically patients with cancer for whom standard treatments have stopped working or do not exist. These patients enroll to pursue an experimental treatment with *therapeutic intent* [9] in the face of unknown efficacy and great uncertainty about how toxicities may manifest and at what doses.

The required therapeutic balance between tolerability and efficacy is typically sought in *dose-escalation* trials that enroll patients serially into an escalating sequence of pre-specified doses. Small *cohorts* of 1–3 patients are enrolled together, all assigned to the same dose level. Although we will demonstrate a pragmatically important generalization to variable cohort sizes, initially we specify uniform cohorts of size 3.

```
allowed_cohort_sizes([3]).
```

Whereas a complete trial protocol will specify complex clinical grading systems for assessing a broad array of distinct toxicities, for dose-escalation purposes each patient's experienced toxicity gets tallied in binary terms, according to whether the toxicity is severe enough to be deemed a 'dose-limiting toxicity' (DLT).

Eliding the time which must elapse from dosing to toxicity assessment, at any given moment each dose will be characterized by a cumulative *toxicity tally* `T/N` denoting that a DLT has occurred in `T` out of the `N` participants enrolled at that dose:

```
valid_tally(T/N) :- valid_denom(N), 0 #=< #T, #T #=< N.
```

Rapid progress being imperative in cancer drug development, phase 1 trials are typically designed to yield a recommended phase 2 dose without enrolling a large number of patients. Thus, *for phase 1 purposes*, the maximum enrollment deemed necessary to characterize any given dose may be quite modest:

```
valid_denom(N) :- N in 0..6.
```

If a dose has current tally `T0/N0`, then enrollment of a new cohort at this same dose will yield a new tally `T/N` in the expected way:

```
enroll(T0/N0, T/N) :-
    allowed_cohort_sizes(Cs),
    member(Nnew, Cs),
    #N #= #N0 + #Nnew,
    valid_denom(N),
    Tnew in 0..Nnew, indomain(Tnew),
    #T #= #T0 + #Tnew.
```

The dose-escalation *state* at any moment consists of accumulated toxicity tallies at each of an ascending sequence of up to 8 doses (a pragmatically relevant maximum rarely exceeded by actual dose-escalation trials), together with an index into this sequence identifying a ‘current dose’. We represent this by a pair of tally-lists, the left-hand `Ls` listing lower doses in *descending* dose order with the current dose at its head, and the right-hand `Hs` listing higher doses in *ascending* order with the next-higher dose at its head.

```
valid_state(Ls - Hs) :-
    append(Ls, Hs, Ds),
    length(Ds, ND), ND in 1..8,
    maplist(valid_tally, Ds).

state_tallies(Ls-Hs, Qs) :-
    valid_state(Ls-Hs),
    reverse(Ls, Js),
    append(Js, Hs, Qs).
```

As tallies accumulate at the various dose levels, the ‘current dose level’ (into which the next cohort is to be enrolled) may be updated by one of 3 possible dose-escalation *decisions*: `esc`alate, `stay` and `de-esc`alate:

```
state0_decision_state(Ls - [H0|Hs], esc, [H|Ls] - Hs) :-
    valid_state(Ls - [H0|Hs]),
    enroll(H0, H).
state0_decision_state([L0|Ls] - Hs, sta, [L|Ls] - Hs) :-
    valid_state([L0|Ls] - Hs),
    enroll(L0, L).
state0_decision_state([L,D0|Ls] - Hs, des, [D|Ls] - [L|Hs]) :-
    valid_state([L,D0|Ls] - Hs),
    enroll(D0, D).
```

The very *data structures* in which we have declared the basic template of a dose-escalation protocol already limit which dose-escalation decisions are *feasible* at any point in the trial. One cannot, for instance, escalate above the highest dose; nor can one de-escalate from the lowest:

```
state0_decision_infeasible_t(_-[], esc, true).
state0_decision_infeasible_t([_]-_, des, true).
```

We encode maximum dose-wise enrollment also as a matter of *feasibility*:

```
state0_decision_infeasible_t(_-[_/N|_], esc, true)   :- #N #>= 6.
state0_decision_infeasible_t([_/N|_]-_, sta, true)   :- #N #>= 6.
state0_decision_infeasible_t([_,_/N|_]-_, des, true) :- #N #>= 6.
```

The `false` clauses of `state0_decision_infeasible_t/3` are obtained by demonstrating that at least one valid subsequent state *does* exist:

```
state0_decision_infeasible_t(S0, E, false) :-
    member(E, [esc,sta,des]),
    Goal = exists_subsequent_state(S0, E),
    once((term_si(Goal), Goal)).

exists_subsequent_state(S0, E) :-
    state0_decision_state(S0, E, _).
```

The existential quantification performed here uses the (generally, impure) predicate `once/1` to avoid redundant solutions for greater efficiency, yet it does so in a safe way: `term_si/1` ensures that `once/1` is only called when its argument is ground and can therefore yield at most one solution. Importantly, `term_si/1` yields an *instantiation error* if its argument is not ground, differing crucially from the standard predicate `ground/1`, which can fail silently even in cases that admit solutions by further instantiations. Therefore our use of `once/1` *never incurs silent loss of solutions.*

In addition to the `esc`, `sta` and `des` decisions which permit the trial to continue, there is also the possibility of `stop`ping with a dose recommendation. The *rules* which relate observed toxicity tallies to these decisions constitute the *design* of a dose-escalation protocol.

