Domains for Higher-Order Games

Matthew Hague[1], **Roland Meyer**[2], **and Sebastian Muskalla**[2]

1 **Royal Holloway University of London,** matthew.hague@rhul.ac.uk
2 **TU Braunschweig,** {roland.meyer, s.muskalla}@tu-braunschweig.de

Abstract

We study two-player inclusion games played over word-generating higher-order recursion schemes. While inclusion checks are known to capture verification problems, two-player games generalise this relationship to include program synthesis. In such games, non-terminals of the grammar are controlled by opposing players. The goal of the existential player is to avoid producing a word that lies outside of a regular language of unsafe words.

We contribute a new domain that provides a representation of the winning region of such games. Our domain is based on (functions over) potentially infinite Boolean formulas with words as atomic propositions. We develop an abstract interpretation framework that we instantiate to abstract this domain into a domain where the propositions are replaced by states of a finite automaton. This second domain is therefore finite and we obtain, via standard fixed point techniques, a direct algorithm for the analysis of two-player inclusion games. We show, via a second instantiation of the framework, that our finite domain can be optimised, leading to a $(k+1)$EXP algorithm for order-k recursion schemes. We give a matching lower bound, showing that our approach is optimal. Since our approach is based on standard Kleene iteration, existing techniques and tools for fixed point computations can be applied.

1 Introduction

Inclusion checking has received considerable recent attention [46, 19, 1, 2, 30]. One of the reasons is a new verification loop, which invokes inclusion as a subroutine in an iterative fashion. The loop has been proposed by Podelski et al. for the safety verification of recursive programs [26], and then been generalized to parallel and parameterized programs [36, 17, 15] and to liveness [16]. The idea of Podelski's loop is to iteratively approximate unsound data flow in the program of interest, and add the approximations to the specification. Consider a program with control-flow language CF that is supposed to satisfy a safety specification given by a regular language R. If the check $CF \subseteq R$ succeeds, then the program is correct as the data flow only restricts the set of computations. If a computation $w \in CF$ is found that lies outside R, then it depends on the data flow whether the proram is correct. If data is handled correctly, w is a counterexample to R. Otherwise, w is generalized to a regular language S of computations that are all infeasible. We set $R = R \cup S$ and repeat the procedure.

Podelski's loop has also been generalized to synthesis [29, 38]. In that setting, the program is assumed to have two kinds of non-determinism. Some of the non-deterministic transitions are understood to be controlled by the environment. They provide inputs that the system has to react to, and are also referred to as demonic non-determinism. In contrast, the so-called angelic non-determinism are the alternatives of the system to react to an input. The synthesis problem is to devise a controller that resolves the angelic non-determinism in a way that a given safety specification is met. Technically, the synthesis problem corresponds to a two-player perfect information game, and the controller implements a winning strategy for the system player. When generalizing Podelski's loop to the synthesis problem, the inclusion check thus amounts to solving a strategy-synthesis problem.

Our motivation is to synthesize functional programs with Podelski's loop. We assume the program to be given as a non-deterministic higher-order recursion scheme where the non-terminals are assigned to two players. One player is the system player who tries to enforce the derivation of words that belong to a given regular language. The other player is the environment, trying to derive a word outside the language. The use of the corresponding strategy-synthesis algorithm in Podelski's loop comes with three characteristics: (1) The algorithm is invoked iteratively, (2) the program is large and the specification is small, and (3) the specification is non-deterministic. The first point means that the strategy synthesis should not rely on costly precomputation. Moreover, it should have the chance to terminate early. The second says that the cost of the computation should depend on the size of the specification, not on the size of the program. Computations on the program, in particular iterative ones, should be avoided. Combined with the third requirement, these two characteristics rule out reductions to reachability games. The required determinisation would mean a costly precomputation, the reduction to reachability a product with the program. This discussion in particular forbids a reduction of the strategy-synthesis problem to higher-order model checking [39], which indeed can be achieved (see Appendix A for a comparison to intersection types [35]). Instead, we need a strategy synthesis that can directly deal with non-deterministic specifications.

We show that the winning region of a higher-order inclusion game wrt. a non-deterministic right-hand side can be computed with a standard fixed-point iteration. Our contribution is a domain suitable for this computation. The key idea is to use Boolean formulas over the states of the targeted finite automaton (so the states are the atomic propositions). While a formula-based domain has recently been proposed for context-free inclusion games [29] (and generalized to infinite words [38]), the generalization to higher order is new. To understand the idea, consider a non-terminal that is ground and for which we have computed a formula. The Boolean structure reflects the alternation among the players in the plays that start from this non-terminal. The leaves abstract the words that are generated along the plays, in terms of sets of states from which these words can be accepted. Checking which player has a winning strategy can by done by evaluating the formula under the assignment that assigns true to sets of states containing the initial state. To our surprise, the above domain did not give the optimal complexity. Instead, it was possible to further optimize it by resolving the determinisation information. Intuitively, the existential player can also resolve the non-determinism given by a set. Crucially, our approach is handles the non-determinism of the specification inside the analysis, without preprocessing.

Besides offering the characteristics that are needed for Podelski's loop, our development also contributes to the research program of *effective denotational semantics*, as recently proposed by Salvati and Walukiewicz [44] and [4] being an early work in this field. The idea is to solve verification problems by computing the semantics of a program in a suitable domain. Salvati and Walukiewicz studied the expressiveness of greatest fixed-point semantics and their correspondence to automata [44], and constructions of enriched Scott models for parity conditions [42] based on recent work of Grellois and Melliès [20, 21]. Hofmann and Chen considered the verification of more restricted ω-path properties focussed on the domain [27]. They show that explicit automata constructions can be avoided and give a domain that directly captures subsets (so-called patches) of the ω-language. The work has been generalized to higher order [28]. Our contribution is related in that we focus on the domain (suitable for capturing plays).

Besides the construction of the domain, the correctness proof may be of interest. We employ fixed-point transfer results as known from abstract interpretation [3]. To be precise, we proceed in three steps. First, we give a semantic characterization showing that the winning region can be captured by an infinite model (a greatest fixed point). This domain has as elements (potentially infinite) sets of (finite) Boolean formulas. The formulas capture plays (up to a certain depth) and the atomic propositions are terminal words. The infinite set structure is to avoid infinite syntax. Then we employ a fixed-point transfer result to replace the terminals by states and get rid of the sets. The third step is another fixed-point transfer that justifies the optimization. We give a matching lower bound. The problem is $(k+1)$EXP-complete for order-k schemes.

Related Work. The relationship between recursion schemes and extensions of pushdown automata has been well studied [12, 13, 31, 23]. This means algorithms for recursion schemes can be transferred to extensions of pushdown automata and vice versa. In the sequel, we will use *pushdown automata* to refer to pushdown automata and their family of extensions.

The decidability of Monadic Second Order Logic (MSO) over trees generated by recursion schemes was first settled in the restricted case of *safe* schemes by Knapik *et al.* [31] and independently by Caucal [11]. This result was generalised to all schemes by Ong [39]. Both of these results consider *deterministic* schemes only.

Related results have also been obtained in the consideration of games played over the configuration graphs of pushdown automata [45, 10, 32, 23]. Of particular interest are *saturation* methods for pushdown games [5, 18, 9, 6, 24, 25, 7]. In these works, automata representing sets of winning configurations are constructed using fixed point computations.

A tightly related approach pioneered by Kobayashi et al. operating directly on schemes is that of *intersection types* [34, 35]. In this setting, types embedding a property automaton are assigned to terms of a scheme. These types are closely related to our domains as we explain later. Recently, saturation techniques were transferred to intersection types by Broadbent and Kobayashi [8]. The typing algorithm is then a least fixed point computation analogous to an optimised version of our Kleene iteration, restricted to deterministic schemes. This has led to one of the most competitive model-checking tools for schemes [33].

One may find solutions to our language inclusion problems by reductions to many of the above works. For example, from an inclusion game for schemes, we may build a game over an equivalent kind of pushdown automaton.

Then, by forming the product of this pushdown automaton and a determinisation of the NFA, one obtains a reachability game over a pushdown automaton that can be solved by any of the above methods. As we discussed above, such constructions are undesirable for iterative invocations as in Podelski's loop.

We already discussed the relationship to model-theoretic verification algorithms. Abstract interpretation has also been used by both Ramsay [41] and Salvati and Walukiewicz [43] for verification. The former used a Galois connection between safety properties (concrete) and equivalence classes of intersection types (abstract) to recreate decidability results known in the literature. The latter gives a semantics capable of computing properties expressed in weak MSO. Indeed, abstract interpretation has long been used for static analysis of higher-order programs [3].

2 Preliminaries

In this section, we introduce notions and notations that are used throughout the paper.

Complete Partial Orders. Let $(D, \leq)$ be a *partial order*, i.e. D is a set and $\leq$ is a relation that is a partial order on D. We call $(D, \leq)$ *pointed* if there is a greatest element, called the *top element* and denoted by $\top \in D$. A *descending chain* in D is a sequence $(d_i)_{i\in\mathbb{N}}$ of elements in D with $d_i \geq d_{i+1}$. We call $(D, \leq)$ *ω-complete* if every descending chain has a greatest lower bound, called the *meet* or the *infimum* of the chain, denoted by $\bigsqcap_{i\in\mathbb{N}} d_i$. If $(D, \leq)$ is pointed and ω-complete, we call it a *pointed ω-complete partial order (cppo)*. In the following, we will only consider partial orders that are cppos. Note that in the literature, the notion cppo is usually used to refer to the dual concept, i.e. for partial orders with a least element and least upper bounds for ascending chains.

