Categories and Preorders in Value Iteration: Fixed Points and Surrogate Models

Internship Report

Louis Rustenholz,
supervised by Ichiro Hasuo and Jérémy Dubut

August 22, 2021

The goal of this paper is to understand and generalise modern model-checking techniques based on fixed point characterisations. To do this, we leverage the general coalgebraic view of processes, and study order structures that may arise.

We encode branching structure in *process types*, additional logical structure in *process-predicate types* and fixed point problems as *process-predicate types with leaf structures.* Moreover, we approximate processes by simpler kind of processes by comparing *process-predicate types with shape structures.*

This paper has two main contributions, which can be used as recipes for value iteration (VI) and bounded value iteration (BVI) algorithms.

The first main contribution is a theorem simplifying the verification of monotonicity hypotheses at the heart of VI algorithms.

It is striking that solutions of most problems about processes can be characterised as fixed points of weakest-precondition transformers, defined on well-chosen modalities. We understand this through the notion of *leaf structure*, which encodes sufficient conditions that make predicate transformers be monotone, thus proving the existence of fixed points through the Knaster-Tarski theorem.

In particular, we generalise the result that, even with infinite state space and branching, transfinite value iteration solves the reachability problem for stochastic games.

The second main contribution is a double approximation theorem based on comparison of process types.

In model checking, it is useful to model complex processes with simpler kind of processes, using surrogate models. Problems in games and decision processes can be related. Problems in infinitely and finitely branching transition systems can be related. Reachability in Markov Decision Processes can be related to widest paths in weighted graphs [1].

The notions of *shape structure*, *connection* and *tight connection* help us to understand those relations, which are more than a theoretical tool: they allow for approximate computations, used to improve value iteration with speed-ups and bounds.

In particular, we prove that when a process-predicate $\mathcal{T}_\downarrow$ with both shape and leaf structure has a "tight overapproximation" $\mathcal{T}_\uparrow$, least fixed points (suprema) in the lower model can be described as infima of a sequence of least fixed points in the higher model, thus paving the way for BVI through surrogate models.

Finally, this theory lays the foundation for future results, e.g. for continuous value iteration and towards a broader class of surrogate model algorithms for model checking.

Contents

1 Introduction

The goal of this internship, described in the abstract, is to find theoretical structures helpful in generalising modern model checking techniques, in particular based on value iteration and surrogate models.

This introduction is divided in five sections. The scientific context is broadly described in Section 1.1. Our main contributions are listed in Section 1.2, and lines of research left open are then discussed in Section 7.1. To simplify navigation in this report, its plan is described in Section 1.3. Finally, Section 1.4 is devoted to acknowledgements.

1.1 Scientific context

1.1.1 Formal methods – Societal relevance

This internship – and the corresponding report – takes place in the context of formal methods, automated verification and static analysis. Those fields, broadly construed, aim at tackling the following questions. Given a *formal model* and a *specification* described as a logical formula, how can we know if the specification holds? Given a specification, can we synthesise a controller that fulfils its requirements? How can we know if a specification is satisfiable? Can we always know it? How fast? More generally, how can we analyse the behaviour of a formal model, especially of computer programs?

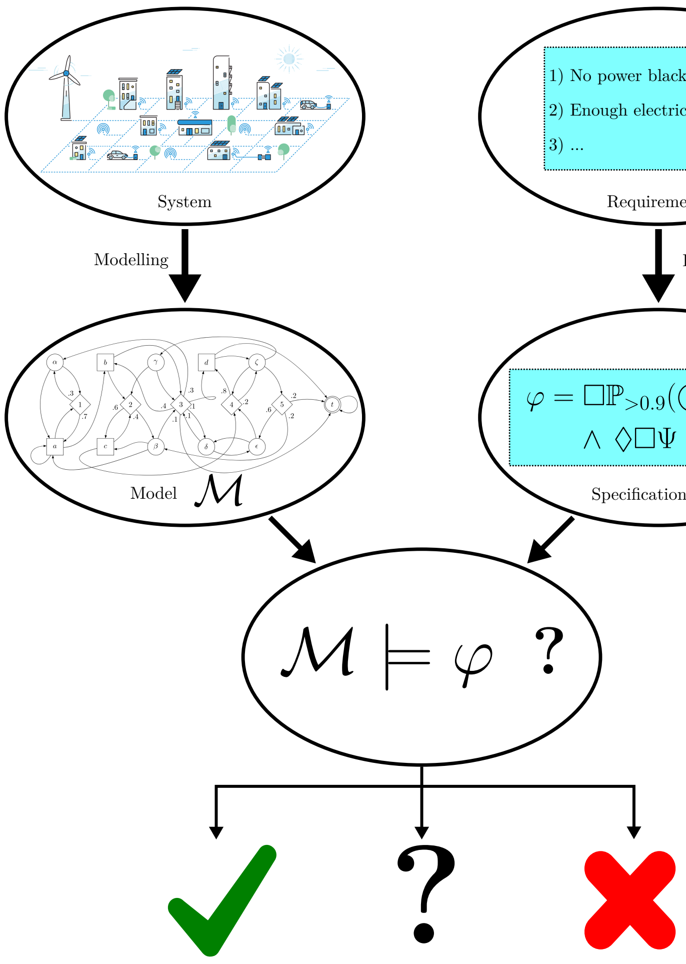

System
1) No power blackout
2) Enough electricity for all
3) ...
Requirements
Modelling
Formalisation
Model $\mathcal{M}$
$\varphi = \Box\mathbb{P}_{>0.9}(\bigcirc A \,\mathrm{U}\, B) \wedge \Diamond\Box\Psi \wedge \Box\neg\Phi$
Specification φ
$\mathcal{M} \models \varphi$?
?

Illustration – We want to check if the model of a system satisfies a formal specification.

Those questions are not only deep and interesting from the point of view of computer science and logics. They have tremendous societal relevance.

First, formal methods can be used to *chase bugs*. In applications that require high level of confidence, bugs can have a huge cost, both from a human and financial point of view. How can we bet our lives on automated railways, autonomous cars, nuclear power stations, biomedical devices, etc. if they contain programs that could be faulty? What about trusting websites and cryptographic protocols dealing with private data, or spacecrafts that have cost billions? Over and over, examples have shown that testing is not enough for safety critical systems, especially against malicious users.

However, formal methods are more than defensive tools that improve quality and reduce debugging costs. They can be used as engines making algorithms possible, e.g. to generate control strategies, synthesise hardware, or optimise compilers.

It must be emphasised that formal methods are not only about "computer programs" in the sense of OS kernels, compilers and difficult algorithms. Formal methods are also used to study communication protocols, (cyber-)physical systems, biochemical networks, games, safe machine learning, etc. Although some of those cases are contemporary scientific challenges, formal methods may be used to deal with systems that are non-deterministic, probabilistic, continuous or concurrent. Application fields include public transports and energy, to safely control complex infrastructures like railway networks or continental-scale electrical grids.

To give one example among others, we can mention that many recent papers, including Section 6.6.1 of [2], [3] and [4], wield model checking and stochastic games to study strategies in electrical grids with demand-side management and renewable energy sources.

1.1.2 Formal methods – A broad set of techniques

Formal methods consist of a fairly broad set of techniques stemming from theoretical computer science and applied logic. This report focuses on the model checking perspective, but many other approaches tackle the problems studied by formal methods. While those subfields are strongly related together, we can isolate and mention some of them.

- **Deductive verification**. In mathematics, it is usual to prove properties by manipulating specifications as *syntactic* objects, through the use of a *proof calculus* and *axioms*. In the context of systems' verification, this often leads to proof obligations, invariants, preconditions and postconditions. Proofs may be done with a high level of confidence by using an *interactive theorem prover*, such as Coq [5] (based on the calculus of inductive constructions), Agda [6], Isabelle [7] or Lean [8]. Human interaction is required.

- **Automated theorem proving**. For some logics, proof calculi and models, the above process may be completely or partially automated. For example, while the *general* question of satisfiability of first-order logic is undecidable ([9], [10], [11]), it is decidable for *particular* first-order theories [12]. Similarly, several modal logics [13] have been introduced as efficiently decidable fragments of first-order logic. The field of descriptive complexity [14] provides valuable insight on the relationship between logics' expressiveness and computational complexity. Automated theorem prover include SAT solvers [15] and SMT solvers (such as Alt-Ergo [16], CVC4 [17] and Z3 [18]).

- **Model checking**. This technique focuses less on the syntactic structure of the logic, and more on exhaustive exploration of the model itself. An emphasis is put on exploration of the state space through efficient algorithms. While graph and automata algorithms are central, model checking is not only about these. To avoid state explosion, model checking tools may simplify the model, use partial order reduction, take advantage of symmetry, find clever abstractions, use symbolic algorithms, etc. Tools include PRISM [19], SPIN [20], TLC model checker for TLA+ [21] and UPPAAL [22].

- **Abstract interpretation**. Since analysis of computer programs and formal models is often undecidable ([23], [11]) or computationally intractable, sound abstractions are required. Abstract interpretation, based on order theory, provides many tools to efficiently obtain partial information. Abstract interpretation is a general theory of sound approximations of semantics, and is based on *abstract domains*, which can often be given geometric interpretation. First developed in the late 1970s [24], it is extensively used in static analysis (e.g. for compiler optimisation or verification) and in cooperation with other formal methods. Tools based on abstract interpretation include Astrée [25], FLUCTUAT [26], or more recently neural network verifiers like ERAN [27] or Sherlock [28].

When applying formal methods, several architectural choices are to be taken into account: the modelling language, the specification language, and the verification tool.

1.1.3 Semantics

Semantics is a field of theoretical computer science, which may be described as the rigorous mathematical study of the *meaning* of programming languages. In particular, it provides techniques to formally define this meaning. This avoids relying on definitions made in prose, which are imprecise, subjective and impossible to analyse.

Over the years, semantics has found applications both to formal verification and to the design of new programming languages, by providing insight and inspiration.

The long-term goal pursued in our paper is to understand the essence of state-of-the-art model checking techniques and to generalise them to new instances. To achieve this goal, we have several reasons to tap into semantics literature. Indeed, semantics is necessary to capture the essence of model checking, and provides valuable tools for generalisation. In particular, we rely extensively on work in denotational semantics based on category theory and order theory.

Three major approaches to semantics can be described.

- **Operational semantics.** Operational methods define the meaning of the language directly through a term rewriting system, working directly on the syntax. They are usually easy to write (this may be seen as describing a low-level interpreter of the language), useful for some applications, but are hard to analyse, debug, and are tightly linked to a low-level description of an inefficient implementation. Examples include small-step and big-step semantics for untyped lambda calculus, also used for LISP [29].
- **Denotational semantics.** Denotational methods try to give a *mathematical meaning* to the language, by associating mathematical objects (denotations) to language constructs. They are high-level, more powerful and less rigid than operational semantics, but can be much harder to design. Denotational semantics are rather easy to analyse mathematically and amenable to techniques like abstract interpretation. Moreover, they are often *compositional*, which gives them great structure and flexibility.
- **Axiomatic semantics.** In this third, coarsest method, no distinction is made between a phrase's meaning and its properties: its meaning is exactly what can be proven about it in some logic. A classical example is Hoare logic [30], based on Hoare triples.

The case of denotational semantics is our main inspiration.

Category theory is an excellent tool for it, since it is especially well-suited to enforce compositionality. A good general introduction to category theory is [31]. Many examples of categorical semantics can be given, e.g. relying on the Curry-Howard-Lambek correspondence.

In the general study of *processes* (such as transition systems, automata, Markov chains, stochastic games, etc.), categorical methods have given rise to the field of coalgebra [32].

Another complementary approach is by order theory, giving rise to domain theory [33] whose methods have much interaction with the theory of abstract interpretation. A recurring theme in domain theory is the notion of *fixed points*.

In this report, we benefit from the categorical perspective of coalgebra theory and the extensive results in order theory obtained in the context of domain theory.

Moreover, we take inspiration from *predicate transformer semantics* [34], Dijkstra's reformulation of Hoare axiomatic semantics, which has been studied in the coalgebraic context by Ichiro Hasuo [35].

1.1.4 Coalgebra

The theory of coalgebra can be seen as a way to study *processes* (e.g. transition systems, automata, Markov chains, stochastic games, etc.) and *infinite structures* (e.g. streams, infinite trees) in a generic way.

Since the initial goal of the internship was to generalise a result about surrogate models in the context of stochastic games and weighted graphs, this theory of processes is a central foundation to our work. This is discussed in Section 2.

An introduction to this theory can be found in the book "Introduction to Coalgebra – Towards Mathematics of States and Observation" [32], written by Bart Jacobs, a founder of the field.

Since coalgebra generalises models that are central in computer science, such as automata, it has found several applications over the years, not only to semantics but also to algorithmics. The following are only a few examples.

- It helps to efficiently define notions of *simulation and bisimulation* in new contexts [36], [37]. (Bi)simulations have several uses in theoretical computer science, and are especially useful to design model *minimisation algorithms* that fight state space explosion in model checking.
- In particular, efficient algorithms for *partition refinement* have been found [38].
- It found applications to *automata learning*, where languages and models are learned from observations [39].
- It has been used to *implement coinductive types* in functional languages, as illustrated by *CoCaml* [40].

Coalgebra theory, born in the late 90s, is thus one of the few major categorical fields in computer science. However, classical coalgebra's power is limited, since it mainly deals with the branching structure of processes. Many modern model checking algorithms manipulate extensively *predicates on the state space*, and even *(alternating) fixed points* as a core technique. For example, this can be used in incremental or approximate methods.

To account for this fact, there are proposals to extend the field of coalgebra with new, more expressive objects, based on the study of *fibrations* ([41], [42], [43], [44], [45]).

In our work – mainly by lack of time – we don't introduce explicitly the formalism of fibrations, even if it can be found implicitly. Formalising the role of fibrations here could simplify our theory and ease its relationship with other work in the coalgebraic field. However, this is left for future work.

1.1.5 Approximate and incremental methods – value iteration

Now that formal methods and foundational theoretical tools have been introduced, let us mention the kind of algorithmic techniques we want to model.

It is increasingly understood that, for the practitioner, it is not always desirable to focus on finding an optimal or exact solution to a problem. In some cases, optimisation or function problems have huge computational complexity, whereas it is possible to find approximate solutions very efficiently.

The theoretical study of approximability and inapproximability is an active field of research. For example, there are NP-hard optimisation problems – such as the knapsack problem – which enjoys Polynomial-Time Approximation Schemes (PTAS), that compute an arbitrarily precise approximation of the optimal solution in polynomial time [46]. On the other hand, inapproximability results can be obtained – a central conjecture is the Unique Game Conjecture [47]. Interestingly, even undecidable problems like the halting problem may be studied from an approximability standpoint [48].

In our case, we are especially interested in iterative incremental methods, the most known of which being *value iteration* (VI) [49]. As explained in Section 4, when the solution of an optimisation problem can be characterised as an extremal fixed point of a monotone function, this provides a VI algorithm, which computes a sequence converging to that solution.

For example, it is possible to compute reachability probabilities in stochastic games – written $V(\mathcal{G})$ – by iterating the *Bellman operator* $\mathcal{B}$. We have $V(\mathcal{G}) = \sup_{n\in\mathbb{N}} \mathcal{B}^n(\bot)$ ([49], [1]), where the sequence $(\mathcal{B}^n(\bot))_{n\in\mathbb{N}}$ is *increasing*, providing *lower bounds* $L_n = \mathcal{B}^n(\bot)$. Notice that $V(\mathcal{G})$ appears as the *least fixed point* (lfp) of $\mathcal{B}$.

$$\bot = L_0 \leq L_1 \leq L_2 \leq \cdots \leq V(\mathcal{G}).$$

For the practitioner, this algorithm has the advantage of being easily tuned for speed versus precision. However, an issue is that, at every step, it is impossible to know how close the current approximation is from the solution. A natural heuristic would be to fix a threshold ϵ and stop the algorithm once $L_{n+1} - L_n < \epsilon$, but this has been shown to provide arbitrarily bad approximations [50].

To solve this problem and provide precision guarantees, *bounded value iteration* (BVI, also called interval iteration) algorithms have been introduced ([50], [51]) Since a monotone operator (here $\mathcal{B}$) is used, we may also compute the *decreasing* sequence of upper bounds $(U_n)_{n\in\mathbb{N}}$ where $U_n := \mathcal{B}^n(\top)$.

$$\begin{tikzcd}[row sep=small]
\bot = L_0 \arrow[r, phantom, "\leq"] & L_1 \arrow[r, phantom, "\leq"] & L_2 \arrow[r, phantom, "\leq"] & \cdots \arrow[dr, phantom, "\leq" sloped] & \\
 & & & & V(\mathcal{G}) \\
\top = U_0 \arrow[r, phantom, "\geq"] & U_1 \arrow[r, phantom, "\geq"] & U_2 \arrow[r, phantom, "\geq"] & \cdots \arrow[ur, phantom, "\geq" sloped] &
\end{tikzcd}$$

An issue is that (U_n) may not converge to $V(\mathcal{G})$, since it converges to the *greatest fixed point* (gfp) of $\mathcal{B}$, where lfp and gfp might differ. It is thus not obvious how to define a good stopping criterion.

$$\begin{tikzcd}
L_n \arrow[r, "n\to\infty"'] & \text{lfp } \mathcal{B} \arrow[d, phantom, "\neq"] \\
U_n \arrow[r, "n\to\infty"] & \text{gfp } \mathcal{B}
\end{tikzcd}$$

In the case of stochastic games, this can be solved by computations of end components ([50], [51]), but this quickly becomes a bottleneck. A novel approach proposed in [1] is to introduce a *surrogate model* to create upper bounds $\tilde{U_n}$ that *do* converge to the lfp $V(\mathcal{G})$.

Moreover, in the general case, *transfinite* value iteration may be needed. This is discussed in Section 4.

1.1.6 Surrogate models

The concept of surrogate model comes from engineering, e.g. in chemistry [52], where there are used for modelisation and simulation.

Surrogate model are *simple* models that mimic the behaviour of complex systems. While developing them can be expensive and requires expertise, they approximate the behaviour of the underlying system to a good precision and are much cheaper to evaluate.

A way to understand this idea is "improve our knowledge about the solution to a difficult problem by solving simple problems".

Even if this idea is very simple and powerful, it is very new to the field of model checking as a design principle.

An early result that may be seen as a simple example of this principle is the fact that many qualitative properties about finite Markov chains can be computed purely by graph analysis, forgetting about transition probabilities [13].

To the best of our knowledge, the only other example is found in the 2020 paper [1], where relations between stochastic games, Markov decision processes, Markov chains and, most interestingly, weighted graphs, are used to create a BVI algorithm for reachability probabilities in stochastic games.

A long-term goal pursued in this internship is to build a good theoretical understanding of this "base model / surrogate model" design principle, with the perspective of finding new instances and creating new model checking algorithms.

1.2 Contributions

Our contributions are the following, the first three being the main ones.

- We propose a generic framework, based on the coalgebraic point of view and on order structures, to study value iteration and surrogate models in processes.

  Those recipes are summarised in Section 6.3.

- We provide a general theorem which packages all assumptions enabling VI algorithms, making them easy to check (Theorems 4.1 and 4.5). In particular, this generalises the result that value iteration solves the reachability problem in all stochastic games.

  The structure enabling VI, which we call *leaf structure*, is understood as the *lifting* of a **Set**-endofunctor to a monotone **Preord**-endofunctor (Definition 3.15).

  We notice (Proposition 4.2) that the least fixed point and greatest fixed point operators are both functorial and monotone.

- We provide an approximation theorem (Theorem 5.22) describing a particular kind of BVI algorithm using surrogate models.

More generally, we study how branching structures impact value functions (Proposition 5.1, Corollary 5.15).

Order structures based on branching structures, which we call *shape structures*, are understood as the *extension* of a **Set**-functor to **Preord** (Definition 3.17).

Moreover, a notion of *connection* and *tight connection* between several process types is studied in Sections 5.2 and 5.3 to gain a first understanding of surrogate models.

- Several instances of the general definitions and results are given throughout the paper, especially in Sections 2.2, 3.1 and 6.
- In the study of preorders, a special place is given to the Egli-Milner and tailwise preorders (Examples 3.18 and 3.19), related to the suprema, infima and expectation modalities. Several monotonicity properties are proven about them.
- Among other things, we give a description of the Bellman transformer as a weakest precondition transformer (Example 3.7).
- We explain why some of our results are optimal by providing counter-examples.
- Two simple examples of *connections* are given, one related to the approximation of a player's strategy in a multiplayer game, the other to the optimal scheduler in infinitely branching transition system (Examples 5.10 and 6.18).

1.3 Overview

This internship report is divided in several sections.

- You are currently in the introduction (Section 1), which discusses the scientific context and our contributions. It also gives an overview of the report and contains acknowledgements.

  Possible lines of research for future work will be discussed in the conclusion (Section 7).
- Section 2 explains how we use the coalgebraic language to describe processes (such as transition systems, automata, Markov chains and stochastic games) in a general way.
  - Important definitions (process types, processes, coalgebra) are in Section 2.1.
  - Intuition and illustrated examples are provided in Section 2.2.
- Section 3 then explains how to turn processes into problems with semantics.
  - The base logical structure consists of predicates, truth objects and weakest precondition transformers, and is explained in Section 3.1.

    This section also gives many examples of what is meant by "a problem". In particular, it explains how the Bellman operator can be seen as a weakest precondition transformer in Example 3.7.
  - Section 3.2 then explains how order structures can be added to process type, and is inspired by other works, including that of Balan and Kurz [64], Jacobs [36] and Hasuo [35].

    To do so, it starts by introducing relevant concepts from order theory and enriched category theory.

    It then describes the two main order structures studied in this report. First, *leaf structures*, that enable monotone predicate transformers, thus paving the way for VI algorithms. Then, *shape structures* that enable comparison of the branching structure of processes, thus paving the way for BVI algorithms based on surrogate models.

    Those two notions are then studied in Sections 4 and 5.
- Section 4 studies more precisely the notion of leaf structure, which gives us our first main result (Theorems 4.1 and 4.5).

    - Section 4.1 explains why leaf structures are interesting.

      When they exist, Knaster-Tarski and Cousot-Cousot theorems apply, thus making fixed point theories possible (Theorem 4.1), even if transfinite iterations may be needed.

      Then, a simple but important functoriality and monotonicity result (Proposition 4.2) is proven. It will be used throughout the paper.

      Finally, the question of convergence speed is studied.
    - Section 4.2 then gives a large class of examples of leaf structures.

      Anything defined on non-deterministic or stochastic branching with modalities based on suprema, infima and expectation can be given a leaf structure by using Egli-Milner and tailwise preorders. (Theorem 4.5).
    - This theorem is proven in detail in Section 4.3.
- Section 5 then studies the notion of shape structure, going towards our second main result (Theorem 5.22).
    - In Section 5.1, a shape-monotonicity results explains how shape structures may be used to compare processes inside a process-type.

      For example, this generalises the result that, in a stochastic game, reachability probabilities decrease if Minimizer loses power.
    - Section 5.2 then bridges the gap between multiple process-types by introducing *connections* (between process-types) and *comparison chains* (between processes of different type).
    - However, connections are not enough to smoothly obtain BVI algorithms. An additional, *tightness* hypothesis is introduced and explained in Section 5.3, and we are finally able to prove Theorem 5.22.
- Finally, Section 6 is devoted to examples, applications, and discusses possible directions that could be pursued to improve our theory.
    - Section 6.1 gives instances of process-predicate types with leaf and shape structures, such as problems on games, transition systems and graphs.
    - Section 6.2 gives examples of connections between process-predicate types with shape structure, thus showing to which extent our theory can go to accommodate base model / surrogate model relationships.
    - Section 6.3 takes a step back, and sums up how our theory can be used by the practitioner: it provides recipes for VI and BVI algorithms.
    - Finally, Section 6.4 discusses how we could extend the scope of our work, by accommodating new, more interesting examples of surrogate models.

1.4 Acknowledgements

First and foremost, I would like to thank my internship advisers, Ichiro Hasuo and Jérémy Dubut. Thank you both for all your constant support and remarks. Even though a global pandemic made it impossible to come to Japan, I hope to be able to work with you again and finally meet you in person.

Thanks Ichiro for making all of this possible, for welcoming me at Erato MMSD, for giving the opportunity to learn and explore a great subject, and for sharing your insights both into various fields of computer science and on managing a diverse and vibrant research team. Thanks for making sure that I could make the most out of my experience despite the time difference. I learned and explored a lot while working under your supervision.

Thanks Jérémy for being always there, for your pedagogy and for our great mathematical discussions, either on intricate conjectures encountered during my internship or on exciting work that you made me discover. It was always fun and helpful to speak with you.

I would also like to thank Samuel Mimram, for making me discover Erato MMSD, and Éric Goubault, for giving references and advice towards this internship. More generally, I would like to thank École polytechnique and the computer science department for creating the conditions making this internship possible.

From Erato MMSD, I would like to thank the whole team for being so welcoming. In particular, thanks to Natsuki Urabe for his help with alternating fixed points and for telling me about various applications of the theory studied in the internship. Thanks to the whole G0 group for their warm welcome and seminar. Thanks to the G1 group, and in particular to Toru Takisaka, for inviting me to discuss my master's project on tropical abstract interpretation at the group's seminar. Thanks to everyone in the student reading group on enriched category theory and in the model checking group for inviting to participate in their activities and to learn with them. Thanks to all of the administrative team for helping me through the process of trying to reach Japan and attending LICS. Finally, a huge thanks to the organisers of the online social events, and in particular to Stefan Klikovits, for making sure that informal discussions took place despite the pandemic, and for their amazing welcome party!

To conclude I would like to thank my jury, Olivier Bournez and François Fages, for taking the time to study this report and helping me progress.

2 Processes

In computer science in general, and in model checking in particular, many kinds of processes are of interest: (in)finite transition systems, (non)deterministic automata, possibly non-terminating computations, multiplayer games, Markov chains, Markov decisions processes, stochastic games, etc. The theory of coalgebras ([53], [54], [32]) makes it possible to deal with such processes in a generic, categorical way.

The branching structure (non-deterministic, probabilistic, etc.) of such processes is encoded in what we call *process types*, which we introduce in this section.

In the rest of the paper, we will put additional structure on process types to deal with modern model checking techniques. Using logical structure and order structure, we will express predicates on the state space of processes, give characterisations as fixed point problems, and model processes by simpler kinds of processes.

$$\begin{tikzcd}
\text{Order} & & \text{Leaf structure} & & \text{Shape structure} \\
\text{Logic} \arrow[u] & & & \text{Process-predicate type} \arrow[ull] \arrow[ur] & \\
\text{Branching} \arrow[u] & & & \text{Process type} \arrow[u] &
\end{tikzcd}$$

The branching strucure corresponds to the process-type. Logical and order structures are put *above* the branching structure. Logical structure will turn a *process* (which has a process type) into a *problem* (which has a process-predicate type) by adding semantics. We will see two kinds of order structure put *above* the logical structure. The first (*leaf structure*) mixes branching and logical structures. The second (*shape structure*) deals only with the branching structure (*shape structure*). In the next sections, we will see that leaf and shape structures interact together.

2.1 Main definitions

Definition 2.1 (Process type)**.** A *process type* $T : \mathbf{Set} \to \mathbf{Set}$ is simply a **Set** endofunctor.

Definition 2.2 (Processes and coalgebra)**.**

A *process* is a function $g : X \to TY$ where T is a process type.

More precisely, it is a tuple $(T : \mathbf{Set} \to \mathbf{Set}, X : \mathbf{Set}, Y : \mathbf{Set}, g : \mathrm{Hom}(X, TY))$.

A *coalgebra* (or *endoprocess*) is a process where $X = Y$.

When $T : \mathbf{Set} \to \mathbf{Set}$ is a process type, the *category of T-processes* $\mathrm{Pr}(T)$ is the category whose objects are processes of type T and whose arrows between $g : X \to TY$ and $g' : X' \to TY'$ are all pairs of functions (φ, ψ) making the following diagram commute.

$$\begin{tikzcd}
X \arrow[r, "g"] \arrow[d, "\varphi"] & TY \arrow[d, "T\psi"] \\
X' \arrow[r, "g'"] & TY'
\end{tikzcd}$$

The *category of T-coalgebras* (or T-endoprocesses) $\mathrm{Coalg}(T)$ is the (neither full nor wide) subcategory of $\mathrm{Pr}(T)$ whose objects are coalgebras on T and whose arrows between $g : X \to TX$ and $g' : X' \to TX'$ are all functions φ making the following diagram commute.

$$\begin{tikzcd}
X \arrow[r, "g"] \arrow[d, "\varphi"] & TX \arrow[d, "T\varphi"] \\
X' \arrow[r, "g'"] & TX'
\end{tikzcd}$$

Remark 2.3 (Base category)**.** Note that we are restricting ourselves to the category **Set**.

We may restrict further to the category of finite sets, $\mathbf{Set}_{\text{fin}}$, and define *finite process types* $T : \mathbf{Set}_{\text{fin}} \to \mathbf{Set}_{\text{fin}}$, *finite processes* $g : X \to TY$ with $T : \mathbf{Set}_{\text{fin}} \to \mathbf{Set}_{\text{fin}}$, $X, Y : \mathbf{Set}_{\text{fin}}$, and *finite coalgebras* $g : X \to TX$ with $T : \mathbf{Set}_{\text{fin}} \to \mathbf{Set}_{\text{fin}}$, $X : \mathbf{Set}_{\text{fin}}$. Finite processes can be seen as processes on finite state space.

In other sections, we will add order structure to process types. Using the category **Preord** of preorders and monotone functions, instead of the family $\mathrm{Hom}(\mathbf{Set}, \mathbf{Set})$ of **Set**-endofunctors, we will consider the family $\mathrm{Hom}_{\mathbf{Preord}}(\mathbf{Preord}, \mathbf{Preord})$ of monotone **Preord**-endofunctors, where "monotone" means that we restrict ourselves to **Preord**-enriched functors.

More generally, we could study processes with type constructed on an arbitrary category, but this is outside the scope of this paper, and all of our examples of process types come from **Set**.

The coalgebraic language of processes can be used to present and unify many results in automata theory, game theory, etc. Before going further, we will describe how this formal notion of "processes" can be interpreted.

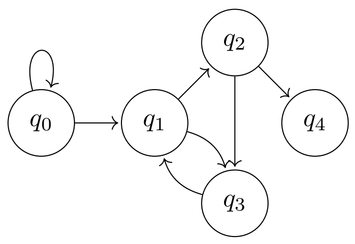
2.2 Intuition and examples

Example 2.4 (Intuition about processes)**.** Let $g : X \to TY$ be a process.

X can be interpreted as the initial state space and Y as the final state space. g describes possible transitions and observations. If x is the current state, $g(x)$ is the "tree" of all properties locally observable (attributes of the current state and possible next states). Coalgebras (endoprocesses) are simply processes with constant state space.

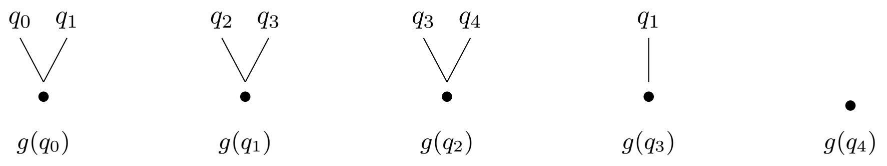
For example, directed graphs can be seen as coalgebras on the covariant powerset functor $\mathcal{P}$.

The following directed graph, on the left, would be represented as the following coalgebra, on the right.

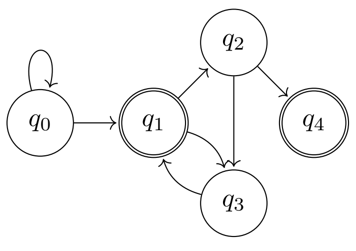


$$\begin{aligned} g : \{q_0, q_1, q_2, q_3, q_4\} &\to \mathcal{P}\{q_0, q_1, q_2, q_3, q_4\} \\ q_0 &\mapsto \{q_0, q_1\} \\ q_1 &\mapsto \{q_2, q_3\} \\ q_2 &\mapsto \{q_3, q_4\} \\ q_3 &\mapsto \{q_1\} \\ q_4 &\mapsto \varnothing \end{aligned}$$

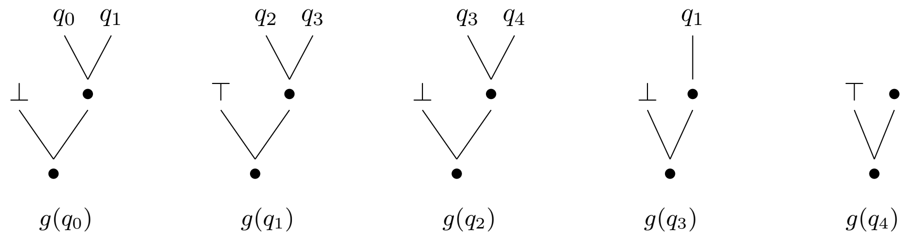
This can also be presented with trees.



We can "enrich the signature" to observe new phenomena. For example, to distinguish accepting states, we can replace $\mathcal{P}$ by $2 \times \mathcal{P}(-)$. Here are three representations of a coalgebra on this functor.



$$
\begin{aligned}
g : \{q_0, q_1, q_2, q_3, q_4\} &\to 2 \times \mathcal{P}\{q_0, q_1, q_2, q_3, q_4\} \\
q_0 &\mapsto (\bot, \{q_0, q_1\}) \\
q_1 &\mapsto (\top, \{q_2, q_3\}) \\
q_2 &\mapsto (\bot, \{q_3, q_4\}) \\
q_3 &\mapsto (\bot, \{q_1\}) \\
q_4 &\mapsto (\top, \varnothing)
\end{aligned}
$$



Remark 2.5 (Algebras and coalgebras)**.** You may be starting to notice how convenient endofunctors are to describe the “signature” of processes. The situation is similar to that of algebras for inductive data.

For $T : \mathbf{Set} \to \mathbf{Set}$, T-coalgebras are $g : X \to TX$, while T-algebras are $\alpha : TX \to X$.

The distinction *algebras versus coalgebras* may be understood as the distinction *data versus machines*, or *construction versus observation*, or (definitions and proofs by) *induction versus coinduction*.

This idea is presented in [54]. For our current purposes, the main insight to remember is that coalgebraic objects have several *observers* (or *destructors*), in the same way as algebraic objects have multiple *constructors*.

The 2000 expository paper “Universal coalgebra: a theory of systems” [53] provides a great introduction to the language of coalgebras, which unites the study of processes in computer science. This theory enjoyed many successes over the years, some of which stemming from its approach of coinduction, bisimulation and process minimisation.

To highlight the generality of this theory of processes, here are a few examples of process types.

Example 2.6.

- Traditionally, (unlabelled) non-deterministic transition systems are described by a tuple $(S : \mathbf{Set}, \to : \mathcal{P}(S \times S))$. As we have already seen, NTS (or graphs, or Kripke structures) can be equivalently seen as coalgebras of the covariant powerset functor $\mathcal{P}$.
- Similarly, we choose to see NTS labelled on the alphabet Σ as coalgebras of process type $\mathcal{P}(\Sigma \times -)$. Recall that traditionally, they were represented by a tuple $(S : \mathbf{Set}, \Sigma : \mathbf{Set}, \to : \mathcal{P}(S \times \Sigma \times S))$.
- If, like in many textbooks, we want to see Kripke structures as NTS where states are labelled with atomic propositions from a set AP, we can define Kripke structures as coalgebras on $\mathcal{P}(AP) \times \mathcal{P}(-)$. Adding $\mathcal{P}(AP) \times \cdot$ to a process type is the most general way of dealing with *state attributes*. We already encountered the case $\mathrm{Card}(AP) = 1$ when we wanted to distinguish *goal states*, and described NTS with goal states as coalgebras on $2 \times \mathcal{P}(-)$.
- Various restrictions can be encoded in the process type. NTS that are only *finitely branching* are modelled by the *finite* powerset functor $\mathcal{P}_{\mathrm{f}}$. NTS without deadlocks are modelled by the non-empty powerset functor $\mathcal{P}_{\neq\varnothing}$.
- Deterministic (unlabelled) transition systems (DTS) are simply coalgebras on the identity functor of **Set**.
- Markov Chains (MC) are modelled by the distribution functor $\mathcal{D}$. The definition of $\mathcal{D}$ is given below at Definition 2.8, and Example 2.9 is an example of MC.
- Possibly non-terminating computations, or (unlabelled) DTS with possible deadlock, can be seen as coalgebras on $\mathcal{L} : X \mapsto 1 + X$.