## 4 The traditional rules of 3 + 3 dose-escalation

The conventional, and still most prevalent, dose-escalation design is the 3 + 3. For an account of this design *in practice*, we are indebted to [4], quoted at length in Figure 1. A similar description from [8] is given in Figure 2.

*"Although there is no canonical 'standard method', we present the design that we have most often encountered, which is used almost universally with minor variations. First, there is a precise [clinical] definition in the protocol of what is considered dose-limiting toxicity (DLT); it may differ in different settings. The dose levels are fixed in advance, with the first patients treated at dose level 1. Initially, one treats 3 patients at a level. If 0 out of 3 patients experiences DLT, one proceeds to the next higher level with a cohort of 3 patients. If 1 out of 3 patients experiences DLT, treat an additional 3 patients at the same dose level. If 1 out of 6 patients experiences DLT at a level, the dose escalates for the next cohort of patients. If* $\geq 2$ *out of 6 patients experience DLT, or* $\geq 2$ *out of 3 patients experience DLT in the initial cohort treated at a dose level, then one has exceeded the MTD. Some investigations will, at this point, declare the previous dose level as the MTD, but a more common requirement is to have 6 patients treated at the MTD (if it is higher than level 0). To satisfy this requirement, one would treat another 3 patients at this previous dose level if there were only 3 already treated. The MTD is then defined as the highest dose level* ($\geq 1$) *in which 6 patients have been treated with* $\leq 1$ *instance of DLT, or dose level 0 if there were* $\geq 2$ *instances of DLT at dose level 1."*

**Fig. 1.** Natural-language description of the 3 + 3 dose-escalation protocol, from [4]. Our specification implements the "more common requirement" described here.

*"In the traditional 3 + 3, phase I cancer trial design, a minimum of three participants are studied at each dose level. If none of these three participants experience a DLT, a subsequent three participants are enrolled onto the next highest dose level. If one of three participants at a dose level experiences a DLT, up to three more participants are enrolled. When a DLT is observed in at least two participants in a cohort of three to six, the MTD is [regarded as having been] exceeded and an additional three participants (up to a total of six) are treated at the next lower dose level. The MTD is defined as the dose level at which none or one of six participants (0% to 17%) experience a DLT, when at least two of three to six participants (33% to 67%) experience a DLT at the next highest dose."*

**Fig. 2.** Natural-language description of the 3 + 3 dose-escalation protocol, from [8]. The "more common" variant of [4] appears to be taken for granted here.

## 5 Indeterminacies in dose escalation

Before we can reconstruct the aforementioned rules, we must appreciate certain essential indeterminacies inherent to dose escalation.

Whereas a dose-escalation decision determines the dose at which the next cohort of trial participants will enroll, it does not *fully* determine the ensuing trial state. This is because there is a nonzero probability of DLT in any participant who receives a nonzero dose. Thus, the toxicity tally in a cohort of size `C` could turn out as any of the `C+1` possibilities `[C/C,...,0/C]`:

```
enroll_tallies(C, Qs) :-
    n_countdown(C, Ts),
    maplist(\T^(=(T/C)), Ts, Qs).

n_countdown(N, [N|Ns]) :- #N #> 0, #Nminus1 #= N - 1,
                          n_countdown(Nminus1, Ns).
n_countdown(0, [0]).
```

Moreover, under an important generalization to be explored in Section 12, the actual number of patients who will be available to enroll after a dose-escalation decision may itself be indeterminate, drawn from a list `[C|Cs]` of length $\geq 2$:

```
enrollments_tallies([C|Cs], Qs) :-
    enroll_tallies(C, Q0s),
    append(Q0s, Q1s, Qs),
    enrollments_tallies(Cs, Q1s).
enrollments_tallies([], []).
```

Thus, the decision to enroll a new cohort into a dose level that already has a `T0/N0` toxicity tally gives rise to multiple possible outcome `Tallies`:

```
tally0_tallies(T0/N0, Tallies) :-
    valid_tally(T0/N0),
    allowed_cohort_sizes(Cs),
    enrollments_tallies(Cs, Qs),
    maplist(\Qnew^Q^(Qnew=Tnew/Nnew,
                     #T #= T0 + Tnew,
                     #N #= N0 + Nnew, Q=T/N),
            Qs, Tallies).
```

## 6 Regret-constrained specification of 3 + 3 designs

We can now introduce what we call *regret-constrained* protocol specifications, which ground dose escalation in specified events deemed to induce clinically meaningful *decisional regret*, thereby constraining what dose-escalation decisions are permissible in the course of a trial. This renders explicit a linkage to underlying aims and clinical motivations left implicit by customary treatments of dose-escalation via painstaking procedural tabulations. We will exhibit a Prolog program that, on the basis of a few suitably calibrated yet clinically intuitive regret-constraints, behaves consistently with the natural-language 3 + 3 design rules quoted above.