A function $f : D \to D$ is $\sqcap$-*continuous* if for all descending chains $(d_i)_{i\in\mathbb{N}}$ we have $f(\bigsqcap_{i\in\mathbb{N}} d_i) = \bigsqcap_{i\in\mathbb{N}} f(d_i)$. We call a function $f : D \to D$ *monotonic* if for all $d, d' \in D$, $d \leq d'$ implies $f(d) \leq f(d')$. Note that any function that is $\sqcap$-continuous is also monotonic. For a monotonic function, the sequence $\top \geq f(\top) \geq f^2(\top) = f(f(\top)) \geq f^3(\top) \geq \ldots$ is a descending chain in D.

If the function is $\sqcap$-continuous, then the greatest lower bound $\bigsqcap_{i\in\mathbb{N}} f^i(\top)$ of this chain is by the Kleene's theorem the greatest fixed point of f, i.e. $f(\bigsqcap_{i\in\mathbb{N}} f^i(\top)) = \bigsqcap_{i\in\mathbb{N}} f^i(\top)$ and $\bigsqcap_{i\in\mathbb{N}} f^i(\top)$ is larger than any other element d with $f(d) = d$. We also say that $\bigsqcap_{i\in\mathbb{N}} f^i(\top)$ is the greatest solution to the equation $x = f(x)$.

We say that a lattice satisfies the *descending chain condition (DCC)* if every descending chain has to be stationary at some point. In this case, we have $\bigsqcap_{i\in\mathbb{N}} f^i(\top) = \bigsqcap_{i=0}^{i_0} f^i(\top)$ for some index i_0 in $\mathbb{N}$. This allows us to actually compute the greatest fixed point: Starting with $\top$, we iteratively apply f until the result does not change anymore. This process is called *Kleene iteration.* Note that finite cppos, i.e. with finitely many elements in D, trivially satisfy the descending chain condition.

Finite automata. A *non-deterministic finite automaton (NFA)* is a tuple $A = (Q_{NFA}, \Gamma, \delta, q_0, Q_f)$ where Q_{NFA} is a finite set of states, Γ is a finite alphabet, $\delta \subseteq Q_{NFA} \times \Gamma \times Q_{NFA}$ is a (non-deterministic) transition relation, $q_0 \in Q_{NFA}$ is the initial state, and $Q_f \subseteq Q_{NFA}$ is a set of final states. We write $q \xrightarrow{a} q'$ to denote $(q, a, q') \in \delta$. Moreover, given a word $w = a_1 \cdots a_\ell$, we write $q \xrightarrow{w} q'$ whenever there is a sequence of transitions, also called *run*, $q_1 \xrightarrow{a_1} q_2 \xrightarrow{a_2} \cdots \xrightarrow{a_\ell} q_{\ell+1}$ with $q_1 = q$ and $q_{\ell+1} = q'$. The run is accepting if $q = q_0$ and $q' \in Q_f$. The language of A is $\mathcal{L}(A) = \{w \mid q_0 \xrightarrow{w} q \in Q_f\}$.

3 Higher-Order Recursion Schemes

We introduce higher-order recursion schemes, *schemes* for short, following the presentation in [22]. Schemes can be understood as grammars generating the computation trees of programs in a functional language. As is common in functional languages, we need a typing discipline. To avoid confusion with type-based approaches to higher-order model checking [34, 40, 35], we refer to types as *kinds.* Kinds define the functionality of terms, without specifying the data domain. Technically, the only data domain is the ground kind o, from which (potentially higher-order) function kinds are derived by composition:

$$\kappa \ ::= \ o \ \mid \ (\kappa_1 \to \kappa_2) \ .$$

We usually omit the brackets and assume that the arrow associates to the right, e.g. $o \to o \to o$ stands for $o \to (o \to o)$. The number of arguments to a kind is called the *arity.* The *order* defines the functionality of the arguments: A first-order kind defines functions that act on values, a second-order kind functions that expect functions as parameters. Formally, we have

$$\begin{aligned} \mathsf{arity}(o) &= 0, & \mathsf{order}(o) &= 0, \\ \mathsf{arity}(\kappa_1 \to \kappa_2) &= \mathsf{arity}(\kappa_2) + 1, & \mathsf{order}(\kappa_1 \to \kappa_2) &= \max(\mathsf{order}(\kappa_1) + 1, \mathsf{order}(\kappa_2)) \ . \end{aligned}$$

Let K be the set of all kinds. Higher-order recursion schemes assign kinds to symbols from different alphabets, namely non-terminals, terminals, and variables. Let Γ be a set of such *kinded symbols.* For each kind κ, we denote by Γ^κ the restriction of Γ to the symbols with kind κ. The *terms* $\mathcal{T}^\kappa(\Gamma)$ of kind κ over Γ are defined by simultaneous induction over all kinds. They form the smallest set satisfying

1. $\Gamma^\kappa \subseteq \mathcal{T}^\kappa(\Gamma)$,
2. $\bigcup_{\kappa_1}\{t\ v \mid t \in \mathcal{T}^{\kappa_1 \to \kappa_2}(\Gamma), v \in \mathcal{T}^{\kappa_1}(\Gamma)\} \subseteq \mathcal{T}^{\kappa_2}(\Gamma)$, and
3. $\{\lambda x.t \mid x \in \mathcal{T}^{\kappa_1}(\Gamma), t \in \mathcal{T}^{\kappa_2}(\Gamma)\} \subseteq \mathcal{T}^{\kappa_1 \to \kappa_2}(\Gamma)$.

If term t is of kind κ, we also write $t\colon \kappa$. We use $\mathcal{T}(\Gamma)$ for the set of all terms over Γ. We say a term is *λ-free* if it contains no sub-term of the form $\lambda x.t$. A term is *variable closed* if all occurring variables are bound a preceding λ-expression.

▶ **Definition 1** (Higher Order Recursion Scheme)**.** A *higher order recursion scheme*, shortly referred to as *scheme*, is a tuple $G = (V, N, T, R, S)$, where V is a finite set of kinded symbols called *variables*, T is a finite set of kinded symbols called *terminals*, and N is a finite set of kinded symbols called *non-terminals* with $S \in N$ the *initial symbol.* The sets V, T, and N are pairwise disjoint. The finite set R consists of *rewriting rules* of the form $F = \lambda x_1 \ldots \lambda x_n.e$, where $F \in N$ is a non-terminal of kind $\kappa_1 \to \ldots \kappa_n \to o$, $x_1, \ldots, x_n \in V$ are variables of the required kinds, and e is a λ-free, variable-closed term of ground kind from $\mathcal{T}^o(T \uplus N \uplus \{x_1\colon \kappa_1, \ldots, x_k\colon \kappa_n\})$.

The semantics of G is defined by rewriting subterms according to the rules in R. A *context* is a term $C[\bullet] \in \mathcal{T}(\Gamma \uplus \{\bullet\colon o\})$ in which $\bullet$ occurs exactly once. Given a context $C[\bullet]$ and a term $t : o$, we obtain $C[t]$ by replacing the unique occurrence of $\bullet$ in $C[\bullet]$ by t. Formally, $t \Rightarrow_G t'$ if there is a context $C[\bullet]$, a rule $F = \lambda x_1 \ldots \lambda x_n.e$, and a term $F\ t_1\ \ldots\ t_n : o$ such that $t = C[F\ t_1\ \ldots\ t_n]$ and $t' = C[e[x_1 \mapsto t_1, \ldots, x_n \mapsto t_n]]$. In other words, we replace one occurrence of F in t by a right-hand side of a rewriting rule, while properly instantiating the variables. We call such a replaceable $F\ t_1\ \ldots\ t_n$ a *reducible expression (redex).* The rewriting step is *outermost to innermost (OI)* if there is no redex that contains the rewritten one as a proper subterm.

The OI-language $\mathcal{L}(G)$ of G is the set of all (finite, ranked, labelled) trees T over the terminal symbols that can be created from the initial symbol S via OI-rewriting steps. We will restrict the rewriting relation to OI-rewritings in the rest of this paper. Haddad [22] has shown this restriction does not influence the language generated by the scheme, i.e. $\mathcal{L}(G)$ is also the set of all trees derivable by arbitrary rewriting steps.

Word-Generating Schemes. We consider schemes that generate words rather than trees. A *word-generating scheme* is a scheme with terminals $T \uplus \{\$: o\}$ where exactly one terminal symbol \$ has kind o and all other terminals are of kind $o \to o$. A tree generated by such a scheme has the shape $a_1\ (a_2\ (\cdots\ (a_k\ \$)))$, and we understand it as the finite word $a_1 a_2 \ldots a_k \in T^*$. We also see $\mathcal{L}(G)$ as a language of finite words.

Determinism. The above schemes are non-deterministic in that there may be several rules to rewrite a non-terminal. We associate with a non-deterministic scheme $G = (V, N, T, R, S)$ a deterministic scheme G^{det} with exactly one rule per non-terminal. Intuitively, G^{det} resolves the non-determinism by making it explicit in terms of new terminal symbols. Formally, let $F : \kappa$ be a non-terminal in G with rules $F = t_1$ to $F = t_\ell$. We may assume that each t_i is of the shape $\lambda x_1 \ldots \lambda x_k.e_i$, where e_i is lambda-free.