  This illustrates how coproducts can be used to model different possibilities in branching.
- Labelled DTS with possible deadlock can be modelled on $1 + \Sigma \times (-)$. An example of coalgebra on this type is $\gamma : \Sigma^\infty \to 1 + \Sigma \times \Sigma^\infty$, where Σ^∞ is the set of finite and infinite strings on Σ and γ takes an empty word to the first $*$ and a non-empty word to the tuple given by its head and tail.
- Composition is useful as well: (discrete) Markov Decision Processes (MDP) can be modelled by $2 \times \mathcal{P}\mathcal{D}(-)$, where both non-deterministic and stochastic branching are observed, and we distinguish goals.
- Alternating stochastic games (ASG) can be modelled by $2 \times \mathcal{P}\mathcal{D}\mathcal{P}\mathcal{D}(-)$.
- Stochastic games (SG) can be modelled by $2 \times 2 \times \mathcal{P}\mathcal{D}(-)$, where the first attribute distinguishes between the two players, and the second attribute distinguishes between goal and non-goal states.
- For a more complete example, consider deterministic finite automata (DFA), on the finite alphabet Σ, with possible deadlocks and distinguished accepting states (but no distinguished initial states) can be modelled by the $\mathbf{Set}_{\mathrm{fin}}$-functor $X \mapsto 2 \times (1 + X)^\Sigma$.

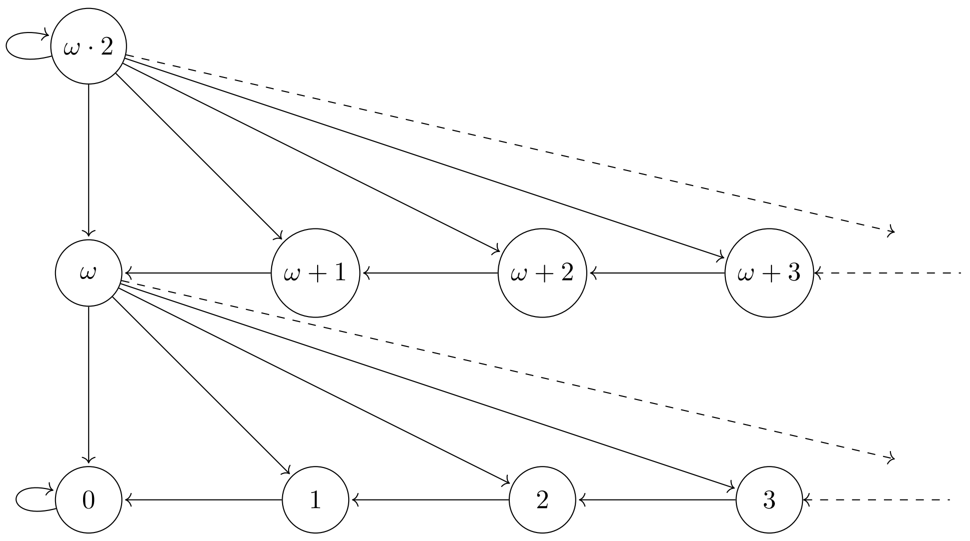
  Obviously, if we remove the finiteness hypothesis of the state space for DFA, we can simply use the corresponding **Set**-functor $2 \times \mathrm{Hom}(\Sigma, 1 + (-))$.
- Similarly, for non-deterministic finite automata (NFA), we can use the $\mathbf{Set}_{\mathrm{fin}}$-functor $2 \times \mathcal{P}(\Sigma \times (-))$.

  If we remove the finiteness hypothesis, we can use the corresponding **Set**-functor. If we want infinite state spaces but only finite branching, we can use $2 \times \mathcal{P}_{\mathrm{f}}(\Sigma \times (-))$.
- Deterministic and non-deterministic (infinite) automata which have both input and *output* in Σ can be modelled by $2 \times \mathrm{Hom}(\Sigma, 1 + (- \times \Sigma))$ and $2 \times \mathcal{P}(\Sigma \times (-) \times \Sigma)$ respectively.
- Weighted directed graphs (WG) with weights in a monoid W, modelled on the *covariant* functor $\mathrm{Hom}_{\mathrm{fin}}(-, W)$ of Definition 2.10, can be seen as generalisations of MP modelled on $\mathcal{D}$.

  Note that WG can also be seen as NTS with labelled transition, i.e. as coalgebras on $\mathcal{P}(W \times -)$. Those two point of view are very different! It can be seen by the fact that there is no *natural* isomorphism between $\mathrm{Hom}_{\mathrm{fin}}(-, W)$ and $\mathcal{P}(W \times -)$. Transitions can be summed and related together in the first case, while no such structure naturally exists in the second case, making transitions totally independent of each other.

To end this section, we develop a few of those examples, which will be used again later.

Example 2.7. Consider the following decision process (or unlabelled NTS, or Kripke structure), which has an infinite state space.



It can be described as a coalgebra $g : X \to \mathcal{P}X$ on the powerset functor, with $X = [0, \omega \cdot 2]$ the set of ordinals smaller or equal than $\omega \cdot 2$, by setting $g(0) = \{0\}$, $g(\omega) = [0, \omega)$, $g(\omega \cdot 2) = [\omega, \omega \cdot 2]$, and, for all $n < \omega$, $g(n + 1) = \{n\}$, $g(\omega + n + 1) = \{\omega + n\}$.

A similar example will be used later. Notice that 0 is *reachable* from all states, and that all executions starting at a state $\alpha < \omega \cdot 2$ *eventually reach* 0 in a finite number of steps. However, not all executions starting at $\omega \cdot 2$ eventually reach 0, since it is possible to loop at $\omega \cdot 2$ forever.

Definition 2.8 (Distribution functor)**.** To describe stochastic branching, we use to distribution functor $\mathcal{D}$, which assigns to a set X the set of all probability distributions on X with finite support.

An element $d : \mathcal{D}X$ can be described as a function $d : X \to [0, 1]$, with finite support $d^{-1}((0, 1])$, such that $\sum_{x \in X} d(x) = 1$. It may also be represented as a finite set of tuples $\{(x, d(x)) \mid d(x) > 0\}$, which enjoys that property that if $(x, a), (x, b)$ both belong to this set then $a = b$.

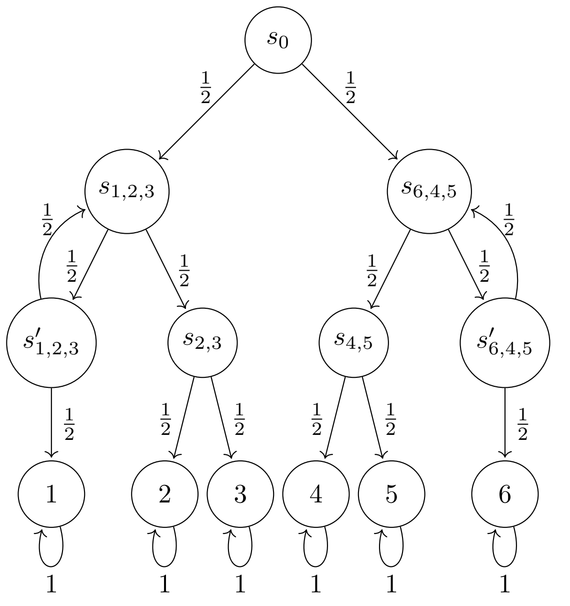
The functor $\mathcal{D}$ takes functions $f : X \to Y$ to

$$\begin{aligned} \mathcal{D}f : \mathcal{D}X &\to \mathcal{D}Y \\ d &\mapsto \Big(y \mapsto \sum_{x \in f^{-1}(y)} d(x)\Big). \end{aligned}$$

Example 2.9 (A Markov Chain)**.**

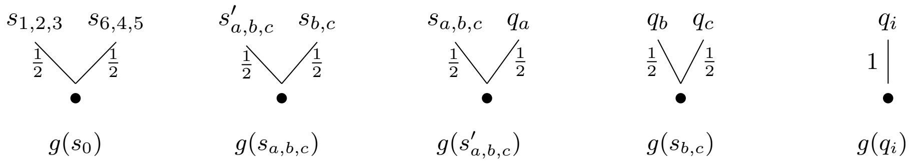
An elementary type of processes with stochastic branching is the type of Markov Chains.

The following Markov chain, presented by Baier and Katoen in their *Principles of model checking* [13], is attributed to Knuth and Yao and simulates dice with a fair coin. We present it visually as a graph, as an explicit $\mathcal{D}$-coalgebra, and with a tree representation of the coalgebra.



$$X := \{s_0, s_{1,2.3}, s'_{1,2,3}, s_{2,3}, s_{6,4,5}, s_{4,5}, s'_{6,4,5}, \\ q_1, q_2, q_3, q_4, q_5, q_6\}$$

$$\begin{aligned} g : X &\to \mathcal{D}X \\ s_0 &\mapsto \{(s_{1,2,3}, 0.5), (s_{4,5,6}, 0.5)\} \\ s_{a,b,c} &\mapsto \{(s'_{a,b,c}, 0.5), (s_{b,c}, 0.5)\} \\ s'_{a,b,c} &\mapsto \{(s_{a,b,c}, 0.5), (q_a, 0.5)\} \\ s_{b,c} &\mapsto \{(q_b, 0.5), (q_c, 0.5)\} \\ q_i &\mapsto \{(q_i, 1)\} \end{aligned}$$



The following definition is non-trivial, and hides additive structure in the functor's behaviour on arrows. Moreover, it *does* define a *covariant* functor (whereas the usual $\mathrm{Hom}(-, X)$ is a *contravariant* functor).

Definition 2.10. We can generalise the distribution functor $\mathcal{D}$ by considering weights in an arbitrary monoid $(W, +)$ and removing the requirement $\sum_x d(x) = 1$.

Let $(W, +)$ be a monoid with identity element 0. We define the *covariant* functor

$$\mathrm{Hom}_{\mathrm{fin}}(-, W) : \mathbf{Set} \to \mathbf{Set}.$$

It takes sets X to the set of all functions $d : X \to W$ such that $d^{-1}(W \setminus \{e\})$ is finite, i.e.

$$\mathrm{Hom}_{\mathrm{fin}}(X, W) = \{d : X \to W \mid d^{-1}(W \setminus \{e\}) \text{ is finite}\},$$

and takes functions $f : X \to Y$ to

$$\begin{aligned} \mathrm{Hom}_{\mathrm{fin}}(f, W) : \mathrm{Hom}_{\mathrm{fin}}(X, W) &\to \mathrm{Hom}_{\mathrm{fin}}(Y, W) \\ d &\mapsto \Big(y \mapsto \sum_{x \in f^{-1}(y)} d(x)\Big). \end{aligned}$$

Remark 2.11. Note that elements $d : \mathrm{Hom}_{\mathrm{fin}}(X, W)$ can be represented as element of $\mathcal{P}_{\mathrm{f}}(W \times X)$, but that this representation is not natural.

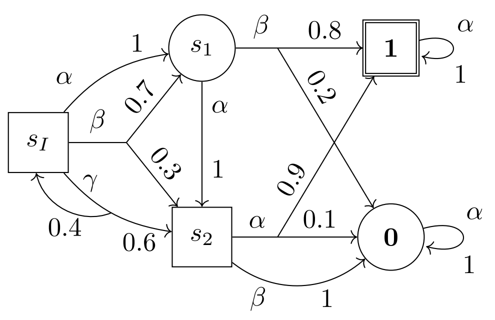
$\mathcal{D}$ is a subfunctor of $\mathrm{Hom}_{\mathrm{fin}}(-, (\mathbb{R}_+, +))$, with the additional requirement that distributions take weights in $[0, 1] \subseteq \mathbb{R}_+$ and have the sum of their values equal to 1. A generalisation of the probabilistic framework to other *effect monoids* can be found in [55].

In this way, Markov chains ($\mathrm{Coalg}(\mathcal{D})$) can be seen as weighted graphs ($\mathrm{Coalg}(\mathrm{Hom}_{\mathrm{fin}}(-, W))$) with additional requirements.

Example 2.12. An application we are especially interested in is that of *stochastic games*. In that case, there is both non-deterministic branching and stochastic branching, and two attributes "current player" and "goal".

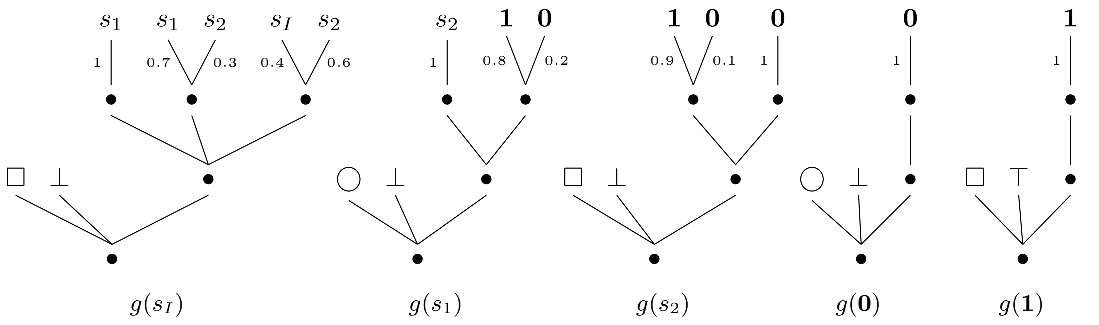
Stochastic games can be described as coalgebras on the functor $2 \times 2 \times \mathcal{P}\mathcal{D}(-)$. Such games are interpreted as 2-player games, where the players are called Maximizer and Minimizer. Each state belongs to a single player, and the next move is played by the player designated by the current state. At each turn, a player chooses an action, which has a random outcome, with probabilities known in advance. Minimizer tries to minimise the probability of reaching a goal state, and Maximizer tries to maximise this probability. In a *play* (finite or infinite sequence of states),

- Maximizer wins if a goal state is reached or Minimizer enters a deadlock,
- Minimizer wins if goal states are never reached or if Maximizer enters a deadlock before the goal.



$$X := \{s_I, s_1, s_2, \mathbf{0}, \mathbf{1}\}$$

$$\begin{aligned}
g : X &\to 2 \times 2 \times \mathcal{PD}X \\
s_I &\mapsto \Big(\square, \bot, \big\{\{(s_1, 1)\}, \\
&\qquad \{(s_1, 0.7), (s_2, 0.3)\}, \{(s_I, 0.4), (s_2, 0.6)\}\big\}\Big) \\
s_1 &\mapsto \Big(\bigcirc, \bot, \big\{\{(s_2, 1)\}, \{(\mathbf{1}, 0.8), (\mathbf{0}, 0.2)\}\big\}\Big) \\
s_2 &\mapsto \Big(\square, \bot, \big\{\{(\mathbf{1}, 0.9), (\mathbf{0}, 0.1)\}, \{(\mathbf{0}, 1)\}, \big\}\Big) \\
\mathbf{0} &\mapsto \Big(\bigcirc, \bot, \big\{\{(\mathbf{0}, 1)\}\big\}\Big) \\
\mathbf{1} &\mapsto \Big(\square, \top, \big\{\{(\mathbf{1}, 1)\}\big\}\Big)
\end{aligned}$$



Here is a possible run of this game. We start at s_I. Maximizer ($\square$) plays β, and reaches s_1 (this outcome had probability 0.7). It is now Minimizer's ($\bigcirc$) turn. He plays α, and reaches s_2. $\square$ then plays α, and reaches $\mathbf{0}$ (this had only probability 0.1). From then on, $\bigcirc$ is the only player and can manage to stay at $\mathbf{0}$ forever, which is not a goal state: *Maximizer has lost.*

Another possible run loops forever at s_I: in that case, *Maximizer loses* as well. This may happen if Maximizer chooses the strategy "play γ as long as possible", but only with probability 0.

One interesting question about this game is the following: *assuming that players play optimally*, if we start at state x, what is the probability that Maximizer will win? In other words and in the language of modal logic, can we compute $\mathbb{P}(\Diamond\mathbf{1}) : X \to [0,1]$?

In the next section, we will study generalisations of this question.

At this point of the paper, we will only state that $\mathbb{P}(\Diamond\mathbf{1})$ can be characterised as the least fixed point of the *Bellman operator* $\mathcal{B} : [0,1]^X \to [0,1]^X$.

The interested reader may prove by hand that $\mathbb{P}(\Diamond\mathbf{1})(\mathbf{0}) = 0$, $\mathbb{P}(\Diamond\mathbf{1})(\mathbf{1}) = 1$, $\mathbb{P}(\Diamond\mathbf{1})(s_2) = 0.9$, $\mathbb{P}(\Diamond\mathbf{1})(s_1) = 0.8$ and $\mathbb{P}(\Diamond\mathbf{1})(s_I) = 0.9$. The situation becomes much harder when we increase the number of possible "loops" between several states.

Remark 2.13. It is interesting to note that the study of processes with both non-determinism and stochasticity is especially hard. A theoretical reason that explains this difficulty is that while $\mathcal{P}$ and $\mathcal{D}$ have monadic structure, there is no monadic structure on the combination $\mathcal{PD}$ [56]. A monadic structure exists on the combination of the multiset functor and distribution functor [57], but applications have not yet been extensively studied.

In the current paper, we don't use any sort on monadic structure of process types $T : \mathbf{Set} \to \mathbf{Set}$. They are however extensively used in the literature, especially in the study of *program semantics* [58]. A big difficulty is that while monadic structures can be summed or tensored, they cannot be composed in general, and this fact has been the motivation of much research [59], [57], [60], [61].

Nevertheless, useful structure arises from monadicity of process types. For example, when (T, η, μ) is a monad, processes $f : X \to TY$ and $g : Y \to TZ$ can be *composed* to $g \odot f := \mu \circ Tg \circ f : X \to TZ$, giving rise to the *Kleisli category* $\mathcal{K}\ell(T)$. This may also provide fruitful categorical insight, such as in [35], whose ideas about predicate transformers and order enrichment are one of the roots of the current paper.

3 Predicates on processes and preorder-enrichments

In the preceding section, we have introduced the notions of *process type* and *coalgebras*, to model various kind of processes, and gave examples such as transition systems (TS), automata, Markov chains (MC), Markov decision processes (MDP) and stochastic games (SG).

In this section, we introduce the necessary additional structure to explore questions about processes. *Process-predicate types* will contain logical structure to talk about predicates on processes, modalities and weakest-precondition predicate transformers. We will then study how order structures can be introduced. This investigation will lead to fixed point characterisations and approximations of process types in Sections 4 and 5.

3.1 Logic on processes

For the moment, process types are only defined by transitions $g : X \to TY$, with state spaces and some sort of constant attributes and branching type. In our last example on stochastic games $g : X \to 2 \times 2 \times \mathcal{PD}X$, we were interested in the study of a *predicate* $\mathbb{P}(\Diamond \mathbf{1}) : X \to [0,1]$, which described a reachability probability under some hypotheses on the behaviour of players. More generally, we are interested in asking questions of reachability, safety, distance, etc. in our models. To do this, we introduce *process-predicate types.*

Definition 3.1 (Process-predicate type)**.** A *process-predicate type* (ppt) $\mathcal{T} := (T, \Omega, \tau)$ is a triple composed of a *process type* $T : \mathbf{Set} \to \mathbf{Set}$, a *truth object* $\Omega : \mathbf{Set}$ and a *modality* $\tau : \mathrm{Hom}(T\Omega, \Omega)$.

Modalities are used to state a weakest-precondition semantics.

Definition 3.2 (Predicates and predicate transformers)**.** Let $\mathcal{T} = (T, \Omega, \tau)$ be a ppt.

For all sets X, Y, elements of the form $p : X \to \Omega$ are called *predicates*, and elements of the form $\Phi : \Omega^Y \to \Omega^X$ are called *predicate transformers.*

Definition 3.3 (Weakest-precondition transformer)**.** Let $\mathcal{T} = (T, \Omega, \tau)$ be a ppt and $g : X \to TY$ be a T-process.

The *weakest-precondition transformer* corresponding to g under the modality τ is the predicate transformer

$$\begin{aligned} g^*_\tau : \Omega^Y &\to \Omega^X \\ p &\mapsto \tau \circ Tp \circ g. \end{aligned}$$

Process-predicate types may be thought of as a basic structure (a process type encoding attributes and branching) additionally equipped with a *semantics* (giving meaning to attributes and to notions like reachability, possibility, behaviour of players, interpretation of randomness, etc.).

We will now show how those definitions are used on simple examples.

Example 3.4 (Reachability and safety in Kripke Structures)**.** Consider the following Kripke structure, defined as a $\mathcal{P}$-coalgebra by $X = [0, \omega \cdot 2]$, $g(0) = \{0\}$, $g(\omega) = [0, \omega)$, $g(\omega \cdot 2) = [\omega, \omega \cdot 2]$, and, for all $n < \omega$, $g(n+1) = \{n\}$, $g(\omega + n + 1) = \{\omega + n\}$.

$$\begin{tikzcd}
\omega\cdot 2 \arrow[loop left] \arrow[d] \arrow[dr] \arrow[drr] \arrow[drrr] \arrow[drrrr, dashed] & & & & \\
\omega \arrow[d] \arrow[dr] \arrow[drr] \arrow[drrr] \arrow[drrrr, dashed] & \omega+1 \arrow[l] & \omega+2 \arrow[l] & \omega+3 \arrow[l] & {} \arrow[l, dashed] \\
0 \arrow[loop left] & 1 \arrow[l] & 2 \arrow[l] & 3 \arrow[l] & {} \arrow[l, dashed]
\end{tikzcd}$$

Let $\Omega = \{\bot, \top\}$ be the set of booleans, and define the modalities *may* $\tau_\exists$ and *must* $\tau_\forall$ by

$$\begin{aligned} \tau_\exists : \mathcal{P}\Omega &\to \Omega \\ \varnothing, \{\bot\} &\mapsto \bot \\ \{\top\}, \{\bot, \top\} &\mapsto \top \end{aligned} \qquad\qquad \begin{aligned} \tau_\forall : \mathcal{P}\Omega &\to \Omega \\ \varnothing, \{\top\} &\mapsto \top \\ \{\bot\}, \{\bot, \top\} &\mapsto \bot. \end{aligned}$$

This thus defines two process-predicate types: $\mathcal{P}_\exists = (\mathcal{P}, \mathrm{Bool}, \tau_\exists)$ and $\mathcal{P}_\forall = (\mathcal{P}, \mathrm{Bool}, \tau_\forall)$.

Suppose that $p : X \to \Omega$ in the indicator function of a set $A \subseteq X$. $g^*_{\tau_\exists}(p)$ is then the indicator function of the set of states from which *A is reachable in one step*, while $g^*_{\tau_\forall}(p)$ is the indicator function of state which *must enter A in the next step or reach a deadlock*.

For example, if $\mathbf{1}_{\{0\}} : X \to \Omega$ is the indicator function of $\{0\}$, $g^*_{\tau_\exists}(\mathbf{1}_{\{0\}}) = \mathbf{1}_{\{0,1,\omega\}}$, since 0 is accessible from all of those three states. However, $g^*_{\tau_\forall}(\mathbf{1}_{\{0\}}) = \mathbf{1}_{\{0,1\}}$, since it is possible to avoid 0 from ω. To describe accessibility in several steps, we can iterate g^*_τ. For example, $(g^*_{\tau_\exists})^n(\mathbf{1}_{\{0\}})$ is the set of states from which 0 is reachable in exactly n steps.

If we consider paths of arbitrary length, it is clear that 0 is actually reachable from all states, and that paths starting at all states except $\omega \cdot 2$ must eventually reach 0. We will understand this in Section 3.2 by doing *transfinite iterations* of g^*_τ and studying *fixed points* of g^*_τ.

Remark 3.5 (Order structure of booleans)**.** Notice that if we see $\Omega = \{\bot, \top\}$ as the complete lattice with $\bot < \top$, then $\tau_\exists$ and $\tau_\forall$ are simply sup and inf, respectively.

Example 3.6 (An example on Markov Chains)**.** Another simple example is the process-predicate type of Markov chains with probabilities and expectations $(\mathcal{D}, [0,1], \mathbb{E})$, where

$$\begin{aligned} \mathbb{E} : \mathcal{D}[0,1] &\to [0,1] \\ d &\mapsto \sum_{p \in [0,1]} p \cdot d(p). \end{aligned}$$

For example, consider the following Markov chain.

$$\begin{tikzcd}
& & & s_0 \arrow[dll, "\frac{1}{2}"'] \arrow[drrr, "\frac{1}{2}"] & & & & \\
& s_{1,2,3} \arrow[dl, "\frac{1}{2}"'] \arrow[dr, "\frac{1}{2}"] & & & & & s_{6,4,5} \arrow[dl, "\frac{1}{2}"'] \arrow[dr, "\frac{1}{2}"] & \\
s'_{1,2,3} \arrow[ur, bend left, "\frac{1}{2}"] \arrow[d, "\frac{1}{2}"] & & s_{2,3} \arrow[dl, "\frac{1}{2}"'] \arrow[dr, "\frac{1}{2}"] & & & s_{4,5} \arrow[dl, "\frac{1}{2}"'] \arrow[dr, "\frac{1}{2}"] & & s'_{6,4,5} \arrow[ul, bend right, "\frac{1}{2}"'] \arrow[d, "\frac{1}{2}"] \\
1 \arrow[loop below, "1"] & 2 \arrow[loop below, "1"] & & 3 \arrow[loop below, "1"] & 4 \arrow[loop below, "1"] & & 5 \arrow[loop below, "1"] & 6 \arrow[loop below, "1"]
\end{tikzcd}$$

Suppose that we want to compute the likeliness to reach an even number from each state.

We may start by setting $p : X \to [0,1]$ to be the indicator function of $\{q_2, q_4, q_6\}$. p is the probability to reach an even number in 0 steps.

$(g^*_{\mathbb{E}})^n(p)(x)$ is then the probability of reaching an even number within n steps when we start from x. We compute it in the following table.

$(g^*_{\mathbb{E}})^n(p)(x)$	q_1	q_2	q_3	q_4	q_5	q_6	$s'_{1,2,3}$	$s_{2,3}$	$s_{4,5}$	$s'_{6,4,5}$	$s_{1,2,3}$	$s_{6,4,5}$	s_0
0	0	1	0	1	0	1	0	0	0	0	0	0	0
1	0	1	0	1	0	1	0	1/2	1/2	1/2	0	0	0
2	0	1	0	1	0	1	0	1/2	1/2	1/2	1/4	1/2	0
3	0	1	0	1	0	1	1/8	1/2	1/2	3/4	1/4	1/2	3/8
4	0	1	0	1	0	1	1/8	1/2	1/2	3/4	5/16	5/8	3/8
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
ω	0	1	0	1	0	1	1/6	1/2	1/2	5/6	1/3	2/3	1/2
$\omega+1$	0	1	0	1	0	1	1/6	1/2	1/2	5/6	1/3	2/3	1/2

Notice that this *local* computation converges by increasing to the correct probabilities at infinity, and that the solution is a fixed point of $g^*_{\mathbb{E}}$.

In Section 3.2, we will make sense of the notation $(g^*_{\mathbb{E}})^\omega(p)$. Later, we will prove that the chain $((g^*_{\mathbb{E}})^\alpha)_{\alpha:\mathbf{Ord}}$, indexed by ordinals, always converges to the correct probability, even for infinite state spaces.

At this point, we have actually already encountered all three basic modalities that will be the main building blocks of all modalities used in this paper, through the examples of the process-predicate types $(\mathcal{P}, \mathrm{Bool}, \sup)$, $(\mathcal{P}, \mathrm{Bool}, \inf)$ and $(\mathcal{D}, [0,1], \mathbb{E})$.

We will now mix those modalities together, explain what we have asserted in the last section. We said that reachability in stochastic games could be computed using the Bellman operator $\mathcal{B}$. We will now explain how $\mathcal{B}$ arises as the weakest-precondition transformer defined by a well-chosen modality.

Example 3.7 (Bellman operator)**.** Stochastic games with distinguished goals can be seen as coalgebras on $2 \times 2 \times \mathcal{PD}$. Since we want to compute probabilities, we will use the truth object $\Omega = [0,1]$.

Since players want to optimise the expectation of the outcome of their actions, we will build $\mathcal{B}$ out of the modalities $\mathbb{E} : \mathcal{D}[0,1] \to [0,1]$, $\inf : \mathcal{P}[0,1] \to [0,1]$, and $\sup : \mathcal{P}[0,1] \to [0,1]$. To take into account distinguished goals, we also use the constant modality $\tau_{\mathbf{1}} : 2 \to [0,1]$ defined by $\tau_{\mathbf{1}}(\bot) = 0$ and $\tau_{\mathbf{1}}(\top) = 1$. The Bellman modality can then be defined as

$$\begin{aligned} \beta : 2 \times 2 \times \mathcal{PD}[0,1] &\to [0,1] \\ (\cdot, \top, \cdot) &\mapsto 1 \\ (\square, \bot, t) &\mapsto \sup_{d \in t} \mathbb{E}(d) \\ (\bigcirc, \bot, t) &\mapsto \inf_{d \in t} \mathbb{E}(d). \end{aligned}$$

For a stochastic game $g : X \to 2 \times 2 \times \mathcal{PD}X$, the weakest-precondition transformer g^*_β is then the usual Bellman operator $\mathcal{B}$, where $\delta(x, a, y)$ denotes the transition probability from x to y when action a is chosen.

$$\begin{aligned} \mathcal{B} = g^*_\beta &: [0,1]^X \to [0,1]^X \\ & p \mapsto \left(x \mapsto \begin{cases} 1 & \text{if } x \text{ is a goal state} \\ \sup_a \sum_y \delta(x,a,y) p(y) & \text{if } x \text{ belongs to Maximizer} \\ \inf_a \sum_y \delta(x,a,y) p(y) & \text{if } x \text{ belongs to Minimizer} \end{cases} \right) \end{aligned}$$

Note that "reader-monad-like notations", with *letters a* used in transitions, are only used here for ease of comparison with the usual Bellman operator. Choice is really modelled here with the powerset monad.

If we want to be precise, we can say that a denotes a *distribution of states.* The function $\delta : (x, a, y) \mapsto a(x)$ takes elements $x \in X$, $a \in \pi_3(g(x)) \subseteq \mathcal{D}X$ and $y \in X$ to the probability $a(x) \in [0,1]$.

Predicates could also be used for other quantitative results. For example, distance, progress functions, etc. can be computed using the truth object $\mathbb{N}_\infty$.

Example 3.8 (Shortest distance in graphs)**.**
Consider once again the case of Kripke structures, here seen as (unweighted) directed graphs.
The Bellman-Ford algorithm (in the dual graph) can then be described as the computation of the sequence $(g_\tau^*)^n(\bot)$, in the process predicate type $(2 \times \mathrm{Id} \times \mathcal{P}(-), \mathbb{N}_\infty, \tau)$, where $\mathbb{N}_\infty = \mathbb{N} \sqcup \{\infty\}$, $\bot : X \to \mathbb{N}_\infty$ is the constant function $x \mapsto \infty$, and τ is the "shortest distance to goal" modality

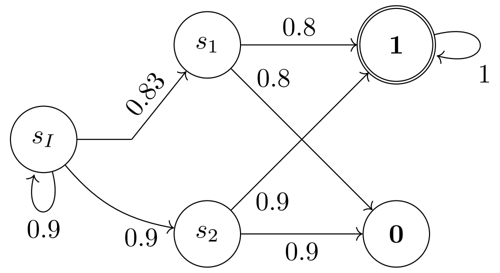
$$\begin{aligned} \tau : 2 \times \mathbb{N}_\infty \times \mathcal{P}\mathbb{N}_\infty &\to \mathbb{N}_\infty \\ (\top, \cdot, \cdot) &\mapsto 0 \\ (\bot, d, D) &\mapsto \min(d, 1 + \min(D)). \end{aligned}$$

More specifically, if we put the order $\infty < \cdots < n < n-1 < \cdots < 1 < 0$ on $\mathbb{N}_\infty$, the shortest distance to a goal can be seen as the *least fixed point* of g_τ^*.
Notice that we have been forced to introduce Id to the process type to make each node x able to observe itself in $g(x)$. We may also prefer to avoid modifying the directed graph process type, and formulate our problem as the ppt $(2 \times \mathcal{P}(-), \mathbb{N}_\infty, \tau)$ where

$$\begin{aligned} \tau : 2 \times \mathcal{P}\mathbb{N}_\infty &\to \mathbb{N}_\infty \\ (\top, \cdot, \cdot) &\mapsto 0 \\ (\bot, D) &\mapsto 1 + \min(D). \end{aligned}$$

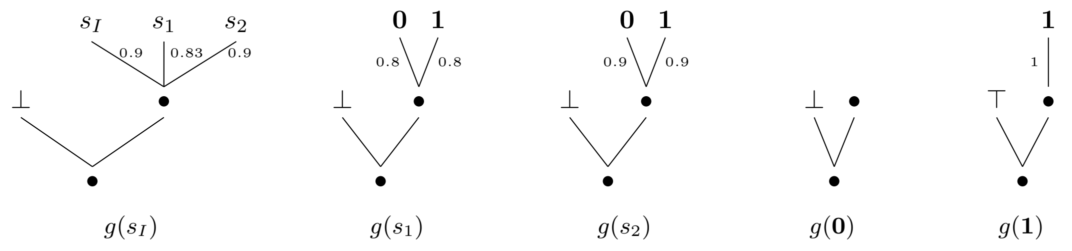
The solution then appears as the lfp of $p \mapsto \min(p, g_\tau^*(p))$ (with min computed pointwise).
Other interesting modalities can be computed on the truth object $\mathbb{N}_\infty$ to model other local computations in graphs.

Example 3.9 (Widest path in weighted graph)**.** Consider the case of weighted graphs with distinguished goals on the monoid $W = ([0,1], \max)$, modelled by the process type $2 \times \mathrm{Hom}_{\mathrm{fin}}(-, W)$.
The *widest path problem*, also called the *maximum capacity path problem*, asks what is the *widest path* from each state to a goal state.
For example, consider the following weighted graph. Note that the quantities above edges don't represent probability transitions anymore, but simply weights, which don't have to sum to 1.



$$X := \{s_I, s_1, s_2, \mathbf{0}, \mathbf{1}\}$$

$$\begin{aligned} g : X &\to 2 \times \mathrm{Hom}_{\mathrm{fin}}(X, [0,1]) \\ s_I &\mapsto \Big(\bot, \big\{(s_I, 0.9), (s_1, 0.83), (s_2, 0.9)\big\}\Big) \\ s_1 &\mapsto \Big(\bot, \big\{(\mathbf{0}, 0.8), (\mathbf{1}, 0.8)\big\}\Big) \\ s_2 &\mapsto \Big(\bot, \big\{(\mathbf{0}, 0.9), (\mathbf{1}, 0.9)\big\}\Big) \\ \mathbf{0} &\mapsto \Big(\bot, 0\Big) \\ \mathbf{1} &\mapsto \Big(\top, \big\{(\mathbf{1}, 1)\big\}\Big) \end{aligned}$$



Any widest path from s_I to $\mathbf{1}$ has width 0.9. An example of such widest path is $s_I \to s_2 \to \mathbf{1}$, whose width is indeed $\max(0.9, 0.9) = 0.9$. Another example of widest path between s_I and $\mathbf{1}$ is $s_I \to s_I \to s_2 \to \mathbf{1} \to \mathbf{1} \to \mathbf{1}$. Any other path has a smaller width. For example, $s_I \to s_1 \to \mathbf{1}$ has width $\max(0.83, 0.8) = 0.8$.