To this end, we state that we `regret` a `Decision` when it yields some undesirable `TallyHistory`, which for the purpose of modeling the 3 + 3 design needs to take into account only the resulting tally `T/N` and the immediately preceding tally `T0/N0`:

```
state0_decision_histories(S0, E, Hs) :-
    (   E = esc,
        S0 = [T0/N0|_] - [Told/Nold|_]
    ;   E = sta,
        S0 = [T0/N0|_] - _,
        Told = T0, Nold = N0
    ;   E = des,
        S0 = [T0/N0, Told/Nold | _] - _
    ),
    tally0_tallies(Told/Nold, Qs),
    maplist(\Q^(=([Q, T0/N0])), Qs, Hs).
```

## 6.1 Specific, clinically-motivated regrets

It is always possible that, after an `esc` decision, all three patients in the new cohort will experience DLTs. Although such an *outcome* is of course regrettable, investigators need not regret *their decision* to escalate if they feel it was justified by a low enough toxicity observed at the previous dose. The judgment as to what constitutes *sufficiently low toxicity to justify escalation* will generally depend on clinical context. The 3 + 3 design embeds a judgment equating this condition with $\mathtt{N0} \geq 3$ and $\mathtt{T0/N0} \leq 1/6 \in \mathbb{Q}$. Accordingly, we regret escalation precisely when this justification is absent:

```
decision_q0_q_regret(esc, T0/N0, _, #\ ( #N0 #>= 3 #/\ #T0 * 6 #=< N0 )).
```

The literature makes equally clear that overly cautious dose-escalation risks exposing too many participants to *subtherapeutic dosing* [7]. Thus, decisional regret can also serve to constrain `des` decisions. The 3 + 3 design is consistent with a judgment regretting de-escalating from a moderately toxic tally $\{\mathtt{T0} \leq 1, \mathtt{N0} \geq 3\}$ upon appreciating very low net toxicity $\mathtt{T/N} < 1/6$ at the new lower dose:

```
decision_q0_q_regret(des, T0/N0, T/N,
                     ( #T0 #=< 1 #/\ #N0 #>= 3 ) #/\
                     ( #N #> 0 #/\ #T * 6 #< #N )
                    ).
```

The literature also exhibits ample evidence of interest in *net* properties of dose-escalation trials *ex post facto* [3]. A 3 + 3 trial run without protocol violations can never record more than 4 DLTs at any single dose level. We express this fact by positing a regret for *any* decision that results in $\geq 5$ toxicities at any dose level, regardless of preceding tally history:

```
decision_q0_q_regret(esc, _, T/_, #T #>= 5).
decision_q0_q_regret(sta, _, T/_, #T #>= 5).
decision_q0_q_regret(des, _, T/_, #T #>= 5).
```

Prolog has allowed us to state these regrets in a very general form: We regret *any* situation that satisfies the given constraints, even if nothing else is known about it. Indeed, we capture an *infinite* set of concrete tallies with each of these clauses of `decision_q0_q_regret/4`.

## 6.2 Reification of regret

To achieve efficiency while retaining the desired generality of our code with the constructs from [5], we *reify* these declared regrets in a predicate, `regret_t/3`. We ensure the `true` and `false` branches of `regret_t/3` are constructed without copy-and-paste error by employing the `term_expansion/2` mechanism to generate the `false` branch of `regret_t/3` as the constructive negation of all `true` clauses, using the reification mechanism of `library(clpz)`:

```
% Generate e_vars_disjunction _at compile time_.
:- dynamic(e_vars_disjunction/3).
term_expansion(generate_clauses, Clauses) :-
    findall(e_vars_disjunction(E, Vars, Disjunction),
            (   member(E, [esc,sta,des]),
                findall(Vs-RC,
                        (   decision_q0_q_regret(E, T0/N0, T/N, RC),
                            Vs = [N0,N,T0,T]
                        ), VsRCs),
                pairs_keys_values(VsRCs, Vs, RCs),
                maplist(=(Vars), Vs),
                foldl(\X^Disj0^Disj^(Disj=(Disj0#\/X)), RCs, 0#=1, Disjunction)
            ),
            Clauses).
generate_clauses.

regret_t(E, H, Truth)  :-
    e_h_disjunction(E, H, Disjunction),
    Disjunction #<==> #B,
    b_t(B, Truth).

b_t(0, false).
b_t(1, true).

e_h_disjunction(E, H, Disjunction) :-
```

```
    H = [T/N, T0/N0],
    e_vars_disjunction(E, [N0,N,T0,T], Disjunction).
```