We introduce a new terminal symbol $op_F : o \to o \to \ldots \to o$ of arity ℓ. Let the set of all these terminals be $T^{det} = \{op_F \mid F \in N\}$. The set of rules R^{det} now consists of a single rule for each non-terminal, namely $F = \lambda x_1 \ldots \lambda x_k.op_F\ e_1\ \cdots\ e_\ell$. The original rules in R are removed. This yields $G^{det} = (V, N, T \uplus T^{det}, R^{det}, S)$. The advantage of resolving the non-determinism explicitly is that we can give a semantics to non-deterministic choices that depends on the non-terminal instead of having to treat non-determinism uniformly.

Semantics. Let $G = (V, N, T, R, S)$ be a deterministic scheme. A *model* of G is a pair $\mathcal{M} = (\mathcal{D}, \mathcal{I})$, where $\mathcal{D}$ is a family of domains $(\mathcal{D}(\kappa))_{\kappa \in K}$ that satisfies the following: $\mathcal{D}(o)$ is a cppo and $\mathcal{D}(\kappa_1 \to \kappa_2) = \mathit{Cont}(\mathcal{D}(\kappa_1), \mathcal{D}(\kappa_2))$. Here, $\mathit{Cont}(A, B)$ is the set of all $\sqcap$-continuous functions from domain A to B. We comment on this lattice in a moment. The interpretation $\mathcal{I} : T \to \mathcal{D}$ assigns to each terminal $s : \kappa$ an element $\mathcal{I}(s) \in \mathcal{D}(\kappa)$.

The ordering on functions is defined component-wise, $f \leq_{\kappa_1 \to \kappa_2} g$ if $(f\ x) \leq_{\kappa_2} (g\ x)$ for all $x \in \mathcal{D}(\kappa_1)$. For each κ, we denote the top element of $\mathcal{D}(\kappa)$ by $\top_\kappa$. For the ground kind, $\top_o$ exists since $\mathcal{D}(\kappa)$ is a cppo, and $\top_{\kappa_1 \to \kappa_2}$ is the function that maps every argument to $\top_{\kappa_2}$. The meet of a descending chain of functions $(f_i)_{i \in \mathbb{N}}$ is the function defined by $(\bigsqcap_{\kappa_1 \to \kappa_2} (f_i)_{i \in \mathbb{N}})\ x = \bigsqcap_{\kappa_2} (f_i\ x)_{i \in \mathbb{N}}$. Note that the sequence on the right-hand side is a descending chain. Note, we often elide the kind information from $\sqcap$ and $\top$.

The *semantics of terms* defined by a model is a function

$$\mathcal{M}[\![-]\!] : \mathcal{T} \to (N \uplus V \nrightarrow \mathcal{D}) \to \mathcal{D}\ .$$

that assigns to each term built over the non-terminals and terminals again a function. This function expects a valuation $\nu : N \uplus V \nrightarrow \mathcal{D}$ of all non-terminals and the free variables, and returns an element from the domain. We lift $\sqcap$ to descending chains of valuations with $(\bigsqcap_{i \in \mathbb{N}} \nu_i)(y) = \bigsqcap_{i \in \mathbb{N}} (\nu_i(y))$ for $y \in N \uplus V$. We obtain that the set of such valuations is a cppo where the greatest elements are those valuations which assign the greatest elements of the appropriate domain to all arguments.

Since the right-hand sides of rules of the scheme are variable-closed, we do not need a variable valuation for them. We need the variable valuation, however, whenever we proceed by induction on the structure of terms. The semantics is defined by such an induction:

$$\begin{aligned} \mathcal{M}[\![s]\!]\ \nu &= \mathcal{I}(s) & \mathcal{M}[\![t_1\ t_2]\!]\ \nu &= (\mathcal{M}[\![t_1]\!]\ \nu)\ (\mathcal{M}[\![t_2]\!]\ \nu) \\ \mathcal{M}[\![x]\!]\ \nu &= \nu(x) & \mathcal{M}[\![\lambda x : \kappa.t_1]\!]\ \nu &= d \in \mathcal{D}(\kappa) \mapsto \mathcal{M}[\![t_1]\!]\ \nu[x \mapsto d] \\ \mathcal{M}[\![F]\!]\ \nu &= \nu(F)\ . \end{aligned}$$

We show that $\mathcal{M}[\![t]\!]$ is $\sqcap$-continuous for all terms t. This primarily follows inductively from continuity of the functions in the domain, but requires some care when handling application.

▶ **Proposition 2.** *For all t, $\mathcal{M}[\![t]\!]$ is $\sqcap$-continuous (in ν) over the respective lattice.*

Given a model $\mathcal{M}$, we understand the rules $F_1 = t_1, \ldots, F_k = t_k$ of the scheme (recalling that we assumed the scheme was deterministic) as a function

$$\mathit{rhs}_{\mathcal{M}} : (N \to \mathcal{D}) \to (N \to \mathcal{D})\ , \quad \text{where} \quad \mathit{rhs}_{\mathcal{M}}(\nu)(F_j) = \mathcal{M}[\![t_j]\!]\ \nu\ .$$

Note that, since the right-hand sides are variable-closed, the $\mathcal{M}[\![t_j]\!]$ are really functions in the non-terminals. Provided $\mathcal{M}[\![t_1]\!]$ to $\mathcal{M}[\![t_k]\!]$ are $\sqcap$-continuous (in the valuation of the non-terminals), the function $\mathit{rhs}_{\mathcal{M}}$ will be $\sqcap$-continuous. This allows us to apply Kleene's fixed point theorem, stating that there is a greatest fixed point of $\mathit{rhs}_{\mathcal{M}}$ and the following iteration converges to it. The initial value is the greatest element $\sigma^0_{\mathcal{M}}$ where $\sigma^0_{\mathcal{M}}(F_j) = \top_j$ with $\top_j$ the top element of $\mathcal{D}(\kappa_j)$. The $(i+1)^{\text{th}}$ approximant is computed by evaluating the right-hand side at the i^{th} solution, $\sigma^{i+1}_{\mathcal{M}} = \mathit{rhs}_{\mathcal{M}}(\sigma^i_{\mathcal{M}})$. The greatest fixed point is the tuple $\sigma_{\mathcal{M}}$ defined by

$$\sigma_{\mathcal{M}} = \bigsqcap_{i \in \mathbb{N}} \sigma^i_{\mathcal{M}} = \bigsqcap_{i \in \mathbb{N}} \mathit{rhs}^i_{\mathcal{M}}(\sigma^0_{\mathcal{M}})\ .$$

It can be understood as the greatest solution to the equation $\nu = \mathit{rhs}_{\mathcal{M}}(\nu)$. We call this greatest solution $\sigma_{\mathcal{M}}$ the *semantics of the scheme* in the model.

Since we have not syntactically defined the evaluation of a λ-term, our development will need a simple substitution lemma.

▶ **Lemma 3.** *For all* $\nu : N \uplus V \nrightarrow \mathcal{D}$*, we have* $\mathcal{M}\llbracket(\lambda x.t)\ t'\rrbracket\ \nu = \mathcal{M}\llbracket t[x \mapsto t']\rrbracket\ \nu$*.*

4 Games

The goal of this paper is to solve higher-order games, games whose game arena is given by a higher-order recursion scheme. We assume that the derivation process is controlled by two players. To this end, we divide the non-terminals of a word-generating scheme into those owned by the existential player $\Diamond$ and those owned by the universal player $\Box$. Whenever a non-terminal is replaced during the derivation, we think of the owner choosing the rule of the non-deterministic scheme. The winning condition of the game is given by an automaton A, Player $\Diamond$ attempts to produce a word that is in $\mathcal{L}(A)$, while Player $\Box$ attempts to produce a word outside of $\mathcal{L}(A)$.

▶ **Definition 4.** A *higher-order game* is a triple $\mathcal{G} = (G, A, O)$ where G is a word-generating scheme, A is an NFA, $O : N \to \{\Diamond, \Box\}$ is a partitioning of the non-terminals of G.

A play of the game is a sequence of OI-rewriting steps. Since terms generate words, it is unambiguous which term forms the next redex to be rewritten. In particular, all terms are of the form $a_1(a_2(\cdots(a_k(t))))$, where t is either \$ or a redex $F\ t_1\ \cdots\ t_m$. If $O(F) = \Diamond$ then Player $\Diamond$ chooses a rule $F = \lambda x_1 \ldots \lambda x_m.e$ to apply, else Player $\Box$ chooses the rule. This moves play to $a_1\ (a_2\ (\cdots\ (a_k\ e[x_1 \mapsto t_1, \ldots, x_m \mapsto t_m])))$.

Each play begins at the initial non-terminal S, and continues either ad infinitum or until a term $a_1\ (a_2\ (\cdots\ (a_k\ \$)))$, understood as the word $w = a_1 \ldots a_k$, is produced. Infinite plays do not produce a word and are won by Player $\Diamond$. Finite maximal plays produce a such a word w. Player $\Diamond$ wins such a play whenever $w \in \mathcal{L}(A)$, Player $\Box$ wins if $w \in \overline{\mathcal{L}(A)}$.

Since these games have a Borel winning condition, they are determined [37]. That is, either Player $\Diamond$ or Player $\Box$ has a winning strategy.

Determining the Winner of a Higher-Order Game (HOG)
Input: A higher-order game $\mathcal{G}$ and an NFA A.
Question: Does Player $\Diamond$ win $\mathcal{G}$?

Our contribution is a fixed-point algorithm to decide HOG. We derive it in three steps. First, we develop a concrete model for higher-order games whose semantics captures the above winning condition. Second, we introduce a framework that for two models and a mapping between them guarantees that the mapping of the greatest fixed point with respect to the one model is the greatest fixed point with respect to the other mode, under the assumption of some properties. Finally, we introduce an abstract model that uses a finite ground domain. The solution of HOG can be read off from the semantics of the abstract model, which in turn can be computed via Kleene iteration. We instantiate the framework for the concrete and abstract model to prove the soundness of the algorithm.