You may have noticed that 0.9, the width of a widest path from s_I to $\mathbf{1}$, is the same as the probability $\mathbb{P}(\Diamond\mathbf{1})(s_I)$ from the stochastic game in Example 2.12. Similarly, in this weighted graph the width of a widest path is 0.8 from s_1 to $\mathbf{1}$, 0.9 from s_2 to $\mathbf{1}$, 1 from $\mathbf{1}$ to $\mathbf{1}$, and 0 from $\mathbf{0}$ to $\mathbf{1}$.

There is a good reason why widest path and reachability probabilities coincide in this example. The WG above was constructed by transforming the SG of Example 2.12, first into a MDP, then into a WG, using the method described in paper [1]. Similar connections, relating SG and MDP, or other kinds of process-predicate types related together, are studied in Section 5.

Like before, the solution to this widest path problem enjoys a fixed point characterisation, and can be computed as $\sup g_\tau^\alpha(\mathbf{0})$, in the process-predicate type $(2 \times \mathrm{Id} \times \mathrm{Hom}_{\mathrm{fin}}(-, W), W, \tau)$, where $\mathbf{0} : x \mapsto 0$ and τ is the *widest path modality*

$$\begin{aligned} \tau : 2 \times W \times \mathrm{Hom}_{\mathrm{fin}}(W, W) &\to W \\ (\top, \cdot, \cdot) &\mapsto 1 \\ (\bot, w_0, d) &\mapsto \max\left(w_0, \max_{w \in W}(\min(w, d(w)))\right). \end{aligned}$$

Like in the previous example, Id was necessary to make nodes able to observe themselves, but we may also choose to reformulate our problem on $2 \times \mathrm{Hom}_{\mathrm{fin}}(-, W)$ and compute the lfp of the new $p \mapsto \max(p, g_\tau^*(p))$.

A useful property on this example is that the problem can be solved directly by graph analysis, in time $O(|E| + |V|\log|V|)$. Relating the difficult problem of reachability probabilities in SG to the simple problem of widest paths in WG is the main technique developed in [1]. Fixed-point characterisations are the main ingredient making this possible, and generalising this method is one of the goals of the current paper.

In 3.1, we have introduced *process-predicate types*, a structure extending that of *process types* with truth objects and modalities, allowing for predicates and weakest-precondition transformers.

We have noticed that many interesting quantities could be computed as *fixed points* of weakest-precondition transformers. We gave a few examples, but it is possible to go much further, e.g. by studying *alternating fixed points semantics* for modal μ-logic (a generalisation of modal logics like LTL, CTL and CTL*) [62], [63].

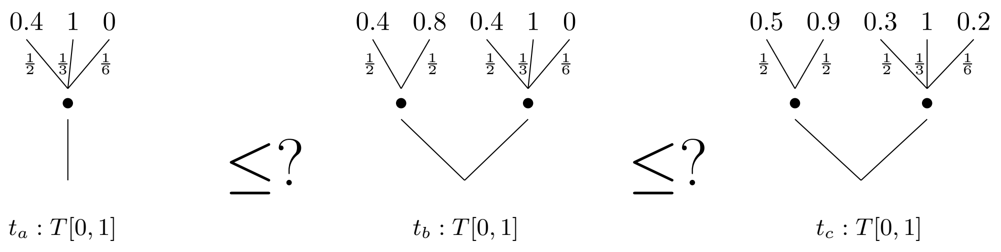
So far, we have introduced half of the structure used in this paper, by adding logic to process types. However, we have already noticed, on examples, the role of fixed point characterisations, which require additional order structure: something is missing. In Section 3.2, we give the remaining half, by exploring how order structures can be added to process(-predicate) types. This will give us not only a theory of fixed points, in Section 4, but also a theory of process type approximation in Section 5.

3.2 Preorder-enrichments

Ordered structures like preorders, partial orders, complete lattices, etc. are highly used in computer science to study iterative processes, incremental methods or (co)inductive definitions. In this section, we explain how they can be used to study processes.

Recall that process types are simply endofunctors $T : \mathbf{Set} \to \mathbf{Set}$. Balan and Kurz [64] studied how to extend and lift such endofunctors to **Preord** and **Poset**, focusing on the case of *finitary functors*. We make use of those ideas, but not of their methods, since we are interested in some non-finitary functors. Lifting and extension of $T : \mathbf{Set} \to \mathbf{Set}$ to order categories appear regularly in coalgebraic studies. Extensions to **Preord** are at the heart of the theory of (bi)simulation studied in [36]. Liftings to categories enriched on **Poset** and **Cppo** appear [35] in the study of weakest-precondition semantics. Those last two examples are very similar to what we will call *shape structure* and *leaf structure* in the rest of this paper.

We start from the tree representation of elements $g(x) : TY$ of processes $g : X \to TY$, and try to see which kinds of order structures may arise.



0.4 1 0
$\frac{1}{2}$ $\frac{1}{3}$ $\frac{1}{6}$
$t_a : T[0,1]$
$\leq?$
0.4 0.8 0.4 1 0
$\frac{1}{2}$ $\frac{1}{2}$ $\frac{1}{2}$ $\frac{1}{3}$ $\frac{1}{6}$
$t_b : T[0,1]$
$\leq?$
0.5 0.9 0.3 1 0.2
$\frac{1}{2}$ $\frac{1}{2}$ $\frac{1}{2}$ $\frac{1}{3}$ $\frac{1}{6}$
$t_c : T[0,1]$

There are two natural ways of putting a preorder structure on TY.

(i) Directly use the preorder structure of Y if it is available.

(ii) Compare elements of TY without using any sort of structure on Y.

We call the first preorder structure a *leaf structure*, and the second preorder structure a *shape structure*.

Leaf structures are described by a *lifting* of the endofunctor $T : \mathbf{Set} \to \mathbf{Set}$ to a *monotone* endofunctor $T_l : \mathbf{Preord} \to \mathbf{Preord}$. Monotone endofunctors are simply endofunctors enriched over **Preord**.

Shape structures are defined by an *extension* of $T : \mathbf{Set} \to \mathbf{Set}$ to $T_s : \mathbf{Set} \to \mathbf{Preord}$.

In Section 4, we will study leaf structures, see how they can be used to recognise situations with fixed point characterisations, and use them to give conditions for predicate transformers to be monotone.

In Section 5, we will study shape structures, and use them to design a generic way of comparing predicate-process types together, with the goal of approximating complex processes by simpler ones.

3.2.1 Reminders about order theory and enriched category theory

The reader acquainted with order theory and enriched category theory may choose to skip Section 3.2.1.

Before stating the definition of leaf structures, we need to recall the definition of an enriched category over **Preord**. The concept of enriched categories is a generalisation of the concept of categories, where *sets* of morphisms are replaced by an object from a different category. A general introduction to enriched category theory can be found at [65], along with precise definitions. Here, we simply unwind those definitions in the case of **Preord**.

Definition 3.10 (Notions for **Preord**)**.** A *preorder* $(A, \leq)$ consists of a set A and a relation $\leq \, \in A \times A$ that is reflexive and transitive (but not necessarily total nor antisymmetric).

A *monotone function* $f : (A, \leq) \to (B, \leq)$ between two preorders is a function $f : A \to B$ such that $\forall a, a' \in A,\, a \leq a' \implies f(a) \leq f(a')$.

Preord is the category whose objects are preorders and whose arrows are monotone functions. It is known to be *complete*, *cocomplete*, and *cartesian closed*. In particular, it is *monoidal* and *symmetric*.

The internal hom-object between two elements $A, B : \mathbf{Preord}$ is the preorder $[A, B] := (\mathrm{Hom}(A, B), \leq)$ defined by the pointwise order

$$\forall f, g : \mathrm{Hom}(A, B), f \leq g \iff \forall a \in A,\; f(a) \leq_B f(b).$$

Note that $\leq_A$ does not appear in this definition. Moreover notice that there was no other choice for the order structure of $\mathrm{Hom}(A, B)$. It is uniquely defined (up to natural isomorphism) by the enrichment. Indeed, internal hom-objects are defined by a product-hom adjunction, and adjoint functors are unique up to natural isomorphism.

The product of **Preord** is called the *product (pre)order*. For $(A, \leq_A)$, $(B, \leq_B)$: **Preord**, the product $(A \times B, \leq_{A \times B})$ is given by $\forall a, a' : A \forall b, b' : B,\; (a, b) \leq_{A \times B} (a', b') \iff a \leq_A a' \wedge b \leq_B b'$.

We have a sequence of discrete-forgetful-trivial adjunction

$$D \vdash U \vdash T,$$

where $U : \mathbf{Preord} \to \mathbf{Set}$ is the forgetful functor $U : (A, \leq) \mapsto A$, $D : A \mapsto (A, =)$ puts the discrete preorder, and $T : A \mapsto (A, A \times A)$ puts the indiscrete preorder.

We will also use the notion of *up-set*. In a preorder $(X, \leq)$ an up-set $U \subseteq X$ is a subset such that

$$\forall x, x' \in X,\, x \in U \wedge x \leq x' \implies x' \in U.$$

The following definition unwraps the general definition of enriched categories.

Definition 3.11 (**Preord**-category)**.** A **Preord**-category $\mathcal{A}$ consists of a class of objects $\mathrm{ob}(\mathcal{A})$ and a **Preord**-object $\mathcal{A}(A, B)$ for each $A, B : \mathrm{ob}(\mathcal{A})$. Moreover, we need to satisfy usual composition, associativity and unit laws.

Furthermore, we ask for composition

$$M_{A,B,C} : \mathcal{A}(B, C) \times \mathcal{A}(A, B) \to \mathcal{A}(A, C)$$

to be monotone.

Remark 3.12. Since **Preord** is cartesian closed, it is enriched over itself, i.e. it has a structure of **Preord**-category given by pointwise orders (internal hom-preorders).

Note that this proves that the composition operation in **Preord** is monotone. In other words, postcomposition by a monotone function is a monotone operation, and similarly for precomposition.

Definition 3.13 (**Preord**-functor)**.** A **Preord**-functor $F : \mathcal{A} \to \mathcal{B}$ between **Preord**-categories is given by functions $F : \mathrm{ob}(\mathcal{A}) \to \mathrm{ob}(\mathcal{B})$ and monotone maps $F : \mathcal{A}(A, B) \to \mathcal{B}(FA, FB)$ for all objects A, B of $\mathcal{A}$, satisfying additional, usual coherence laws (relative to composition and units).

We note $\mathrm{Hom}_{\mathbf{Preord}}(\mathcal{A}, \mathcal{B})$ the class of all such functors, and $\mathrm{Hom}_{\mathbf{Set}}(\mathcal{A}, \mathcal{B})$ the class of usual **Set**-functors between the underlying categories $\mathcal{A}_0$ and $\mathcal{B}_0$ (forgetting about structure on homsets).

Preord-functors will also be called *monotone* functors.

In particular, the class $\mathrm{Hom}_{\mathbf{Preord}}(\mathbf{Preord}, \mathbf{Preord})$ of monotone endofunctors on **Preord** is a class of **Preord**-endofunctors.

To complete Section 3.2.1, we will recall Knaster-Tarski's theorem and its Cousot-Cousot counterpart.

Theorem 3.14 (Knaster-Tarski and Cousot-Cousot)**.**
Let $(A, \leq)$ be a complete lattice (a partial *order such that every subset has (unique) infimum and supremum). Then, for every monotone function $f : (A, \leq) \to (A, \leq)$, the set of fixed points of f is itself a complete lattice. In particular, f has least and greatest fixed points, written* $\mathrm{lfp}(f)$ *and* $\mathrm{gfp}(f)$ *respectively.*
Moreover,

$$\mathrm{lfp}(f) = \inf\{a \in A \mid a \geq f(a)\}$$
$$\mathrm{gfp}(f) = \sup\{a \in A \mid a \leq f(a)\}$$

and

$$\mathrm{lfp}(f) = \sup_{\alpha:\mathbf{Ord}} f^{\alpha}(\bot)$$
$$\mathrm{gfp}(f) = \inf_{\alpha:\mathbf{Ord}} f^{\alpha}(\top)$$

where $\bot = \inf \varnothing$ is the least element of A, $\top = \sup \varnothing$ is the top element of A, and α goes over all ordinals, where the transfinite iteration *f^{α} is defined by transfinite induction: $f^0 = \mathrm{id}_A$, $\forall \alpha\, f^{\alpha+1} = f \circ f^{\alpha}$, and for each limit ordinal λ, $f^{\lambda} = \sup_{\alpha<\lambda} f^{\alpha}$, where* $\mathrm{Hom}(A, A)$ *is equipped with the pointwise order (making it a complete lattice).*
Moreover, $(f^{\alpha}(\bot))_{\alpha}$ is increasing, and $(f^{\alpha}(\top))_{\alpha}$ is decreasing.

3.2.2 Leaf and shape structure

With all this vocabulary, we are finally ready to introduce the characterisation of leaf and shape structures.

In Section 3.2.2 we will define leaf structures and shapes structures through lifting and extension of process predicate types. After this general definition, examples of those structures will be given in elementary cases, on functors of interest like $\mathcal{P}$ and $\mathcal{D}$, and in cases of interest like MDP and SG.

Leaf structures are obtained by putting an order structure on each $T\Omega$ where $(\Omega, \leq)$ is a preorder. Moreover, we ask for Ω to be a lattice, and for τ to be compatible with this structure. This is an order-enrichment of process types that is based on the order structure of the truth object.

Definition 3.15 (Leaf structure)**.** We say that a process-predicate type $\mathcal{T} = (T, \Omega, \tau)$ *has leaf structure* whenever we have a $\mathcal{T}_l = (T_l, \Omega_l, \tau_l)$ such that the following hold.

- $T_l : \mathrm{End}_{\mathbf{Preord}}(\mathbf{Preord})$ is a *lifting* of T, i.e. the following diagram commutes.

$$\begin{tikzcd}
\mathbf{Preord} \arrow[r, "T_l"] \arrow[d, "U"] & \mathbf{Preord} \arrow[d, "U"] \\
\mathbf{Set} \arrow[r, "T"] & \mathbf{Set}
\end{tikzcd}$$

- Ω_l is a *complete lattice* (*partial* order with all sup and inf) with underlying set Ω.

$$U\Omega_l = \Omega$$

- $\tau_l : T_l\Omega_l \to \Omega_l$ is monotone, and is a lifting of τ, i.e.

$$U(T_l\Omega_l \xrightarrow{\tau_l} \Omega_l) = (T\Omega \xrightarrow{\tau} \Omega).$$

The following characterisation makes it easier to check that T_l is a **Preord**-lifting of T, without going through all assumptions of the general case of enriched category theory. This is rather easy to prove, using $D \vdash U$ and the fact that $U : \mathbf{Preord} \to \mathbf{Set}$ is faithful.

Proposition 3.16 (Lifting from **Set** to **Preord**)**.** *An "assignment on objects" T_l : **Preord** $\to$ **Preord** is a **Preord**-lifting of T : **Set** $\to$ **Set** if and only if all of the following hold.*

(i) T_l is a functor, i.e. for all monotone functions f, $T_l f$ is monotone.
(It is unnecessary to check $T_l\mathrm{id} = \mathrm{id}$ and $T_l(f \circ g) = T_l f \circ T_l g$ when it is known for T.)

(ii) T_l is a lifting, i.e. $TU = UT_l$.

*(iii) For all X : **Set**, $(B, \leq_B)$: **Preord**,*

$$\begin{aligned} T_l(-) : (\mathrm{Hom}(X, B), \leq_B) &\to (\mathrm{Hom}(TX, T_l B), \leq_{T_l B}) \\ f &\mapsto T_l f \end{aligned}$$

is monotone (where pointwise order are used). This can be checked pointwise for each $t : TX$.

Leaf structures will be studied in Section 4 to give sufficient conditions for the construction of a fixed point theory.

Before giving a few examples, we will state how we describe the *second* kind of order enrichment for process types, the one that does not use at all the structure of the truth object, and uses only the branching structure.

Definition 3.17 (Shape structure)**.** We say that a process-predicate type $\mathcal{T} = (T, \Omega, \tau)$ *has shape structure* whenever we have a $\mathcal{T}_s = (T_s, \Omega_s, \tau_s)$ such that the following hold.

- T_s : **Set** $\to$ **Preord** is an *extension* of T, i.e. the following diagram commutes.

$$\begin{tikzcd}
 & \mathbf{Preord} \arrow[d, "U"] \\
\mathbf{Set} \arrow[ur, "T_s"] \arrow[r, "T"] & \mathbf{Set}
\end{tikzcd}$$

- Ω_s is a *preorder* with underlying set Ω.

$$U\Omega_s = \Omega$$

- $\tau_s : T_s\Omega \to \Omega_s$ is monotone, and is a lifting of τ (coincides with τ on elements), i.e.

$$U(T_s\Omega \xrightarrow{\tau_s} \Omega_s) = (T\Omega \xrightarrow{\tau} \Omega).$$

Shape structures will be studied in Section 5. There, we will construct a notion of comparison between ppt with shape structure, on which approximation algorithms can be based.

Before starting this deeper study, we exhibit a few examples of process-predicate types with leaf or shape structure, so as to know what to have in mind when reading the general theory.

Example 3.18 (Leaf structure on $\mathcal{P}$ – Egli-Milner preorder)**.** Let $(\Omega, \leq)$ be a preorder. The *Egli-Milner* preorder on $(\Omega, \leq)$ is the preorder $(\mathcal{P}\Omega, \preceq)$ defined by

$$\begin{aligned}\forall A, B \in \mathcal{P}\Omega,\ A \preceq B \iff &(\forall a \in A,\ \exists b \in B,\ a \leq b)\\ &\wedge (\forall b \in B,\ \exists a \in A,\ a \leq b).\end{aligned}$$

In other words $A \preceq B$ iff there exists an increasing $i : A \to B$ and decreasing $j : B \to A$.

Note that if Ω has a least element $\bot$, then for all $A \in \mathcal{P}_{\neq\varnothing}X$, $\{\bot\} \preceq A$, and similarly if there is a greatest element $\top$. $\varnothing$ is never comparable to any other subset.

Write $\mathcal{P}_{\text{em}} : (\Omega, \leq) \mapsto (\mathcal{P}\Omega, \preceq)$ the functor which assign to each preorder the Egli-Milner preorder on $\mathcal{P}\Omega$.

It can be shown (cf. Propositions 4.10 and 4.11) that $\mathcal{P}_{\text{em}}$ is a **Preord**-functor that is a lifting of $\mathcal{P}$, and that we obtain leaf structures for the modalities sup and inf for all complete lattices Ω.

- $(\mathcal{P}_{\text{em}}, \Omega, \inf)$ is a ppt with leaf structure.
- $(\mathcal{P}_{\text{em}}, \Omega, \sup)$ is a ppt with leaf structure.

To see how the Egli-Milner preorder behaves on an example, let $(\Omega, \leq) = ([0,1], \leq)$ be the unit segment with the usual order (which is a complete totally ordered lattice). For $0 \leq a < b < c < d \leq 1$, we have $[a,c] \preceq [b,d]$, but $[b,d] \not\preceq [a,d]$. $[a,b] \preceq [c,d]$, but the opposite is false. $[b,c]$ and $[a,d]$ are not comparable through $\preceq$. $[a,c) \preceq [a,c] \preceq (a,c]$, but the opposite is false. $[a,d]$ and $[a,b] \cup [c,d]$ are equivalent through $\preceq$, i.e. $[a,d] \preceq [a,b] \cup [c,d] \preceq [a,d]$. This shows that $\preceq$ can be only a preorder even when $\leq$ has very strong regularity properties.

The notions of suprema, infima and continuity for Egli-Milner preorder are interesting, but left for further research. This is related to conjectures mentioned in Section 7.1.

Example 3.19 (Leaf structure on $\mathcal{D}$ – Tailwise preorder)**.** Consider a preorder $(\Omega, \leq)$. The *tailwise* preorder of $(\Omega, \leq)$ is the preorder $(\mathcal{D}\Omega, \preceq)$ defined by

$$\forall d, d' \in \mathcal{D}\Omega,\ d \preceq d' \iff \Big(\forall U \subseteq \Omega,\ U \text{ is an up-set } \implies \sum_{y \in U} d(y) \leq \sum_{y \in U} d'(y)\Big).$$

$\preceq$ may be seen as the pointwise order on the tail distributions $x \mapsto d(\uparrow_x)$ where $\uparrow_x := \{y \mid x \leq y\}$, with additional hypotheses.

Write $\mathcal{D}_{\text{tl}} : (\Omega, \leq) \mapsto (\mathcal{D}\Omega, \preceq)$ the functor which assign to each preorder the tailwise preorder on $\mathcal{D}\Omega$.

It can be shown (cf. Corollary 4.17) that $\mathcal{D}_{\text{tl}}$ is a **Preord**-functor that is a lifting of $\mathcal{D}$, and that we obtain leaf structures for the expectation modality on $([0,1], \leq)$.

- $(\mathcal{D}_{\text{tl}}, [0,1], \mathbb{E})$ is a ppt with leaf structure.

Example 3.20 (Inductive construction of leaf structures)**.** It will be shown in Section 4 that leaf structures can be added, multiplied, composed, etc. They can thus be put on a large family of functors.

In particular, stochastic games (modelled on $2 \times 2 \times \mathcal{PD}(-)$) and MDP (modelled $2 \times \mathcal{PD}(-)$) enjoy a leaf structure for the reachability modality, composed using the tailwise order on $\mathcal{D}[0,1]$ and the Egli-Milner preorder on $\mathcal{PD}[0,1]$.

Unwinding this definition, for $t, t' : \mathcal{PD}[0,1]$, we will write that $t \preceq t'$ if and only if

$$\forall d \in t,\ \exists d' \in t',\ \ \forall x \in [0,1],\ d(\uparrow_x) \leq d'(\uparrow_x) \quad \text{and} \quad \forall d' \in t',\ \exists d \in t,\ \forall x \in [0,1],\ d(\uparrow_x) \leq d'(\uparrow_x).$$

For shape structures, a few simple examples will be enough for us.

Example 3.21 (Some shape structures)**.**

- For any process-predicate type (T, Ω, τ) and every Ω_s : **Preord**, we may always construct a shape structure by putting the discrete preorder on every TX.

- For any process-predicate type (T, Ω, τ), if Ω_s is the indiscrete preorder on Ω, then every extension of T to **Preord** (including the one that puts the indiscrete order everywhere) induces a shape structure.
- More interestingly, inclusion preorder induces a shape structure for $(\mathcal{P}, \Omega, \sup)$.
- Similarly, reverse inclusion preorder induces a shape structure for $(\mathcal{P}, \Omega, \inf)$.
- Because of this, $MDP := (2 \times \mathcal{PD}(-), [0,1], \beta)$ (where β is the Bellman modality) may be equipped with a shape structure defined by inclusion.

  $$\forall X : \mathbf{Set},\ \forall t_1, t_2 \in 2 \times \mathcal{PD}X,\ t_1 \sqsubseteq t_2 \iff (\pi_1 t_1 = \pi_1 t_2) \wedge (\pi_2 t_1 \subseteq \pi_2 t_2).$$

  On MDPs $g_1, g_2 : X \to 2 \times \mathcal{PD}X$, this can be interpreted as $g_1 \sqsubseteq g_2$ if and only if there are more choices in g_2 than in g_1.
- Similarly, we may define a shape structure on stochastic games ($SG := (2 \times 2 \times \mathcal{PD}(-), [0,1], \beta)$) by using reverse inclusion for Minimizer's choices, and equality for Maximizer's choices.

  $\forall X : \mathbf{Set},\ \forall t_1, t_2 \in 2 \times 2 \times \mathcal{PD}X,$

  $t_1 \sqsubseteq t_2 \iff (\pi_1 t_1 = \pi_1 t_2) \wedge (\pi_2 t_1 = \pi_2 t_2) \wedge \Big(\big(\pi_1 t_1 = \square \wedge \pi_3 t_1 = \pi_3 t_2\big) \vee \big(\pi_1 t_1 = \bigcirc \wedge \pi_3 t_1 \supseteq \pi_3 t_2\big) \Big).$

  On stochastic games $g_1, g_2 : X \to 2 \times 2 \times \mathcal{PD}X$, this is interpreted by saying that $g_1 \sqsubseteq g_2$ if and only if Maximizer has the same choices in both games and Minimizer has *fewer* choices in g_2 than in g_1.

In Sections 5 and 6, those shape structures on MDPs and SGs will allow us to connect the computations of reachability probabilities $\mathbb{P}(\Diamond \mathbf{1})$ in those two types of processes. We will then work toward a similar goal to create a connection between shapes structure of finite and infinite branching ($\mathcal{P}$ and $\mathcal{P}_\mathrm{f}$). Relating MDPs with WGs is discussed in Section 6.4.2, but left for further research. Finding additional examples is also left to further research, and could lead to new model checking algorithms based on surrogate models.

4 Fixed-point computations – leaf structure

We are now going to explain how the existence of a *leaf structure* on a process-predicate type, introduced in the last section, can be used as a sufficient condition to construct fixed points of the predicate transformer. The role of this structure, defined by a lifting of $T : \mathbf{Set} \to \mathbf{Set}$ to a monotone $T_l : \mathbf{Preord} \to \mathbf{Preord}$ and a monotone modality, is to package all assumptions making Theorem 4.1 be true.

Thanks to that, for applications, it will be easy to construct process-predicate types on which the predicate transformer $g^*_\mathcal{T}$ has *fixed points*, computable by *transfinite induction*: it is enough to build process types and modalities out of the examples of Theorem 4.5. The hardest part of the theory of leaf structures will actually be to prove that those canonical examples fit all assumptions to enjoy a leaf structure.

In Example 4.4, we will emphasise that transfinite induction is really needed, since an arbitrary number of limit cases may be necessary. Those speeds are measured using *convergence times*, defined in 4.3.

4.1 Leaf structures induce monotone predicate transformers

Theorem 4.1 (Monotone predicate transformers)**.** *Let $\mathcal{T} = (T, \Omega, \tau)$ be a process-predicate type. If $\mathcal{T}$ admits a leaf structure, then for all processes $g : X \to TY$, the predicate transformer $g^*_\mathcal{T}$ is monotone.*

Thus, by the Knaster-Tarski theorem, if $g : X \to TX$ is a coalgebra, fixed points of $g^*_\mathcal{T}$ form a complete lattice.

In particular, $g^*_\mathcal{T}$ then admits a least fixed point and a greatest fixed point which can be computed by transfinite induction.

$$V(g) := \mathrm{lfp}(g^*_\mathcal{T}) = \sup_{\alpha:\mathbf{Ord}} g^{\alpha}_\mathcal{T}(\bot) \quad \textit{and} \quad \mathrm{gfp}(g^*_\mathcal{T}) = \inf_{\alpha:\mathbf{Ord}} g^{\alpha}_\mathcal{T}(\top)$$

More precisely, those lfp and gfp are computed in the lattice $(\mathrm{Hom}(X, \Omega_l), \leq_{\Omega_l})$ with the pointwise order induced by the leaf structure.

Proof. Suppose that $\mathcal{T} = (T, \Omega, \tau)$ is a process-predicate type with leaf structure $\mathcal{T}_l = (T_l, \Omega_l, \tau_l)$.

Let $g : X \to TY$ be a process. Recall that $g^*_\tau := (p : \Omega^Y) \mapsto (\tau \circ Tp \circ g : \Omega^X)$.

Notice that for the pointwise (pre)order, precomposition by arbitrary functions and postcomposition by monotone functions preserves monotonicity.

Since by hypothesis both $T(-) : \mathrm{Hom}(Y, \Omega_l) \to \mathrm{Hom}(TY, T_l\Omega_l)$ and $\tau_l : T_l\Omega_l \to \Omega_l$ are monotone, we deduce that $g^*_\tau : \mathrm{Hom}(X, \Omega_l) \to \mathrm{Hom}(Y, \Omega_l)$ is monotone. □

The following properties will be used in many proofs about least fixed points of weakest-precondition transformers. Of course, the same holds for greatest fixed points.

Proposition 4.2 (Functoriality and monotonicity of value). *Let $\mathcal{T} = (T, \Omega, \tau)$ be a process-predicate type with leaf structure $\mathcal{T}_l$.*

- *Then,*

$$V : \mathrm{Coalg}(T) \longrightarrow \mathbf{Set}/\Omega$$
$$g \longmapsto \sup_\alpha g^\alpha_\tau(\bot)$$

$$\begin{tikzcd} X \arrow[r, "\varphi"] \arrow[d, "c"] & Y \arrow[d, "d"] & \arrow[r, mapsto] & & X \arrow[rr, "\varphi"] \arrow[dr, "V(c)"'] & & Y \arrow[dl, "V(d)"] \\ TX \arrow[r, "T\varphi"] & TY & & & & \Omega & \end{tikzcd}$$

is a well-defined **Set***-functor.*

In other words, for any two $c, d : \mathrm{Coalg}(T)$, whenever the diagram on the left commutes, the diagram on the right commutes as well, and $V(d)$ can be known from $V(c)$.

$$\begin{tikzcd} X \arrow[r, "\varphi"] \arrow[d, "c"] & Y \arrow[d, "d"] & & X \arrow[rr, "\varphi"] \arrow[dr, "V(c)"'] & & Y \arrow[dl, "V(d)"] \\ TX \arrow[r, "T\varphi"] & TY & & & \Omega & \end{tikzcd}$$

- *Moreover, if $c^*_\tau \leq d^*_\tau$ (pointwise, for the order of $\mathrm{Hom}(X, \Omega_l)$), then $V(c) \leq V(d)$ (pointwise, for the order of Ω_l).*

Proof. We will check each hypothesis making $V : \mathrm{Coalg}(T) \to \mathbf{Set}/\Omega$ a **Set**-functor, and then prove that it is monotone in the sense of the theorem.

- The main thing to do is to prove that V is actually well-defined on $\mathrm{Coalg}(T)$-arrows.

  Let $\varphi : X \to Y$ be a (**Set**)-function.

  First, notice that for any $f : Y \to \Omega$, $c^*_\tau(f \circ \varphi) = d^*_\tau(f) \circ \varphi$.

  Indeed,
$$c^*_\tau(f \circ \varphi) = \tau \circ Tf \circ T\varphi \circ c = \tau \circ Tf \circ d \circ \varphi = d^*_\tau(f) \circ \varphi.$$

  Then, define two chains of functions $f_\alpha : Y \to \Omega$, $g_\alpha : X \to \Omega$, indexed by ordinals α, by $f_0 = \bot$, $f_{\alpha+1} = d^*_\tau(f_\alpha)$, $f_\lambda = \sup_{\alpha<\lambda} f_\alpha$, and always setting $g_\alpha = f_\alpha \circ \varphi$.

  By definition,
$$\sup f_\alpha = V(d)$$

  Notice that $g_0 = \bot_{\mathrm{Hom}(X,\Omega_l)}$. Indeed, for all $x \in X$, $g_0(x) = f_0(\varphi(x)) = \bot_{\Omega_l}$.

  Moreover, for each α,
$$g_{\alpha+1} = d^*_\tau(f_\alpha) \circ \varphi = c^*_\tau(f_\alpha \circ \varphi) = c^*_\tau(g_\alpha).$$

  Additionally, for limit ordinals λ,
$$g_\lambda = f_\lambda \circ \varphi = (\sup_{\alpha<\lambda} f_\alpha) \circ \varphi = \sup_{\alpha<\lambda}(f_\alpha \circ \varphi) = \sup_{\alpha<\lambda} g_\alpha,$$

where precomposition by φ is continuous (commutes with sup) by definition of the pointwise order.
Thus, $\sup g_\alpha = V(c)$. Therefore,

$$V(c) = \sup(f_\alpha \circ \varphi) = \sup(f_\alpha) \circ \varphi = V(d) \circ \varphi,$$

where we once again used the continuity of precomposition.

- Now that we have proven that V is well-defined both on Coalg(T)-objects and Coalg(T)-arrows, it is straightforward to prove that it preserves identities and compositions.

  Indeed, for each $g : X \to TX$,

$$V\big(\mathrm{id}_g^{\mathrm{Coalg}(T)}\big) = V\left(\begin{tikzcd} X \arrow[r, "\mathrm{id}_X"] \arrow[d, "g"] & X \arrow[d, "g"] \\ TX \arrow[r, "T\mathrm{id}_X"] & TX \end{tikzcd}\right) = \begin{tikzcd} X \arrow[rr, "\mathrm{id}_X"] \arrow[dr, "V(g)"'] & & X \arrow[dl, "V(g)"] \\ & \Omega & \end{tikzcd} = \mathrm{id}_{V(g)}^{\mathbf{Set}/\Omega}.$$

  Moreover, given two arrows $f \to g$ and $g \to h$ in Coalg(T) described by $\varphi : X \to Y$ and $\psi : Y \to Z$,

$$V\left(\begin{tikzcd} Y \arrow[r, "\psi"] \arrow[d, "g"] & Z \arrow[d, "h"] \\ TY \arrow[r, "T\psi"] & TZ \end{tikzcd} \circ \begin{tikzcd} X \arrow[r, "\varphi"] \arrow[d, "f"] & Y \arrow[d, "g"] \\ TX \arrow[r, "T\varphi"] & TY \end{tikzcd}\right) = V\left(\begin{tikzcd} X \arrow[r, "\psi\circ\varphi"] \arrow[d, "f"] & Z \arrow[d, "h"] \\ TX \arrow[r, "T(\psi\circ\varphi)"] & TZ \end{tikzcd}\right)$$

$$= \begin{tikzcd} X \arrow[rr, "\psi\circ\varphi"] \arrow[dr, "V(f)"'] & & Z \arrow[dl, "V(h)"] \\ & \Omega & \end{tikzcd}$$

$$= \begin{tikzcd} Y \arrow[rr, "\psi"] \arrow[dr, "V(g)"'] & & Z \arrow[dl, "V(h)"] \\ & \Omega & \end{tikzcd} \circ \begin{tikzcd} X \arrow[rr, "\varphi"] \arrow[dr, "V(f)"'] & & Y \arrow[dl, "V(g)"] \\ & \Omega & \end{tikzcd}$$

$$= V\left(\begin{tikzcd} Y \arrow[r, "\psi"] \arrow[d, "g"] & Z \arrow[d, "h"] \\ TY \arrow[r, "T\psi"] & TZ \end{tikzcd}\right) \circ V\left(\begin{tikzcd} X \arrow[r, "\varphi"] \arrow[d, "f"] & Y \arrow[d, "g"] \\ TX \arrow[r, "T\varphi"] & TY \end{tikzcd}\right).$$

- Finally, we will now prove our monotonicity result. Suppose that $c_\tau^* \leq d_\tau^*$.

  Set p_α, q_α two chains of functions so that $V(c) = \sup p_\alpha$, $V(d) = \sup q_\alpha$ like in 4.1. By transfinite induction, $p_\alpha \leq q_\alpha$ for each ordinal α.

  Indeed, $p_0 = \bot = q_0$.

  If $p_\alpha \leq q_\alpha$, then $p_{\alpha+1} = (c_\tau^*)(p_\alpha) \leq (c_\tau^*)(q_\alpha) \leq (d_\tau^*)(q_\alpha) = q_{\alpha+1}$.

  Finally, if λ is a limit ordinal and if we have $p_\alpha \leq q_\alpha$ for each $\alpha \leq \lambda$, then

$$p_\lambda = \sup_{\alpha<\lambda} p_\alpha \leq \sup_{\alpha<\lambda} q_\alpha = q_\lambda$$

  by the properties of sup.

  In conclusion, $V(c) = \sup p_\alpha \leq \sup q_\alpha = V(d)$.

□

Definition 4.3 (Convergence time)**.** Let $(\Omega, \leq)$ be a complete lattice, and let $(x_\alpha)_{\alpha:\mathbf{Ord}}$ be a (monotone) chain of elements of Ω.

We define the *convergence time* of $(x_\alpha)_\alpha$ as

$$\min\{\alpha : \mathbf{Ord} \mid x_{\alpha+1} = x_\alpha\}.$$

As was explained in the introduction, convergence time to lfp and gfp may be arbitrarily long, and involve arbitrarily many limit cases.