The `e_vars_disjunction/3` clauses thus generated are:

```
?- listing(e_vars_disjunction/3).
 e_vars_disjunction(esc,[A,B,C,D],0#=1#\/ #\ (#A#>=3#/\ #C*6#=<A)#\/ #D#>=5).
 e_vars_disjunction(sta,[A,B,C,D],0#=1#\/ #D#>=5).
 e_vars_disjunction(des,[A,B,C,D],
                 0#=1#\/ #C#=<1#/\ #A#>=3#/\(#B#>0#/\ #D*6#< #B)#\/ #D#>=5).
    true.
```

## 6.3 Anticipatory regret

Whereas `regret_t/3` *looks backwards* to already-realized toxicity tallies, it is also possible to *anticipate* all possible `Regrets`:

```
state0_decision_regrets(S0, E, Regrets) :-
    state0_decision_histories(S0, E, Hs),
    tfilter(regret_t(E), Hs, Regrets).
```

A dose-escalation decision is `regrettable` iff the list of possible `Regrets` is nonempty:

```
state0_decision_regrettable_t(S0, E, false) :-
    state0_decision_regrets(S0, E, []).
state0_decision_regrettable_t(S0, E, true) :-
    state0_decision_regrets(S0, E, [_|_]).
```

By conducting dose escalation to *avert* anticipated regret, resolving degenerate choices by therapeutically motivated preference relations `esc` $\succ$ `sta` $\succ$ `des` that favor exploration of higher doses, we obtain an effective protocol specification.

```
state0_nextdecision(S0, E) :-
    if_((   state0_decision_infeasible_t(S0, esc)
        ;   state0_decision_regrettable_t(S0, esc)),
        if_((   state0_decision_infeasible_t(S0, sta)
            ;   state0_decision_regrettable_t(S0, sta)),
            if_((   state0_decision_infeasible_t(S0, des)
                ;   state0_decision_regrettable_t(S0, des)),
                E = stop,
                E = des),
            E = sta),
        E = esc).
```

Finally, we obtain all possible trial paths using a DCG `path//1` which describes all paths of the trial from the current state `S0`:

```
path(S0) --> { state0_nextdecision(S0, E),
               state0_decision_state(S0, E, S) },
             [E, S],
             path(S).
path(S0) --> { state0_nextdecision(S0, stop),
               stopstate_rec(S0, Rec) },
             [stop, recommend_dose(Rec)].
path(recommend_dose(_)) --> [].
```

When none of the dose-escalation decisions `[esc,sta,des]` remains feasible, the trial `stop`s, yielding a *dose recommendation.* The relation between the final state and this recommendation is specified by `stopstate_rec/2`:

```
stopstate_rec([]-_, 0).
stopstate_rec([T/N|TNs]-_, Rec) :-
        (   #T * 6 #> #N, length(TNs, Rec)
        ;   #T * 6 #=< #N, length([_|TNs], Rec)
        ).
```

## 7 Algebraic properties of our formulation

With the lone exception of `once/1`, which is guaranteed to be used in a safe way as explained in Section 3, the resulting program consists exclusively of *monotonic* Prolog constructs. Our program thus *by construction* provides algebraic properties that we consider essential in this safety-critical application area:

- If an answer says that no (further) solutions exist to a query, then there are truly no (further) solutions, no matter which additional constraints we add. In other words, our program cannot be "tricked" into yielding solutions that other queries deny.
- The predicates of `library(debug)` can be used for *declarative debugging* of the program itself.
- Every query either shows *all* possible answers, or yields an *instantiation error* if the query is not sufficiently instantiated.
- Reordering any clauses or goals in the program can affect termination or the existence of *instantiation errors*, but it does not change the set of described solutions.

In addition, the Scryer Prolog toplevel always shows all pending constraints and can therefore be used like a theorem prover: When a query succeeds unconditionally, a solution is guaranteed to exist.

These properties ensure that our program constitutes a declarative specification that can be queried, run and reasoned about in different ways, and yields only correct results in all possible usage modes. The only possible remaining source of logic errors is a mistake in the protocol formulation itself.

## 8 Relation of our specification to natural-language descriptions

From this program, we can elicit concretely much of the meaning of the text in Figure 1, which we exhibit for the case of D = 3 doses:

– *The dose levels are fixed in advance, with the first patients treated at dose level 1.*
```
?- setof(E^S1, Etc^(phrase(path([0/0]-[0/0,0/0]), [E, S1 | Etc])),
         Starts).
   Starts = [sta^([0/3]-[0/0,0/0]),sta^([1/3]-[0/0,0/0]),
             sta^([2/3]-[0/0,0/0]),sta^([3/3]-[0/0,0/0])].
```
– *If 0 out of 3 patients experiences DLT, one proceeds to the next higher level with a cohort of 3 patients.*
```
?- setof(E, Path^Ls^H^Hs^(phrase(path([0/0]-[0/0,0/0]), Path),
             phrase((..., [[0/3|Ls]-[H|Hs]], [E], ...), Path)), Es).
   Es = [esc].
```
– *If 1 out of 3 patients experiences DLT, treat an additional 3 patients at the same dose level.*
```
?- setof(E, Path^Ls^H^Hs^(phrase(path([0/0]-[0/0,0/0]), Path),
             phrase((..., [[1/3|Ls]-[H|Hs]], [E], ...), Path)), Es).
   Es = [sta].
```
– *If 1 out of 6 patients experiences DLT at a level, the dose escalates for the next cohort of patients.*
```
?- setof(E, Path^Ls^H^Hs^(phrase(path([0/0]-[0/0,0/0]), Path),
                              H = 0/0,
               phrase((...,[[1/6|Ls]-[H|Hs]],[E],...), Path)), Es).
   Es = [esc].
```

Notably, eliciting the intended meaning of the text in the last case required positing that the next-higher dose H had not yet been enrolled. Without this proviso, stop would also have been permissible.

From the decision cascade in the body of state0_nextdecision/2 it is evident that our formulation yields a unique decision in any concrete situation (with ground S0); we therefore consider it a complete specification. The Appendix includes a complete enumeration of all 46 paths for the D = 2 case, easily checked vis-à-vis the text of Figure 1.

## 9 Verification of safety properties

The text of Figure 1 trends from language that is initially imperative, toward declarative statements near the end. The latter reflect clinical investigators' intense interest in asserting *safety properties* for their trials and (by extension) for the conclusions drawn from them. Informally, a safety property states that *bad things do not happen.* For example, the "more common requirement" *to have 6 patients treated at the MTD (if it is higher than level 0)*

expresses a concern to avoid recommending a dose on the strength of too little experience.

Our executable specification exhibits safety properties expressed in Figure 1 through statements about 'the MTD'. For example,

– *If $\geq 2$ out of 6 patients experience DLT, or $\geq 2$ out of 3 patients experience DLT in the initial cohort treated at a dose level, then one has exceeded the MTD.*

```
recommends_dose_exceeding_mtd              :-
   D in 1..8, indomain(D),       % For trials of up to D=8 doses
   InitD = [Q]-Qs, length([Q|Qs], D), % .. that start from the lowest dose
   maplist(=(0/0), [Q|Qs]),      % .. with no prior toxicity information,
   phrase(path(InitD), Path),    % does any Path exist
   phrase((..., [Ls-_], ...,     % .. on which a state Ls-_ appears,
           [recommend_dose(Rec)] % .. such that the recommended dose Rec
          ), Path),
   length(Ls,X), Rec #>= X,      % .. was at least the current dose X,
   Ls = [T/_|_], #T #> 1.        % .. yet X 'exceeded MTD' per protocol?
%?- time(recommends_dose_exceeding_mtd).
%@    % CPU time: 574.029s, 2_701_969_903 inferences
%@    false. % This safety property is verified for trials of up to 8 doses.
```

Note that the featured statement effectively asserts a property of the desirable *dose recommendation* (which it refers to as 'the MTD'), namely that it should be less than any dose at which $\geq 2$ toxicities have been tallied.

## 10 Verification of liveness properties

Although clinical investigators' keen attention to safety is more readily apparent in the literature, any sensible dose-escalation design must also guarantee *liveness* properties. Informally, liveness properties state that *good things do happen.* One such property applicable to these trials is that they *yield a recommendation, and then immediately stop*:

```
trial_fails_to_conclude_with_unique_rec :-
   D in 1..8, indomain(D), length([Q|Qs], D), maplist(=(0/0), [Q|Qs]),
   phrase(path([Q]-Qs), Path),
   (   phrase((..., [recommend_dose(Rec)], [_], ...), Path)
   ;   \+ phrase((..., [recommend_dose(Rec)]), Path)
   ).
%?- time(trial_fails_to_conclude_with_unique_rec).
%@    % CPU time: 560.946s, 2_675_988_416 inferences
%@    false. % All trials of up to 8 doses yield a recommendation, then stop.
```

## 11 Generality and versatility of the specification

The DCG introduced in Section 9 is remarkably general and versatile. It is an *executable specification* of the sequences of events which may occur in the trial, in the sense that it states *what holds*, and it can be used in multiple *modes*. This generality allows us to answer a variety of questions with comparatively simple queries. For example, in one specific mode, we can use our specification as an 'expert system' that automatically determines what the next decision should be at a point where we have reached the highest dose of a `D` $= 3$ trial, having recorded a $0/3$ tally at each dose:

```
?- setof(E, Path^(phrase(path([0/0]-[0/0,0/0]), Path),
                  phrase((...,[[0/3,0/3,0/3]-[], E], ...), Path)), Es).
   Es = [sta].
```

We can as easily adopt a retrospective point of view, to ask what dose-escalation decisions *could have preceded* this state:

```
?- setof(E0, Path^(phrase(path([0/0]-[0/0,0/0]), Path),
                   phrase((...,[E0, [0/3,0/3,0/3]-[]],...), Path)), E0s).
   E0s = [esc].
```