Concrete Semantics. Consider a HOG instance $\mathcal{G} = (G, A, O)$. Let G^{det} be the determinized version of G. Our goal is to define a model $\mathcal{M}^C = (\mathcal{D}^C, \mathcal{I}^C)$ such that the semantics of G^{det} in this model allows us to decide $\mathcal{G}$. Recall that we only have to define the ground domain. For composed kinds, we use the functional lifting discussed in Section 3.

Our idea is to associate to kind o the set of positive Boolean formulas where the atomic propositions are words in T^*. To be able to reuse the definition, we define formula domains in more generality as follows.

Given a (potentially infinite) set P of atomic propositions, the *positive Boolean formulas* $\mathsf{PBool}(P)$ over P are defined to contain true, false, every proposition p from P, and compositions of formulas via conjunction and disjunction. We work up to logical equivalence, which means we treat formulas ϕ_1 and ϕ_2 as equal as long as they are logically equivalent.

Unfortunately, if the set P is infinite, $\mathsf{PBool}(P)$ is not a cppo, since the meet of a descending chain of formulas might not be a finite formula. The idea of our domain is to have infinite conjunctions of formulas. As common in logic, we represent them as infinite sets. Therefore, we consider the set of all sets of (finite) positive Boolean formulas $\mathcal{P}(\mathsf{PBool}(T^*)) \setminus \{\emptyset\}$ factorized modulo logical equivalence. To be precise, the sets may be finite or infinite, but they must be non-empty.

To define the factorization, let an assignment to the atomic propositions be given by a subset of $P' \subseteq P$. The atomic proposition p is true if $p \in P'$. An assignment satisfies a Boolean formula, if the formula evaluates to true in that assignment. It satisfies a set of Boolean formulas, if it satisfies all elements. Given two sets of formulas Φ_1 and Φ_2, we write $\Phi_1 \Rightarrow \Phi_2$, if every assignment that satisfies Φ_1 also satisfies Φ_2. Two sets of formulas are equivalent, denoted $\Phi_1 \Leftrightarrow \Phi_2$, if $\Phi_1 \Rightarrow \Phi_2$ and $\Phi_2 \Rightarrow \Phi_1$ holds.

The ordering on these factorized sets is implication (which by transitivity is independent of the representative). The top element is the set $\{\mathsf{true}\}$, which is implied by every set. The conjunction of two sets is union. Note that it forms the meet in the partial order, and moreover note that meets over arbitrary sets exist, in particular the domain is a cppo. We will also need an operation of disjunction. In the binary, it is defined by $\Phi_1 \vee \Phi_2 = \{\phi_1 \vee \phi_2 \mid \phi_1 \in \Phi_1, \phi_2 \in \Phi_2\}$. We will also use disjunctions of higher (but finite) arity where convenient. Note that the disjunction on finite formulas is guaranteed to result in a finite formula. Therefore, the above is well-defined.

Alltogether, the ground domain of the model is $\mathcal{D}^C(o) = (\mathcal{P}(\mathsf{PBool}(T^*)) \setminus \{\emptyset\})/_{\Leftrightarrow}$. With this ground domain in place, the higher-order domains are defined as continuous functions as in Section 3. We make explicit the conjunction of functions $f, g \in \mathcal{D}^C(\kappa_1 \to \kappa_2)$, which corresponds to the meet. The result is the function $f \wedge g$ which forwards the operation to the domain κ_2. Formally, for $x \in \mathcal{D}^C(\kappa_1)$ we have $(f \wedge g)\ x = f(x) \wedge g(x)$. This concept can be extended to provide the meet for infinite sets of functions. Furthermore, finite disjunction of functions, e.g. $f \vee g$, are defined similarly.

In our case, the assignment $P' \subseteq T^*$ of interest is the language of the automaton A. Player $\Diamond$ will win the game if the concrete semantics assigns a set of formulas to S that is true with respect to $\mathcal{L}(A)$. Concerning the interpretation, the endmarker \$ yields the set of formulas consisting of the single literal that is the empty word, $\mathcal{I}^C(\$) = \{\varepsilon\}$. A first-order terminal $a : o \to o$ prepends a to a given word w. When applied to formulas, $\mathsf{prepend}_a$ distributes over conjunction and disjunction:

$$\mathsf{prepend}_a(\phi) = \begin{cases} aw & \phi = w\ , \\ \mathsf{prepend}_a(\phi_1)\ op\, \mathsf{prepend}_a(\phi_2) & \phi = \phi_1\ op\, \phi_2 \text{ and } \ op \in \{\wedge, \vee\}\ , \\ \phi & \phi = \mathsf{true}\ . \end{cases}$$

Applying the operation to sets of formulas means applying it to every element. We write $\mathsf{prepend}_w$ for $w = a_1 \ldots a_n$ as an abbreviation for $\mathsf{prepend}_{a_1} \circ \cdots \circ \mathsf{prepend}_{a_n}$. The interpretation of op_N of arity ℓ is an ℓ-ary conjunction (resp. disjunction) if Player $\Box$ (resp. $\Diamond$) owns N.

For $\mathcal{M}^C = (\mathcal{D}^C, \mathcal{I}^C)$ to be a model, we need our interpretation of terminals to be $\sqcap$-continuous. This follows largely by the distributivity of our definitions.

▶ **Lemma 5.** *For all non-ground terminals s, $\mathcal{I}^C(s)$ is $\sqcap$-continuous.*

We need to show that the concrete semantics matches the original semantics of the game.

▶ **Theorem 6.** *$\sigma_{\mathcal{M}^C}(S)$ is satisfied by $\mathcal{L}(A)$ iff there is a winning strategy for Player $\Diamond$.*

To show that if $\sigma_{\mathcal{M}^C}(S)$ is satisfied by $\mathcal{L}(A)$ there is a winning strategy for $\Diamond$ we show that $\Diamond$ can maintain the invariant that the semantics of any term constructed during a play is satisfied by $\mathcal{L}(A)$. Thus, if play terminates, it is guaranteed to be in a term representing a word accepted by A. The proof that $\sigma_{\mathcal{M}^C}(S)$ being unsatisfied implies a winning strategy for $\Box$ is more involved and requires the definition of a "correctness" relation between semantics and terms that is lifted to the level of functions, and shown to hold inductively.

5 Framework

Consider the deterministic scheme G together with two models (left and right) $\mathcal{M}_l = (\mathcal{D}_l, \mathcal{I}_l)$ and $\mathcal{M}_r = (\mathcal{D}_r, \mathcal{I}_r)$. Our goal is to relate the semantics in these models in the sense that $\sigma_{\mathcal{M}_r} = \alpha(\sigma_{\mathcal{M}_l})$, the semantics in $\mathcal{M}_r$ is the image of the semantics in $\mathcal{M}_l$ under a function α. Such fixed-point transfer results are well-known in abstract interpretation [3]. We provide a handy generalization to higher order. It is handy in that we give easy to instantiate conditions on α, $\mathcal{M}_l$, and $\mathcal{M}_r$ that yield the above equality.

For the terminology, an *abstraction* is a function $\alpha : \mathcal{D}_l(o) \to \mathcal{D}_r(o)$. We impose some requirements below. To lift the abstraction to function domains, we define the notion of *being compatible with* α. By definition, all ground elements $v_l \in \mathcal{D}_l(o)$ are compatible with α. For function domains, compatibility and the abstraction are defined as follows.

▶ **Definition 7.** Assume α and the notion of compatibility are defined on $\mathcal{D}_l(\kappa_1)$ and $\mathcal{D}_l(\kappa_2)$. Let $\top^l_\kappa$ (resp. $\top^r_\kappa$) be the greatest element of $\mathcal{D}_l(\kappa)$ (resp. $\mathcal{D}_r(\kappa)$) for each κ.

1. Function $f \in \mathcal{D}_l(\kappa_1 \to \kappa_2)$ is compatible with α, if
   - a. for all compatible $v_l, v'_l \in \mathcal{D}_l(\kappa_1)$ with $\alpha(v_l) = \alpha(v'_l)$ we have $\alpha(f\ v_l) = \alpha(f\ v'_l)$, and
   - b. for all compatible $v_l \in \mathcal{D}_l(\kappa_1)$ we have that $f\ v_l$ is compatible.
2. We define $\alpha(f) \in \mathcal{D}_r(\kappa_1 \to \kappa_2)$ as follows.
   - a. If f is compatible, we set $\alpha(f)\ v_r = \alpha(f\ v_l)$, provided there is a compatible $v_l \in \mathcal{D}_l(\kappa_1)$ with $v_r = \alpha(v_l)$, and $\alpha(f)\ v_r = \top^r_{\kappa_2}$ otherwise.
   - b. If f is not compatible, $\alpha(f) = \top^r_{\kappa_1 \to \kappa_2}$.

We lift α to valuations $\nu : N \uplus V \nrightarrow \mathcal{D}_l$ by $\alpha(\nu)(F) = \alpha(\nu(F))$ and similar for x. We also lift compatibility to valuations $\nu : N \uplus V \nrightarrow \mathcal{D}_l$ by requiring $\nu(F)$ to be compatible for all $F \in N$ and similar for $x \in V$.

Compatibility intuitively states that the function on the concrete domain is not more precise than what the abstraction function distinguishes. This allows us to define the abstraction of a function by applying the function and abstracting the result, $\alpha(f)\ \alpha(v_l) = \alpha(f\ v_l)$. Note that compatibility ensures the independence of the choice of v_l. In our development, we will only be interested in compatible functions.