Example 4.4 (Late convergence)**.** Let $\alpha > 0$ be an ordinal.

Consider the case of the ppt $(2 \times \mathcal{P}(-), \mathrm{Bool}, \tau)$ where $\tau(\top, \cdot) = \top$ and $\tau(\bot, t) = \inf t$, which can be seen as a game where only Minimizer is playing, trying to avoid goal states.

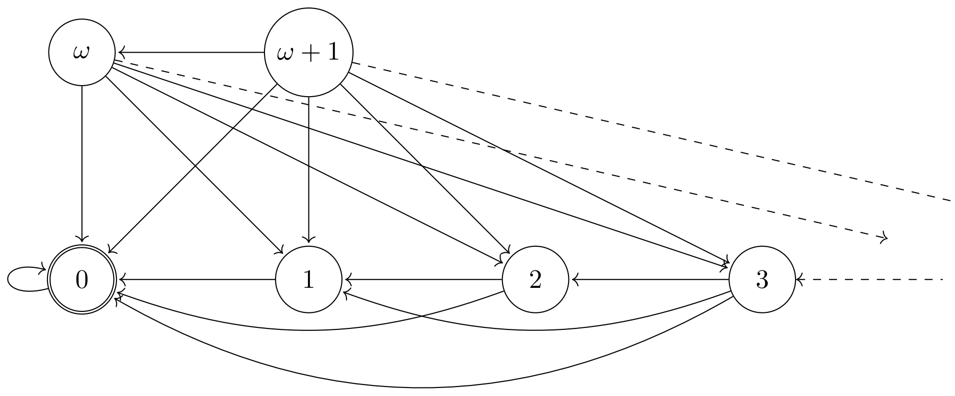
We can construct a process g where the convergence time $(g_\tau^\beta)(\bot)$, which converges to $V(g)$, is exactly α.

Let $X = [0, \alpha)$ be the set of ordinals strictly smaller than α. Define the coalgebra

$$\begin{aligned} g : X &\to 2 \times \mathcal{P}X \\ 0 &\mapsto (\top, \{0\}) \\ \beta &\mapsto (\bot, [0, \beta)) \text{ otherwise.} \end{aligned}$$

Here, $V(g) = \top$. (constant everywhere).

Moreover, for each $\beta \leq \alpha$, $(g_\tau^\beta)(\bot) = \mathbf{1}_{[0,\beta)}$.



The case $\alpha = \omega + 2$, with convergence in $\omega + 2$ steps.

Notice that in the example above, after ω steps, the row below is full of $\top$, but all other nodes are still at $\bot$. We may interpret this example by saying that Minimizer is trying to avoid the goal state. After $n < \omega$ iterations, i.e. at $g_\tau^n(\bot)$, she knows the consequences of her actions only up to n time steps, and so still believes that if she starts at ω, she can transition to states m with $n \ll m < \omega$ (m much greater than n) and avoid a disaster . It takes an *induction* (a transfinite iteration to the limit case ω for $g_\tau^\omega(\bot)$) to realise that the whole first row if off-limit. The consequences of this fact for ω and $\omega + 1$ are only discovered at step $\omega + 1$ and $\omega + 2$: two more successor cases after a limit case.

Recall that for applications, we are interested in techniques like value iteration (VI). We want to compute an increasingly better approximation of a value $V(g)$ through a sequence of lower bounds $(g_\tau^n(\bot))_{n\in\mathbb{N}}$. However, we just saw that the number of steps required for convergence could actually be transfinite, and that limit cases (inductions) must be performed by the implementation.

Transfinite value iteration has been studied before, e.g. for MDPs in [66], [67], but it is unclear how to use it directly in implementations.

To avoid this difficulty, we may prove by hand that predicate transformers are also continuous, or introduce additional *continuity* hypotheses to our leaf structures. This is left for further research.

4.2 A family of examples

The rest of this section is dedicated to the proof of the following theorem, providing a large class of examples of ppt with leaf structure.

Theorem 4.5. *The family $\mathcal{F}_1$, defined inductively, is a family of process-predicate with leaf structure.*

$$\frac{\Omega : \mathbf{CLat},\ A : \mathbf{Set},\ \tau : DA \to \Omega}{(\Delta_A, \Omega, \tau) : \mathcal{F}_1}\textit{(Constant ppt)} \qquad \frac{\Omega : \mathbf{CLat},\ A : \mathbf{Set},\ \tau : \mathrm{Hom}(A, \Omega) \to \Omega}{(\mathrm{Hom}(A, -), \Omega, \tau) : \mathcal{F}_1}\textit{(Reader ppt)}$$

$$\frac{\Omega : \mathbf{CLat}}{(\mathcal{P}_{\mathrm{em}}, \Omega, \sup) : \mathcal{F}_1}\textit{(Max game)} \qquad \frac{\Omega : \mathbf{CLat}}{(\mathcal{P}_{\mathrm{em}}, \Omega, \inf) : \mathcal{F}_1}\textit{(Min game)} \qquad \frac{}{(\mathcal{D}_{\mathrm{tl}}, [0,1], \mathbb{E}) : \mathcal{F}_1}\textit{(Random process)}$$

$$\frac{(W, +) : \mathbf{Monoid},\ (W, \leq_+) : \mathbf{CLat},\ \tau : (\mathrm{Hom}_{\mathrm{fin}}(W, W), \preceq_{tl}) \to (W, \leq_+)}{(\mathrm{Hom}_{\mathrm{fin}}(-, W), W, \tau) : \mathcal{F}_1}\textit{(Generalised random process)}$$

$$\frac{\mathcal{T}_a = (T_a, \Omega_a, \tau_a) : \mathcal{F}_1,\ \mathcal{T}_b = (T_b, \Omega_b, \tau_b) : \mathcal{F}_1}{\mathcal{T}_a \times \mathcal{T}_b := (T_a \times T_b, \Omega_a \times \Omega_b, \tau_a \times \tau_b) : \mathcal{F}_1}\textit{(Product)} \qquad \frac{\mathcal{T}_a = (T_a, \Omega, \tau_a) : \mathcal{F}_1,\ \mathcal{T}_b = (T_b, \Omega, \tau_b) : \mathcal{F}_1}{\mathcal{T}_a + \mathcal{T}_b := (T_a + T_b, \Omega, \tau_a + \tau_b) : \mathcal{F}_1}\textit{(Coproduct)}$$

$$\frac{\mathcal{T}_a = (T_a, \Omega, \tau_a) : \mathcal{F}_1,\ \mathcal{T}_b = (T_b, \Omega, \tau_b) : \mathcal{F}_1}{\mathcal{T}_a \circ \mathcal{T}_b := (T_a \circ T_b, \Omega, \tau_a \circ T_a\tau_b) : \mathcal{F}_1}\textit{(Composition)} \qquad \frac{\mathcal{T} = (T, \Omega, \tau) : \mathcal{F}_1,\ \tilde{\tau} : T\Omega \to \Omega}{(T, \Omega, \tilde{\tau}) : \mathcal{F}_1}\textit{(Modality replacement)}$$

Note that the following definitions and conventions are used.

- **CLat** *is the category of complete lattices.*

  The truth object Ω of a ppt with leaf structure will thus always be a complete lattice.

- *Modalities used in the hypotheses, which are written $\tau : A \to B$, with A, B two preorders, are always supposed monotone ($\tau : A \to B$ is a* **Preord***-arrow).*

- Constant ppt. *For any A : Set, Δ_A is the constant functor to the discrete order on A.*

$$\begin{aligned} \Delta_A : \mathbf{Preord} &\to \mathbf{Preord} \\ X &\mapsto DA \\ (f : X \to Y) &\mapsto \mathrm{id}_A \end{aligned}$$

  (Δ_A, Ω, τ) is a ppt with leaf structure for any application $\tau : DA \to \Omega$.

  In particular, we may choose to use the terminal complete lattice $\Omega = \{*\}$ and get the ppt with leaf structure $(\Delta_A, 1, 1)$.

- Reader ppt. *For any A : Set,* $\mathrm{Hom}(A, -)$ *is the functor*

$$\begin{aligned} \mathrm{Hom}(A, -) : \mathbf{Preord} &\to \mathbf{Preord} \\ X &\mapsto \mathrm{Hom}(A, X) \\ (f : X \to Y) &\mapsto ((g : A \to X) \mapsto (f \circ g : A \to Y)) \end{aligned}$$

  where the pointwise preorder is put on $\mathrm{Hom}(A, X)$*.* $(\mathrm{Hom}(A, -), \Omega, \tau)$ *is a ppt with leaf structure for any monotone $\tau : \mathrm{Hom}(A, \Omega) \to \Omega$. Examples include function application $\delta_a : (f : A \to \Omega) \mapsto (f(a) : \Omega)$.*

- Covariant powerset functor with Egli-Milner preorder. *Cf. Example 3.18.*

- Distribution functor with tailwise preorder. *Cf. Example 3.19.*

- $\mathrm{Hom}_{\mathrm{fin}}(-, W)$ with tailwise preorder. *Cf. Definition 2.10. $(W, +)$ is a monoid such that $(W, \leq_+)$ is a complete lattice, where we define*

$$\forall x, y \in W,\ x \leq_+ y \iff \exists z \in W,\ x + z = y.$$

Note that this "monoid preorder" coincides with the usual order in the cases of $([0,1], \max)$ *and* $(\mathbb{R}_+, +)$*.*

$$\textit{For } d, d' : \mathrm{Hom}_{\mathrm{fin}}(X, W),\ d \preceq_{tl} d' \iff \forall U \subseteq X \text{ up-set},\ d(U) \leq d'(U).$$

$(\mathrm{Hom}_{\mathrm{fin}}(-, W), W, \tau)$ *is then a ppt with leaf structure when* $\tau : \mathrm{Hom}_{\mathrm{fin}}(W, W) \to W$ *is monotone for the leaf structure. Examples include expectation* $\mathbb{E}$ *for* $W = (\overline{\mathbb{R}_+}, +)$ *and* $W = (\mathbb{N}_\infty, +)$*.*

- Product. *The product of two ppt with leaf structure* $\mathcal{T}_a = (T_a, \Omega_a, \tau_a)$*,* $\mathcal{T}_b = (T_b, \Omega_b, \tau_b)$*, where the lattices* Ω_a *and* Ω_b *may be different, is defined using the* product order *(cf. Definition 3.10), as*

$$(T_a \times T_b, \Omega_a \times \Omega_b, \tau_a \times \tau_b).$$

- Coproduct. *Similarly, if* $\mathcal{T}_a$ *and* $\mathcal{T}_b$ *share the same complete lattice* Ω*, we may define their coproduct*

$$\mathcal{T}_a + \mathcal{T}_b = (T_a + T_b, \Omega, \tau_a + \tau_b).$$

- Composition. *Similarly, if* $\mathcal{T}_a$ *and* $\mathcal{T}_b$ *share the same complete lattice* Ω*, we can define the composition*

$$\mathcal{T}_a \circ \mathcal{T}_b = (T_a \circ T_b, \Omega, \tau_a \circ T_a \tau_b).$$

- Constant modality. *If* $\mathcal{T} = (T_l, \Omega, \tau) : \mathcal{F}_l$*, we may replace* τ *by any increasing modality. Trivial examples include constant modalities* $\delta_u : (t : T_l\Omega) \mapsto u$*, where* $u \in \Omega$ *is a constant. We write this* (T_l, Ω, δ_u)*.*

This theorem will be proven in the rest of the section.

Remark 4.6 (Interpretation of Theorem 4.5)**.** We may interpret Theorem 4.5 by saying that "anything constructed on" modalities sup, inf and $\mathbb{E}$, using Egli-Milner preorder and tailwise preorder, induces a leaf structure.

Thus, for any process type T constructed (in a polynomial way) on $\mathcal{P}$ and $\mathcal{D}$, and for any modality τ constructed on sup, inf and $\mathbb{E}$, for all $g : X \to TX$, g^*_τ has fixed points reachable by (transfinite) value iteration.

Example 4.7. Stochastic games modelled on $\{\Box, \bigcirc\} \times \{\bot, \top\} \times \mathcal{PD}(-)$ can be given a leaf structure by the following construction.

$$\begin{tikzcd}
 & & \big(\{\Box,\bigcirc\} \times \{\bot,\top\} \times \mathcal{P}_{\mathrm{em}}\mathcal{D}_{\mathrm{tl}}(-), [0,1], \beta\big) \arrow[d, no head] & \\
 & & + \arrow[dl, no head] \arrow[dr, no head] & \\
 & \big(\{\Box,\bigcirc\} \times \mathcal{P}_{\mathrm{em}}\mathcal{D}_{\mathrm{tl}}(-), [0,1], \tilde{\beta}\big) \arrow[d, no head] & & \big(\{\Box,\bigcirc\} \times \mathcal{P}_{\mathrm{em}}\mathcal{D}_{\mathrm{tl}}(-), [0,1], \delta_1\big) \\
 & + \arrow[dl, no head] \arrow[dr, no head] & & \\
\big(\mathcal{P}_{\mathrm{em}}\mathcal{D}_{\mathrm{tl}}, [0,1], \sup \circ \mathcal{P}\mathbb{E}\big) \arrow[d, no head] & & \big(\mathcal{P}_{\mathrm{em}}\mathcal{D}_{\mathrm{tl}}, [0,1], \inf \circ \mathcal{P}\mathbb{E}\big) \arrow[d, no head] & \\
\circ \arrow[d, no head] \arrow[dr, no head] & & \circ \arrow[d, no head] \arrow[dr, no head] & \\
\big(\mathcal{P}_{\mathrm{em}}, [0,1], \sup\big) & \big(\mathcal{D}_{\mathrm{tl}}, [0,1], \mathbb{E}\big) & \big(\mathcal{P}_{\mathrm{em}}, [0,1], \inf\big) & \big(\mathcal{D}_{\mathrm{tl}}, [0,1], \mathbb{E}\big)
\end{tikzcd}$$

Notice that we obtain the Bellman modality.

This proves that the reachability probabilities $\mathbb{P}(\Diamond \mathbf{1})$ in SG can be computed by value iteration of g^*_β, even for infinite state spaces, i.e.

$$\mathbb{P}_{[g]}(\Diamond \mathbf{1}) = \sup_{\alpha : \mathbf{Ord}} g^\alpha_\beta(\bot).$$

4.3 Proof of Theorem 4.5

We will now prove all ingredients of Theorem 4.5.

We start by a trivial case to remember what there is to prove. It may be a bit wordy: we will allow ourselves to skip some details in the next cases.

Proposition 4.8 (The discrete constant ppt has leaf structure)**.**
Let A be a set, Ω be a complete lattice, and $\tau : A \to \Omega$ be an application.
Then, (Δ_A, Ω, τ) is a ppt with leaf structure.

Proof. We will use Definition 3.15 and Proposition 3.16.

First, we check that Δ_A, the constant functor to DA, is a **Preord**-functor.

- Indeed, $\Delta_A : \mathbf{Preord} \to \mathbf{Preord}$ is a lifting of the constant functor $\Delta_A : \mathbf{Set} \to \mathbf{Set}$.
- Moreover, if $f : X \to Y$ is a monotone function between preorders, $\Delta_A f = \mathrm{id}_{DA}$ is monotone (for all $a, b \in A$, if $a \leq b$, $\mathrm{id}_A a \leq \mathrm{id}_A b$).
- Finally, let X be a set, $(B, \leq_B)$ be a preorder, and let $f, g : X \to B$ be two functions such that $f \leq_B g$ pointwise. Let $t \in \Delta_A X$.

  We want to prove that $(\Delta_A f)(t) \leq (\Delta_A g)(t)$. This is obvious, since $\Delta_A f = \Delta_A g = \mathrm{id}_A$.

There is one more point to prove: $\tau : \Delta_A \Omega \to \Omega$ must be monotone. This is obvious as well, since $\Delta_A \Omega$ is discrete, so whenever $x, y \in \Delta_A \Omega$ are such that $x \leq y$, we have $x = y$, and thus $\tau x = \tau y$. In particular $\tau x \leq \tau y$. □

Proposition 4.9 (The reader ppt has leaf structure)**.**
Let A be a set, Ω be a complete lattice, and $\tau : \mathrm{Hom}(A, \Omega) \to \Omega$ be a monotone application.
Then, $(\mathrm{Hom}(A, -), \Omega, \tau)$ is a ppt with leaf structure.

Proof. τ is monotone by assumption, so we only have to prove that $\mathrm{Hom}(A, -)$ (with the pointwise order) is a **Preord**-functor.

- $\mathrm{Hom}(A, -) : \mathbf{Preord} \to \mathbf{Preord}$ is a lifting of the constant functor $\mathrm{Hom}(A, -) : \mathbf{Set} \to \mathbf{Set}$.
- Let $f : X \to Y$ be a monotone function between preorders. Let $g, h : A \to X$ be such that $g \leq h$ pointwise. Then, for all $a \in A$, $f(g(a)) \leq f(h(a))$, thus $\mathrm{Hom}(A, f)(g) \leq \mathrm{Hom}(A, f)(h)$.
- Let $f, g : X \to Y$ be two monotone functions between preorders such that $f \leq g$ pointwise. Let $h : A \to X$ be a function. For all $a \in A$, we have $f(h(a)) \leq g(h(a))$, thus $\mathrm{Hom}(A, f) \leq \mathrm{Hom}(A, g)$.

□

For the cases $(\mathcal{P}_{\mathrm{em}}, \Omega, \sup)$ and $(\mathcal{P}_{\mathrm{em}}, \Omega, \inf)$, we will start by checking that $\mathcal{P}_{\mathrm{em}}$ is indeed a **Preord**-lifting of $\mathcal{P}$, and then check that sup and inf are indeed monotone for the Egli-Milner preorder.

Proposition 4.10 (Egli-Milner lifting of $\mathcal{P}$)**.** *The covariant powerset functor with Egli-Milner preorder $\mathcal{P}_{\mathrm{em}}$: **Preord** $\to$ **Preord** is a **Preord**-lifting of $\mathcal{P}$: **Set** $\to$ **Set**.*

Proof.
- It is clear that $U\mathcal{P}_{\mathrm{em}} = \mathcal{P}U$.
- Let $f : X \to Y$ be a monotone function between preorders. Let $A, B : \mathcal{P}X$ be to subsets of X such that $A \preceq B$ (for the Egli-Milner preorder). We have an increasing $i : A \to B$ and a decreasing $j : B \to A$. If $A = \varnothing = B$ there is nothing to prove, so we suppose $A \neq \varnothing \neq B$.

  Let $x \in f(A)$, and let $a \in f^{-1}(x) \cap A$. Then, $x = f(a) \leq f(i(a)) \in f(B)$.

  Let $y \in f(B)$, and let $b \in f^{-1}(y) \cap B$. Then $f(A) \ni f(j(b)) \leq f(b) = y$.

  Thus, $f(A) \preceq f(B)$.

- Let X be a set, $(B, \leq_B)$ be a preorder, and $f, g : X \to (B, \leq_B)$ be functions such that $f \leq_B g$ pointwise. We want to prove that $\mathcal{P}f \preceq \mathcal{P}g$. Let $A \in \mathcal{P}X$. If $A = \varnothing$ there is nothing to prove, so we suppose $A \neq \varnothing$.

  Let $x \in f(A)$, and let $a \in f^{-1}(x) \cap A$. Then, $x = f(a) \leq_B g(a) \in g(A)$.

  Let $y \in g(A)$, and let $a \in g^{-1}(y) \cap A$. Then, $f(A) \ni f(a) \leq_B g(a) = y$.

  Thus, $f(A) \preceq g(A)$.

□

Proposition 4.11 (inf and sup are monotone for the Egli-Milner preorder)**.** *Let Ω be a complete lattice.* $\inf : \mathcal{P}_{\mathrm{em}}\Omega \to \Omega$ *and* $\sup : \mathcal{P}_{\mathrm{em}}\Omega \to \Omega$ *are monotone.*

Proof.
- sup. Let $A, B \in \mathcal{P}\Omega$ be such that $A \preceq B$ for the Egli-Milner preorder.

  If $A = \varnothing = B$, $\sup A = \sup B = \bot$ and there is nothing to prove.

  Otherwise, let $i : A \to B$ be increasing. We have

$$\sup_{a \in A} a \leq \sup_{a \in A} i(a) \leq \sup_{b \in B} b.$$

- inf. Let $A, B \in \mathcal{P}\Omega$ be such that $A \preceq B$ for the Egli-Milner preorder.

  If $A = \varnothing = B$, $\inf A = \inf B = \top$ and there is nothing to prove.

  Otherwise, let $j : B \to A$ be decreasing. We have

$$\inf_{a \in A} a \leq \inf_{b \in B} j(b) \leq \inf_{b \in B} b.$$

□

Corollary 4.12. $(\mathcal{P}_{\mathrm{em}}, \Omega, \sup)$ *and* $(\mathcal{P}_{\mathrm{em}}, \Omega, \inf)$ *are ppt with leaf structure.*

We will now prove that $\mathrm{Hom}_{\mathrm{fin}}(-, W)$ is indeed a **Preord**-functor for the tailwise order, and that expectation $\mathbb{E} : \mathrm{Hom}_{\mathrm{fin}}(W, W) \to W$ is monotone for some particular W.
From this, we deduce the corresponding results for $\mathcal{D}_{\mathrm{tl}}$ and $\mathbb{E} : \mathcal{D}[0, 1] \to [0, 1]$.

Proposition 4.13 ($\mathrm{Hom}_{\mathrm{fin}}(-, W)$ is a **Preord**-functor)**.** *Let $(W, +)$ be a monoid. Consider the "monoid preorder" $(W, \leq)$ defined by*

$$\forall x, y \in W,\ x \leq y \iff \exists z \in W,\ x + z = y.$$

Then, $\mathrm{Hom}_{\mathrm{fin}}(-, W)$ *is a* **Preord**-*functor for the tailwise preorder based on the monoid preorder.*

Proof.
- It is easy to check that $\mathrm{Hom}_{\mathrm{fin}}(-, W) : \mathbf{Set} \to \mathbf{Set}$ is indeed a functor. This implies that the version defined from **Preord** to **Preord** will then satisfy basic identity and composition axioms as well.

- Let $f : X \to Y$ be a monotone function between preorders. Let $d, d' : \mathrm{Hom}_{\mathrm{fin}}(X, W)$ be such that $d \preceq d'$ for the tailwise order, i.e. $\forall U \subseteq X$ up-set, $d(U) \leq d'(U)$.

  Let $V \subseteq Y$ be an up-set. We have

$$\mathrm{Hom}_{\mathrm{fin}}(f, W)(d)(Y) = \sum_{x \in f^{-1}(V)} d(x) \leq \sum_{x \in f^{-1}(V)} d'(x) = \mathrm{Hom}_{\mathrm{fin}}(f, W)(d')(V),$$

  where the middle inequality comes from the fact that $f^{-1}(V)$ is an up-set, by monotonicity of f. Indeed, if $x, x' \in X$ are such that $x \leq x'$ and $f(x) \in V$, $f(x) \leq f(x')$, thus $f(x') \in V$.

  Thus, $\mathrm{Hom}_{\mathrm{fin}}(f, W)$ is monotone.

- Let X be a set, $(B, \leq_B)$ be a preorder, and $f, g : X \to B$ be two functions such that $f \leq_B g$ pointwise. Let $d \in \mathrm{Hom}_{\mathrm{fin}}(X, W)$ and $U \subseteq B$ be an up-set. We have

$$\mathrm{Hom}_{\mathrm{fin}}(f, W)(d)(U) = \sum_{x \in f^{-1}(U)} d(x) \leq \sum_{x \in g^{-1}(U)} d(x) = \mathrm{Hom}_{\mathrm{fin}}(g, W)(d)(U),$$

where the middle inequality comes from the fact that $f^{-1}(U) \subseteq g^{-1}(U)$ by $f \leq g$. Indeed, let $x \in f^{-1}(U)$. We have $f(x) \leq_B g(x)$, thus $g(x) \in U$, i.e. $x \in g^{-1}(U)$.

Thus, $\mathrm{Hom}_{\mathrm{fin}}(-, W)$ is monotone on functions.

□

In both of the following propositions, $\mathbb{N}_\infty = \mathbb{N} \cup \{+\infty\}$ and $\overline{\mathbb{R}_+} = \mathbb{R}_+ \cup \{+\infty\}$ are equipped with the usual sum and the usual product. We use the convention $0 \times \infty = 0$.

Lemma 4.14 (Tail integration).

- *Let $W = (\mathbb{N}_\infty, +)$. The monoid order is the usual order $(\mathbb{N}_\infty, \leq)$.*

  For all $d : \mathrm{Hom}_{\mathrm{fin}}(W, W)$, we have
  $$\mathbb{E}(d) = \sum_{n \in \mathbb{N}} d((n, +\infty]).$$

- *Let $W = (\overline{\mathbb{R}_+}, +)$. The monoid order is the usual order $(\overline{\mathbb{R}_+} \leq)$.*

  For all $d : \mathrm{Hom}_{\mathrm{fin}}(W, W)$, we have
  $$\mathbb{E}(d) = \int_{x \in [0, +\infty)} d((x, +\infty])\, dx.$$

Proposition 4.15.
$\mathbb{E} : \mathrm{Hom}_{\mathrm{fin}}(W, W) \to W$ is monotone for the tailwise order when $W = (\mathbb{N}_\infty, +)$ or $W = (\overline{\mathbb{R}_+}, +)$.

Proof. Let $d, d' : \mathrm{Hom}_{\mathrm{fin}}(W, W)$. In the case $W = (\mathbb{N}, +)$, we have
$$\mathbb{E}(d) = \sum_{n \in \mathbb{N}} d((n, +\infty]) \leq \sum_{n \in \mathbb{N}} d'((n, +\infty]) = \mathbb{E}(d'),$$
and in the case $W = (\mathbb{R}_+, +)$ we have
$$\mathbb{E}(d) = \int_{x \in [0, +\infty)} d((x, +\infty])\, dx \leq \int_{x \in [0, +\infty)} d'((x, +\infty])\, dx = \mathbb{E}(d').$$

The positivity of d and d' is used to make sure that the sums and integrals are well-defined, and the fact that $(n, +\infty]$, $(x, +\infty]$ are up-sets provides local inequalities. □

Corollary 4.16.
$(\mathrm{Hom}_{\mathrm{fin}}(-, \mathbb{N}_\infty), \mathbb{N}_\infty, \mathbb{E})$ and $(\mathrm{Hom}_{\mathrm{fin}}(-, \overline{\mathbb{R}_+}), \overline{\mathbb{R}_+}, \mathbb{E})$ are ppt with leaf structure for the tailwise order.

Corollary 4.17.

- *$\mathcal{D}_{\mathrm{tl}} : \mathbf{Preord} \to \mathbf{Preord}$ is a $\mathbf{Preord}$-functor.*
- *For $d : \mathcal{D}[0, 1]$,*
  $$\mathbb{E}(d) = \int_0^1 d(]x, 1])\, dx.$$
- *$\mathbb{E} : \mathcal{D}[0, 1] \to [0, 1]$ is monotone for the tailwise order.*
- *$(\mathcal{D}_{\mathrm{tl}}, [0, 1], \mathbb{E})$ is a ppt with leaf structure.*

Proof.

- Same proof as Proposition 4.13, checking hypotheses for only some of the $d : \mathrm{Hom}_{\mathrm{fin}}(W, W)$.
- Special case of Lemma 4.14.
- Special case of Proposition 4.15, using only some of the $d : \mathrm{Hom}_{\mathrm{fin}}(W, W)$.
- Combine the preceding points.

□

We now proceed by showing that ppt with leaf structure can be multiplied, summed and composed.

Proposition 4.18 (Product of ppt with leaf structure)**.** *Let $\mathcal{T}_a = (T_a, \Omega_a, \tau_a)$ and $\mathcal{T}_b = (T_b, \Omega_b, \tau_b)$ be two ppt with leaf structure, where the lattices Ω_a and Ω_b may be different.*
Then, $(T_a \times T_b, \Omega_a \times \Omega_b, \tau_a \times \tau_b)$ is a ppt with leaf structure.

Proof.

- The product of two **Set**-functors is again a **Set**-functor.

  Moreover, if $f : X \to Y$ is a monotone function between preorders, and $(t_a, t_b), (t'_a, t'_b) \in T_aX \times T_bX$ are such that $(t_a, t_b) \leq_{(T_a \times T_b)X} (t'_a, t'_b)$, then $t_a \leq_{T_aX} t'_a$ and $t_b \leq_{T_bX} t'_b$, thus $f(t_a) \leq_{T_aY} f(t'_a)$ and $f(t_b) \leq_{T_bY} f(t'_b)$, thus $f((t_a, t_b)) \leq_{(T_a \times T_b)Y} f((t'_a, t'_b))$.

  Finally, for any X, Y : **Preord**, $(T_a \times T_b)(-) : \mathrm{Hom}(X, Y) \to \mathrm{Hom}((T_a \times T_b)X, (T_a \times T_b)Y)$ is monotone as the product of the two monotone functions $T_a(-)$ and $T_b(-)$.

  The product of two elements of $\mathrm{Hom}_{\mathbf{Preord}}(\mathbf{Preord}, \mathbf{Preord})$ is thus still in $\mathrm{Hom}_{\mathbf{Preord}}(\mathbf{Preord}, \mathbf{Preord})$.

- The product of the two complete lattices $\Omega_a \times \Omega_b$ is a complete lattice, where sup and inf are computed element-wise.

- $\tau_a \times \tau_b : T_a\Omega_a \times T_b\Omega_b \to \Omega_a \times \Omega_b$ is monotone as the product of two monotone applications.

□

Proposition 4.19 (Coproduct of ppt with leaf structure)**.** *Let $\mathcal{T}_a = (T_a, \Omega, \tau_a)$ and $\mathcal{T}_b = (T_b, \Omega, \tau_b)$ be two ppt with leaf structure sharing the same complete lattice.*
Then, $(T_a + T_b, \Omega, \tau_a + \tau_b)$ is a ppt with leaf structure.

Proof.

- The coproduct of two **Set**-functors is again a **Set**-functor.

  Moreover, if $f : X \to Y$ is a monotone function between preorders, and $t, t' \in T_aX + T_bX$ are such that $t \leq t'$, then $t, t' \in T_aX$ or $t, t' \in T_bX$. Without loss of generality, we assume $t, t' \in T_aX$, thus $(T_a + T_b)(f)(t) = (T_af)(t) \leq (T_af)(t') = (T_a + T_b)(f)(t')$.

  Finally, for any X, Y : **Preord**, $(T_a + T_b)(-) : \mathrm{Hom}(X, Y) \to \mathrm{Hom}((T_a + T_b)X, (T_a + T_b)Y)$ is monotone as the coproduct of the two monotone functions $T_a(-)$ and $T_b(-)$.

  The coproduct of two elements of $\mathrm{Hom}_{\mathbf{Preord}}(\mathbf{Preord}, \mathbf{Preord})$ is thus still in $\mathrm{Hom}_{\mathbf{Preord}}(\mathbf{Preord}, \mathbf{Preord})$.

- $\tau_a + \tau_b : T_a\Omega + T_b\Omega \to \Omega$, defined by case, is monotone since for each $t, t' \in T_a\Omega + T_b\Omega$, $t \preceq t'$ if and only if we have both $t, t' \in T_a\Omega$ and $t, t' \in T_b\Omega$. We can thus rely on the monotonicity of τ_a and τ_b.

□

Proposition 4.20 (Composition of ppt with leaf structure)**.** *Let $\mathcal{T}_a = (T_a, \Omega, \tau_a)$ and $\mathcal{T}_b = (T_b, \Omega, \tau_b)$ be two ppt with leaf structure sharing the same truth object.*
Then, $(T_a \circ T_b, \Omega, \tau_a \circ T_a\tau_b)$ is a ppt with leaf structure.

Proof.

- The composition of two **Set**-functors is again a **Set**-functor.

  Moreover, if $f : X \to Y$ is a monotone function between preorders, T_bf is monotone, thus $(T_aT_b)(f) = T_a(T_bf)$ is monotone.

  Finally, for any X, Y : **Preord**, $(T_a \circ T_b)(-) : \mathrm{Hom}(X, Y) \to \mathrm{Hom}((T_a + T_b)X, (T_a + T_b)Y)$ is monotone as the composition of the two monotone functions $T_a(-)$ and $T_b(-)$.

  The composition of two elements of $\mathrm{Hom}_{\mathbf{Preord}}(\mathbf{Preord}, \mathbf{Preord})$ is thus still in $\mathrm{Hom}_{\mathbf{Preord}}(\mathbf{Preord}, \mathbf{Preord})$.

- Notice that $\tau_a \circ T_a\tau_b : T_aT_b\Omega \to \Omega$ typechecks. Moreover, it is monotone as the composition of two monotone functions.

□

Proposition 4.21. *If $\mathcal{T} = (T, \Omega, \tau)$ is a ppt with leaf structure, for any monotone $\tilde{\tau} : T\Omega \to \Omega$, $(T, \Omega, \tilde{\tau})$ is a ppt with leaf structure.*

Proof. Trivial. □

We have finally proven all points of Theorem 4.5.

5 Approximation of process-predicate types – shape structure

This section is devoted to the study of *shape structures* on process-predicate types and of *connections* between different ppt with shape structures.

While the role of *leaf structure* was to provide fixed points of predicate transformers for a single process, *shape structures* and *connections* enable the comparison of several processes on the same type or on different types.

We will outline main definitions, results about *shape structures* and *connections*, and illustrate them in the case of stochastic games (SG) and Markov decision processes (MDP). Several additional examples and applications are provided in Section 6.

Recall that *shape structures* on ppts have been described in Definition 3.17 as tuples (T_s, Ω_s, τ_s), where T_s : **Set** $\to$ **Preord** is a **Preord**-extension of a process type T : **Set** $\to$ **Set**, and where $\tau_s : T_s\Omega \to \Omega_s$ is monotone.

In Section 5.1, we use shape structures to compare processes on the same type. This models simple phenomena like the fact that in a SG, if Minimizer loses power, then reachability probabilities of the goal increase.

In Section 5.2, we then introduce the notion of *connection* between process-predicate types with shape structure (Definition 5.7). An example of such connection is the “Minimizer restriction” relationship between SG and MDP. Other examples are given in Section 6.

This makes it possible to approximate solutions of problems expressed as a ppt by solutions of problems expressed on *another* ppt. This is stated in Corollary 5.15 as a comparison result between processes on *different* types. As explained in Remark 5.16, this improves value iteration algorithms (VI) provided by leaf structure (cf. Section 4), by adding upper bounds to lfp and lower bounds to gfp.

To go from VI to bounded value iteration (BVI), we need to go a bit further. Section 5.3 states assumptions that turn a *connection* into a *tight connection* (Definition 5.17). When those assumptions are met, we obtain Theorem 5.22, a strong approximation theorem making BVI possible by connecting two ppt together.

Theorem 5.22 states that when we have a connection $\mathcal{T}_\downarrow \lhd \mathcal{T}_\uparrow$ between ppt with both shape and leaf structures, if $\mathcal{T}_\uparrow$ is a *tight* upper approximation of $\mathcal{T}_\downarrow$, then lfp in $\mathcal{T}_\downarrow$ can be described as both suprema (using leaf structure) and infima (of lfp in $\mathcal{T}_\uparrow$, using the connection).

For example, in Example 5.23, we obtain a form of BVI for SG using the connection $SG \lhd MDP$, which meets our tightness assumptions in the case of finite branching. The importance of convergence speed for BVI is explained in Remark 5.24 and Proposition 5.25. This limits the applicability of this example, and more powerful applications are described in Section 6.

5.1 Shape-monotonicity of predicate transformers

We start by the following simple result, showing how shape structure and leaf structure interact.