We are also able to look far ahead, anticipating all recommendations that remain possible at any time in an ongoing trial. Having recorded tallies of $0/3$, $0/3$ and $2/6$ at dose levels 1, 2 and 3 of a `D` $= 3$ trial, for example:

```
?- setof(Rec, Path^(phrase(path([0/0]-[0/0,0/0]), Path),
                    phrase((..., [[2/6,0/3,0/3]-[]],
                            ..., [recommend_dose(Rec)]), Path)), Recs).
   Recs = [0,1,2]. % Only dose level 3 has been ruled out so far.
```

Finally, the DCG allows complete enumeration of *all admissible trials*:

```
?- J+\time((D = 8, length([Q|Qs], D), maplist(=(0/0), [Q|Qs]),
            setof(Path, phrase(path([Q]-Qs), Path), Paths),
            length(Paths, J))).
%@    % CPU time: 337.638s, 1_573_175_565 inferences
%@    J = 16138.
```

This capability can confer substantial benefits even on analyses of a statistical nature. For example, [6] employs such enumeration to derive statistical characterizations of 3 + 3 trials, exempt from Monte Carlo error. The generality and versatility of our Prolog code enables more usage modes than a procedural implementation can provide. We regard the run-times of our program in all these modes as commensurate to the importance of reliable guarantees in this domain, and therefore do not pursue any further performance considerations in the present paper.

## 12 Flexibly accommodating protocol variations

As shown by the expansive tabulations in [8] and [2], when working within the tradition of informal protocol specifications, much labor is needed to adapt the typically lumpy cohorts of dose-escalation designs to trial enrollment occurring as a rather fluid *arrival process* in real time. Our RC specification framework, by contrast, shows promise for being readily adaptable to this challenge.

If we recompile our program with an expanded set of possible cohort sizes,

```
allowed_cohort_sizes([3,2,1]).
```

then, as the following query shows, this added flexibility renders the protocol adequate to admit paths in which participants enroll either singly or in pairs, rather than waiting for arrival of a third participant to complete the traditional cohort of 3. Such just-in-time or 'rolling' [8] enrollment may enable de-escalation decisions to occur sooner, avoiding dosing patients at levels that are no longer viable dose recommendations. This improves safety, and also accelerates progress of the trial.

```
?- phrase(path([0/3,0/3,0/3]-[]), [sta, [2/5,0/3,0/3]-[],
                                   des, [0/6,0/3]-[2/5],
                                   stop, recommend_dose(2)]).
   true % The above-described path is admissible.
;  ... .
```

## 13 Conclusion

Possibly the most essential aspect of our program's *declarativeness* is its generality. This quality has allowed us to examine our program in numerous ways to probe comprehensively for various forms of error. We can investigate individual predicates with general queries that expose their full meaning *as implemented*, to check for departures from *intended* meaning. Yet the very same program allows us to recapitulate the best available natural-language description of the 3 + 3 protocol, and indeed in so doing to recognize deficiencies in that description and *correct* them. Not only does this reduce the likelihood that we have failed to model 'the true' 3 + 3 protocol, but it gives us a claim to have definitively superseded the extant natural-language descriptions.

The *internal* consistency of our program, on the other hand, derives from its very construction using a pure monotonic subset of Prolog. This enables us to trust `false` answers from queries that search for violations of safety and liveness properties, rendering such answers *guarantees*.

**Acknowledgments.** We thank Ulrich Neumerkel and several anonymous reviewers for their valuable comments.

**Disclosure of Interests.** The authors have no competing interests to declare that are relevant to the content of this article. This work received no external funding.