The conditions needed for the fixed-point transfer are the following.

▶ **Definition 8.** Function α is *precise* for $\mathcal{M}_l$ and $\mathcal{M}_r$, if

(P1) $\alpha(\mathcal{D}_l(o)) = \mathcal{D}_r(o)$,
(P2) $\alpha : \mathcal{D}_l(o) \to \mathcal{D}_r(o)$ is $\sqcap$-continuous,
(P3) $\alpha(\top^l_o) = \top^r_o$,

(P4) $\alpha(\mathcal{I}_l(s)) = \mathcal{I}_r(s)$ for all terminals $s\colon o$, and similarly $\alpha(\mathcal{I}_l(s)\ v_l) = \mathcal{I}_r(s)\ \alpha(v_l)$ for all terminals $s : \kappa_1 \to \kappa_2$ and all compatible $v_l \in \mathcal{D}_l(\kappa_1)$,

(P5) $\mathcal{I}_l(s)\ v_l$ is compatible for all terminals $s : \kappa_1 \to \kappa_2$, and all compatible $v_l \in \mathcal{D}_l(\kappa_1)$.

In the case of games, we introduce terminals of higher order to resolve the non-determinism in the scheme, which is why we give the definition in this generality. (P1) is surjectivity of the abstraction. (P2) states that the abstraction is well-behaved wrt. meets. Requirement (P3) says that the greatest element is mapped as expected. Note that all three requirements are only posed for the ground domain. Lemmas 9, 10, and 11 show that they generalize to function domains by the definition of function abstraction. (P4) is that the interpretations of terminals in concrete and abstract model are suitably related. The last requirement (P5) is compatibility. Both (P4) and (P5) are generalized to terms in Lemma 12.

▶ **Lemma 9.** *If* (P1) *holds, then for every $\kappa \in K$ and every $v_r \in \mathcal{D}_r(\kappa)$ there is a compatible $v_l \in \mathcal{D}_l(\kappa)$ with $\alpha(v_l) = v_r$.*

▶ **Lemma 10.** *If* (P1) *and* (P2) *hold, then for all $\kappa \in K$ and all descending chains of compatible elements $(f_i)_{i\in\mathbb{N}}$ in $\mathcal{D}(\kappa)$, we have $\bigsqcap_{i\in\mathbb{N}} f_i$ compatible and $\alpha(\bigsqcap_{i\in\mathbb{N}} f_i) = \bigsqcap_{i\in\mathbb{N}} \alpha(f_i)$.*

▶ **Lemma 11.** *If* (P3) *holds, then $\alpha(\top^l_\kappa) = \top^r_\kappa$ for all $\kappa \in K$.*

To prove $\alpha(\sigma_{\mathcal{M}_l}) = \sigma_{\mathcal{M}_r}$, we need that $rhs_{\mathcal{M}_r}$ is an exact abstract transformer of $rhs_{\mathcal{M}_l}$. The following lemma states this for all terms t, in particular those that occur in the equations. The generalization to product domains is immediate. Note that the result is limited to compatible valuations, but this will be sufficient for our purposes. Due to space reasons, we have relegated the proof of this lemma to Appendix D.4. It proceeds by induction on the structure of terms, while simultaneously proving $\mathcal{M}_l[\![t]\!]$ compatible with α.

▶ **Lemma 12.** *Assume* (P1), (P4), *and* (P5) *hold. For all terms t and all compatible ν, we have $\mathcal{M}_l[\![t]\!]\ \nu$ compatible and $\alpha(\mathcal{M}_l[\![t]\!]\ \nu) = \mathcal{M}_r[\![t]\!]\ \alpha(\nu)$.*

We are now prepared to show our fixed-point transfer result.

▶ **Theorem 13** (Fixed-Point Transfer)**.** *Let G be a scheme with models $\mathcal{M}_l$ and $\mathcal{M}_r$. Let σ_l and σ_r be the corresponding semantics. If $\alpha : \mathcal{D}_l \to \mathcal{D}_r$ is precise, we have $\sigma_r = \alpha(\sigma_l)$.*

6 Domains for Higher-Order Games

We propose two domains, called abstract and optimized, that allow us to compute the winning regions of higher-order games. The computation is a standard fixed-point iteration, and the optimized domain is such that this iteration has optimal complexity. Correctness follows by instantiating the previous framework.

Abstract Semantics. Our goal is to define an abstract model for games that (1) suitably relates to the concrete model from Section 4 and (2) is computable. By a suitable relation, we mean the two models should relate via an abstraction function. Provided the conditions on precision hold, correctness of the abstraction then follows from Theorem 13. Combined with Theorem 6, this will allow us to solve HOG. Computable in particular means the domain should be finite and the operations should be efficiently computable.

We define the $\mathcal{M}^A = (\mathcal{D}^A, \mathcal{I}^A)$ as follows. Again, we resolve the non-determinism into Boolean formulas. But rather than tracking the precise words generated by the scheme, we only track the current set of states of the automaton. To achieve the surjectivity required by precision, we restrict the powerset to those sets of states from which a word is accepted. Let $\mathsf{acc}(w) = \{q \mid q \xrightarrow{w} q_f \in Q_f\}$. For a language L we have $\mathsf{acc}(L) = \{\mathsf{acc}(w) \mid w \in L\}$. The abstract domain for terms of ground kind is $\mathcal{D}^A(o) = \mathsf{PBool}(\mathsf{acc}(T^*))$. The lifting to functions is as explained in Section 3. Satisfaction is now defined relative to a set Ω of elements of $\mathcal{P}(Q_{NFA})$ (cf. Section 4). With finitely many atomic propositions, there are only finitely many formulas (up to logical equivalence). This means we no longer need sets of formulas to represent infinite conjunctions, but can work with plain formulas. The ordering is thus the ordinary implication with the meet being conjunction and top being true.

The interpretation of ground terms is $\mathcal{I}^A(\$) = Q_f$ and $\mathcal{I}^A(a) = \mathsf{pre}_a$. Here pre_a is the predecessor computation under label a, $\mathsf{pre}_a(Q) = \{q' \in Q_{NFA} \mid q' \xrightarrow{a} q \in Q\}$. It is lifted to formulas by distributing it over conjunction and disjunction. The composition operators are again interpreted as conjunctions and disjunctions, depending on the owner of the non-terminal. Since we restrict the atomic propositions to the acceptance sets of words, we have to show that the interpretations stay inside this restricted set.

▶ **Lemma 14.** *The interpretations are defined on the abstract domain.*

The proof of $\sqcap$-continuity is standard.

▶ **Lemma 15.** *For all terminals s, $\mathcal{I}^A(s)$ is $\sqcap$-continuous over the respective lattices.*

Recall that the concrete model for games is $\mathcal{M}^C = (\mathcal{D}^C, \mathcal{I}^C)$, where the domain is $\mathcal{D}^C = \mathcal{P}(\mathsf{PBool}(T^*))$. To relate this model to the abstract one from above, we define the abstraction function $\alpha : \mathcal{D}^C(o) \to \mathcal{D}^A(o)$. It leaves the Boolean structure of a formula unchanged but maps every word (which is an atomic proposition) to the set of states from which this word is accepted. For a set of formulas, we take the conjunction of the abstraction of the elements. This conjunction is finite as we work over a finite domain, so there is no need to worry about infinite syntax. Technically, we define α on $\mathsf{PBool}(T^*)$ by

$$\alpha(\phi) = \begin{cases} \mathsf{acc}(w) & \text{if } \phi = w, \\ \alpha(\phi_1)\; op\, \alpha(\phi_2) & \text{if } \phi = \phi_1\; op\, \phi_2 \text{ and } op \in \{\wedge, \vee\}, \\ \phi & \text{if } \phi = \mathsf{true}. \end{cases}$$

Given a set of formulas $\Phi \in \mathcal{P}(\mathsf{PBool}(T^*))$, we define $\alpha(\Phi) = \bigwedge_{\phi \in \Phi} \alpha(\phi)$.

This definition is suitable in that $\alpha(\sigma_{\mathcal{M}^C}) = \sigma_{\mathcal{M}^A}$ entails the following.

▶ **Theorem 16.** *$\sigma_{\mathcal{M}^A}(S)$ is satisfied by $\{Q \in \mathsf{acc}(T^*) \mid q_0 \in Q\}$ iff Player $\diamond$ wins $\mathcal{G}$.*

To see that the theorem is a consequence of the fixed-point transfer, observe that

$$\{Q \in \mathsf{acc}(T^*) \mid q_0 \in Q\} = \mathsf{acc}(\mathcal{L}(A)).$$

Then, by $\sigma_{\mathcal{M}^A} = \alpha(\sigma_{\mathcal{M}^C})$ we have $\mathsf{acc}(\mathcal{L}(A))$ satisfies $\sigma_{\mathcal{M}^A}(S)$ iff it also satisfies $\alpha(\sigma_{\mathcal{M}^C}(S))$. This holds iff $\mathcal{L}(A)$ satisfies $\sigma_{\mathcal{M}^C}(S)$ (a simple induction over formulas). By Theorem 6, this occurs iff Player $\diamond$ wins the game.

It remains to establish $\alpha(\sigma_{\mathcal{M}^C}) = \sigma_{\mathcal{M}^A}$. With the framework developed in the previous section, the fixed-point transfer is a consequence of precision, Theorem 13. The proof of the following proposition is a routine calculation.