Proposition 5.1 (Shape-monotonicity of predicate transformers)**.** *Let Ω be a complete lattice, $\Omega_0 = U\Omega$ the underlying set of Ω and $\mathcal{T} = (T, \Omega_0, \tau)$ be a ppt with shape structure $\mathcal{T}_s = (T_s, \Omega, \tau_s)$. Recall that this means that T_s :* **Set** $\to$ **Preord** *is an extension of T and that $\tau_s : (T_s\Omega_0, \sqsubseteq) \to (\Omega, \leq)$ is monotone.*

Then, for any two processes $g, g' : X \to TY$, if $g \sqsubseteq g'$ (for the shape preorder), we have

$$g^*_\tau \leq g'^*_\tau.$$

In particular, by Proposition 4.2, if $\mathcal{T}$ also has leaf structure $\mathcal{T}_l = (T_l, \Omega, \tau_l)$, and if g, g' are coalgebras, we thus have the following inequalities of value functions.

$$V(g) \leq V(g')$$

The same holds for greatest fixed points.

$$\mathrm{gfp}(g) \leq \mathrm{gfp}(g')$$

Proof. Look at the following diagram.

$$\begin{tikzcd}
& X \arrow[ld, "g"'] \arrow[rd, "g'"] & \\
TY \arrow[rr, dotted, "\mathrm{id}"] \arrow[d, "T_s f"'] & & TY \arrow[d, "T_s f"] \\
T_s\Omega_0 \arrow[rr, dotted, "\mathrm{id}"] \arrow[rd, "\tau_s"'] & & T_s\Omega_0 \arrow[ld, "\tau_s"] \\
& \Omega &
\end{tikzcd}$$

Since $g \sqsubseteq g'$, we have an inequality $\sqsubseteq$ on the top triangle.

Since $T_s f$ is monotone (by functoriality), this lifts to a relation $\sqsubseteq$ with the top triangle and center square.

Since τ_s is monotone for $\sqsubseteq$, post-composition by τ is increasing, and we thus have an inequality $\leq$ on the whole diagram. □

Remark 5.2. Notice that there is only one compatibility condition between the shape structure $\mathcal{T}_s$ and the leaf structure $\mathcal{T}_l$: there are defined on the *same* order Ω.

Both $\tau_s : T_s\Omega_0 \to \Omega$ and $\tau_l : T_l\Omega \to \Omega$ are monotone.

All comparison are made on predicate transformers $X \to \Omega$, so the order structure of the shared Ω can be used, and we can rely on the second point of proposition 4.2 which states that if $g^*_\tau \leq g'^*_\tau$ (pointwise for the order of $\mathrm{Hom}(X, \Omega)$), then $V(g) \leq V(g')$ (pointwise for the order of Ω).

This proposition models simple properties on the branching structure of our processes, such as the fact that when Minimizer loses choices in a SG, reachability probabilities of the goal increase, or similarly that they increase when the player of a MDP obtains new choices.

MDP and SG will serve as illustration to most properties in this example. Thus, before going further, we recall Example 3.21, and describe precisely the shape structures we put on MDPs and SGs,

Example 5.3 (A shape structure for MDPs: choice inclusion)**.** Let MDP be the process-predicate type defined by $(2 \times \mathcal{PD}(-), [0,1], \beta)$, where β is the Bellman modality ($\beta(\top, -) = 1$, $\beta(\bot, t) = \sup_{d \in t} \mathbb{E}(d)$).

This ppt models the problem of goal reachability in Markov decision processes, and can be given a leaf structure similarly as in Example 4.7, using Egli-Milner preorder and tailwise preorder.

Egli-Milner preorder is chosen, like it was in Section 4, so as to make sup and inf be monotone.

We define a shape structure for MDPs in the following way.

First, for all sets X, define the preorder $(2 \times \mathcal{PD}X, \sqsubseteq)$ by

$$\forall t_1, t_2 \in 2 \times \mathcal{PD}X,\ t_1 \sqsubseteq t_2 \iff (\pi_1 t_1 = \pi_1 t_2) \wedge (\pi_2 t_1 \subseteq \pi_2 t_2).$$

This defines a **Preord**-extension T_s of $2 \times \mathcal{PD}(-)$:

$$\begin{aligned} T_s : \mathbf{Set} &\to \mathbf{Preord} \\ X &\mapsto (2 \times \mathcal{PD}X, \sqsubseteq) \\ (f : X \to Y) &\mapsto 2 \times \mathcal{PD}f. \end{aligned}$$

Indeed, it is easy to check that if $f : X \to Y$, $T_s f = 2 \times \mathcal{PD}f$ is monotone for $\sqsubseteq$.

Let t_1, t_2 be in $T_s X$, such that $t_1 \sqsubseteq t_2$, i.e. $\pi_1 t_1 = \pi_1 t_2$ and $\{d : \mathcal{D}X \mid d \in \pi_2 t_1\} \subseteq \{d : \mathcal{D}X \mid d \in \pi_2 t_2\}$. We then have $(T_s f)(t_1) = (\pi_1 t_1, \{\mathcal{D}(f)d : \mathcal{D}Y \mid d \in \pi_2 t_1\})$ and $(T_s f)(t_2) = (\pi_1 t_2, \{\mathcal{D}(f)d : \mathcal{D}Y \mid d \in \pi_2 t_2\})$, where $\{\mathcal{D}(f)d : \mathcal{D}Y \mid d \in \pi_2 t_1\} \subseteq \{\mathcal{D}(f)d : \mathcal{D}Y \mid d \in \pi_2 t_2\}$, thus $(T_s f)(t_1) \sqsubseteq (T_s f)(t_2)$.

$\Omega_s = ([0,1], \leq)$ is a complete lattice.

Furthermore, the Bellman modality $\beta_s : (2 \times \mathcal{PD}[0,1], \sqsubseteq) \to ([0,1], \leq)$ is monotone. Indeed, there is nothing to check for the case of goal states $\top$. In the other case, we use the fact that for any two sets $A \subseteq B \subset [0,1]$, $\sup A \leq \sup B$.

Thus, the "choice inclusion" preorder $\sqsubseteq$ induces a ppt with shape structure (T_s, Ω_s, β_s). □

We define a slightly different kind of shape structure for SGs, using only reverse inclusion on Minimizer's choices, and equality on Maximizer's choices. This will be useful in Section 5.3 to connect SGs with MDPs.

Example 5.4 (A shape structure for SGs: Minimizer restriction)**.** Let SG be the process-predicate type defined by $(2 \times 2 \times \mathcal{PD}(-), [0,1], \beta)$, where β is the Bellman modality ($\beta(\top, -, -) = 1$, $\beta(\Box, \bot, t) = \sup_{d \in t} \mathbb{E}(d)$, $\beta(\bigcirc, \bot, t) = \inf_{d \in t} \mathbb{E}(d)$).

This ppt models the problem of goal reachability in stochastic games, and is given a leaf structure in Example 4.7, using Egli-Milner preorder and tailwise preorder.

We define a shape structure for SGs in the following way.

First, for all sets X, define the preorder $(2 \times 2 \times \mathcal{PD}X, \sqsubseteq)$ by

$$\forall t_1, t_2 \in 2 \times 2 \times \mathcal{PD}X,$$
$$t_1 \sqsubseteq t_2 \iff (\pi_1 t_1 = \pi_1 t_2) \wedge (\pi_2 t_1 = \pi_2 t_2) \wedge \Big(\big(\pi_1 t_1 = \Box \wedge \pi_3 t_1 = \pi_3 t_2\big) \vee \big(\pi_1 t_1 = \bigcirc \wedge \pi_3 t_1 \supseteq \pi_3 t_2\big) \Big).$$

This defines a **Preord**-extension T_s of $2 \times 2 \times \mathcal{PD}(-)$:

$$\begin{aligned} T_s : \mathbf{Set} &\to \mathbf{Preord} \\ X &\mapsto (2 \times \mathcal{PD}X, \sqsubseteq) \\ (f : X \to Y) &\mapsto 2 \times \mathcal{PD}f. \end{aligned}$$

Like before, it is easy to check that if $f : X \to Y$, $T_s f = 2 \times \mathcal{PD}f$ is monotone for $\sqsubseteq$. $\Omega_s = ([0,1], \leq)$ is still a complete lattice, and the Bellman modality $\beta_s : (2 \times 2 \times \mathcal{PD}[0,1], \sqsubseteq) \to ([0,1], \leq)$ is still monotone, where we use the fact that for any two sets $[0,1] \supseteq A \supseteq B$, $\inf A \leq \inf B$.

Thus, the "Minimizer restriction" preorder $\sqsubseteq$ induces a ppt with shape structure (T_s, Ω_s, β_s). $\Box$

Now that we have shape structures in mind for MDPs and SGs, we can illustrate the simple Proposition 5.1 on shape-monotonicity of predicate transformers.

Notice that we don't prove anything new or especially interesting: it is Proposition 5.1 that is a generalisation of those results.

Example 5.5 (Shape-monotonicity for choice inclusion in MDPs)**.** Let $\mathcal{T} = (2 \times \mathcal{PD}(-), [0,1], \beta)$ be the ppt of MDPs (for goal reachability), and $\mathcal{T}_s$ be its shape structure defined by choice inclusion like in 5.3.

Let $g, g' : X \to 2 \times \mathcal{PD}X$ be two MDPs such that $g \sqsubseteq g'$.

This means that for every state $x \in X$, $\pi_1 g_(x) = \pi_1 g'(x)$ and $\pi_2 g(x) \subset \pi_2 g'(x)$, i.e. goal states are the same in both MDPs and the player has more choices in g' than in g.

Proposition 5.1 show that this implies the relation $g^*_\beta \leq g'^*_\beta$ on predicate transformers. This can be written as $\mathcal{B}_g \leq \mathcal{B}_{g'}$, where $\mathcal{B}_h$ is the Bellman operator on the MDP h. More precisely, for every predicate $p : X \to [0,1]$ and every $x \in X$ which isn't a goal state,

$$\sup_{d \in \pi_2 g(x)} \sum_{y \in X} p(y) d(y) \leq \sup_{d \in \pi_2 g'(x)} \sum_{y \in X} p(y) d(y).$$

In particular, since $\mathcal{T}$ can be given a leaf structure, with $V(g)(x) = \mathbb{P}_g(\Diamond \mathbf{1})(x)$, we have that for all $x \in X$,

$$\mathbb{P}_g(\Diamond \mathbf{1})(x) \leq \mathbb{P}_{g'}(\Diamond \mathbf{1})(x).$$

Example 5.6 (Shape-monotonicity for Minimizer restriction in SGs)**.** Let $\mathcal{T} = (2 \times 2 \times \mathcal{PD}(-), [0,1], \beta)$ be the ppt of SGs (for goal reachability), and $\mathcal{T}_s$ be its shape structure defined by Minimizer restriction like in 5.4.

Let $g, g' : X \to 2 \times \mathcal{PD}X$ be two MDPs such that $g \sqsubseteq g'$.

This means that g and g' share the same goal states, and that states in g and g' are assigned to the same player. Moreover, for any $x \in X$, if $\pi_1 g(x) = \Box$ then $\pi_3 g(x) = \pi_3 g'(x)$, and if $\pi_1 g(x) = \bigcirc$ then $\pi_3 g(x) \supseteq \pi_3 g'(x)$. In other words, Maximizer has the same choices in g and g', and Minimizer has *fewer* choices in g' than in g.

Proposition 5.1 shows that this implies the relation $g^*_\beta \leq g'^*_\beta$ on predicate transformers. This can be written as $\mathcal{B}_g \leq \mathcal{B}_{g'}$, where $\mathcal{B}_h$ is the Bellman operator on the SG h. More precisely, for every predicate $p : X \to [0,1]$ and every $x \in X$ which isn't a goal state,

$$\begin{aligned} \sup_{d \in \pi_3 g(x)} \sum_{y \in X} p(y) d(y) &\leq \sup_{d \in \pi_3 g'(x)} \sum_{y \in X} p(y) d(y) && \text{if } x \text{ belongs to Maximizer} \\ \inf_{d \in \pi_3 g(x)} \sum_{y \in X} p(y) d(y) &\leq \inf_{d \in \pi_3 g'(x)} \sum_{y \in X} p(y) d(y) && \text{if } x \text{ belongs to Minimizer.} \end{aligned}$$

(We actually have an equality in the case of Maximizer's states).

In particular, since $\mathcal{T}$ can be given a leaf structure, with $V(g)(x) = \mathbb{P}_g(\Diamond\mathbf{1})(x)$, we have that for all $x \in X$,

$$\mathbb{P}_g(\Diamond\mathbf{1})(x) \leq \mathbb{P}_{g'}(\Diamond\mathbf{1})(x).$$

5.2 Connections between ppt with shape structure

We now introduce a notion of *connection* between ppt with shape structures. This allows us to relate ppt (with shape structure) together, and to compare processes on different ppt, paving the way for surrogate models and bounded value iteration.

Note that since our shape structures are very much related to the "orders on functors" described in [36], an interesting direction of research would be to construct *connections* between the many examples described in second section of [36].

Definition 5.7 (Connections and approximations)**.** Let $\mathcal{T}_\downarrow = (T_\downarrow, \Omega, \tau_\downarrow)$, $\mathcal{T}_\uparrow = (T_\uparrow, \Omega, \tau_\uparrow)$ be process-predicate types with shape structure on the same truth object.

We say that $\mathcal{T}_\downarrow$ and $\mathcal{T}_\uparrow$ are *connected* whenever there exists a ppt with shape structure $\mathcal{T}_* = (T_*, \Omega, \tau_*)$ such that the following holds.

We write $\Omega_0 := U\Omega$, $T_{\downarrow,0} := UT_\downarrow$, $T_{*,0} := UT_*$, $T_{\uparrow,0} := UT_\uparrow$ the underlying set and process types.

- The **Set** endofunctor $T_{*,0}$ is a cospan of $T_{\downarrow,0}$ and $T_{\uparrow,0}$, i.e. there is a diagram with two natural transformations $\alpha_\downarrow$, $\alpha_\uparrow$ of the following shape.

$$T_{\downarrow,0} \xRightarrow{\alpha_\downarrow} T_{*,0} \xLeftarrow{\alpha_\uparrow} T_{\uparrow,0}.$$

- We have $\tau_\downarrow \leq \tau_* \circ \alpha_\downarrow$ and $\tau_* \circ \alpha_\uparrow \leq \tau_\uparrow$ for the pointwise order on Ω.

$$\begin{tikzcd}
T_\downarrow\Omega_0 \arrow[r, "\alpha_{\downarrow,\Omega_0}"] \arrow[dr, "\tau_\downarrow"'] \arrow[r, phantom, shift right=3, "\leq" near end] & T_*\Omega_0 \arrow[d, "\tau_*"] & T_\uparrow\Omega_0 \arrow[l, "\alpha_{\uparrow,\Omega_0}"'] \arrow[dl, "\tau_\uparrow"] \arrow[l, phantom, shift left=3, "\leq" near end] \\
& \Omega &
\end{tikzcd}$$

In that case we say that $\mathcal{T}_*$ is the *connector* between $\mathcal{T}_\downarrow$ and $\mathcal{T}_\uparrow$, that $\mathcal{T}_\downarrow$ and $\mathcal{T}_\uparrow$ are *connected* through $\mathcal{T}_*$, that $\mathcal{T}_\downarrow$ is a *lower approximation* of $\mathcal{T}_\uparrow$, and that $\mathcal{T}_\uparrow$ is an *upper approximation* of $\mathcal{T}_\downarrow$.

When this holds, we write

$$\mathcal{T}_\downarrow \lhd_{\mathcal{T}_*} \mathcal{T}_\uparrow,$$

or simply $\mathcal{T}_\downarrow \lhd \mathcal{T}_\uparrow$ to signify that such a connection exists.

Remark 5.8. We could make a more general definition by using a zigzag instead of a cospan, but the cospan definition covers our examples.

Several examples of connections are given in Section 6.2.

A simple way of constructing such a connection is to simply have a natural transformation between process types. More sophisticated connectors simply serve as a generalisation of the following situation.

Proposition 5.9 (Connections through natural transformations)**.** *Let $\mathcal{T}_\downarrow = (T_\downarrow, \Omega, \tau_\downarrow)$, $\mathcal{T}_\uparrow = (T_\uparrow, \Omega, \tau_\uparrow)$ be process-predicate types with shape structure on the same truth object.*

Suppose that there exists a natural transformation $\alpha : UT_\downarrow \Rightarrow UT_\uparrow$ such that $\tau_\downarrow \leq \tau_\uparrow \circ \alpha_{\Omega_0}$. Then,

$$\mathcal{T}_\downarrow \lhd_{\mathcal{T}_\uparrow} \mathcal{T}_\uparrow.$$

Similarly, if there exists $\alpha : UT_\uparrow \Rightarrow UT_\downarrow$ such that $\tau_\downarrow \circ \alpha_{\Omega_0} \leq \tau_\uparrow$, then

$$\mathcal{T}_\downarrow \lhd_{\mathcal{T}_\downarrow} \mathcal{T}_\uparrow.$$

Proof. For the other natural transformation, take the identity. □

We give an example by connecting SGs with MDPs.

Example 5.10 ($SG \lhd_{MDP} MDP$)**.** Consider the two ppt with shape structure MDP and SG defined in Examples 5.3 and 5.4. We have a connection

$$SG \lhd_{MDP} MDP.$$

In other words, stochastic games can be connected to Markov decision processes in the sense of connections between ppt with shape structure.

Proof. Recall that MDP is defined on the ppt $(2 \times \mathcal{PD}(-), [0,1], \beta_{MDP})$, with the shape structure induced by choice inclusion, and that SG is defined on the ppt $(2 \times 2 \times \mathcal{PD}(-), [0,1], \beta_{SG})$, with the shape structure induced by Minimizer restriction.

Define a natural transformation α by

$$\begin{aligned} \alpha_X : \{\Box, \bigcirc\} \times \{\bot, \top\} \times \mathcal{PD}X &\to \{\bot, \top\} \times \mathcal{PD}X \\ (\bigcirc, g, \varnothing) &\mapsto (\top, \varnothing) \\ (p, g, t) &\mapsto (g, t) \qquad \text{if } (p,t) \neq (\bigcirc, \varnothing). \end{aligned}$$

α forgets, for each state, to which player it belongs, and deals with the special case where Minimizer is in a deadlock by turning it into a goal. It is straightforward to verify that α is indeed natural.

We thus only have to prove that $\beta_{SG} \leq \beta_{MDP} \circ \alpha_{[0,1]}$.

Let $u = (p, g, t) \in 2 \times 2 \times \mathcal{PD}[0,1]$ and $v = \alpha_{[0,1]}(u) = (g, t) \in 2 \times \mathcal{PD}[0,1]$.

If $g = \top$ (if u stems from a goal state), we directly have $\beta_{SG}(u) = 1 = \beta_{MDP}(v)$.

Similarly, if $g = \bot$ and $p = \Box$, $\beta_{SG}(u) = \sup_{d \in t} \mathbb{E}(d) = \beta_{MDP}(v)$.

If $g = \bot$, $p = \bigcirc$, $t \neq \varnothing$, we can check that $\beta_{SG}(u) = \inf_{d \in t} \mathbb{E}(d) \leq \sup_{d \in t} \mathbb{E}(d) = \beta_{MDP}(v)$.

Finally, if $p = \bigcirc$ with $t = \varnothing$, we have $\beta_{SG}(u) = \inf \varnothing = 1 = \beta_{MDP}(v)$. □

The names *lower* and *upper approximations* can be justified by the following property.

Proposition 5.11 (Predicate transformers going through the connector)**.** *Let $\mathcal{T}_\downarrow$, $\mathcal{T}_*$ and $\mathcal{T}_\uparrow$ be ppt with shape structure such that $\mathcal{T}_\downarrow \lhd_{\mathcal{T}_*} \mathcal{T}_\uparrow$.*

Let $g_\downarrow : X \to T_\downarrow Y$ and $g_\uparrow : X \to T_\uparrow Y$ be two processes. Then,

$$(g_\downarrow)^*_{\mathcal{T}_\downarrow} \leq (\alpha_\downarrow \circ g_\downarrow)^*_{\mathcal{T}_*} \quad \text{and} \quad (\alpha_\uparrow \circ g_\uparrow)^*_{\mathcal{T}_*} \leq (g_\uparrow)^*_{\mathcal{T}_\uparrow}.$$

Moreover,

- *If $g_\downarrow$ and $g_\uparrow$ coincide in $\mathcal{T}_*$, i.e. if $\alpha_\downarrow \circ g_\downarrow = \alpha_\uparrow \circ g_\uparrow$ then*
$$(g_\downarrow)^*_{\mathcal{T}_\downarrow} \leq (g_\uparrow)^*_{\mathcal{T}_\uparrow}.$$
- *If $\mathcal{T}_\downarrow$, $\mathcal{T}_*$, $\mathcal{T}_\uparrow$ also have leaf structure (with the same order structure on Ω), and if $g_\downarrow, g_\uparrow$ are coalgebras, then*
$$V(g_\downarrow) \leq V(\alpha_\downarrow \circ g_\downarrow) \quad \text{and} \quad V(\alpha_\uparrow \circ g_\uparrow) \leq V(g_\uparrow),$$
and the same holds for greatest fixed points.
- *Finally, if all the hypotheses above hold, i.e. if $g_\downarrow$ and $g_\uparrow$ coincide in $\mathcal{T}_*$, if we have leaf structures, and if the processes are coalgebras, then*
$$V(g_\downarrow) \leq V(g_\uparrow),$$
and the same holds for greatest fixed points.

Proof. Let $p : Y \to \Omega$ be a predicate, and look at the following diagram.

$$\begin{tikzcd}
 & X \arrow[dl, "g_\downarrow"'] \arrow[dr, "\alpha_{\downarrow,Y} \circ g_\downarrow"] & \\
T_\downarrow Y \arrow[rr, "\alpha_{\downarrow,Y}"'] \arrow[d, "T_\downarrow p"'] & & T_* Y \arrow[d, "T_* p"] \\
T_\downarrow \Omega_0 \arrow[rr, "\alpha_{\downarrow,\Omega_0}"] \arrow[dr, "\tau_\downarrow"'] & & T_* \Omega_0 \arrow[dl, "\tau_*"] \\
 & \Omega &
\end{tikzcd}$$

The triangle on the top is commutative by definition (composition).
The square in the middle is commutative by naturality of α.
The last triangle is $\leq$ by definition of the connection through T_*, τ_*.
This lifts to a $\leq$ relation on the full diagram since precomposition is increasing.
The same proof can be done on this diagram

$$\begin{tikzcd}
 & X \arrow[dl, "\alpha_{\uparrow,Y}\circ g_\uparrow"'] \arrow[dr, "g_\uparrow"] & \\
T_*Y \arrow[d, "T_*p"'] & & T_\uparrow Y \arrow[ll, "\alpha_{\uparrow,Y}"] \arrow[d, "T_\uparrow p"] \\
T_*\Omega_0 \arrow[dr, "\tau_*"'] & & T_\uparrow\Omega_0 \arrow[ll, "\alpha_{\uparrow,\Omega_0}"'] \arrow[dl, "\tau_\uparrow"] \\
 & \Omega &
\end{tikzcd}$$

□

This can be exemplified in the connection $SG \lhd_{MDP} MDP$.

Example 5.12 (Minimizer restriction increases reachability probabilities)**.** Consider once again the case of SG and MDP, using the connection $SG \lhd_{MDP} MDP$ defined in Example 5.10 with a natural transformation $\alpha : 2 \times 2 \times \mathcal{PD}(-) \Longrightarrow 2 \times \mathcal{PD}(-)$.

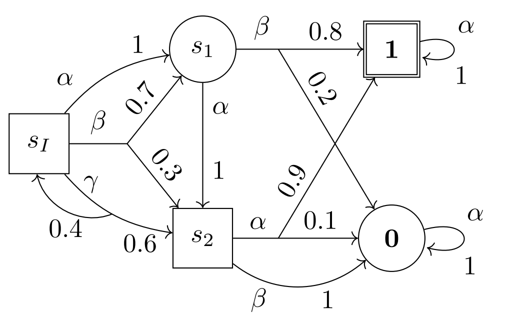
Let $g : X \to 2 \times 2 \times \mathcal{PD}X$ be a SG. Then, $\alpha_X \circ g : X \to 2 \times \mathcal{PD}X$ is a MDP, where all states that used to belong to Minimizer (trying to avoid goal states) now belong to a player trying to reach goal states.

Without surprises, Proposition 5.11 states that $g^*_{\beta_{SG}} \leq (\alpha_X \circ g)^*_{\beta_{MDP}}$, and in particular $V_{SG}(g) \leq V_{MDP}(\alpha_X \circ g)$, i.e.

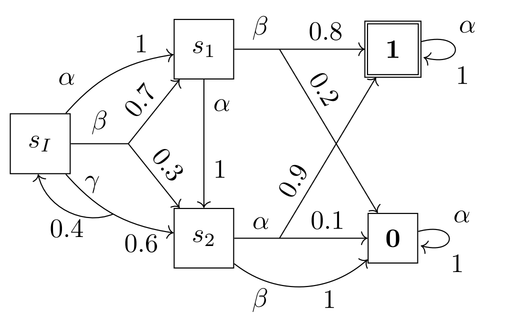
$$\mathbb{P}_{SG,\,g}(\Diamond\mathbf{1}) \leq \mathbb{P}_{MDP,\,\alpha_X\circ g}(\Diamond\mathbf{1}).$$

In other words, giving all states to Maximizer increased the probability of reaching a goal.

More generally, let $g_\uparrow : X \to 2 \times \mathcal{PD}$ be a MDP such that $\alpha_X \circ g \sqsubseteq_{MDP} g_\uparrow$. This corresponds to saying that, while going from g to $g_\uparrow$, all states have been given to Maximizer, and Maximizer may then have been given even more choices at each state. In that case, Proposition 5.11 states that we also have $g^*_{\beta_{SG}} \leq (g_\uparrow)^*_{\beta_{MDP}}$.



$g : X \to 2 \times 2 \times \mathcal{PD}X$



$\alpha_X \circ g : X \to 2 \times \mathcal{PD}X$

$$V_{SG}(g) = \begin{pmatrix} s_I & s_1 & s_2 & \mathbf{0} & \mathbf{1} \\ 0.9 & 0.8 & 0.9 & 0 & 1 \end{pmatrix} \qquad \leq \qquad V_{MDP}(\alpha_X \circ g) = \begin{pmatrix} s_I & s_1 & s_2 & \mathbf{0} & \mathbf{1} \\ 0.9 & 0.9 & 0.9 & 0 & 1 \end{pmatrix}$$

In this example, the single player chooses α at s_1, when Minimizer would have played β.

It is actually unnecessary to ensure that $g_\downarrow$ and $g_\uparrow$ coincide in $\mathcal{T}_*$. More generally, we may simply try to prove that $g_\downarrow$ is “smaller” than $g_\uparrow$, in the following way.

Definition 5.13 (Comparison chains)**.** Let $\mathcal{T}_\downarrow$, $\mathcal{T}_*$ and $\mathcal{T}_\uparrow$ be ppt with shape structure such that $\mathcal{T}_\downarrow \triangleleft_{\mathcal{T}_*} \mathcal{T}_\uparrow$.
For any two processes $g_\downarrow : X \to T_\downarrow Y$ and $g_\uparrow : X \to T_\uparrow Y$, we write

$$g_\downarrow \underset{(g'_\downarrow, \mathcal{T}_*, g'_\uparrow)}{\lhd} g_\uparrow$$

whenever we have a *comparison chain*

$$g_\downarrow \sqsubseteq_\downarrow g'_\downarrow, \quad \alpha_\downarrow \circ g'_\downarrow \sqsubseteq_* \alpha_\uparrow \circ g'_\uparrow, \quad g'_\uparrow \sqsubseteq_\uparrow g_\uparrow,$$

where $g'_\downarrow : X \to T_\downarrow Y$, $g'_\uparrow : X \to T_\downarrow Y$, and $\sqsubseteq_\downarrow, \sqsubseteq_*, \sqsubseteq_\uparrow$ are the shape preorders.
In that case, we say that $g_\downarrow$ and $g_\uparrow$ are *connected*, that $g_\downarrow$ is a *lower approximation* of $g_\uparrow$ and that $g_\uparrow$ is an *upper approximation* of $g_\downarrow$.
To say that such a connection exists, we may also simply write

$$g_\downarrow \lhd_{\mathcal{T}_*} g_\uparrow \quad \text{or} \quad g_\downarrow \lhd g_\uparrow.$$

Example 5.14 (Shape monotonicity between SG and MDP)**.** In the case of SG and MDP (cf. Example 5.10), the formula

$$g_\downarrow \underset{(g'_\downarrow, \mathcal{T}_*, g'_\uparrow)}{\lhd} g_\uparrow$$

means that we go from a SG $g_\downarrow$ to a MDP $g_\uparrow$ through the following steps.

- From $g_\downarrow$ to another SG $g'_\downarrow$, you may remove some of Minimizer's choices ($g_\downarrow \sqsubseteq_{SG} g'_\downarrow$).
- From the SG $g'_\downarrow$ to the MDP $g'_\uparrow$, give all of Minimizer's states to the single player ($\alpha_X \circ g'_\downarrow$), turning Minimizer's deadlocks into goal states.

  You may then give some new options at all states to this single player ($\alpha_X \circ g'_\downarrow \sqsubseteq_{MDP} \mathrm{id}_X \circ g'_\uparrow = g'_\uparrow$).
- Once again, you may give new options to the player ($g'_\uparrow \sqsubseteq_{MDP} g_\uparrow$).

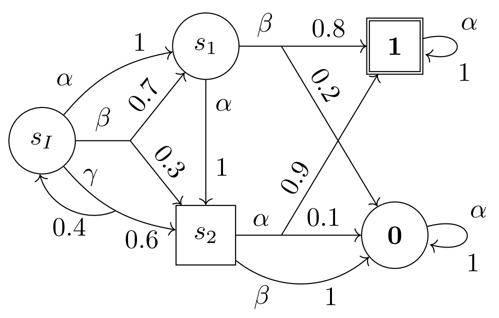
In that case, the following corollary states that

$$(g_\downarrow)^*_{\tau_{SG}} \leq (g_\uparrow)^*_{\tau_{MDP}},$$

and in particular

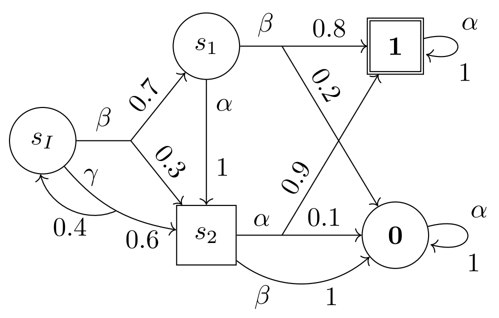
$$V_{SG}(g_\downarrow) \leq V_{MDP}(g_\uparrow).$$

We illustrate this on an example (note that we made Minimizer slightly more powerful in $g_\downarrow$ than in our previous examples).



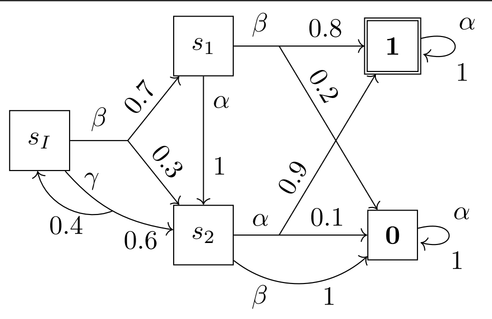
$g_{\downarrow} : X \to 2 \times 2 \times \mathcal{PD}X$

$$V_{SG}(g_{\downarrow}) = \begin{pmatrix} s_I & s_1 & s_2 & \mathbf{0} & \mathbf{1} \\ 0.8 & 0.8 & 0.9 & 0 & 1 \end{pmatrix}$$



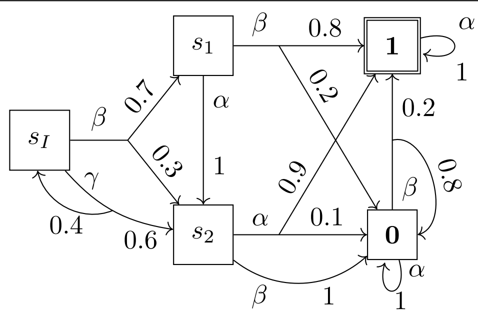
$g'_{\downarrow} : X \to 2 \times \mathcal{PD}X$

$$V_{SG}(g'_{\downarrow}) = \begin{pmatrix} s_I & s_1 & s_2 & \mathbf{0} & \mathbf{1} \\ 0.83 & 0.8 & 0.9 & 0 & 1 \end{pmatrix}$$



$\alpha_X \circ g'_{\downarrow} : X \to 2 \times \mathcal{PD}X$

$$V_{MDP}(\alpha_X \circ g'_{\downarrow}) = \begin{pmatrix} s_I & s_1 & s_2 & \mathbf{0} & \mathbf{1} \\ 0.9 & 0.9 & 0.9 & 0 & 1 \end{pmatrix}$$



$\mathrm{id}_X \circ g'_{\uparrow} = g'_{\uparrow} = g_{\uparrow} : X \to 2 \times \mathcal{PD}X$

$$V_{MDP}(g'_{\uparrow}) = V_{MDP}(g_{\uparrow}) = \begin{pmatrix} s_I & s_1 & s_2 & \mathbf{0} & \mathbf{1} \\ 0.92 & 0.92 & 0.92 & 0.2 & 1 \end{pmatrix}$$

Choices are removed from Min, then states are given to Max, then choices are given to Max.

Corollary 5.15 (Shape-monotonicity between different types)**.** *Let $\mathcal{T}_{\downarrow}$, $\mathcal{T}_{*}$ and $\mathcal{T}_{\uparrow}$ be ppt with shape structure such that $\mathcal{T}_{\downarrow} \lhd_{\mathcal{T}_{*}} \mathcal{T}_{\uparrow}$.*

Let $g_{\downarrow} : X \to T_{\downarrow}Y$, $g_{\uparrow} : X \to T_{\uparrow}Y$ be two processes such that $g_{\downarrow} \lhd g_{\uparrow}$. Then

$$(g_{\downarrow})^{*}_{\tau_{\downarrow}} \leq (g_{\uparrow})^{*}_{\tau_{\uparrow}}.$$

Moreover, if $\mathcal{T}_{\downarrow}$, $\mathcal{T}_{\uparrow}$ have leaf structure and if $g_{\downarrow}$, $g_{\uparrow}$ are coalgebras,

$$V(g_{\downarrow}) \leq V(g_{\uparrow}),$$

and the same holds for greatest fixed points.

Proof. By Proposition 5.1, $(g_{\downarrow})^{*}_{\tau_{\downarrow}} \leq (g'_{\downarrow})^{*}_{\tau_{\downarrow}}$ and $(g'_{\uparrow})^{*}_{\tau_{\uparrow}} \leq (g_{\uparrow})^{*}_{\tau_{\uparrow}}$.

By Proposition 5.11, $(g'_{\downarrow})^{*}_{\tau_{\downarrow}} \leq (\alpha_{\downarrow} \circ g'_{\downarrow})^{*}_{\tau_{*}}$ and $(\alpha_{\uparrow} \circ g'_{\uparrow})^{*}_{\tau_{*}} \leq (g'_{\uparrow})^{*}_{\tau_{\uparrow}}$.

We then conclude by transitivity. □

Remark 5.16 (Upper bounds in value iteration)**.** Corollary 5.15 has strong consequences. Suppose that we are interested in solving a problem which can be described by a process-predicate type $\mathcal{T} = (T, \Omega, \tau)$. An instance of the problem is given by a coalgebra $g : \mathrm{Coalg}(T)$, and the goal is to compute $V(g)$.

From Section 4, we know that $V(g)$ can be computed as the (transfinite) limit of the increasing chain $(g^{\alpha}_{\mathcal{T}}(\bot))_{\alpha:\mathbf{Ord}}$. In particular, this gives us a way of computing a *lower bound* of $V(g)$, and incrementally make it better: this is the technique called *value iteration.*

Corollary 5.15 points toward a way of computing *upper bounds* of $V(g)$. Indeed, suppose that $\mathcal{T}$ has shape structure, and that we can find an upper approximation $\mathcal{T}_{\uparrow}$ of $\mathcal{T}$, i.e. another ppt with shape structure such that $\mathcal{T} \lhd \mathcal{T}_{\uparrow}$ (cf. Definition 5.7).