## References

1. Edler, L.: Statistical requirements of phase I studies. Onkologie **13**(2), 90–95 (Apr 1990). https://doi.org/10.1159/000216733
2. Frankel, P.H., Chung, V., Tuscano, J., Siddiqi, T., Sampath, S., Longmate, J., Groshen, S., Newman, E.M.: Model of a Queuing Approach for Patient Accrual in Phase 1 Oncology Studies. JAMA Network Open **3**(5), e204787–e204787 (May 2020). https://doi.org/10.1001/jamanetworkopen.2020.4787, https://doi.org/10.1001/jamanetworkopen.2020.4787
3. Italiano, A., Massard, C., Bahleda, R., Vataire, A.L., Deutsch, E., Magné, N., Pignon, J.P., Vassal, G., Armand, J.P., Soria, J.C.: Treatment outcome and survival in participants of phase I oncology trials carried out from 2003 to 2006 at Institut Gustave Roussy. Annals of Oncology: Official Journal of the European Society for Medical Oncology **19**(4), 787–792 (Apr 2008). https://doi.org/10.1093/annonc/mdm548
4. Korn, E.L., Midthune, D., Chen, T.T., Rubinstein, L.V., Christian, M.C., Simon, R.M.: A comparison of two phase I trial designs. Statistics in Medicine **13**(18), 1799–1806 (Sep 1994). https://doi.org/10.1002/sim.4780131802, http://doi.wiley.com/10.1002/sim.4780131802
5. Neumerkel, U., Kral, S.: Indexing dif/2. CoRR **abs/1607.01590** (2016), http://arxiv.org/abs/1607.01590
6. Norris, D.C.: What Were They Thinking? Pharmacologic priors implicit in a choice of 3+3 dose-escalation design. arXiv:2012.05301 [stat.ME] (Dec 2020), https://arxiv.org/abs/2012.05301
7. Simon, R., Freidlin, B., Rubinstein, L., Arbuck, S.G., Collins, J., Christian, M.C.: Accelerated titration designs for phase I clinical trials in oncology. Journal of the National Cancer Institute **89**(15), 1138–1147 (Aug 1997). https://doi.org/10.1093/jnci/89.15.1138
8. Skolnik, J.M., Barrett, J.S., Jayaraman, B., Patel, D., Adamson, P.C.: Shortening the Timeline of Pediatric Phase I Trials: The Rolling Six Design. Journal of Clinical Oncology **26**(2), 190–195 (Jan 2008). https://doi.org/10.1200/JCO.2007.12.7712, http://ascopubs.org/doi/full/10.1200/JCO.2007.12.7712
9. Weber, J.S., Levit, L.A., Adamson, P.C., Bruinooge, S.S., Burris, H.A., Carducci, M.A., Dicker, A.P., Gönen, M., Keefe, S.M., Postow, M.A., Thompson, M.A., Waterhouse, D.M., Weiner, S.L., Schuchter, L.M.: Reaffirming and Clarifying the American Society of Clinical Oncology's Policy Statement on the Critical Role of Phase I Trials in Cancer Research and Treatment. Journal of Clinical Oncology **35**(2), 139–140 (Nov 2016). https://doi.org/10.1200/JCO.2016.70.4692, http://ascopubs.org/doi/full/10.1200/JCO.2016.70.4692