▶ **Proposition 17.** *α is precise. Hence, $\alpha(\sigma_{\mathcal{M}^C}) = \sigma_{\mathcal{M}^A}$.*

Optimized Semantics. The above model yields a decision procedure for HOG. Unfortunately, the complexity does not match the lower bound that we prove in the appendix. Therefore, we now show an optimized model, based on the above one, that is able to close the gap.

The idea of the optimized model is to resolve the atomic propositions in $\mathcal{M}^A$, which are sets of states, into disjunctions among the states. The reader familiar with inclusion algorithms will find this decomposition odd. Surprisingly, it works for games.

We first define $\alpha : \mathsf{PBool}(\mathsf{acc}(T^*)) \to \mathsf{PBool}(Q_{NFA})$. The optimized domain will then be based on the image of α. This guarantees surjectivity. For a set of states Q, we define $\alpha(Q) = \bigvee Q = \bigvee_{q \in Q} q$. For a formula, the abstraction function is defined to distribute over conjunction and disjunction. The optimized model is $\mathcal{M}^O = (\mathcal{D}^O, \mathcal{I}^O)$ with ground domain $\alpha(\mathsf{PBool}(\mathsf{acc}(T^*)))$. The interpretation is $\mathcal{I}^O(\$) = \bigvee Q_f$. For a, we resolve the set of predecessors into a disjunction, $\mathcal{I}^O(a)\ q = \bigvee \mathsf{pre}_a(\{q\})$. The function distributes over conjunction and disjunction. Finally, $\mathcal{I}^O(op_F)$ is conjunction or disjunction of formulas, depending on the owner of the non-terminal. Since we ause a restricted domain, we have to argue that the operations are actually defined in the sense that they do not leave the domain.

▶ **Lemma 18.** *The interpretations are defined on the optimized domain.*

The following property is straightforward to prove.

▶ **Lemma 19.** *For all terminals s, $\mathcal{I}^O(s)$ is $\sqcap$-continuous over the respective lattices.*

We again show precision. This will allow us to apply the fixed-point transfer and compute the image of the abstract semantics as the semantics in the optimized model.

▶ **Proposition 20.** *α is precise. Hence, $\alpha(\sigma_{\mathcal{M}^A}) = \sigma_{\mathcal{M}^O}$.*

▶ **Theorem 21.** *$\sigma_{\mathcal{M}^O}(S)$ is satisfied by $\{q_0\}$ iff Player $\Diamond$ wins $\mathcal{G}$.*

It is sufficient to show $\sigma_{\mathcal{M}^A}(S)$ is satisfied by $\{Q \in \mathsf{acc}(T^*) \mid q_0 \in Q\}$ iff $\sigma_{\mathcal{M}^O}(S)$ is satisfied by $\{q_0\}$. The statement then follows from Theorem 16. Propositions Q in $\sigma_{\mathcal{M}^A}(S)$ are resolved into disjunctions $\bigvee Q$ in $\sigma_{\mathcal{M}^O}(S)$. For such a proposition, we have $Q \in \{Q \in \mathsf{acc}(T^*) \mid q_0 \in Q\}$ iff $\bigvee Q$ is satisfied by $\{q_0\}$. This equivalence propagates to the formulas $\sigma_{\mathcal{M}^A}(S)$ and $\sigma_{\mathcal{M}^O}(S)$ as the Boolean structure coincides. The latter follows from $\alpha(\sigma_{\mathcal{M}^A}(S)) = \sigma_{\mathcal{M}^O}(S)$.

Complexity. It remains to consider the complexity of the problem and of the suggested algorithm. To solve HOG, we compute the semantics $\sigma_{\mathcal{M}^O}$ with respect to the optimised model and then evaluate $\sigma_{\mathcal{M}^O}(S)$ at the assignment $\{q_0\}$. Assume that the highest order of any non-terminal in the input scheme $\mathcal{G}$ is k.

▶ **Proposition 22.** *The semantics $\sigma_{\mathcal{M}^O}$can be computed in $(k+1)$EXP.*

Crucial to the proof is showing that the number of iterations needed to compute the greatest fixed point is at most $(k+1)$-times exponential. To this end, we show a suitable upper bound on the length of strictly descending chains in the domains assigned by $\mathcal{D}^O$.

▶ **Proposition 23.** *Determining whether Player $\Diamond$ wins $\mathcal{G}$ is $(k+1)$EXP-hard for $k > 0$.*

The lower bound is via a reduction from the word membership problem for alternating k-iterated pushdown automata with polynomially bounded auxiliary work-tape. This problem was shown by Engelfriet to be kEXP-hard. We can reduce this problem to HOG via well-known translations between iterated stack automata and recursion schemes, using the regular language A to help simulate the work-tape.

Together, these results show the following corollary.

▶ **Corollary 24.** HOG *is $(k+1)$EXP-complete for order-k schemes andn $k > 0$.*

References

1 P. A. Abdulla, Y. Chen, L. Clemente, L. Holík, C.-D. Hong, R. Mayr, and T. Vojnar. Simulation subsumption in Ramsey-based Büchi automata universality and inclusion testing. In *CAV*, volume 6174 of *LNCS*, pages 132–147. Springer, 2010.

2 P. A. Abdulla, Y. Chen, L. Clemente, L. Holík, C.-D. Hong, R. Mayr, and T. Vojnar. Advanced Ramsey-based Büchi automata inclusion testing. In *CONCUR*, volume 6901 of *LNCS*, pages 187–202. Springer, 2011.

3 S. Abramsky and C. Hankin. An introduction to abstract interpretation. In *Abstract Interpretation of declarative languages*, volume 1, pages 63–102. Ellis Horwood, 1987.

4 K. Aehlig. A finite semantics of simply-typed lambda terms for infinite runs of automata. *Logical Methods in Computer Science*, 3(3), 2007.

5 A. Bouajjani, J. Esparza, and O. Maler. Reachability analysis of pushdown automata: Application to model-checking. In *CONCUR*, volume 1243 of *LNCS*, pages 135–150. Springer, 1997.

6 A. Bouajjani and A. Meyer. Symbolic Reachability Analysis of Higher-Order Context-Free Processes. In *FSTTCS*, volume 3328 of *LNCS*. Springer, 2004.

7 C. Broadbent, A. Carayol, M. Hague, and O. Serre. A saturation method for collapsible pushdown systems. In *ICALP*, volume 7392 of *LNCS*, pages 165–176. Springer, 2012.

8 C. Broadbent and N. Kobayashi. Saturation-based model checking of higher-order recursion schemes. In *CSL*, volume 23 of *LIPIcs*, pages 129–148. Dagstuhl, 2013.

9 T. Cachat. Symbolic strategy synthesis for games on pushdown graphs. In *ICALP*, volume 2380 of *LNCS*, pages 704–715. Springer, 2002.

10 Th. Cachat. Higher order pushdown automata, the Caucal hierarchy of graphs and parity games. In *ICALP*, volume 2719 of *LNCS*, pages 556–569. Springer, 2003.

11 D. Caucal. On infinite terms having a decidable monadic theory. In *MFCS*, volume 2420 of *LNCS*, pages 165–176. Springer, 2002.

12 W. Damm. The IO- and OI-hierarchies. *Theor. Comp. Sci.*, 20:95–207, 1982.

13 W. Damm and A. Goerdt. An automata-theoretical characterization of the oi-hierarchy. *Inf. Comp.*, 71:1–32, 1986.

14 J. Engelfriet. Iterated stack automata and complexity classes. *Inf. Comput.*, 95(1):21–75, 1991.

15 A. Farzan, Z. Kincaid, and A. Podelski. Proof spaces for unbounded parallelism. In *POPL*, pages 407–420. ACM, 2015.

16 A. Farzan, Z. Kincaid, and A. Podelski. Proving liveness of parameterized programs. In *LICS*. IEEE, 2016.

17 Azadeh Farzan, Zachary Kincaid, and Andreas Podelski. Proofs that count. In *POPL*, pages 151–164. ACM, 2014.

18 A. Finkel, B. Willems, and P. Wolper. A direct symbolic approach to model checking pushdown systems. In *INFINITY*, volume 9, pages 27–37, 1997.

19 S. Fogarty and M. Y. Vardi. Efficient Büchi universality checking. In *TACAS*, volume 6015 of *LNCS*, pages 205–220. Springer, 2010.

20 C. Grellois and P.-A. Melliès. An infinitary model of linear logic. In *FoSSaCS*, volume 9034 of *LNCS*, pages 41–55. Springer, 2015.

21 C. Grellois and P.-A. Melliès. Relational semantics of linear logic and higher-order model checking. In *CSL*, volume 41 of *LIPIcs*, pages 260–276. Dagstuhl, 2015.

22 A. Haddad. IO vs OI in higher-order recursion schemes. In *FICS*, pages 23–30, 2012.

23 M. Hague, A. S. Murawski, C.-H. L. Ong, and O. Serre. Collapsible pushdown automata and recursion schemes. In *LICS*, pages 452–461, 2008.

24 M. Hague and C.-H. L. Ong. Symbolic backwards-reachability analysis for higher-order pushdown systems. In *FoSSaCS*, pages 213–227, 2007.

25 M. Hague and C.-H. Luke Ong. Winning regions of pushdown parity games: A saturation method. In *CONCUR*, pages 384–398, 2009.

26 M. Heizmann, J. Hoenicke, and A. Podelski. Nested interpolants. In *POPL*, pages 471–482. ACM, 2010.

27 M. Hofmann and W. Chen. Abstract interpretation from Büchi automata. In *CSL-LICS*, pages 51:1–51:10, 2014.

28 M. Hofmann and J. Ledent. A cartesian closed category for higher-order model checking. In *LICS*, 2017. To appear.

29 L. Holík, R. Meyer, and S. Muskalla. Summaries for context-free games. In *FSTTCS*, volume 65 of *LIPIcs*, pages 41:1–41:16. Dagstuhl, 2016.