If we can furthermore compute a $g_\uparrow$: Coalg($T_\uparrow$) such that $g \lhd g_\uparrow$ (cf. Definition 5.13), we will then have $V(g) \leq V(g_\uparrow)$.

If this $V(g_\uparrow)$ can be computed efficiently, e.g. if the upper bound $\mathcal{T}_\uparrow$ enjoys good algorithmic properties, we will have effectively computed an *upper bound* of $V(g)$.

5.3 An approximation theorem in the case of tight connections

As explained in Remark 5.16, thanks to Corollary 5.15, we can obtain upper bounds of least fixed points (and similarly, lower bounds of greatest fixed points) by using connections between ppt with shape structures (Definition 5.7). This is an improvement upon Section 4, where lfp (resp. gfp) could only be computed as suprema (resp. infima) of transfinite sequences, thus giving only lower bounds (resp. upper bounds).

However, the bounds from Section 4 were *converging*, and this is not the case of the bound obtained in Remark 5.16. To obtain convergence from this other direction, we need more than simple *connection*.

The following axiom, exemplified in 5.20 and other applications of Section 6, gives us what we hope.

Definition 5.17 (Tight connection)**.**

Let $\mathcal{T}_\downarrow = (T_\downarrow, \Omega, \tau_\downarrow)$, and $\mathcal{T}_\uparrow = (T_\uparrow, \Omega, \tau_\uparrow)$ be ppt with shape structure such that $\mathcal{T}_\downarrow \lhd \mathcal{T}_\uparrow$.

We say that $\mathcal{T}_\uparrow$ is a *tight overapproximation* of $\mathcal{T}_\downarrow$ whenever for all sets X, for all coalgebras $g_\downarrow : X \to T_\downarrow X$, for all predicates $p : X \to \Omega$, there exists a $g_\uparrow : X \to T_\uparrow X$ such that

$$g_\downarrow \lhd g_\uparrow \quad \text{and} \quad (g_\downarrow)^*_{\tau_\downarrow}(p) = (g_\uparrow)^*_{\tau_\uparrow}(p).$$

Similarly, we say that $\mathcal{T}_\downarrow$ is a *tight underapproximation* of $\mathcal{T}_\uparrow$ whenever for all sets X, for all coalgebras $g_\uparrow : X \to T_\uparrow X$, for all predicates $p : X \to \Omega$, there exists a $g_\downarrow : X \to T_\downarrow X$ such that

$$g_\downarrow \lhd g_\uparrow \quad \text{and} \quad (g_\downarrow)^*_{\tau_\downarrow}(p) = (g_\uparrow)^*_{\tau_\uparrow}(p).$$

Remark 5.18. By Corollary 5.15, we can reformulate Definition 5.17 by saying that $\mathcal{T}_\uparrow$ is a *tight overapproximation* of $\mathcal{T}_\downarrow$ if and only if for all X, $g_\downarrow : X \to T_\downarrow X$ and $p : X \to \Omega$, there exists a $g_\uparrow : X \to T_\uparrow X$ such that

$$g_\downarrow \lhd g_\uparrow \quad \text{and} \quad (g_\downarrow)^*_{\tau_\downarrow}(p) \geq (g_\uparrow)^*_{\tau_\uparrow}(p).$$

Similarly, $\mathcal{T}_\downarrow$ is a *tight underapproximation* of $\mathcal{T}_\uparrow$ if and only if for all X, $g_\uparrow : X \to T_\uparrow X$ and $p : X \to \Omega$, there exists a $g_\downarrow : X \to T_\downarrow X$ such that

$$g_\downarrow \lhd g_\uparrow \quad \text{and} \quad (g_\downarrow)^*_{\tau_\downarrow}(p) \geq (g_\uparrow)^*_{\tau_\uparrow}(p).$$

Since the concept of tight connection gives us one of the central results of this paper, we repackage its definition and explain it more plainly in the following remark.

Remark 5.19 (Tight connection reformulated)**.**

This remark provides an equivalent, although imprecise, definition to that of 5.17.

Let $\mathcal{T}_\downarrow$ and $\mathcal{T}_\uparrow$ be two ppt (representing process types, with additional semantic to model two different problems on the same truth object). We say that $\mathcal{T}_\uparrow$ is a tight overapproximation of $\mathcal{T}_\downarrow$ whenever the following hold.

- We can say when a process $g_\uparrow$ in $\mathcal{T}_\uparrow$ is an "overapproximation" of a $g_\downarrow$ in $\mathcal{T}_\downarrow$, thanks to a *connection* $\mathcal{T}_\downarrow \lhd \mathcal{T}_\uparrow$.

  The definition says that, in that case, $g_\uparrow$ overapproximates $g_\downarrow$ whenever there is some relationship between the branching structures of $g_\downarrow$ and $g_\uparrow$ (check if there is a comparison chain like in Definition 5.13).

  This implies that the predicate transformer $(g_\uparrow)^*_{\tau_\uparrow}$ computes a solution to the problem of $\mathcal{T}_\uparrow$ that is larger than the solution computed by $(g_\downarrow)^*_{\tau_\downarrow}$ to the problem of $\mathcal{T}_\downarrow$.

- Moreover, we ask that for every $g_\downarrow$ in $\mathcal{T}_\downarrow$, there exists an overapproximation $g_\uparrow$ in $\mathcal{T}_\uparrow$, i.e. $g_\downarrow \lhd g_\uparrow$.

  This provides upper bounds to solutions of problems in $\mathcal{T}_\downarrow$.

- Finally, we strengthen the last requirement by asking for overapproximations that are *locally exact*.

  For each $g_\downarrow$ in $\mathcal{T}_\downarrow$ on the state space X and every predicate $p : X \to \Omega$, we need to be able to provide an approximation $g_\uparrow$ such that $(g_\downarrow)^*_{\tau_\downarrow}(p) = (g_\uparrow)^*_{\tau_\uparrow}(p)$, and not only $(g_\downarrow)^*_{\tau_\downarrow}(p) \leq (g_\uparrow)^*_{\tau_\uparrow}(p)$.

  Thus, $g_\uparrow$ is a process in the problem $\mathcal{T}_\uparrow$ such that *no information is lost* with regard to p while going from $g_\downarrow$ to its approximation $g_\uparrow$.

This concept will give us a theorem which enables bounded value iteration (BVI). Before stating the theorem, we note that in the connection $SG \lhd_{MDP} MDP$, MDP is a tight overapproximation of SG when we restrict ourselves to finite branching.

Example 5.20 (Tight connection between SG and MDP)**.** In this example, we provide the necessary constructions to prove that MDP is a tight overapproximation of SG in the case of finite branching.

We still use definitions of Example 5.10, but restrict the functors to *finite branching*: the ppt considered are $(2 \times \mathcal{P}_{\mathrm{f}}\mathcal{D}(-), [0,1], \beta_{MDP})$ and $(2 \times 2 \times \mathcal{P}_{\mathrm{f}}\mathcal{D}(-), [0,1], \beta_{SG})$.

Consider $g_\downarrow : X \to 2 \times 2 \times \mathcal{P}_{\mathrm{f}}\mathcal{D}X$ a *finitely branching* stochastic game on a (possibly infinite) state space X. Let $p : X \to [0,1]$ be a predicate. We want to construct a (finitely branching) MDP $g_\uparrow : X \to 2 \times \mathcal{P}_{\mathrm{f}}\mathcal{D}X$ that overapproximates $g_\downarrow$ (we have $g_\downarrow \lhd g_\uparrow$) such that no information about the behaviour of $g_\uparrow$ on p is lost (we have $(g_\downarrow)^*_{\tau_\downarrow}(p) = (g_\uparrow)^*_{\tau_\uparrow}(p)$).

Construct the MDP $g_\uparrow$ from $g_\downarrow$ by *keeping only Minimizer's moves that are optimal with respect to p.*

$$\begin{aligned}
&g_\uparrow : X \to 2 \times \mathcal{P}_{\mathrm{f}}\mathcal{D}X \\
&\quad x \mapsto \big(\pi_2 g_\downarrow(x), \pi_3 g_\downarrow(x)\big) && \text{if } \pi_1 g_\downarrow(x) = \square \\
&\quad x \mapsto \Big(\pi_2 g_\downarrow(x), \big\{d \in \pi_3 g_\downarrow(x) \,\big|\, \mathbb{E}((\mathcal{D}p)d) = \inf_{d' \in \pi_3 g_\downarrow(x)} \mathbb{E}((\mathcal{D}p)d')\big\}\Big) && \text{if } \pi_1 g_\downarrow(x) = \bigcirc \text{ and } \pi_3 g_\downarrow(x) \neq \varnothing \\
&\quad x \mapsto \big(\top, \varnothing\big) && \text{if } \pi_1 g_\downarrow(x) = \bigcirc \text{ and } \pi_3 g_\downarrow(x) = \varnothing
\end{aligned}$$

Notice how, indeed, no information has been lost with respect to p. We prove that $\tau_\downarrow \circ T_\downarrow p \circ g_\downarrow = \tau_\uparrow \circ T_\uparrow p \circ g_\uparrow$.

Let $x \in X$. If x is a goal state, $(g_\downarrow)^*_{\tau_\downarrow}(p)(x) = 1 = (g_\uparrow)^*_{\tau_\uparrow}(p)(x)$.

Otherwise, if x belongs to Max, $(g_\downarrow)^*_{\tau_\downarrow}(p)(x) = \sup_{d \in \pi_3 g_\downarrow(x)} \mathbb{E}((\mathcal{D}p)d) = (g_\uparrow)^*_{\tau_\uparrow}(p)(x)$.

If x is a deadlock for Min, $(g_\downarrow)^*_{\tau_\downarrow}(p)(x) = \inf \varnothing = 1 = (g_\uparrow)^*_{\tau_\uparrow}(p)(x)$. Finally, if x belongs to Min with $\pi_3 g_\downarrow(x) \neq \varnothing$, $(g_\downarrow)^*_{\tau_\downarrow}(p)(x) = \inf_{d \in \pi_3 g_\downarrow(x)} \mathbb{E}((\mathcal{D}p)d) = \inf_{d \in \pi_2 g_\uparrow(x)} \mathbb{E}((\mathcal{D}p)d) = (g_\uparrow)^*_{\tau_\uparrow}(p)(x)$.

Notice that in the last case, we have used the fact that $\pi_3 g_\downarrow(x)$ is finite to prove that there exists an optimal choice $d \in \pi_3 g_\downarrow(x)$ such that $\mathbb{E}((\mathcal{D}p)d) = \inf_{d' \in \pi_3 g_\downarrow(x)} \mathbb{E}((\mathcal{D}p)d')$.

By construction, we do have $g_\downarrow \lhd g_\uparrow$. Indeed, we have the comparison chain

$$g_\downarrow \sqsubseteq_{SG} g'_\downarrow, \quad \alpha_X \circ g'_\downarrow = \mathrm{id}_x \circ g_\uparrow = g_\uparrow,$$

where we only used Minimizer restriction by going from $g_\downarrow$ to

$$\begin{aligned}
&g'_\downarrow : X \to 2 \times 2 \times \mathcal{P}_{\mathrm{f}}\mathcal{D}X \\
&\quad x \mapsto g_\downarrow(x) && \text{if } \pi_1 g_\downarrow(x) = \square \\
&\quad x \mapsto \Big(\pi_1 g_\downarrow(x), \pi_2 g_\downarrow(x), \big\{d \in \pi_3 g_\downarrow(x) \,\big|\, \mathbb{E}((\mathcal{D}p)d) = \inf_{d' \in \pi_3 g_\downarrow(x)} \mathbb{E}((\mathcal{D}p)d')\big\}\Big) && \text{if } \pi_1 g_\downarrow(x) = \bigcirc.
\end{aligned}$$

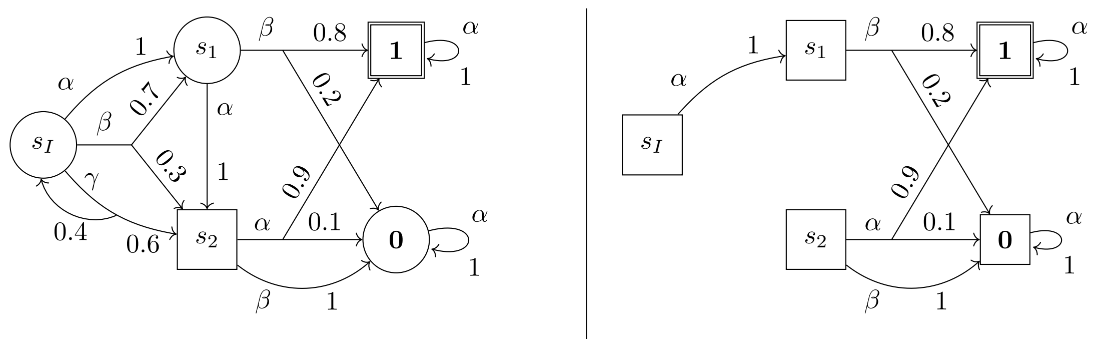
Thus, MDP_{f} is a tight overapproximation of SG_{f}. □

In the following example, we show what this locally lossless approximation looks like in the cases

$$p = V_{SG}(g_\downarrow) = \begin{pmatrix} s_I & s_1 & s_2 & \mathbf{0} & \mathbf{1} \\ 0.8 & 0.8 & 0.9 & 0 & 1 \end{pmatrix} \quad \text{and} \quad p = (g_\downarrow)^2_{\tau_\downarrow}(\bot) = \begin{pmatrix} s_I & s_1 & s_2 & \mathbf{0} & \mathbf{1} \\ 0 & 0.8 & 0.9 & 0 & 1 \end{pmatrix}.$$

We can interpret this setting in the following way. In the first case, Minimizer has understood everything about Maximizer's strategy, knows what the consequences of its actions will be even after ω time steps, and thus really knows what its optimal choices are.

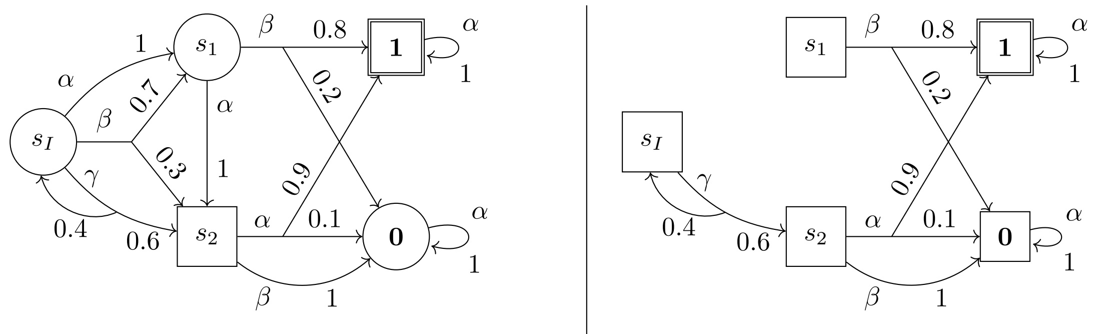
In the second case, however, Minimizer has only looked at the consequences of his actions with a horizon of 2 time steps. Minimizer will still try to follow an optimal strategy, but will only be able to do so based on his limited knowledge. Here, Minimizer will still believe that choosing γ in s_I is a viable strategy for staying in an infinite loop.



$g_{\downarrow} : X \to 2 \times 2 \times \mathcal{PD}X$ $\qquad$ $g_{\uparrow} : X \to 2 \times \mathcal{PD}X$

$$(g_{\downarrow})^*_{\beta_{SG}}(V_{SG}(g_{\downarrow})) = V_{SG}(g_{\downarrow}) = \left(\begin{array}{ccccc} s_I & s_1 & s_2 & \mathbf{0} & \mathbf{1} \\ 0.8 & 0.8 & 0.9 & 0 & 1 \end{array}\right) \quad = \quad (g_{\uparrow})^*_{\beta_{MDP}}(V_{SG}(g_{\downarrow})) = \left(\begin{array}{ccccc} s_I & s_1 & s_2 & \mathbf{0} & \mathbf{1} \\ 0.8 & 0.8 & 0.9 & 0 & 1 \end{array}\right)$$

Overapproximation of $g_{\downarrow}$ exact at $p = V_{SG}(g_{\downarrow})$. Only Min's optimal choices (with respect to $V_{SG}(g_{\downarrow})$) are kept.



$g_{\downarrow} : X \to 2 \times 2 \times \mathcal{PD}X$ $\qquad$ $g_{\uparrow} : X \to 2 \times \mathcal{PD}X$

$$(g_{\downarrow})^*_{\beta_{SG}}\big((g_{\downarrow})^2_{\beta_{SG}}(\bot)\big) = \left(\begin{array}{ccccc} s_I & s_1 & s_2 & \mathbf{0} & \mathbf{1} \\ 0.54 & 0.8 & 0.9 & 0 & 1 \end{array}\right) \quad = \quad (g_{\uparrow})^*_{\beta_{MDP}}\big((g_{\downarrow})^2_{\beta_{SG}}(\bot)\big) = \left(\begin{array}{ccccc} s_I & s_1 & s_2 & \mathbf{0} & \mathbf{1} \\ 0.54 & 0.8 & 0.9 & 0 & 1 \end{array}\right)$$

$$V_{SG}(g_{\downarrow}) = \left(\begin{array}{ccccc} s_I & s_1 & s_2 & \mathbf{0} & \mathbf{1} \\ 0.8 & 0.8 & 0.9 & 0 & 1 \end{array}\right) \quad \leq \quad V_{MDP}(g_{\uparrow}) = \left(\begin{array}{ccccc} s_I & s_1 & s_2 & \mathbf{0} & \mathbf{1} \\ 0.9 & 0.8 & 0.9 & 0 & 1 \end{array}\right)$$

Overapproximation of $g_{\downarrow}$, where only Min's optimal choices (with respect to $(g_{\downarrow})^2_{\beta_{SG}}(\bot)$) are kept.

Notice that computing the overapproximation (exact at p) $g_{\downarrow}$ from $g_{\uparrow}$ and p was very simple in our case: it could be done in linear time in the size of the MDP (for finite state spaces). This is encouraging, and shows that the tight connection $SG_{\mathrm{f}} \lhd MDP_{\mathrm{f}}$ can be used in applications. It is indeed used, e.g. in [1], where it is also composed with a connection between MDP and WG.

This emphasises an important point about our theory. Finding a tight connection tells us what objects can be used for applications such as BVI. However, the theory doesn't provide a constructive, fast algorithm to compute those objects: this is still up to the implementer. Our order theory is *structural*, but not *algorithmic*.

Remark 5.21 (Local computation of truth values, local approximation)**.**

We could axiomatise the fact that locally exact upper bounds are computed *locally*, to motivate the fact that they can be computed efficiently.

Up until now, we only used *local computation of truth values*, using

$$\begin{array}{rccccc} \tau \circ T(-): & \Omega^X & \to & TX & \to & \Omega \\ & p & \mapsto & (t & \mapsto & (\tau \circ Tp)(t)) \end{array}$$

which gave us our predicate transformers (written in currified form)

$$\begin{array}{rccccc} (-)^*_\tau(-): & \Omega^X & \to & (TX)^X & \to & \Omega^X \\ & p & \mapsto & (g & \mapsto & \tau \circ Tp \circ g) \end{array}$$

which compute local propagation of truth values, and can be studied with leaf structure.

Similarly, in the case $\mathcal{T}_\uparrow$ a tight overapproximation of $\mathcal{T}_\downarrow$, we could ask for *local approximation of processes*, using a

$$\varphi : \Omega^X \to T_\downarrow X \to T_\uparrow X$$

such that

$$\forall p : \Omega^X,\ \forall t : T_\downarrow X,\ \big(t \triangleleft \varphi(p)(t)\ \wedge\ (\tau_\downarrow \circ T_\downarrow p)(t) = (\tau_\uparrow \circ T_\uparrow p \circ \varphi(p))(t)\big).$$

This would give us a

$$\begin{array}{rccccc} \Phi: & \Omega^X & \to & (T_\downarrow X)^X & \to & (T_\uparrow X)^X \\ & p & \mapsto & (g_\downarrow & \mapsto & \varphi(p) \circ g_\downarrow) \end{array}$$

computing overapproximations $\Phi(p, g_\downarrow) : X \to T_\uparrow X$ of $g_\downarrow$ exact at p.

Notice that the construction above for SG/MDP was local, with

$$\begin{aligned} \varphi : \Omega^X \to 2 \times 2 \times \mathcal{P}_\mathrm{f}\mathcal{D}X &\to 2 \times \mathcal{P}_\mathrm{f}\mathcal{D}X \\ (p, (g, \square, t)) &\mapsto (g, t) \\ (p, (g, \bigcirc, \varnothing)) &\mapsto (\top, \varnothing) \\ (p, (g, \bigcirc, t)) &\mapsto \Big(g, \big\{d \in t \,\big|\, \mathbb{E}((\mathcal{D}p)d) = \inf_{d' \in t} \mathbb{E}((\mathcal{D}p)d')\big\}\Big) \quad \text{if } t \neq \varnothing. \end{aligned}$$

□

The following theorem gives us sequences converging to fixed point from both below and above.

Theorem 5.22 (Double approximation theorem)**.**
Let $\mathcal{T}_\downarrow = (T_\downarrow, \Omega, \tau_\downarrow)$, and $\mathcal{T}_\uparrow = (T_\uparrow, \Omega, \tau_\uparrow)$ be ppt with both shape and leaf structure such that $\mathcal{T}_\downarrow \triangleleft \mathcal{T}_\uparrow$.
This supposes that the complete lattice Ω is shared between the shape and leaf structures of both ppt.
Moreover, suppose that $\mathcal{T}_\uparrow$ is a tight *overapproximation of $\mathcal{T}_\downarrow$.*
Let $g_\downarrow : X \to T_\downarrow X$ be a coalgebra. For each ordinal α, we can define $p_\alpha = (g_\downarrow)^\alpha_{\tau_\downarrow}(\bot)$, and choose $g_\alpha : X \to T_\uparrow X$ with $g_\downarrow \triangleleft g_\alpha$ such that $(g_\downarrow)^*_{\tau_\downarrow}(p_\alpha) = (g_\alpha)^*_{\tau_\uparrow}(p_\alpha)$. Then,

$$V(g_\downarrow) := \mathrm{lfp}\big((g_\downarrow)^*_{\tau_\downarrow}\big) = \sup_{\alpha:\mathbf{Ord}} p_\alpha = \inf_{\alpha:\mathbf{Ord}} V(g_\alpha).$$

Furthermore, the convergence time of $(V(g_\alpha))_{\alpha:\mathbf{Ord}}$ is smaller than the convergence time of $(p_\alpha)_{\alpha:\mathbf{Ord}}$.

Similarly, suppose that $\mathcal{T}_\downarrow$ is a tight *underapproximation of $\mathcal{T}_\uparrow$.*
Let $g_\uparrow : X \to T_\uparrow X$ be a coalgebra. For each ordinal α, we can define $p_\alpha = (g_\uparrow)^\alpha_{\tau_\uparrow}(\top)$, and choose $g_\alpha : X \to T_\downarrow X$ with $g_\alpha \triangleleft g_\uparrow$ such that $(g_\alpha)^*_{\tau_\downarrow}(p_\alpha) = (g_\uparrow)^*_{\tau_\uparrow}(p_\alpha)$. Then,

$$\mathrm{gfp}\big((g_\uparrow)^*_{\tau_\uparrow}\big) = \inf_{\alpha:\mathbf{Ord}} p_\alpha = \sup_{\alpha:\mathbf{Ord}} \mathrm{gfp}\big((g_\alpha)^*_{\tau_\downarrow}\big).$$

Furthermore, the convergence time of $(V(g_\alpha))_{\alpha:\mathbf{Ord}}$ is smaller than the convergence time of $(p_\alpha)_{\alpha:\mathbf{Ord}}$.

Proof. We only have to prove the first result, the other case is symmetric.
Suppose that $\mathcal{T}_\uparrow$ is a *tight* overapproximation of $\mathcal{T}_\downarrow$, and let $g_\downarrow : X \to T_\downarrow X$ be a coalgebra.
Define $p_\alpha = (g_\downarrow)^\alpha_{\tau_\downarrow}(\bot)$, such that $V(g_\downarrow) = \sup p_\alpha$ by Theorem 4.1.
For each α, let $g_\alpha : X \to T_\uparrow X$ such that $g_\downarrow \triangleleft g_\alpha$ and $(g_\downarrow)^*_{\tau_\downarrow}(p_\alpha) = (g_\alpha)^*_{\tau_\uparrow}(p_\alpha)$.

- By Corollary 5.15, we already know that for each α, $V(g_\downarrow) \leq V(g_\alpha)$, thus
$$V(g_\downarrow) \leq \inf_{\alpha:\mathbf{Ord}} V(g_\alpha).$$

- Let α_{lim} be the convergence time of (p_α), and let $\alpha \geq \alpha_{lim}$. We have $p_\alpha = V(g_\downarrow)$, thus
$$(g_\alpha)^*_{\tau_\uparrow}(p_\alpha) = (g_\downarrow)^*_{\tau_\downarrow}(p_\alpha) = (g_\downarrow)^*_{\tau_\downarrow}(V(g_\downarrow)) = V(g_\downarrow) = p_\alpha.$$
Thus, p_α is a fixed point of $(g_\alpha)^*_{\tau_\uparrow}$!
This implies that the least fixed point $V(g_\alpha)$ of $(g_\alpha)^*_{\tau_\uparrow}$ is smaller than $p_\alpha = V(g_\downarrow)$.
$$V(g_\downarrow) \geq V(g_\alpha) \geq \inf_{\alpha:\mathbf{Ord}} V(g_\alpha).$$

□

Notice what this gives us on our running example.

Example 5.23 (Theorem 5.22 for $SG \lhd MDP$)**.** Let $g_\downarrow$ be a SG, $p_\alpha = (g_\downarrow)^\alpha_{\beta_{SG}}(\bot)$ be our usual chain of lower bounds of $V_{SG}(g_\downarrow)$, and for each α let g_α be a MDP, like in Example 5.20 overapproximating $g_\downarrow$ with no loss of information at p_α.

Then,
$$\begin{aligned} V_{SG}(g_\downarrow) &= \sup_{\alpha:\mathbf{Ord}} p_\alpha \\ &= \inf_{\alpha:\mathbf{Ord}} V_{MDP}(g_\alpha) = \inf_{\alpha:\mathbf{Ord}} \sup_{\beta:\mathbf{Ord}} (g_\alpha)^\beta_{\beta_{MDP}}(\bot). \end{aligned}$$

The sequence $(\mathbb{P}_{MDP,\, g_\alpha}(\Diamond \mathbf{1}))_\alpha$ of value functions for MDPs converges from above to the value function $\mathbb{P}_{SG,\, g_\downarrow}(\Diamond \mathbf{1})$ of the SG that interests us.

Notice how this gives us a form of BVI for reachability in stochastic games.

- Take the SG $g_\downarrow$ as input, initialise $p_0 = \bot$ and the upper bound at $\top$.
- At each inductive step $\alpha + 1$,
  - Compute a new lower bound $p_{\alpha+1} = (g_\downarrow)^*_{\beta_{SG}}(p_\alpha)$.
  - Compute the MDP $g_{\alpha+1}$ like in Example 5.20.
  - Compute a new upper bound $V_{MDP}(g_{\alpha+1})$.
- If a limit case $\lambda = \sup_{\alpha<\lambda}$ is needed, be prepared to
  - Compute a new lower bound $p_\lambda = \sup_{\alpha<\lambda} p_\alpha$.
  - Compute a new upper bound $V_{MDP}(g_\lambda)$ like before.
- The chain $(p_\alpha)_\alpha$ converges from below to $V_{SG}(g_\downarrow)$, and $(V_{MDP}(g_\alpha))_\alpha$ converges from above to $V_{SG}(g_\downarrow)$.
  Thus, we know the value of $V_{SG}(g_\downarrow)$ with precision ϵ whenever $|V_{MDP}(g_\alpha) - p_\alpha|_\infty \leq \epsilon$.

Just like in Example 5.20, g_α in a MDP obtained by fixing the strategy of Minimizer on $g_\downarrow$, with a strategy converging to the optimal strategy.

This BVI algorithm has some merits, but there are some obvious shortcomings.

- We may have to deal with transfinite induction (but it can be shown that this only appears on infinite state spaces).
- We have to compute many V_{MDP} to compute a single V_{SG}. This supposes that we have a great algorithm for MDP that didn't work for SG.

This last point shows that such BVI algorithm mainly works if we are able to connect our problem (here reachability in SG) to a problem that is very simple, such as one that can be solved by graph analysis.

Remark 5.24 (Convergence speeds)**.** In the theorem above, notice that the quantity $V(g_\downarrow)$, which we want to compute, is equal to both $\sup_{\alpha:\mathbf{Ord}} p_\alpha$ and $\inf_{\alpha:\mathbf{Ord}} V(g_\alpha)$.

If $(p_\alpha)_\alpha$ and $(V(g_\alpha))_\alpha$ have the same convergence speed, this simply provides a BVI algorithm.

However, we might imagine that $(V(g_\alpha))_\alpha$ converges *faster* than $(p_\alpha)_\alpha$. In this situation, the relationship $V(g_\downarrow) = \inf_{\alpha:\mathbf{Ord}} V(g_\alpha)$ can provide an iterative algorithm to compute $V(g_\downarrow)$ that is actually faster than the VI algorithm provided by the relationship $V(g_\downarrow) = \sup_{\alpha:\mathbf{Ord}} p_\alpha$. Of course, this requires fast algorithms to compute each approximation g_α and each value function $V(g_\alpha)$.

Interestingly, in the case of MDP and SG on finite state spaces, we *do* have the relation

$$\text{convergence\_time\_of}\Big(\big(V_{MDP}(g_\alpha)\big)_\alpha\Big) < \omega,$$

shown in Lemma 4.5 of [1], while it is possible to have

$$\text{convergence\_time\_of}\Big(\big((g_\downarrow)^\alpha_{\beta_{SG}}(\bot)\big)_{\alpha:\mathbf{Ord}}\Big) = \omega.$$

In other words, we need only a *finite* number of approximations by MDPs g_α, while we may have needed an *infinite* number of lower bounds p_α.

Notice that we are not saying that each $V_{MDP}(g_\alpha)$ can be computed quickly, but only that the *chain* of all such values converges quickly.

We may wonder if the inequality

$$\text{convergence\_time\_of}\Big(\big(V_\uparrow(g_\alpha)\big)_\alpha\Big) < \text{convergence\_time\_of}\Big(\big((g_\downarrow)^\alpha_{\mathcal{T}_\downarrow}(\bot)\big)_{\alpha:\mathbf{Ord}}\Big)$$

holds in the general case. Unfortunately, the following example shows that it is not true.

Proposition 5.25 (Slow convergence of approximations)**.** *In general, the inequality*

$$\text{convergence\_time\_of}\Big(\big(V_\uparrow(g_\alpha)\big)_\alpha\Big) \leq \text{convergence\_time\_of}\Big(\big((g_\downarrow)^\alpha_{\mathcal{T}_\downarrow}(\bot)\big)_{\alpha:\mathbf{Ord}}\Big)$$

may be saturated.

More precisely, for any ordinal $\alpha_{lim} > 0$, there exists a connection $\mathcal{T}_\downarrow \lhd \mathcal{T}_\uparrow$ such that $\mathcal{T}_\uparrow$ is a tight overapproximation of $\mathcal{T}_\downarrow$ and such that there exists a coalgebra $g_\downarrow$ on $T_\downarrow$ such that

- *The convergence time of the chain $(p_\alpha)_{\alpha:\mathbf{Ord}}$ defined by $p_\alpha = (g_\downarrow)^\alpha_{\mathcal{T}_\downarrow}(\bot)$ is equal to α_{lim}.*
- *There exists a set of coalgebras $(g_\alpha)_{\alpha\leq\alpha_{lim}}$ on $T_\uparrow$ which is such that, for each α we have $g_\downarrow \lhd g_\alpha$ and $(g_\downarrow)^*_{\mathcal{T}_\downarrow}(p_\alpha) = (g_\alpha)^*_{\mathcal{T}_\uparrow}(p_\alpha)$, and which verifies that*

$$\text{convergence\_time\_of}\Big(\big(V_\uparrow(g_\alpha)\big)_\alpha\Big) = \alpha_{lim}.$$

Proof. The counter-example consists of taking the NTS of Example 4.4 about late convergence, adding a loop on the highest state, and comparing the problems of (un)safety and reachability.

Details are provided below.

Consider the following example, similar in nature to MDP/SG without stochasticity.

Let $\mathcal{T}_\downarrow = (2 \times \mathcal{P}, \text{Bool}, 1 + \inf)$ and $\mathcal{T}_\uparrow = (2 \times \mathcal{P}, \text{Bool}, 1 + \sup)$ be the ppt with shape structures given by reverse inclusion for $\mathcal{T}_\uparrow$, and inclusion for $\mathcal{T}_\downarrow$.

$\mathcal{T}_\downarrow$ can be interpreted as the problem of (un)safety in a non-deterministic transition system with goals (NTS), and $\mathcal{T}_\uparrow$ as the reachability problem on the same process-type.

The proof that they are both ppt with shape structures is similar to the corresponding fact for MDP and SG. Both ppt can be given leaf structure based on Egli-Milner preorder using Theorem 4.5.

Like in MDP/SG, we have $\mathcal{T}_\downarrow \lhd_{\mathcal{T}_\uparrow} \mathcal{T}_\uparrow$, through the use of the natural transformation α defined by the maps

$$\begin{aligned} \alpha_X : 2 \times \mathcal{P}X &\to 2 \times \mathcal{P}X \\ (g, \varnothing) &\mapsto (\top, \varnothing) \\ (g, t) &\mapsto (g, t) \quad \text{if } t \neq \varnothing. \end{aligned}$$

Using the finiteness of the complete total order Bool, we can easily show that $\mathcal{T}_\uparrow$ is a tight overapproximation of $\mathcal{T}_\downarrow$. Indeed, for any $g_\downarrow : X \to 2 \times \mathcal{P}X$ and any $p : X \to \text{Bool}$, we may define

$$\begin{aligned} g'_\downarrow : X &\to 2 \times \mathcal{P}X \\ x &\mapsto \Big(\pi_1 g_\downarrow(x), \big\{y \in \pi_2 g_\downarrow(x) \,\big|\, p(y) = \inf_{z \in \pi_2 g_\downarrow(x)} p(z)\big\}\Big), \end{aligned}$$

and

$$\begin{aligned} g_\uparrow := \alpha_X \circ g'_\downarrow : X &\to 2 \times \mathcal{P}X \\ x &\mapsto \big(\top, \varnothing\big) && \text{if } \pi_2 g_\downarrow(x) = \varnothing \\ x &\mapsto \Big(\pi_1 g_\downarrow(x), \big\{y \in \pi_2 g_\downarrow(x) \,\big|\, p(y) = \inf_{z \in \pi_2 g_\downarrow(x)} p(z)\big\}\Big) && \text{if } \pi_2 g_\downarrow(x) \neq \varnothing. \end{aligned}$$

With those definitions, we can easily check that $(g_\downarrow)^*_{\mathcal{T}_\downarrow}(p) = (g_\uparrow)^*_{\mathcal{T}_\uparrow}(p)$.