## Appendix

A complete listing of all 46 paths of the 3 + 3 design for the `D` = 2 case. These may be checked one-by-one against the text of Figure 1.

```
:- use_module(library(format)).
?- J+\(setof(Path, (phrase(path([0/0]-[0/0]), Path)), Paths)
      , maplist(portray_clause, Paths), length(Paths, J)).
[sta,[0/3]-[0/0],esc,[0/3,0/3]-[],sta,[0/6,0/3]-[],stop,recommend_dose(2)].
[sta,[0/3]-[0/0],esc,[0/3,0/3]-[],sta,[1/6,0/3]-[],stop,recommend_dose(2)].
[sta,[0/3]-[0/0],esc,[0/3,0/3]-[],sta,[2/6,0/3]-[],des,[0/6]-[2/6],stop,recommend_dose(1)].
[sta,[0/3]-[0/0],esc,[0/3,0/3]-[],sta,[2/6,0/3]-[],des,[1/6]-[2/6],stop,recommend_dose(1)].
[sta,[0/3]-[0/0],esc,[0/3,0/3]-[],sta,[2/6,0/3]-[],des,[2/6]-[2/6],stop,recommend_dose(0)].
[sta,[0/3]-[0/0],esc,[0/3,0/3]-[],sta,[2/6,0/3]-[],des,[3/6]-[2/6],stop,recommend_dose(0)].
[sta,[0/3]-[0/0],esc,[0/3,0/3]-[],sta,[3/6,0/3]-[],des,[0/6]-[3/6],stop,recommend_dose(1)].
[sta,[0/3]-[0/0],esc,[0/3,0/3]-[],sta,[3/6,0/3]-[],des,[1/6]-[3/6],stop,recommend_dose(1)].
[sta,[0/3]-[0/0],esc,[0/3,0/3]-[],sta,[3/6,0/3]-[],des,[2/6]-[3/6],stop,recommend_dose(0)].
[sta,[0/3]-[0/0],esc,[0/3,0/3]-[],sta,[3/6,0/3]-[],des,[3/6]-[3/6],stop,recommend_dose(0)].
[sta,[0/3]-[0/0],esc,[1/3,0/3]-[],sta,[1/6,0/3]-[],stop,recommend_dose(2)].
[sta,[0/3]-[0/0],esc,[1/3,0/3]-[],sta,[2/6,0/3]-[],des,[0/6]-[2/6],stop,recommend_dose(1)].
[sta,[0/3]-[0/0],esc,[1/3,0/3]-[],sta,[2/6,0/3]-[],des,[1/6]-[2/6],stop,recommend_dose(1)].
[sta,[0/3]-[0/0],esc,[1/3,0/3]-[],sta,[2/6,0/3]-[],des,[2/6]-[2/6],stop,recommend_dose(0)].
[sta,[0/3]-[0/0],esc,[1/3,0/3]-[],sta,[2/6,0/3]-[],des,[3/6]-[2/6],stop,recommend_dose(0)].
[sta,[0/3]-[0/0],esc,[1/3,0/3]-[],sta,[3/6,0/3]-[],des,[0/6]-[3/6],stop,recommend_dose(1)].
[sta,[0/3]-[0/0],esc,[1/3,0/3]-[],sta,[3/6,0/3]-[],des,[1/6]-[3/6],stop,recommend_dose(1)].
[sta,[0/3]-[0/0],esc,[1/3,0/3]-[],sta,[3/6,0/3]-[],des,[2/6]-[3/6],stop,recommend_dose(0)].
[sta,[0/3]-[0/0],esc,[1/3,0/3]-[],sta,[3/6,0/3]-[],des,[3/6]-[3/6],stop,recommend_dose(0)].
[sta,[0/3]-[0/0],esc,[1/3,0/3]-[],sta,[4/6,0/3]-[],des,[0/6]-[4/6],stop,recommend_dose(1)].
[sta,[0/3]-[0/0],esc,[1/3,0/3]-[],sta,[4/6,0/3]-[],des,[1/6]-[4/6],stop,recommend_dose(1)].
[sta,[0/3]-[0/0],esc,[1/3,0/3]-[],sta,[4/6,0/3]-[],des,[2/6]-[4/6],stop,recommend_dose(0)].
[sta,[0/3]-[0/0],esc,[1/3,0/3]-[],sta,[4/6,0/3]-[],des,[3/6]-[4/6],stop,recommend_dose(0)].
[sta,[0/3]-[0/0],esc,[2/3,0/3]-[],des,[0/6]-[2/3],stop,recommend_dose(1)].
[sta,[0/3]-[0/0],esc,[2/3,0/3]-[],des,[1/6]-[2/3],stop,recommend_dose(1)].
[sta,[0/3]-[0/0],esc,[2/3,0/3]-[],des,[2/6]-[2/3],stop,recommend_dose(0)].
[sta,[0/3]-[0/0],esc,[2/3,0/3]-[],des,[3/6]-[2/3],stop,recommend_dose(0)].
[sta,[0/3]-[0/0],esc,[3/3,0/3]-[],des,[0/6]-[3/3],stop,recommend_dose(1)].
[sta,[0/3]-[0/0],esc,[3/3,0/3]-[],des,[1/6]-[3/3],stop,recommend_dose(1)].
[sta,[0/3]-[0/0],esc,[3/3,0/3]-[],des,[2/6]-[3/3],stop,recommend_dose(0)].
[sta,[0/3]-[0/0],esc,[3/3,0/3]-[],des,[3/6]-[3/3],stop,recommend_dose(0)].
[sta,[1/3]-[0/0],sta,[1/6]-[0/0],esc,[0/3,1/6]-[],sta,[0/6,1/6]-[],stop,recommend_dose(2)].
[sta,[1/3]-[0/0],sta,[1/6]-[0/0],esc,[0/3,1/6]-[],sta,[1/6,1/6]-[],stop,recommend_dose(2)].
[sta,[1/3]-[0/0],sta,[1/6]-[0/0],esc,[0/3,1/6]-[],sta,[2/6,1/6]-[],stop,recommend_dose(1)].
[sta,[1/3]-[0/0],sta,[1/6]-[0/0],esc,[0/3,1/6]-[],sta,[3/6,1/6]-[],stop,recommend_dose(1)].
[sta,[1/3]-[0/0],sta,[1/6]-[0/0],esc,[1/3,1/6]-[],sta,[1/6,1/6]-[],stop,recommend_dose(2)].
[sta,[1/3]-[0/0],sta,[1/6]-[0/0],esc,[1/3,1/6]-[],sta,[2/6,1/6]-[],stop,recommend_dose(1)].
[sta,[1/3]-[0/0],sta,[1/6]-[0/0],esc,[1/3,1/6]-[],sta,[3/6,1/6]-[],stop,recommend_dose(1)].
[sta,[1/3]-[0/0],sta,[1/6]-[0/0],esc,[1/3,1/6]-[],sta,[4/6,1/6]-[],stop,recommend_dose(1)].
[sta,[1/3]-[0/0],sta,[1/6]-[0/0],esc,[2/3,1/6]-[],stop,recommend_dose(1)].
[sta,[1/3]-[0/0],sta,[1/6]-[0/0],esc,[3/3,1/6]-[],stop,recommend_dose(1)].
[sta,[1/3]-[0/0],sta,[2/6]-[0/0],stop,recommend_dose(0)].
[sta,[1/3]-[0/0],sta,[3/6]-[0/0],stop,recommend_dose(0)].
[sta,[1/3]-[0/0],sta,[4/6]-[0/0],stop,recommend_dose(0)].
[sta,[2/3]-[0/0],stop,recommend_dose(0)].
[sta,[3/3]-[0/0],stop,recommend_dose(0)].
   J = 46.
```