30 Lukás Holík and Roland Meyer. Antichains for the verification of recursive programs. In *NETYS*, volume 9466 of *LNCS*, pages 322–336. Springer, 2015.

31 T. Knapik, D. Niwinski, and P. Urzyczyn. Higher-order pushdown trees are easy. In *FoSSaCS*, pages 205–222, 2002.

32 T. Knapik, D. Niwiński, P. Urzyczyn, and I. Walukiewicz. Unsafe grammars and panic automata. In *ICALP*, volume 3580 of *LNCS*, pages 1450–1461. Springer, 2005.

33 N. Kobayashi. HorSat2: A model checker for HORS based on SATuration. A tool available at `http://www-kb.is.s.u-tokyo.ac.jp/~koba/horsat2/`.

34 N. Kobayashi. Types and higher-order recursion schemes for verification of higher-order programs. In *POPL*, pages 416–428. ACM, 2009.

35 N. Kobayashi and C.-H. L. Ong. A type system equivalent to the modal mu-calculus model checking of higher-order recursion schemes. In *LICS*, pages 179–188. IEEE, 2009.

36 Z. Long, G. Calin, R. Majumdar, and R. Meyer. Language-theoretic abstraction refinement. In *FASE*, volume 7212 of *LNCS*, pages 362–376. Springer, 2012.

37 D. A. Martin. Borel determinacy. *Annals of Mathematics*, 102(2):363–371, 1975.

38 R. Meyer, S. Muskalla, and E. Neumann. Liveness verification and synthesis: New algorithms for recursive programs. `https://arxiv.org/abs/1701.02947`.

39 C.-H. L. Ong. On model-checking trees generated by higher-order recursion schemes. In *LICS*, pages 81–90. IEEE, 2006.

40 S. J. Ramsay. *Intersection-Types and Higher-Order Model Checking.* PhD thesis, Oxford University, 2013.

41 S. J. Ramsay. Exact intersection type abstractions for safety checking of recursion schemes. In *PPDP*, pages 175–186, 2014.

42 S. Salvati and I. Walukiewicz. A model for behavioural properties of higher-order programs. In *CSL*, pages 229–243, 2015.

43 S. Salvati and I. Walukiewicz. Typing weak MSOL properties. In *FoSSaCS*, pages 343–357, 2015.

44 S. Salvati and I. Walukiewicz. Using models to model-check recursive schemes. *Logical Methods in Computer Science*, 11(2), 2015.

45 I. Walukiewicz. Pushdown processes: Games and model-checking. *Information and Computation*, 164(2):234 – 263, 2001.

46 M. Wulf, L. Doyen, T. A. Henzinger, and J.-F. Raskin. Antichains: A new algorithm for checking universality of finite automata. In *CAV*, volume 4144 of *LNCS*. Springer, 2006.

A Relation to Higher-Order Model Checing

We elaborate on the relation of our work to the influential line of research on intersection types as pioneered by [35]. With intersection types, it is usually proven that *there is* a word or tree derivable by a HORS that is accepted by an automaton, i.e. a well-typed type environment can be certificate for the non-emptiness of the intersection $\mathcal{L}(scheme) \cap \mathcal{L}(Automaton) \neq \emptyset$. If the HORS is deterministic, $\mathcal{L}(scheme)$ consists of a single tree, so this is also decides the inclusion $\mathcal{L}(scheme) \subseteq \mathcal{L}(Automaton)$. If we naively extend intersection types to non-deterministic schemes, this is not true anymore. To prove the inclusion in this case, we will need to complement the automaton and prove the emptiness of the intersection, i.e. $\mathcal{L}(scheme) \cap \mathcal{L}(\overline{Automaton}) = \emptyset$. Note that a well-typing (a well-typed type environment) cannot prove the emptiness by itself: If the type for the initial symbol does not contain a transition from a final to an initial state, that can either stem from the non-existence of a such a transition sequence, or from the typing not being strong enough. For example, the empty typing that does not assign any type to any symbol (or the empty intersection, if you want), is a well-typing and does not prove anything. Therefore, an algorithm that decides the non-emptiness of the intersection by using intersection-types has to guarantee that it constructs a well-typing strong enough to prove the existence of an accepting transition sequence if such a sequence exists. Note that algorithms that compute intersection types usually allow alternating automata as the specification. It is conceptually easier to complement an alternating automaton than it is to complement a non-deterministic automaton: The transition for each origin and label is given as a Boolean formula, and we can get the complement automaton by considering the dual formula (i.e. the formula in which conjunctions and disjunctions are swapped). Note that usually, the transition formulas are normalized to disjunctive normal form (DNF), so computing the dual formula (which will then be in CNF) and re-normalizing it to DNF can lead to an exponential blowup.

B Proofs for Section 3

B.1 Proof of Proposition 2

Proof. Let $(\nu_i)_{i\in\mathbb{N}}$ be a descending chain of evaluations, i.e. $\nu_i \geq \nu_{i+1}$ for all $i \in \mathbb{N}$. It is to show that for all t, $\mathcal{M}[\![t]\!]$ is $\sqcap$-continuous (in the argument ν) over the respective lattice, i.e.

$$\mathcal{M}[\![t]\!]\left(\bigsqcap_{i\in\mathbb{N}} \nu_i\right) = \bigsqcap_{i\in\mathbb{N}} (\mathcal{M}[\![F]\!]\ \nu_i)\ .$$

We proceed by induction over t.

1. **Case $t = F$ or $t = x$.**
   Both of these cases are identical, hence we only show the former. We have

   $$\mathcal{M}[\![F]\!]\ (\bigsqcap_{i\in\mathbb{N}} \nu_i) = (\bigsqcap_{i\in\mathbb{N}} \nu_i)(F) = \bigsqcap_{i\in\mathbb{N}}(\nu_i(F)) = \bigsqcap_{i\in\mathbb{N}}(\mathcal{M}[\![F]\!]\ \nu_i)$$

   where the first and final equalities are by definition of the concrete semantics, and the second is by definition of $\sqcap$ over valuations ν_i.
2. **Case $t = s$ for some terminal s.**
   Similar to the previous case, we have

   $$\mathcal{M}[\![s]\!]\ (\bigsqcap_{i\in\mathbb{N}} \nu_i) = \mathcal{I}(s) = \bigsqcap_{i\in\mathbb{N}}(\mathcal{M}[\![s]\!]\ \nu_i)$$

by definition.

3. **Case** $t = t_1\ t_2$.
We have

$$\begin{aligned}
&\mathcal{M}[\![t_1\ t_2]\!]\ (\bigsqcap_{i\in\mathbb{N}} \nu_i)\\
\text{(Definition of semantics)} &= (\mathcal{M}[\![t_1]\!]\ (\bigsqcap_{i\in\mathbb{N}} \nu_i))\ (\mathcal{M}[\![t_2]\!]\ (\bigsqcap_{i\in\mathbb{N}} \nu_i))\\
\text{(Induction hypothesis)} &= (\bigsqcap_{i\in\mathbb{N}}(\mathcal{M}[\![t_1]\!]\ \nu_i))\ (\bigsqcap_{i\in\mathbb{N}}(\mathcal{M}[\![t_2]\!]\ \nu_i))\\
\text{(Definition of } \sqcap \text{ for functions)} &= \bigsqcap_{i\in\mathbb{N}}((\mathcal{M}[\![t_1]\!]\ \nu_i)\ (\bigsqcap_{i\in\mathbb{N}}(\mathcal{M}[\![t_2]\!]\ \nu_i)))\\
\text{(Continuity of } \mathcal{M}[\![t_1]\!]\ \nu_i \in \mathcal{D}) &= \bigsqcap_{i\in\mathbb{N}}\bigsqcap_{j\in\mathbb{N}}((\mathcal{M}[\![t_1]\!]\ \nu_i)\ (\mathcal{M}[\![t_2]\!]\ \nu_j)))\\
\text{(Argued below)} &= \bigsqcap_{i\in\mathbb{N}}((\mathcal{M}[\![t_1]\!]\ \nu_i)\ (\mathcal{M}[\![t_2]\!]\ \nu_i)))\\
\text{(Definition of semantics)} &= \bigsqcap_{i\in\mathbb{N}}(\mathcal{M}[\![t_1\ t_2]\!]\ \nu_i)\ .
\end{aligned}$$

We have to argue the step indicated above. That is,

$$\bigsqcap_{i\in\mathbb{N}}\bigsqcap_{j\in\mathbb{N}}((\mathcal{M}[\![t_1]\!]\ \nu_i)\ (\mathcal{M}[\![t_2]\!]\ \nu_j))) = \bigsqcap_{i\in\mathbb{N}}((\mathcal{M}[\![t_1]\!]\ \nu_i)\ (\mathcal{M}[\![t_2]\!]\ \nu_i)))\ .$$

The right-hand side is greater than the left-hand side, because terms of the form $((\mathcal{M}[\![t_1]\!]\ \nu_i)\ (\mathcal{M}[\![t_2]\!]\ \nu_j))$ where $\nu_i \neq \nu_j$ are missing in the RHS. To see that it is in fact equal, note that for two indices $i, j \in \mathbb{N}$, we have either $\nu_i \leq \nu_j$ or $\nu_j \leq \nu_i$, since the valuations form a descending chain. Let $m = \min\{i, j\}$. We now use that $\sqcap$-continuity implies monotonicity, and thus we have

$$((\mathcal{M}[\![t_1]\!]\ \nu_m)\ (\mathcal{M}[\![t_2]\!]\ \nu_m)) \leq ((\mathcal{M}[\![t_1]\!]\ \nu_i)\ (\mathcal{M}[\![t_2]\!]\ \nu_j))\ .$$

Hence, for any expression $((\mathcal{M}[\![t_1]\!]\ \nu_i)\ (\mathcal{M}[\![t_2]\!]\ \nu_j))$ that is missing in the meet in the RHS, the meet in the RHS contains an expression that is smaller, hence, they are equal.