We can finally get to our example of process badly approached by approximations, which behaves similarly to Example 2.7. Let $X = [0, \alpha_{lim}]$, and

$$\begin{aligned} g_\downarrow : X &\to 2 \times \mathcal{P}X \\ 0 &\mapsto (\top, \{0\}) \\ \alpha &\mapsto (\bot, [0, \alpha)) \quad \text{if } 0 < \alpha < \alpha_{lim} \\ \alpha_{lim} &\mapsto (\bot, X). \end{aligned}$$

It is clear that 0 can be avoided only from α_{lim}, by doing an infinite loop. Thus, $V_\downarrow(g_\downarrow) = \mathbf{1}_{[0,\alpha_{lim})}$. Moreover, for each $\alpha \le \alpha_{lim}$, $p_\alpha := (g_\downarrow)^\alpha_{\mathcal{T}_\downarrow}(\bot) = \mathbf{1}_{[0,\alpha)}$. Thus,

$$\text{convergence\_time\_of}\big((p_\alpha)_\alpha\big) = \alpha_{lim}.$$

Now, for each $0 \le \alpha < \alpha_{lim}$, if we construct the approximation $g \lhd g_\alpha$ like described above, we obtain

$$\begin{aligned} g_\alpha : X &\to 2 \times \mathcal{P}X \\ 0 &\mapsto (\top, \{0\}) \\ \beta &\mapsto (\bot, [0, \beta)) \quad \text{if } 0 < \beta \le \alpha \\ \beta &\mapsto (\bot, [\alpha, \beta)) \quad \text{if } \alpha < \beta < \alpha_{lim} \\ \alpha_{lim} &\mapsto (\bot, [\alpha, \alpha_{\text{lim}}]), \end{aligned}$$

thus $V_\uparrow(g_\alpha) = \mathbf{1}_X = \top$. It is easy to check that, indeed, $(g_\alpha)^*_{\mathcal{T}_\uparrow}(p_\alpha) = (g_\downarrow)^*_{\mathcal{T}_\downarrow}(p_\alpha)$.

Put in words, if $\beta \le \alpha$, Minimizer has already realised that he can only lose, but if $\alpha < \beta < \alpha_{lim}$, Minimizer still wrongly believes that there is a way out, because he hasn't realised that α is a state from which we will necessarily reach 0.

We only reach $V_\downarrow(g_\downarrow) = \mathbf{1}_{[0,\alpha_{lim})}$ at step α_{lim}, where

$$\begin{aligned} g_{\alpha_{lim}} : X &\to 2 \times \mathcal{P}X \\ 0 &\mapsto (\top, \{0\}) \\ \beta &\mapsto (\bot, [0, \beta)) \quad \text{if } 0 < \beta < \alpha_{lim} \\ \alpha_{lim} &\mapsto (\bot, \{\alpha_{\text{lim}}\}). \end{aligned}$$

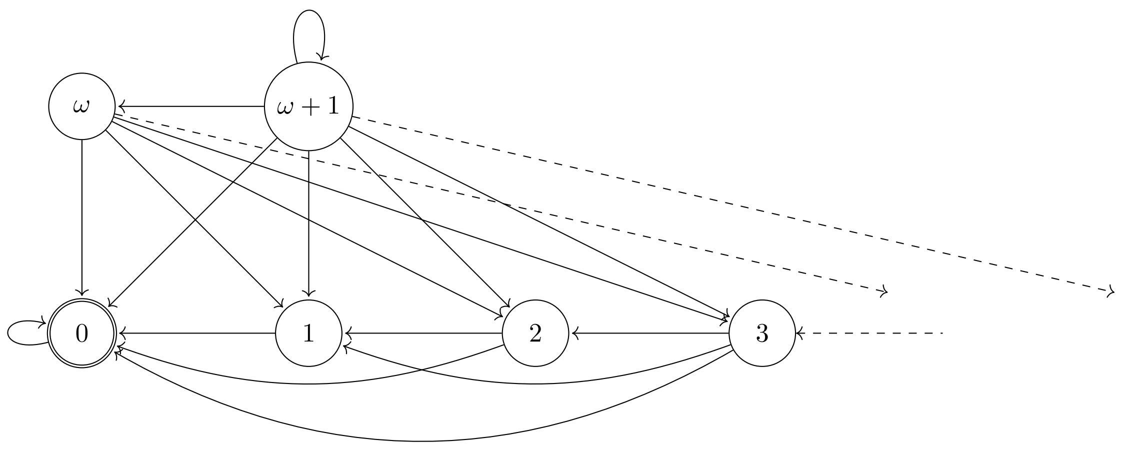
Minimizer finally gives up everywhere but at the maximal state, and we have $V_\uparrow(g_{\alpha_{lim}}) = \mathbf{1}_{[0,\alpha_{lim})} = V_\downarrow(g_\downarrow)$, as expected from Theorem 5.22.

Thus,

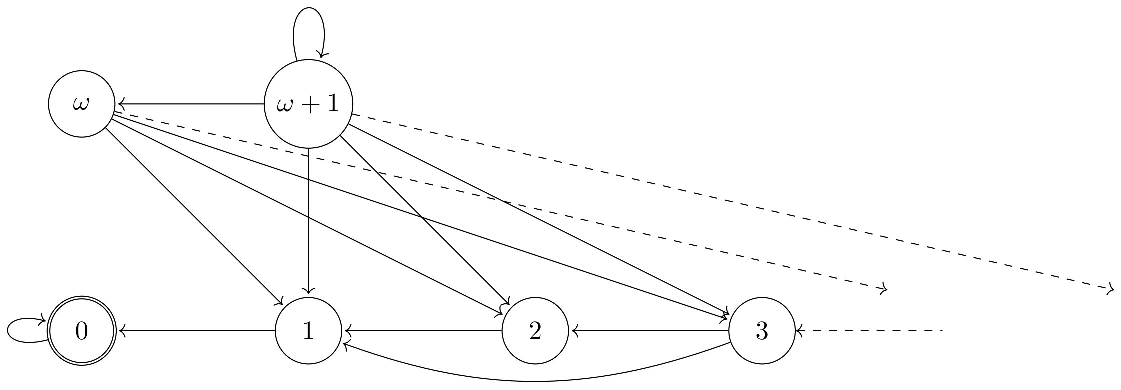
$$\text{convergence\_time\_of}\Big(\big(V_\uparrow(g_\alpha)\big)_\alpha\Big) = \alpha_{lim} = \text{convergence\_time\_of}\big((p_\alpha)_\alpha\big).$$

□

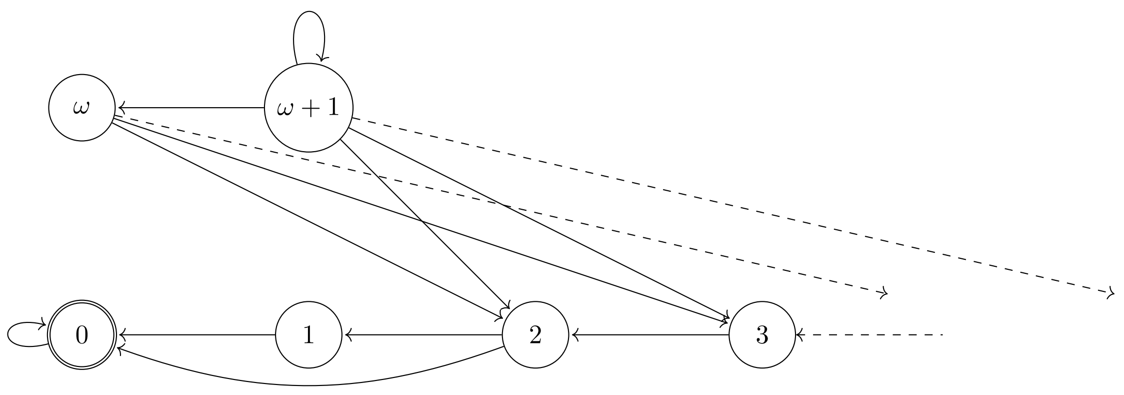
Example 5.26. We illustrate the proof above in the case $\alpha_{lim} = \omega + 1$, by showing $g_\downarrow$ and some of the g_α.



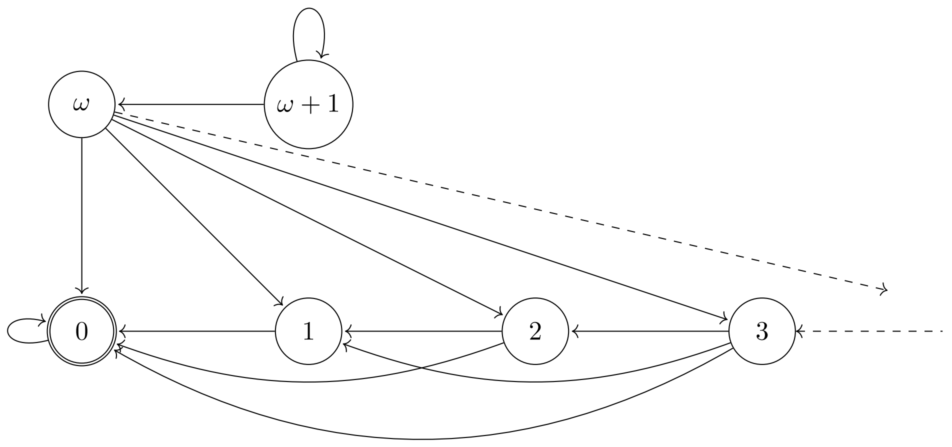
The process $g_\downarrow$ when $\alpha_{lim} = \omega + 1$. Only $\omega + 1$ can avoid 0. For $\alpha \leq \omega + 1$, $p_\alpha = \mathbf{1}_{[0,\alpha)}$.



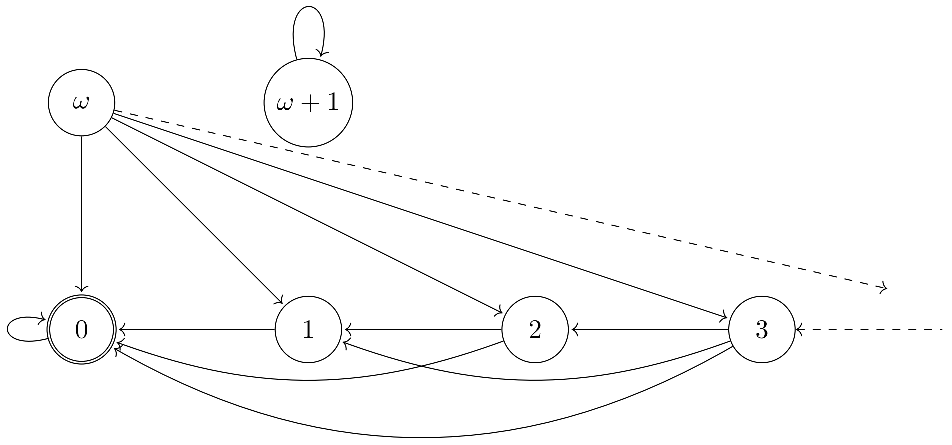
The process g_1. All states can reach 0. Minimizer made the mistake of thinking that 1 was a safe spot.



The process g_2. All states can reach 0. Minimizer made the mistake of thinking that 2 was a safe spot.



The process g_ω. All states can reach 0. Minimizer made the mistake of thinking that ω was a safe spot.



The process $g_{\omega+1}$. 0 is unreachable from $\omega+1$. Minimizer finally made no mistake.

Remark 5.27. It is interesting to note that in the above example, while the convergence of $\big(V_\uparrow(g_\alpha)\big)_\alpha$ is slow, the computation of each $V_\uparrow(g_\alpha)$ is very fast.

Indeed, $V_\uparrow(g_\alpha) = \sup_{\beta:\mathbf{Ord}}(g_\alpha)^\beta_{\tau_\uparrow}(\bot)$, where convergence_time_of$\Big(\big((g_\alpha)^\beta_{\tau_\uparrow}(\bot)\big)_\beta\Big) \leq 2$ (in the worst case, go from $\beta > \alpha$ to α in one step, and then from α to 0 in a second step).

This contrasts with the case of SG / MDP on finite state spaces, where each $V_\uparrow(g_\alpha)$ can take ω steps to be computed, but $(V_\uparrow(g_\alpha))_{\alpha:\mathbf{Ord}}$ converges in finitely many steps (Cf. Lemma 4.5 of [1]).

The main part of this report has now been completed. We have seen in Section 4 how to axiomatise properties making VI iteration work, and Section 5 just explained how connections could be found between models to allow BVI. This point is made explicit in Section 6.3.

For the moment, we have mainly illustrated our discussion with the examples of SG and MDP. In Section 6, new examples are provided, and an opening towards "base model / surrogate model" situations not yet covered by our theory is discussed.

6 Examples and applications

This section provides material that can be used to illustrate the theory developed in this paper.

In Section 6.1, several examples of *process-predicate type with leaf and/or shape structure* are given. They model problems on SG, MDP, MC (Section 6.1.1), on (in)finitely branching transition systems (Section 6.1.2), and on (un)weighted graphs (Section 6.1.3).

Section 6.2 then provides three examples of *connections*.

- Reachability in SG and MDP.
- Reachability in MC and MDP.
- Safety in finitely branching and infinitely branching transition systems.

A few applications of the theory developed in this paper are summed up in Section 6.3. We have obtained a *recipe* to design VI and BVI algorithms: we have obtained sufficient conditions for their construction.

Our BVI algorithms use surrogate models, but we have only used one particular kind of recipe. While the idea of using surrogate models in model-checking is still new, other examples exist. Those future line of research are discussed in Section 6.4, where we talk about what might be needed to make our theory able to deal with those new kinds of connections.

6.1 Examples of ppt with leaf and shape structure

In Section 6.1, we provide a short list of examples of problems that can be described as process-predicate type. For each example,

- We give a ppt representing the problem,
- We explain what leaf structure can be used to prove that we have a VI algorithm to solve the problem,
- If useful, we describe a shape structure to be used for connections (Section 6.2).

6.1.1 SG, MDP, MC

Example 6.1 ($\mathbb{P}(\Diamond \mathbf{1})$ in SG)**.**

- **Ppt**. $(2 \times 2 \times \mathcal{PD}(-), [0,1], \beta)$, using lfp, where

$$\begin{aligned} \beta : 2 \times 2 \times \mathcal{PD}\Omega &\to \Omega \\ (p, \top, t) &\mapsto 1 \\ (\Box, \bot, t) &\mapsto \sup_{d \in t} \mathbb{E}(d) \\ (\bigcirc, \bot, t) &\mapsto \inf_{d \in t} \mathbb{E}(d) \end{aligned}$$

- **Leaf structure**. Egli-Milner preorder and tailwise preorder.

- **Shape structure**. Reverse inclusion on Min's choices.

Proof. There is nothing to do to prove that we have described a process-predicate type.

The fact that $\mathbb{P}(\lozenge\mathbf{1})$ is computed as the lfp can be seen as a *definition* of $\mathbb{P}(\lozenge\mathbf{1})$. Proofs that this coincides with other definitions can be found in [49].

The proofs that we have really obtained leaf structure and shape structure is detailed in Examples 4.7 and 5.4. □

Example 6.2 ($\mathbb{P}(\lozenge\mathbf{1})$ in MDP)**.**

- **Ppt**. $(2 \times \mathcal{PD}(-), [0,1], \beta)$, using lfp, where

$$\begin{aligned} \beta : 2 \times \mathcal{PD}\Omega &\to \Omega \\ (\top, t) &\mapsto 1 \\ (\bot, t) &\mapsto \sup_{d \in t} \mathbb{E}(d) \end{aligned}$$

- **Leaf structure**. Egli-Milner preorder and tailwise preorder.
- **Shape structure**. Inclusion on choices.

Proof. There is nothing to do to prove that we have described a process-predicate type.

The proof that we have really obtained a shape structure is detailed in Example 5.3.

The proof that we have really obtained a leaf structure is similar to that of Example 4.7 and is direct by using Theorem 4.5: we only used modalities sup and $\mathbb{E}$. □

Example 6.3 ($\mathbb{P}(\lozenge\mathbf{1})$ in MC)**.**

- **Ppt**. $(2 \times \mathcal{D}(-), [0,1], 1 + \mathbb{E})$, using lfp, where

$$\begin{aligned} 1 + \mathbb{E} : 2 \times \mathcal{D}\Omega &\to \Omega \\ (\top, d) &\mapsto 1 \\ (\bot, d) &\mapsto \mathbb{E}(d) \end{aligned}$$

- **Leaf structure**. Tailwise preorder.
- **Shape structure**. Equality.

Proof. Similar to the previous cases. For leaf structure, use Theorem 4.5. For shape structure, we are in a trivial case: it is true that

$$\begin{aligned} D : \mathbf{Set} &\to \mathbf{Preord} \\ X &\mapsto (X, =) \end{aligned}$$

defines a shape structure, since we trivially have that $x = y$ implies $\tau(x) \leq \tau(y)$ for any modality τ. □

Remark 6.4.

Above and below, we can obtain finitely branching versions of SG, MDP, using $\mathcal{P}_\mathrm{f}$ instead of $\mathcal{P}$.

We may even restrict ourselves to *finite* versions of those ppt with both leaf and shape structure, by using endofunctors $\mathbf{Set}_\mathrm{fin} \to \mathbf{Set}_\mathrm{fin}$.

Example 6.5 ($\mathbb{P}(\square\mathbf{1})$ in MC)**.**

- **Ppt**. $(2 \times \mathcal{D}(-), [0,1], \tau_\square)$, using gfp, where

$$\begin{aligned} \tau_\square : 2 \times \mathcal{D}\Omega &\to \Omega \\ (\top, t) &\mapsto 1 \\ (\bot, t) &\mapsto \inf_{d \in t} \mathbb{E}(d) \end{aligned}$$

- **Leaf structure**. Egli-Milner preorder and Tailwise preorder.
- **Shape structure**. Inclusion on choices.

Remark 6.6. At the moment, we didn't go far in studying how alternating fixed points (afp) behave in our framework.

Afp are useful for properties such as repeated reachability $\Box\Diamond\mathbf{1}$ or persistence $\Diamond\Box\mathbf{1}$. More generally, they can be used for acceptance conditions of Büchi automata and parity automata.

The papers [62] and [63] provide valuable insights on the use of alternating fixed points in the coalgebraic context.

To imagine what this might look like, the set of states in $X = X_1 \sqcup X_2$ satisfying the Büchi acceptance condition $\Box\Diamond X_2$ could be computed in our framework as the set U_{sol}, in an equation similar to the following.

$$\begin{aligned}
f_1 &= g^*_{\tau_\exists} \wedge g^*_{\tau_1} \\
f_2 &= g^*_{\tau_\exists} \wedge g^*_{\tau_2} \\
\lambda_1 &= \lambda U.\mu V.f_1(U \vee V) \\
U_2^{sol} &= \nu(f_2 \circ (\mathrm{id} \vee \lambda_1)) \\
U_1^{sol} &= \lambda_1(U_2^{sol}) \\
U^{sol} &= (\mathrm{id} \vee \lambda_1)(\nu(f_2 \circ (\mathrm{id} \vee \lambda_1)))
\end{aligned}$$

6.1.2 Liveness and safety for (in)finitely branching transition systems

Example 6.7 (E$\Diamond\mathbf{1}$ in NTS)**.**

- **Ppt**. $(2 \times \mathcal{P}(-), \Omega, 1 + \sup)$, using lfp, where

$$\begin{aligned}
1 + \sup : 2 \times \mathcal{P}\Omega &\to \Omega \\
(\top, t) &\mapsto 1 \\
(\bot, t) &\mapsto \sup t
\end{aligned}$$

- **Leaf structure**. Egli-Milner preorder and Tailwise preorder.
- **Shape structure**. Inclusion on choices.

Example 6.8 (A$\Diamond\mathbf{1}$ in NTS)**.**

- **Ppt**. $(2 \times \mathcal{P}(-), \Omega, 1 + \inf)$, using lfp, where

$$\begin{aligned}
1 + \inf : 2 \times \mathcal{P}\Omega &\to \Omega \\
(\top, t) &\mapsto 1 \\
(\bot, t) &\mapsto \inf t
\end{aligned}$$

- **Leaf structure**. Egli-Milner preorder and Tailwise preorder.
- **Shape structure**. Reverse inclusion on choices.

Remark 6.9. Like before, we can model and solve the same problem for *finitely* branching transition systems by using $\mathcal{P}_\mathrm{f}$ instead of $\mathcal{P}$.

The case where Ω is a total order behaves especially well, even more so if it is finite.

In particular, interesting examples are $\Omega = \mathrm{Bool}$ and $\Omega = [0, 1]$.

6.1.3 Graphs

Example 6.10 (Widest path in weighted graphs)**.**
Let $W = (\overline{\mathbb{R}_+}, +, *)$, with the usual addition and multiplication with $0 \times \infty = 0$.

- **Ppt**. $(2 \times \mathrm{Id} \times \mathrm{Hom}_{\mathrm{fin}}(-, W), W, \tau)$, using lfp, where

$$\begin{aligned}\tau : 2 \times W \times \mathrm{Hom}_{\mathrm{fin}}(W, W) &\to W\\ (\top, \cdot, \cdot) &\mapsto 1\\ (\bot, w_0, d) &\mapsto \max\left(w_0, \max_{w \in W}(\min(w, d(w)))\right).\end{aligned}$$

- **Leaf structure**. Tailwise preorder.

Example 6.11 (Shortest path in graphs)**.** Let W be the monoid $(\mathbb{N}_\infty, +)$.

- **Ppt**. $(2 \times \mathrm{Id} \times \mathcal{P}_{\mathrm{f}}(-), \mathbb{N}_\infty, \tau)$, using lfp, where

$$\begin{aligned}\tau : 2 \times \mathbb{N}_\infty \times \mathcal{P}_{\mathrm{f}}\mathbb{N}_\infty &\to \mathbb{N}_\infty\\ (\top, \cdot, \cdot) &\mapsto 0\\ (\bot, d, D) &\mapsto \min(d, 1 + \min(D)).\end{aligned}$$

- **Leaf structure**. Egli-Milner preorder.

Remark 6.12. In this paper, we haven't worked a lot on examples where the truth object is $\mathbb{N}_\infty$.
Doing so could be useful for applications such as progress measures.
This is the context of *quantitative verification*, for which many VI algorithms are described in [49].

Remark 6.13. Both examples above give us local computations techniques, but more efficient global algorithms for graph analysis exist.

6.2 Connections

Example 6.14 (Reachability for SG and MDP)**.** For the reachability problem,

$$SG \triangleleft_{MDP} MDP.$$

Furthermore, if we restrict ourselves to finite branching, MDP_f is a tight overapproximation of SG_f which is computable locally.

As a consequence, for any finitely branching stochastic game g, we have

$$V_{SG}(g) = \sup_{\alpha:\mathbf{Ord}} p_\alpha = \inf_{\alpha:\mathbf{Ord}} V_{MDP}(g_\alpha),$$

where $p_\alpha := (g^*_{\beta_{SG}})^\alpha(\bot)$ and where $g_\alpha := \Phi(p_\alpha, g)$ is a MDP locally computed through Φ such that $g \triangleleft g_\alpha$ and $(g_\alpha)^*_{\beta_{MDP}}(p_\alpha) = g^*_{\beta_{SG}}(p_\alpha)$.

Proof. Cf. Example 5.10.
We use the natural transformation

$$\begin{aligned}\alpha_X : \{\square, \bigcirc\} \times \{\bot, \top\} \times \mathcal{PD}X &\to \{\bot, \top\} \times \mathcal{PD}X\\ (\bigcirc, g, \varnothing) &\mapsto (\top, \varnothing)\\ (p, g, t) &\mapsto (g, t) \quad \text{ if } (p, t) \neq (\bigcirc, \varnothing),\end{aligned}$$

and the local approximation

$$\begin{aligned}\varphi : \Omega^X \to 2 \times 2 \times \mathcal{P}_{\mathrm{f}}\mathcal{D}X &\to 2 \times \mathcal{P}_{\mathrm{f}}\mathcal{D}X\\ (p, (g, \square, t)) &\mapsto (g, t)\\ (p, (g, \bigcirc, \varnothing)) &\mapsto (\top, \varnothing)\\ (p, (g, \bigcirc, t)) &\mapsto \left(g, \left\{d \in t \,\middle|\, \mathbb{E}((\mathcal{D}p)d) = \inf_{d' \in t} \mathbb{E}((\mathcal{D}p)d')\right\}\right) \quad \text{ if } t \neq \varnothing,\end{aligned}$$

like in Remark 5.21.

It is easy to check that α is natural and that φ does indeed verify

$$\forall p : \Omega^X,\, \forall t : T_\downarrow X,\ \big(t \lhd \varphi(p)(t) \ \wedge\ (\tau_\downarrow \circ T_\downarrow p)(t) = (\tau_\uparrow \circ T_\uparrow p \circ \varphi(p))(t)\big).$$

It is done in Examples 5.10 and 5.20. $\square$

Example 6.15 (Reachability for MDP and MC)**.** For the reachability problem,

$$MC \lhd_{MDP} MDP.$$

Furthermore, if we restrict ourselves to finite branching, MC_f is a tight underapproximation of MDP_f which is computable locally.

As a consequence, for any finitely branching Markov decision process $\mathcal{M}$, we have

$$\mathrm{gfp}\,\mathcal{M}^*_\beta = \inf_{\alpha:\mathbf{Ord}} p_\alpha = \sup_{\alpha:\mathbf{Ord}} \mathrm{gfp}\ (\mathcal{C}_\alpha)^*_{1+\mathbb{E}},$$

where $p_\alpha := (\mathcal{M}^*_{\beta_{MDP}})^\alpha(\top)$ and where $\mathcal{C}_\alpha := \Phi(p_\alpha, \mathcal{M})$ is a MC locally computed through Φ such that $g_\alpha \lhd g$ and $(\mathcal{C}_\alpha)^*_{1+\mathbb{E}}(p_\alpha) = \mathcal{M}^*_{\beta_{MDP}}(p_\alpha)$.

However, *a priori*, we only have

$$V_{MDP,\Diamond\mathbf{1}}(\mathcal{M}) = \mathrm{lfp}\,\mathcal{M}^*_\beta = \sup_{\alpha:\mathbf{Ord}} q_\alpha \geq \sup_{\alpha:\mathbf{Ord}} \mathrm{lfp}\ (\mathcal{C}'_\alpha)^*_{1+\mathbb{E}} = \sup_{\alpha:\mathbf{Ord}} V_{MC,\Diamond\mathbf{1}}(\mathcal{C}'_\alpha)$$

when we instead fix $q_\alpha := (\mathcal{M}^*_{\beta_{MDP}})^\alpha(\bot)$ and $\mathcal{C}'_\alpha := \Phi(q_\alpha, \mathcal{M})$. Using some properties of MC described in Remark 6.16, this issue can often be solved in implementations.

Proof. We use the natural transformation

$$\begin{aligned}\alpha_X : 2 \times \mathcal{D}X &\to 2 \times \mathcal{P}\mathcal{D}X\\ (p, d) &\mapsto (p, \{d\}),\end{aligned}$$

and, for each MDP $\mathcal{M}$ and predicate $p : X \to \Omega$, the locally computed approximation

$$\begin{aligned}\Phi(p, \mathcal{M}) : X &\to 2 \times \mathcal{D}X\\ x &\mapsto (g, \mathbf{1}_{\{x\}}) && \text{if } \mathcal{M}(x) = (g, \varnothing)\\ x &\mapsto (g, \{d\}) \text{ for a } d \in \operatorname*{argmax}_{d' \in t}\big(\mathbb{E}((\mathcal{D}p)d')\big) && \text{if } \mathcal{M}(x) = (g, t), \text{ with } t \neq \varnothing,\end{aligned}$$

which computes an optimal scheduler.

It is easy to check that α is natural, that

$$(1 + \mathbb{E}) \leq \beta_{MDP} \circ \alpha_{[0,1]}$$

(they are actually equal), and that Φ does indeed verify that for all MDP $\mathcal{M}$ and for all $p : X \to \Omega$,

$$\Phi(p, \mathcal{M}) \lhd \mathcal{M} \quad \text{ and } \quad \big(\Phi(p, \mathcal{M})\big)^*_{1+\mathbb{E}}(p) = \mathcal{M}^*_\beta(p).$$

$\square$

Remark 6.16. A good property of MC is that, *when the state space is finite*, it is easy to find an equivalent situation where $\mathrm{lfp}\,\mathcal{C}^*_{1+\mathbb{E}} = \mathrm{gfp}\,\mathcal{C}^*_{1+\mathbb{E}}$, by restricting the state space to

$$S_{\Diamond\mathbf{1}} = \{x \in X \mid \mathbb{P}(\Diamond\mathbf{1})(x) > 0\},$$

where $S_{\Diamond\mathbf{1}}$ can be computed in linear time using graph analysis. Cf. [13], and in particular Theorem 10.19 and Corollary 10.31, for an explanation of this technique.

Thanks to that, efficient algorithms exist for computing $\mathbb{P}(\Diamond\mathbf{1})$ in MC, e.g. by BVI or by solving a linear system.

Moreover, the fact that $\mathrm{lfp}\,\mathcal{C}^*_{1+\mathbb{E}} = \mathrm{gfp}\,\mathcal{C}^*_{1+\mathbb{E}}$ can be useful in proofs about surrogate models, like in the MDP-WG example of Section 6.4.

In the next example, we prove that we can restrict the safety problem of NTS to finitely branching NTS.

Going to the surrogate model provides a large speed-up, since the VI convergence time of the base model on $\mathcal{P}$ can be arbitrary (cf. Example 4.4), whereas the VI convergence time of the surrogate model on $\mathcal{P}_{\mathrm{f}}$ is always smaller than ω.

However, for implementations, we need a way of computing φ efficiently. It is only provided here using the axiom of choice.

Remark 6.17. The following result notices something elementary: if we restrict to the optimal scheduler, then $\mathcal{P}$ and $\mathcal{P}_{\mathrm{f}}$ are the same, since they can both accommodate the optimal scheduler.

Notice that it would *not* be the case for $\Omega = [0, 1]$!

Example 6.18 (Safety for $\mathcal{P}$ and $\mathcal{P}_{\mathrm{f}}$)**.** Let Ω be a complete lattice.

For the reachability problem, we have a connection between infinitely and finitely branching NTS.

$$(2 \times \mathcal{P}, \Omega, 1 + \inf) \lhd (2 \times \mathcal{P}_{\mathrm{f}}, \Omega, 1 + \inf)$$

Moreover, if Ω is *total* and *finite*, $(2 \times \mathcal{P}_{\mathrm{f}}, \Omega, 1 + \inf)$ is a tight overapproximation that is locally computable.

In particular, this is the case for $\Omega = \mathrm{Bool}$.

Thus, if $g : X \to 2 \times \mathcal{P}X$ is an NTS, for the safety problem expressed on Bool,

$$\mathrm{Sat}_{\neg\Box(\neg\mathbf{1})}(g) = \sup_{\alpha:\mathbf{Ord}} p_\alpha = \inf_{\alpha:\mathbf{Ord}} \mathrm{Sat}_{\neg\Box(\neg\mathbf{1})}(g_\alpha),$$

where $p_\alpha := (g^*_{1+\inf})^\alpha(\bot)$ and where $g_\alpha := \Phi(p_\alpha, g)$ is a finitely branching NTS locally computed through Φ such that $g \lhd g_\alpha$ and $(g_\alpha)^*_\tau(p_\alpha) = g^*_\tau(p_\alpha)$.

Note that, however, the name "locally computable" may be misguiding, since we may need the axiom of choice to obtain Φ non-constructively.

Proof. We use the natural transformation defined by inclusion

$$\begin{aligned} \alpha_X : 2 \times \mathcal{P}_{\mathrm{f}}X &\to 2 \times \mathcal{P}X \\ (g, t) &\mapsto (g, t). \end{aligned}$$

It is obvious that α is natural and that $(1 + \inf) \leq (1 + \inf) \circ \alpha_\Omega$, since they are equal.

When Ω is total and final, such as when $\Omega = \mathrm{Bool}$, we prove that $2 \times \mathcal{P}_{\mathrm{f}}$ is a tight approximation of $2 \times \mathcal{P}$ by using

$$\begin{aligned} \varphi : \Omega^X \to 2 \times \mathcal{P}X &\to 2 \times \mathcal{P}_{\mathrm{f}}X \\ (p, (g, \varnothing)) &\mapsto (g, \varnothing) \\ (p, (g, t)) &\mapsto (g, \{y\}), \text{ where } y \text{ is a single element chosen in } \operatorname*{argmin}_{z \in t} p(z). \end{aligned}$$

The existence of φ is provided by the axiom of choice.

It is easy to check that α is natural and that φ does verify

$$\forall p : \Omega^X, \forall t : T_\downarrow X, \; \big(t \lhd \varphi(p)(t) \;\wedge\; (\tau_\downarrow \circ T_\downarrow p)(t) = (\tau_\uparrow \circ T_\uparrow p \circ \varphi(p))(t)\big).$$

Indeed, for each $p : \Omega^X$, $t : \mathcal{P}X$, $\varphi(p)(t)$ is an element of $\mathcal{P}_{\mathrm{f}}X$ such that

- $\pi_1 t = \pi_1 \varphi(p)(t)$ and $\pi_2 t \supset \pi_2 \varphi(p)(t)$, thus $t \sqsubseteq \alpha_X \circ \varphi(p)(t)$, thus $t \lhd \varphi(p)(t)$.
- If $g = \top$, $((1 + \inf) \circ \mathcal{P}p)(t) = 1 = ((1 + \inf) \circ \mathcal{P}_{\mathrm{f}}p \circ \varphi(p))(t)$ and
  if $g = \bot$, $((1 + \inf) \circ \mathcal{P}p)(t) = \min_{z \in t} p(z) = ((1 + \inf) \circ \mathcal{P}_{\mathrm{f}}p \circ \varphi(p))(t)$.

□

Remark 6.19. We could generalise this result to the case where Ω is total but infinite, by creating a notion of "ϵ-tight approximation". This is left for further research.

6.3 Applications of our theory: recipe for VI and BVI algorithm

In this paper, we have obtained a *recipe* to design VI and BVI algorithms: we have obtained sufficient conditions for their construction.

Remark 6.20 (VI)**.** To construct a value iteration (VI) algorithm, it is sufficient to do the following.

- Describe the process type as an endofunctor $T : \mathbf{Set} \to \mathbf{Set}$. Name Ω the truth object.
- Find a candidate $\tau : T\Omega \to \Omega$ giving *local* optimal solutions to the problem.
- Check that (T, Ω, τ) has leaf structure.

  This is always the case when T and τ can be constructed using the ingredients of Theorem 4.5, such as $\mathcal{P}$, $\mathcal{D}$, inf, sup, $\mathbb{E}$, and polynomial constructions.

Having leaf structure means, like in Section 4, that T can be lifted to a **Preord**-enriched **Preord** endofunctor, and that $\tau : (T\Omega, \preceq) \to (\Omega, \leq)$ is monotone.

Remark 6.21 (BVI)**.** To turn a VI algorithm into a bounded value iteration (BVI) algorithm using a surrogate model, it is sufficient to do the following. We will suppose that our VI algorithm computes a lfp, the other case is symmetric.

Our initial VI algorithm is described by the ppt $\mathcal{T}_\downarrow = (T_\downarrow, \Omega, \tau_\downarrow)$.

- Find a candidate surrogate model $\mathcal{T}_\uparrow = (T_\uparrow, \Omega, \tau_\uparrow)$. It represents a new problem for which a VI algorithm exists.

  It is good if there are tricks to solve this problem efficiently (e.g. using graph analysis).

- Find a way of relating processes $\mathcal{T}_\downarrow$ and $\mathcal{T}_\uparrow$, describing this with *shape structures* and a *connection.*

  This can be achieved in two steps, to be performed in parallel.

  On the one hand, extend $T_\downarrow$ and $T_\uparrow$ to functors $\mathbf{Set} \to \mathbf{Preord}$ such that both $\tau : (T\Omega, \sqsubseteq) \to (\Omega, \leq)$ are monotone.

  On the other hand, find a (T_*, Ω, τ_*) (it can be $\mathcal{T}_\downarrow$ or $\mathcal{T}_\uparrow$) such that we have the following commutative diagrams.

$$T_{\downarrow,0} \overset{\alpha_\downarrow}{\Longrightarrow} T_{*,0} \overset{\alpha_\uparrow}{\Longleftarrow} T_{\uparrow,0}.$$

$$\begin{tikzcd}
T_\downarrow\Omega_0 \arrow[r, "\alpha_{\downarrow,\Omega_0}"] \arrow[dr, "\tau_\downarrow"'] \arrow[r, phantom, shift right=3, "\leq"] & T_*\Omega_0 \arrow[d, dashed, "\tau_*"] & T_\uparrow\Omega_0 \arrow[l, "\alpha_{\uparrow,\Omega_0}"'] \arrow[dl, "\tau_\uparrow"] \arrow[l, phantom, shift left=3, "\leq"] \\
 & \Omega &
\end{tikzcd}$$

  This is written

$$\mathcal{T}_\downarrow \lhd_{\mathcal{T}_*} \lhd \mathcal{T}_\uparrow.$$

  Processes $g_\downarrow : X \to T_\downarrow X$ and $g_\uparrow : X \to T_\uparrow X$ will be deemed comparable, which is written $g_\downarrow \lhd g_\uparrow$, if and only if there exists a comparison chain

$$g_\downarrow \sqsubseteq_\downarrow g'_\downarrow, \quad \alpha_\downarrow \circ g'_\downarrow \sqsubseteq_* \alpha_\uparrow \circ g'_\uparrow, \quad g'_\uparrow \sqsubseteq_\uparrow g_\uparrow.$$

- Make sure that the connection is *tight*, by showing that for each $g_\downarrow : X \to T_\downarrow X$ and each predicate $p : X \to \Omega$, a (good) approximation $g_\uparrow = \Phi(p, g_\downarrow)$ can be locally computed.

  It is enough to construct a $\varphi : \Omega^X \to T_\downarrow X \to T_\uparrow X$ such that

$$\forall p : \Omega^X, \forall t : T_\downarrow X, \big(t \lhd \varphi(p)(t) \wedge (\tau_\downarrow \circ T_\downarrow p)(t) = (\tau_\uparrow \circ T_\uparrow p \circ \varphi(p))(t)\big).$$

In that case, for any process $g_\downarrow : X \to T_\downarrow X$,

$$V_\downarrow(g_\downarrow) = \sup_{\alpha:\mathbf{Ord}} p_\alpha = \inf_{\alpha:\mathbf{Ord}} V_\uparrow(g_\alpha),$$

where $p_\alpha = (g_\downarrow)^*_{\tau_\downarrow}$ like in the VI algorithm, and $g_\alpha : X \to T_\uparrow X$ is defined by $g_\alpha = \Phi(p_\alpha, g)$, like in 5.21.

Remark 6.22. In the description above, we obtained

$$V_\downarrow(g_\downarrow) = \sup_{\alpha:\mathbf{Ord}} p_\alpha = \inf_{\alpha:\mathbf{Ord}} V_\uparrow(g_\alpha),$$

To make this a useful BVI algorithm, it is necessary to be able to compute each $V_\uparrow(g_\alpha)$ in a fast way. It is reasonable to do it only for some of the α.

If, by chance, the upper chain $(V_\uparrow(g_\alpha))_{\alpha:\mathbf{Ord}}$ converges much faster than the lower chain $(p_\alpha)_{\alpha:\mathbf{Ord}}$, we get even better than a BVI algorithm: a fast way of computing $V_\downarrow(g_\downarrow)$ through a surrogate model.

For example, this happens in the case of reachability for finite SG and MDP: the upper chain converges in finite time, whereas the lower chain often takes infinitely many steps.

6.4 Extensions to the theory: new kind of connections for more surrogate models

At this point, our theory handles a large amount of examples for VI, and a few simple examples of BVI using surrogate models. However, more could be done.

The concept of surrogate models is still quite new in the realm of model-checking, but the idea is powerful and promising. Given a *difficult* base problem, is there a way to relate it to a *simple* problem (the surrogate model) whose solutions can be efficiently computed and approximate that of the base problem? Given a difficult problem (base model), can we know in advance who will be good candidate for surrogate models? Is there a way to relate this to abstract interpretation [24]?

Generalising those ideas would be a good step toward implementing the "base model / surrogate model" design principle to discover new efficient algorithms for model checking.

The work done in this report in a first step towards that. Our notion of *connection* between *ppt with shape structure* has been shown to accommodate at least some of the "base model / surrogate model" relationships.

However, we only applied it in one way. Even if it can be used in several examples, we only obtained one kind of relationship.

- Consider a base problem g in a base model type $\mathcal{T}_\downarrow$: we want to compute the lfp of g^*_τ.
- Find a surrogate model type $\mathcal{T}_\uparrow$, and a *connection* $\mathcal{T}_\downarrow \lhd \mathcal{T}_\uparrow$ (clever shape structures may have to be found). Find a way to locally compute approximations $\Phi : X^\Omega \to (T_\downarrow X)^X \to (T_\uparrow X)^X$.
- At *each step* α, compute $p_\alpha = (g^*_\tau)^\alpha(\bot)$.
- At *each step* α, compute $g_\alpha = \Phi(p, g)$ and lfp $(g_\alpha)^*_\tau$.
- Obtain BVI
$$\text{lfp } g^*_\tau = \sup_{\alpha:\mathbf{Ord}} p_\alpha = \inf_{\alpha:\mathbf{Ord}} \text{lfp } (g_\alpha)^*_\tau.$$

This is great, and can prove useful both in applications and in proofs. However, notice that a surrogate problem must be studied *at each step* α: this could be extremely expensive if the problem type isn't easy enough.

Moreover, we have not accommodated all kind of base model / surrogate model relationships.

In Section 6.4, we describe two other examples of base model / surrogate model relationship. Accommodating them, especially the MDP-WG one, is an important next step for further research.

6.4.1 A classical surrogate model: graph analysis of MC for almost sure properties

For finite Markov chains – the process type is $2^{AP} \times \mathcal{D}(-)$ – a large class of *almost sure properties* can be checked in linear time. In other words, given a MC $\mathcal{C}$ on the state space X and a property ψ belonging to the family (like reachability, repeated reachability, persistence, etc.), computing

$$\{x \in X \,|\, \mathbb{P}_{\mathcal{C}}(\psi)(x) = 1\}$$

can be done in linear time.

The method can be seen as a use of surrogate models in classical model checking. Indeed, the idea, described in section 10.2 of [13], is to replace the MC by a directed graph, and then perform graph analysis (backward analysis, computing bottom strongly connected components, etc.).

$$\begin{aligned}\alpha_X : 2^{AP} \times \mathcal{D}X &\to 2^{AP} \times \mathcal{P}_{\mathrm{f}}X\\ (p,d) &\mapsto \Big(p, \big\{x \in X \,\big|\, d(x) > 0\big\}\Big)\end{aligned}$$

For *finite* MC, most qualitative problems can be solved by forgetting about the probability transitions and then doing graph analysis. Studying this relationship could be a good first step toward generalising our theory. Notice that the surrogate model is built *only once, without using any predicate* X^Ω.

It is important to note that this relationship does *not* hold for infinite MC.

6.4.2 An innovative surrogate model: weighted graph analysis of MDP for reachability

In [1], an algorithm is given to compute $V_{SG}(\mathcal{G})$ for $\mathcal{G} : X \to 2 \times 2 \times \mathcal{PD}X$, where X is finite.

It uses (tight) connections (between SG, MDP and MC), but also two other kind of local approximation, which may be seen as new kind of connections.

In particular, (widest paths in) weighted graphs are used as surrogate models for (reachability in) MDP, while using properties of MC to make this relationship work.

The full algorithm and proof of correctness can be found in [1], but we explain the main ideas here in our theory's language, as a first step toward accommodating this powerful algorithm based on the "base model / surrogate model" design principle.

The algorithm and its proof can be described in the following way.

Remark 6.23 (Surrogate model for $V(\mathcal{G})$ by global propagation in WG)**.** Have in mind the process types $2 \times 2 \times \mathcal{PD}(-)$ for SG, $2 \times \mathcal{PD}(-)$ for MDP, $2 \times \mathcal{D}(-)$ for MC, and $2 \times \mathrm{Hom}_{\mathrm{fin}}(-, [0,1])$.

- The goal is to compute $V_{SG}(\mathcal{G})$ for $\mathcal{G}$ a stochastic game.

  Using the method of tight connections on $SG \lhd MDP$, we reduce the problem to the computation of $V_{MDP}(\mathcal{M})$ for $\mathcal{M}$ a Markov decision process.

  This is done in the previous sections. The local approximation is computed by

$$\begin{aligned}\varphi_{SG\to MDP} : \Omega^X \to 2 \times 2 \times \mathcal{PD}X &\to 2 \times \mathcal{PD}X\\ (p,(g,\Box,t)) &\mapsto (g,t)\\ (p,(g,\bigcirc,\varnothing)) &\mapsto (\top,\varnothing)\\ (p,(g,\bigcirc,t)) &\mapsto \Big(g, \big\{d \in t \,\big|\, \mathbb{E}((\mathcal{D}p)d) = \inf_{d' \in t} \mathbb{E}((\mathcal{D}p)d')\big\}\Big) \quad \text{if } t \neq \varnothing,\end{aligned}$$

  where we implicitly decurrified $\Omega^X \to 2 \times 2 \times \mathcal{PD}X \to 2 \times \mathcal{PD}X$ to $\Omega^X \times (2 \times 2 \times \mathcal{PD}X) \to 2 \times \mathcal{PD}X$ in the declaration of $\varphi_{SG\to MDP}$.

- The algorithm will use

$$\begin{aligned}\varphi_{MDP\to WG} : \Omega^X \to 2 \times \mathcal{PD}X &\to 2 \times \mathrm{Hom}_{\mathrm{fin}}(X,\Omega)\\ (p,(g,t)) &\mapsto \Big(g, x \mapsto \big\{\mathbb{E}((\mathcal{D}p)d) \,\big|\, d \in t,\, d(x) > 0\big\}\Big)\end{aligned}$$

to compute a surrogate model for global propagation.

The convergence proof will also use

$$\begin{aligned}\varphi_{MDP\to MC} : \Omega^X &\to 2\times\mathcal{P}_{\neq\varnothing}\mathcal{D}X \to 2\times\mathcal{D}X\\ (p,(g,t)) &\mapsto (g,\{d\}),\quad \text{where } d \text{ belongs to } \operatorname*{argmax}_{d'\in t}\mathbb{E}((\mathcal{D}p)(d')).\end{aligned}$$

For all three $\varphi : \Omega^X \to T_1X \to T_2X$, we write

$$\begin{aligned}\Phi : \Omega^X &\to (T_1X)^X \to (T_2X)^X\\ (p,g) &\mapsto \varphi(p)\circ g.\end{aligned}$$

- Three special properties of our models are used.
  - $\varphi_{SG\to MDP}$ and $\varphi_{MDP\to MC}$ are computable, since we have finite branching.
  - Given any MDP $\mathcal{M} : X \to 2\times\mathcal{P}\mathcal{D}X$ on X finite, there is only a finite number of MC $\mathcal{C} : X \to 2\times\mathcal{D}X$ such that
    $$\mathcal{C} \triangleleft \mathcal{M}.$$
    (In other words, there is only a finite number of ways to fix the strategy in $\mathcal{M}$.)
  - Given a MC $\mathcal{C} : X \to 2\times\mathcal{D}X$, it is easy to compute the set
    $$X' = \{x\in X \mid \mathbb{P}_{\mathcal{C}}(\Diamond\mathbf{1})(x) > 0\}.$$
    In that case, we obtain a single fixed point (for the problem of reachability in MC).
    $$\operatorname{lfp}\big((\mathcal{C})^*_{1+\mathbb{E}}\big) = \operatorname{gfp}\big((\mathcal{C})^*_{1+\mathbb{E}}\big)$$
    Cf. section 10.1.2 of [13] to see how to compute X' in linear time by double backward analysis on the underlying graph of $\mathcal{C}$.
- The algorithm and the proof than proceed in the following way.
  - Let
    $$\begin{aligned}t^\star_{\mathcal{M}} : \Omega^X &\to \Omega^X\\ p &\mapsto V_{WG}(\Phi_{MDP\to WG}(p,\mathcal{M})),\end{aligned}$$
    where $V_{WG} : (X \to 2\times\mathrm{Hom}_{\mathrm{fin}}(X,\Omega)) \to \Omega^X$ computes the widest path in a weighted graph. V_{WG} is a lfp, but may be computed quickly by global graph analysis.
    $t^\star_{\mathcal{M}}$ is monotone.
    We will prove that
    $$V_{MDP}(\mathcal{M}) = \operatorname{lfp}\mathcal{M}^*_\beta = \operatorname{gfp} t^\star_{\mathcal{M}},$$
    thus providing upper bounds.
  - Notice that
    $$V_{MDP}(\mathcal{M}) \le t^\star_{\mathcal{M}}(V_{MDP}(\mathcal{M})).$$
    This proves that
    $$V_{MDP}(\mathcal{M}) \le \operatorname{gfp} t^\star_{\mathcal{M}}.$$
  - Notice that, for each $p : X \to \Omega$, the MC $\mathcal{C} := \Phi_{MDP\to MC}(p,\mathcal{M})$ is such that $\mathcal{C}\triangleleft\mathcal{M}$ and
    $$t^\star_{\mathcal{M}}(p) \le \mathcal{C}^*_{1+\mathbb{E}}(p).$$
    By finiteness, we can choose a MC $\mathcal{C}$ such that for infinitely many $p_\alpha := (t^\star_{\mathcal{M}})^\alpha(\top)$ we have
    $$t^\star_{\mathcal{M}}(p_\alpha) \le \mathcal{C}^*_{1+\mathbb{E}}(p_\alpha),$$
    thus
    $$\operatorname{gfp} t^\star_{\mathcal{M}} \le \operatorname{gfp}\mathcal{C}^*_{1+\mathbb{E}}.$$
    We keep naming this MC $\mathcal{C}$ in the following points.

- We check that nothing is lost by restricting to $X' = \{x \in X \mid \mathbb{P}_{\mathcal{C}}(\Diamond \mathbf{1})(x) > 0\}$. Thus,
$$\operatorname{gfp} \mathcal{C}^*_{1+\mathbb{E}} = \operatorname{lfp} \mathcal{C}^*_{1+\mathbb{E}}.$$
- Since $\mathcal{C} \lhd \mathcal{M}$, we have (cf. 5.15)
$$\operatorname{lfp} \mathcal{C}^*_{1+\mathbb{E}} \leq \operatorname{lfp} \mathcal{M}^*_{\beta}.$$
- We can now conclude. We have proven
$$\operatorname{lfp} \mathcal{C}^*_{1+\mathbb{E}} \leq \operatorname{lfp} \mathcal{M}^*_{\beta} \leq \operatorname{gfp} t^\star_{\mathcal{M}} \leq \operatorname{gfp} \mathcal{C}^*_{1+\mathbb{E}} = \operatorname{lfp} \mathcal{C}^*_{1+\mathbb{E}},$$
thus
$$V_{MDP}(\mathcal{M}) = \operatorname{lfp} \mathcal{M}^*_{\beta} = \operatorname{gfp} t^\star_{\mathcal{M}}.$$

Remark 6.24. Two things happened here that are well understood by our theory : the connection $MC \lhd MDP$ (related to $\Phi_{MDP \to MC}$) and the tight connection $SG \lhd MDP$ (defined by $\Phi_{SG \to MDP}$).
However, our theory doesn't yet deal well with $t^\star_{\mathcal{M}} := p \mapsto V_{WG}(\Phi_{MDP \to WG}(p, \mathcal{M}))$. Notice how our theory performs only *local computations* in the g^*_τ, while a *global propagation* (V_{WG}) is computed inside $t^\star_{\mathcal{M}}$.
The two following points are still to be generalised to the abstract setting.

- For each $p : X \to \Omega$, $\mathcal{C} := \Phi_{MDP \to MC}(p, \mathcal{M})$ is such that
$$t^\star_{\mathcal{M}}(p) \leq \mathcal{C}^*_{1+\mathbb{E}}(p).$$
- We have
$$V_{MDP}(\mathcal{M}) \leq t^\star_{\mathcal{M}}(V_{MDP}(\mathcal{M})).$$

Notice that, given a MDP $\mathcal{M}$, the evolution of the surrogate model $\mathcal{W}$ is independent of computations performed on $\mathcal{M}$. We compute $(\mathcal{M}^*_\beta)^\alpha(\bot)$ on one hand, and $(t^\star_{\mathcal{M}})^\alpha(\top)$ on the other hand.
This contrasts with connections studied in Section 5, where at each step the surrogate model was updated based on the result $p_\alpha = (g^*_\tau)^\alpha(\bot)$.
At this point of our work, exploring the MDP-WG relationship is left for further research.

7 Conclusion

7.1 Next steps

Our work leaves the road wide open for further research.

Global propagation, new instances

In our work, we accommodate some of the relationships between base models and surrogate models, and this provides BVI algorithms.
However, we do not accommodate all such relationships. In particular, we do not fully reach the initial goal of the internship, since we do not accommodate the "global propagation" relationship between MDP and WG studied in [1].
The main issue is that "tight connections" based on weakest precondition transformers are too weak: they leave all global computations outside of the predicate transformer. This is discussed in Section 6.4, and more specifically in Remark 6.24.
New kind of connections must be described!

This line of research is extremely important to the further development and applicability of our theory and of the surrogate model design principle.
More generally, a goal is to find new instances of the general notions studied in this report.

Continuity

In this report, general conditions enabling VI algorithm are described. Those conditions imply *monotone* predicate transformers, thus *transfinite* value iteration may be needed, as discussed in Example 4.4.

To strengthen our results, it would be interesting to design *continuous leaf structures* enabling *Scott-continuous predicate transformers.*

This would lead to properties like the following.

Conjecture 7.1.
The Bellman operator is Scott-continuous in all finitely branching stochastic games, even on infinite state space. Thus, value iteration converges in at most ω steps.

Several weeks of the internship were devoted to this question, but too many holes remained in the proofs to make it presentable.

The main thing to study are the (Scott-)continuity properties of Egli-Milner preorder for subsets, tailwise preorder for distribution, and modalities including suprema, infima and expectation.

For example, one of the proofs' holes could be filled by proving the following conjecture, which we didn't manage to do.

Conjecture 7.2. *Let $\preceq$ be the Egli-Milner preorder (defined in 3.18).*
$(\Omega, \leq)$ is a complete lattice (not necessarily total nor finite), and $(\mathcal{P}\Omega, \preceq)$ is defined by

$$\begin{aligned} \forall A, B \in \mathcal{P}\Omega,\ A \preceq B \iff & (\forall a \in A,\ \exists b \in B,\ a \leq b) \\ & \wedge (\forall b \in B,\ \exists a \in A,\ a \leq b). \end{aligned}$$

Let $(A_i)_{i \in I}$ be a chain *of $(\mathcal{P}\Omega, \preceq)$, i.e. a chain of subsets of Ω.*
Then, (A_i) admits at least one infimum (in the sense of preorders), which can be described as

$$\left\{ \inf_{i \in I} x_i \;\middle|\; (x_i)_{i \in I} \in \Omega^I \text{ and } \forall i \in I,\ x_i \in A_i \right\}.$$

Various ideas were explored in those continuity proofs.

For example, to prove the continuity of $\mathcal{D}(-) : \mathrm{Hom}(X, [0,1]) \to \mathrm{Hom}(\mathcal{D}X, \mathcal{D}[0,1])$ in the sense of the tailwise order used pointwise, the proof relied on (simple) ideas from the theory of càdlàg functions, such as "wiggling in both space and time".

Those results may be put together in further work if buggy proofs are repaired.

Fibrations and categorical viewpoint

We implicitly work extensively with the fibration

$$\begin{tikzcd} \mathbf{Pred}_\Omega \arrow[d] \\ \mathbf{Set} \end{tikzcd}$$

where $\mathbf{Pred} := \mathbf{Set}/\Omega$ is the category of predicates $p : X \to \Omega$.

We noted (Proposition 4.2) that for a process-predicate type $\mathcal{T} = (T, \Omega, \tau)$ with leaf structure, computations of lfp and gfp gave rise to functors

$$\mathrm{Coalg}(T) \to \mathbf{Pred}.$$

Functorial properties were used in key proofs, but we didn't go any further in studying the fibrational structures that appear in our work.

Doing so could be useful to connect with current development in coalgebraic theory (Cf. Section 1.1.4) and to synthesise our work. What would we gain by using fibrational structures like codensity liftings?

For the sake of simplicity, all monadic hypotheses about process types were left aside in our work. However, they are common in the literature. How would we accommodate them, and what new results would we gain? We could study T-processes $g : X \to TY$ as Kleisli arrows, composable in the Kleisli category $\mathcal{K}\ell(T)$, would this give us any insight?

Moreover, we noticed glimpses of structure in several objects encountered in this internship, but didn't always create a big picture.

Leaf structures can be summed, composed, etc. can we extract useful structure out of this? For any **Set**-endofunctor T, there are coarsest and finest liftings to **Preord**, what can we deduce out of this, knowing that modalities impose restrictions on possible liftings?

What about shape structure? All of them were hand-made for our proofs, can we design a more generic architectural process, like we did for leaf structures?

We have a relationship $\mathcal{T}_{\downarrow} \lhd \mathcal{T}_{\uparrow}$ between process-predicate type with both leaf and shape structure. This relation is reflexive and transitive, do we gain anything by seeing it as a preorder? Could we make this a category? What is the overall structure of relationships between models of processes?

Alternating fixed points

For the moment, our theory studies least fixed points and greatest fixed points, but not alternating fixed points, i.e. alternation of lfp and gfp.

However, many problems require some alternation depth to be described with a fixed point characterisation. This is illustrated by the survey paper [49] on value iteration.

We shortly mention this question in Remark 6.6. Work on alternating fixed points in the coalgebraic context can be found in [62] and [63].

Approximately tight connections

Our notion of tight connection (Cf. definition 5.17) requires an equality (written $(g_{\downarrow})^*_{\mathcal{T}_{\downarrow}}(p) = (g_{\uparrow})^*_{\mathcal{T}_{\uparrow}}(p)$), but it would be possible to work with arbitrarily good approximations.

This is quickly mentioned in Remark 6.19.

Such incremental techniques could help us to deal better with infinite truth objects, such as $\Omega = [0, 1]$ for probabilistic verification.

Quantitative computations: $\mathbb{N}_{\infty}$ as a truth object

In this report, we mainly work with boolean ($\Omega = \{\bot, \top\}$) and probabilistic ($\Omega = [0, 1]$) truth domains.

Some examples use the quantitative truth domain ($\Omega = \mathbb{N}_{\infty}$), but we didn't focus on the specific properties of this case.

This could be a large source of examples and application cases, including of surrogate models well-suited to graph analysis.

Examples of VI algorithm for the quantitative truth domain $\overline{\mathbb{R}_+}$ are discussed in the survey paper [49].

7.2 Epilogue

During this internship, we have studied categorical structures appearing in incremental and approximate algorithms, focusing on value iteration and surrogate models.

Using coalgebras, order theory and predicate transformers, we described a general theory of processes able to deal with predicates on the state space.

Two kinds of order structures appeared: *leaf structures*, which ensure convergence of VI algorithms, and *shape structures*, which help us to compare processes together, paving the way for surrogate models understood with *connections*.

We thus obtained recipes for VI and BVI algorithms, based on two main theorems.

However, as discussed in Section 7.1, this is only the beginning of the road: new insights are to be found, and new algorithms are to be designed.

References

[1] Kittiphon Phalakarn, Toru Takisaka, Thomas Haas, and Ichiro Hasuo. Widest paths and global propagation in bounded value iteration for stochastic games. In *Computer Aided Verification*, pages 349–371. Springer International Publishing, 2020.

[2] A Simaitis. *Automatic verification of competitive stochastic systems.* PhD thesis, University of Oxford, 2014.

[3] S. Rasoul Etesami, Walid Saad, Narayan B. Mandayam, and H. Vincent Poor. Stochastic games for the smart grid energy management with prospect prosumers. *IEEE Transactions on Automatic Control*, 63(8):2327–2342, 2018.

[4] Souymodip Chakraborty, Joost-Pieter Katoen, Falak Sher, and Martin Strelec. Modelling and statistical model checking of a microgrid. *International Journal on Software Tools for Technology Transfer*, 17, 01 2014.

[5] Yves Bertot and Pierre Castéran. *Interactive theorem proving and program development: Coq'Art: the calculus of inductive constructions.* Springer Science & Business Media, 2013.

[6] Ana Bove, Peter Dybjer, and Ulf Norell. A brief overview of agda–a functional language with dependent types. In *International Conference on Theorem Proving in Higher Order Logics*, pages 73–78. Springer, 2009.

[7] Tobias Nipkow, Lawrence C Paulson, and Markus Wenzel. *Isabelle/HOL: a proof assistant for higher-order logic*, volume 2283. Springer Science & Business Media, 2002.

[8] Leonardo de Moura, Soonho Kong, Jeremy Avigad, Floris Van Doorn, and Jakob von Raumer. The lean theorem prover (system description). In *International Conference on Automated Deduction*, pages 378–388. Springer, 2015.

[9] D. Hilbert and W. Ackermann. *Grundzüge der Theoretischen Logik.* Springer Berlin Heidelberg, 1928.

[10] Alonzo Church. An unsolvable problem of elementary number theory. *American Journal of Mathematics*, 58(2):345, April 1936.

[11] A. M. Turing. On computable numbers, with an application to the entscheidungsproblem. *Proceedings of the London Mathematical Society*, s2-42(1):230–265, 1937.

[12] J. Donald Monk. *Mathematical Logic.* Springer New York, 1976.

[13] Christel Baier and Joost-Pieter Katoen. *Principles of Model Checking (Representation and Mind Series).* The MIT Press, 2008.

[14] Neil Immerman. *Descriptive complexity.* Springer Science & Business Media, 2012.

[15] Marijn JH Heule, Matti Järvisalo, and Martin Suda. Sat competition 2018. *Journal on Satisfiability, Boolean Modeling and Computation*, 11(1):133–154, 2019.

[16] Sylvain Conchon, Albin Coquereau, Mohamed Iguernlala, and Alain Mebsout. Alt-ergo 2.2. In *SMT Workshop: International Workshop on Satisfiability Modulo Theories*, 2018.

[17] Clark Barrett, Christopher L Conway, Morgan Deters, Liana Hadarean, Dejan Jovanović, Tim King, Andrew Reynolds, and Cesare Tinelli. Cvc4. In *International Conference on Computer Aided Verification*, pages 171–177. Springer, 2011.

[18] Leonardo De Moura and Nikolaj Bjørner. Z3: An efficient smt solver. In *International conference on Tools and Algorithms for the Construction and Analysis of Systems*, pages 337–340. Springer, 2008.

[19] M. Kwiatkowska, G. Norman, and D. Parker. PRISM 4.0: Verification of probabilistic real-time systems. In G. Gopalakrishnan and S. Qadeer, editors, *Proc. 23rd International Conference on Computer Aided Verification (CAV'11)*, volume 6806 of *LNCS*, pages 585–591. Springer, 2011.

[20] Mordechai Ben-Ari. *Principles of the Spin model checker.* Springer Science & Business Media, 2008.

[21] Yuan Yu, Panagiotis Manolios, and Leslie Lamport. Model checking tla+ specifications. In *Advanced Research Working Conference on Correct Hardware Design and Verification Methods*, pages 54–66. Springer, 1999.

[22] Gerd Behrmann, Alexandre David, Kim Guldstrand Larsen, John Håkansson, Paul Pettersson, Wang Yi, and Martijn Hendriks. Uppaal 4.0. 2006.

[23] H. G. Rice. Classes of recursively enumerable sets and their decision problems. *Transactions of the American Mathematical Society*, 74(2):358–358, February 1953.

[24] P. Cousot and R. Cousot. Abstract interpretation: a unified lattice model for static analysis of programs by construction or approximation of fixpoints. In *Conference Record of the Fourth Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, pages 238–252, Los Angeles, California, 1977. ACM Press, New York, NY.

[25] Patrick Cousot, Radhia Cousot, Jerôme Feret, Laurent Mauborgne, Antoine Miné, David Monniaux, and Xavier Rival. The ASTREé analyzer. In *Programming Languages and Systems*, pages 21–30. Springer Berlin Heidelberg, 2005.

[26] Olivier Bouissou, Eric Goubault, Sylvie Putot, Karim Tekkal, and Franck Vedrine. Hybridfluctuat: A static analyzer of numerical programs within a continuous environment. In *Proceedings of the 21st International Conference on Computer Aided Verification*, CAV '09, page 620–626, Berlin, Heidelberg, 2009. Springer-Verlag.

[27] SRILab ETHZ. ERAN. `https://github.com/eth-sri/eran`.

[28] Souradeep Dutta, Xin Chen, Susmit Jha, Sriram Sankaranarayanan, and Ashish Tiwari. Sherlock - a tool for verification of neural network feedback systems: Demo abstract. In *Proceedings of the 22nd ACM International Conference on Hybrid Systems: Computation and Control*, HSCC '19, page 262–263, New York, NY, USA, 2019. Association for Computing Machinery.

[29] John McCarthy. Recursive functions of symbolic expressions and their computation by machine, part i. *Communications of the ACM*, 3(4):184–195, April 1960.

[30] C. A. R. Hoare. An axiomatic basis for computer programming. *Communications of the ACM*, 12(10):576–580, October 1969.

[31] Emily Riehl. *Category theory in context.* Courier Dover Publications, 2017.

[32] Bart Jacobs. *Introduction to Coalgebra: Towards Mathematics of States and Observation.* Cambridge Tracts in Theoretical Computer Science. Cambridge University Press, 2016.

[33] Samson Abramsky and Achim Jung. Domain theory. 1994.

[34] Edsger W. Dijkstra. Guarded commands, nondeterminacy and formal derivation of programs. *Communications of the ACM*, 18(8):453–457, August 1975.

[35] Ichiro Hasuo. Generic weakest precondition semantics from monads enriched with order. *Theoretical Computer Science*, 604:2–29, November 2015.

[36] Bart Jacobs and Jesse Hughes. Simulations in coalgebra. *Electronic Notes in Theoretical Computer Science*, 82(1):128–149, 2003. CMCS'03, Coalgebraic Methods in Computer Science (Satellite Event for ETAPS 2003).

[37] Sam Staton. Relating coalgebraic notions of bisimulation. In *Algebra and Coalgebra in Computer Science*, pages 191–205. Springer Berlin Heidelberg, 2009.

[38] Ulrich Dorsch, Stefan Milius, Lutz Schröder, and Thorsten Wißmann. Efficient Coalgebraic Partition Refinement. In Roland Meyer and Uwe Nestmann, editors, *28th International Conference on Concurrency Theory (CONCUR 2017)*, volume 85 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 32:1–32:16, Dagstuhl, Germany, 2017. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik.

[39] Bart Jacobs and Alexandra Silva. Automata learning: A categorical perspective. In *Horizons of the Mind. A Tribute to Prakash Panangaden*, pages 384–406. Springer, 2014.

[40] Jean-Baptiste Jeannin, Dexter Kozen, and Alexandra Silva. Cocaml: Functional programming with regular coinductive types. *Fundamenta Informaticae*, 150(3-4):347–377, 2017.

[41] Shin-ya Katsumata and Tetsuya Sato. Codensity liftings of monads. In *6th Conference on Algebra and Coalgebra in Computer Science (CALCO 2015).* Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2015.

[42] David Sprunger, Shin ya Katsumata, Jérémy Dubut, and Ichiro Hasuo. Fibrational bisimulations and quantitative reasoning. In *Coalgebraic Methods in Computer Science*, pages 190–213. Springer International Publishing, 2018.

[43] Yuichi Komorida, Shin ya Katsumata, Nick Hu, Bartek Klin, and I. Hasuo. Codensity games for bisimilarity. *2019 34th Annual ACM/IEEE Symposium on Logic in Computer Science (LICS)*, pages 1–13, 2019.

[44] Alejandro Aguirre and Shin ya Katsumata. Weakest preconditions in fibrations. *Electronic Notes in Theoretical Computer Science*, 352:5–27, 2020. The 36th Mathematical Foundations of Programming Semantics Conference, 2020.

[45] Mayuko Kori, Ichiro Hasuo, and Shin ya Katsumata. Fibrational initial algebra-final coalgebra coincidence over initial algebras: Turning verification witnesses upside down, 2021.

[46] Vijay V. Vazirani. *Approximation Algorithms.* Springer Berlin Heidelberg, 2003.

[47] Subhash Khot. On the unique games conjecture (invited survey). In *2010 IEEE 25th Annual Conference on Computational Complexity.* IEEE, June 2010.

[48] Sven Köhler, Christian Schindelhauer, and Martin Ziegler. On approximating real-world halting problems. In *Fundamentals of Computation Theory*, pages 454–466. Springer Berlin Heidelberg, 2005.

[49] Krishnendu Chatterjee and Thomas A. Henzinger. *Value Iteration*, page 107–138. Springer-Verlag, Berlin, Heidelberg, 2008.

[50] Serge Haddad and Benjamin Monmege. Interval iteration algorithm for MDPs and IMDPs. *Theoretical Computer Science*, 735:111–131, July 2018.

[51] Edon Kelmendi, Julia Krämer, Jan Křetínský, and Maximilian Weininger. Value iteration for simple stochastic games: Stopping criterion and learning algorithm. In *Computer Aided Verification*, pages 623–642. Springer International Publishing, 2018.

[52] Mandar N. Thombre, Heinz A. Preisig, and Misganaw B. Addis. Developing surrogate models via computer based experiments. In *12th International Symposium on Process Systems Engineering and 25th European Symposium on Computer Aided Process Engineering*, pages 641–646. Elsevier, 2015.

[53] J.J.M.M. Rutten. Universal coalgebra: a theory of systems. *Theoretical Computer Science*, 249(1):3–80, 2000. Modern Algebra.

[54] B. Jacobs and J. Rutten. A tutorial on (co)algebras and (co)induction. 1997.

[55] Bart Jacobs. Probabilities, distribution monads, and convex categories. *Theoretical Computer Science*, 412(28):3323–3336, 2011. Festschrift in Honour of Jan Bergstra.

[56] Maaike Zwart and Dan Marsden. No-go theorems for distributive laws. In *Proceedings of the 34th Annual ACM/IEEE Symposium on Logic in Computer Science*, LICS '19. IEEE Press, 2019.

[57] Bart Jacobs. From multisets over distributions to distributions over multisets. *CoRR*, abs/2105.06908, 2021.

[58] Bart Jacobs and Erik Poll. Coalgebras and monads in the semantics of java. *Theoretical Computer Science*, 291(3):329–349, 2003. Algebraic Methodology and Software Technology.

[59] Jon Beck. Distributive laws. In B. Eckmann, editor, *Seminar on Triples and Categorical Homology Theory*, pages 119–140, Berlin, Heidelberg, 1969. Springer Berlin Heidelberg.

[60] Maaike Zwart. *On the Non-Compositionality of Monads via Distributive Laws.* PhD thesis, Department of Computer Science, University of Oxford, September 2020.

[61] Louis Parlant. *Monad Composition via Preservation of Algebras.* PhD thesis, 10 2020.

[62] Natsuki Urabe and Ichiro Hasuo. Categorical buechi and parity conditions via alternating fixed points of functors, 2018.

[63] Quentin Aristote. Fibrational framework for nested alternating fixed points and(bi)simulation notions for büchi automata, 2020.

[64] Adriana Balan and Alexander Kurz. Finitary functors: From set to preord and poset. In *Proceedings of the 4th International Conference on Algebra and Coalgebra in Computer Science*, CALCO'11, page 85–99, Berlin, Heidelberg, 2011. Springer-Verlag.

[65] G. KELLY. The basic concepts of enriched category theory. *Reprints in Theory and Applications of Categories [electronic only]*, 2005, 01 2005.

[66] A. Maitra and W. Sudderth. The optimal reward operator in negative dynamic programming. *Math. Oper. Res.*, 17(4):921–931, November 1992.

[67] Huizhen Yu. On convergence of value iteration for a class of total cost markov decision processes. *SIAM Journal on Control and Optimization*, 53(4):1982–2016, January 2015.