4. **Case** $t = \lambda x.t'$.
We have

$$\begin{aligned}
&\mathcal{M}[\![\lambda x.t']\!]\ (\bigsqcap_{i\in\mathbb{N}} \nu_i)\\
\text{(Definition of semantics)} &= v \mapsto (\mathcal{M}[\![t']\!]\ (\bigsqcap_{i\in\mathbb{N}} \nu_i)[x \mapsto v])\\
\text{(Induction hypothesis)} &= v \mapsto (\bigsqcap_{i\in\mathbb{N}}(\mathcal{M}[\![t']\!]\ \nu_i[x \mapsto v]))\\
\text{(Definition of } \sqcap \text{ for functions)} &= \bigsqcap_{i\in\mathbb{N}}((v \mapsto \mathcal{M}[\![t_1]\!]\ \nu_i[x \mapsto v]))\\
\text{(Definition of semantics)} &= \bigsqcap_{i\in\mathbb{N}}(\mathcal{M}[\![\lambda x.t']\!]\ \nu_i)\ .
\end{aligned}$$

◀

B.2 Proof of Lemma 3

Proof. We show that for all $\nu : N \cup V \nrightarrow \mathcal{D}$ and all suitable terms t, t', we have

$$\mathcal{M}[\![(\lambda x.t)\ t']\!]\ \nu = \mathcal{M}[\![t[x \mapsto t']]\!]\ \nu\ .$$

We have by definition

$$\mathcal{M}[\![(\lambda x.t)\ t']\!]\ \nu = (\mathcal{M}[\![(\lambda x.t)]\!]\ \nu)\ (\mathcal{M}[\![t']\!]\ \nu) = \mathcal{M}[\![t]\!]\ (\nu[x \mapsto \mathcal{M}[\![t']\!]\ \nu])$$

and show by induction over t that

$$\mathcal{M}[\![t]\!]\ \nu[x \mapsto \mathcal{M}[\![t']\!]\ \nu] = \mathcal{M}[\![t[x \mapsto t']]\!]\ \nu\ .$$

In the base cases we have

1. $\mathcal{M}[\![F]\!]\ \nu[x \mapsto \mathcal{M}[\![t']\!]\ \nu] = (\nu[x \mapsto \mathcal{M}[\![t']\!]\ \nu])(F) = \nu(F) = \mathcal{M}[\![F]\!]\ \nu = \mathcal{M}[\![F[x \mapsto t']]\!]\ \nu$,
2. $\mathcal{M}[\![s]\!]\ \nu[x \mapsto \mathcal{M}[\![t']\!]\ \nu] = \mathcal{I}(s) = \mathcal{M}[\![s]\!]\ \nu = \mathcal{M}[\![s[x \mapsto t']]\!]\ \nu$,
3. $\mathcal{M}[\![x]\!]\ \nu[x \mapsto \mathcal{M}[\![t']\!]\ \nu] = \mathcal{M}[\![t']\!]\ \nu = \mathcal{M}[\![x[x \mapsto t']]\!]\ \nu$, and
4. $\mathcal{M}[\![y]\!]\ \nu[x \mapsto \mathcal{M}[\![t']\!]\ \nu] = (\nu[x \mapsto \mathcal{M}[\![t']\!]\ \nu])(y) = \nu(y) = \mathcal{M}[\![y]\!]\ \nu = \mathcal{M}[\![y[x \mapsto t']]\!]\ \nu$, for variable $y \neq x$.

Then, for the induction step, we first consider application. That is

$$\mathcal{M}[\![t_1\ t_2]\!]\ \nu[x \mapsto \mathcal{M}[\![t']\!]\ \nu] = (\mathcal{M}[\![t_1]\!]\ \nu[x \mapsto \mathcal{M}[\![t']\!]\ \nu])\ (\mathcal{M}[\![t_2]\!]\ \nu[x \mapsto \mathcal{M}[\![t']\!]\ \nu])$$

which is equal to, by induction,

$$(\mathcal{M}[\![t_1[x \mapsto t']]\!]\ \nu)\ (\mathcal{M}[\![t_2[x \mapsto t']]\!]\ \nu) = \mathcal{M}[\![t_1[x \mapsto t']\ t_2[x \mapsto t']]\!]\ \nu = \mathcal{M}[\![(t_1\ t_2)[x \mapsto t']]\!]\ \nu\ .$$

Finally, for abstraction, we can assume by α-conversion that $y \neq x$, and we have

$$\mathcal{M}[\![\lambda y.t_1]\!]\ \nu[x \mapsto \mathcal{M}[\![t']\!]\ \nu] = v \mapsto \mathcal{M}[\![t_1]\!]\ \nu[x \mapsto \mathcal{M}[\![t']\!]\ \nu, y \mapsto v]$$

which is by induction equal to the function

$$v \mapsto \mathcal{M}[\![t_1[x \mapsto t']]\!]\ \nu[y \mapsto v] = \mathcal{M}[\![\lambda y.t_1[x \mapsto t']]\!]\ \nu = \mathcal{M}[\![(\lambda y.t_1)[x \mapsto t']]\!]\ \nu\ .$$

Thus, by induction, we have the lemma as required. ◀

C Proofs for Section 4

C.1 Proof of Lemma 5

Proof. We show that for all non-ground terminals s, $\mathcal{I}^C(s)$ is $\sqcap$-continuous. We need to treat the terminals $a : o \to o$ of the original scheme and the terminals op_F that were introduced for the determinisation separately. In each case, assume a descending chain of arguments $(x_i)_{i \in \mathbb{N}}$.

1. **Case** $s = a$**.**
   Since $\mathcal{I}^C(a) = \mathsf{prepend}_a$ and we have

   $$\mathsf{prepend}_a(\bigsqcap_{i \in \mathbb{N}} x_i) = \bigsqcap_{i \in \mathbb{N}}(\mathsf{prepend}_a\ x_i)$$

   by definition of $\mathsf{prepend}_a$, we have the property as required.

2. **Case** $s = op_F$.
   We show the property when op_F is owned by $\Diamond$, and thus interpreted as ℓ-fold disjunction, conjunction is similar. We proceed by induction on the arity ℓ. In the base case $\ell = 1$, $\mathcal{I}^C(op_F)$ is the identity function that is $\sqcap$-continuous
   Now assume op_F has arity $\ell + 1$, and $\mathcal{I}^C(op_F) = \bigvee_{\ell+1}$ is an $\ell + 1$-fold disjunction. We have

$$\begin{aligned}
& \textstyle\bigvee_{\ell+1}(\bigsqcap_{i\in\mathbb{N}} x_i) \\
\text{(Definition of } \textstyle\bigvee_{\ell+1}) &= y_1,\dots,y_\ell \;\mapsto\; (\textstyle\bigsqcap_{i\in\mathbb{N}} x_i) \vee \textstyle\bigvee_\ell\, y_1 \;\cdots\; y_\ell \\
\text{(Distributivity (see below))} &= y_1,\dots,y_\ell \;\mapsto\; \textstyle\bigsqcap_{i\in\mathbb{N}}(x_i \vee \textstyle\bigvee_\ell\, y_1 \;\cdots\; y_\ell) \\
\text{(Definition of } \sqcap \text{ for functions)} &= \textstyle\bigsqcap_{i\in\mathbb{N}}(y_1,\dots,y_\ell \;\mapsto\; x_i \vee \textstyle\bigvee_\ell\, y_1 \;\cdots\; y_\ell) \\
\text{(Definition of } \textstyle\bigvee_{\ell+1}) &= \textstyle\bigsqcap_{i\in\mathbb{N}} \textstyle\bigvee_{\ell+1}\, x_i
\end{aligned}$$

   In the above we required $\vee$ to distribute over $\sqcap$, which can be seen by induction over types. In the base case, that $\vee$ distributes over $\sqcap = \wedge$ is standard. For the step case, we have for all f_i, g, and v

$$\begin{aligned}
& \big((\textstyle\bigsqcap_{i\in\mathbb{N}} f_i) \vee g\big)\; v \\
\text{(Definition of } \vee \text{ and } \sqcap) &= \big(\textstyle\bigsqcap_{i\in\mathbb{N}}(f_i\; v)\big) \vee (g\; v) \\
\text{(Induction)} &= \textstyle\bigsqcap_{i\in\mathbb{N}} \big((f_i\; v) \vee (g\; v)\big) \\
\text{(Definition of } \vee \text{ and } \sqcap) &= \big(\textstyle\bigsqcap_{i\in\mathbb{N}}(f_i \vee g)\big)\; v\ .
\end{aligned}$$

◀

C.2 Proof of Theorem 6

We are required to show $\sigma_{\mathcal{M}^C}(S)$ is satisfied by $\mathcal{L}(A)$ iff there is a winning strategy foryer Player $\Diamond$. The theorem is shown in the following two lemmas.

▶ **Lemma 25** (Player $\Diamond$). *If $\sigma_{\mathcal{M}^C}(S)$ is satisfied by $\mathcal{L}(A)$ there is a winning strategy for $\Diamond$.*

Proof. In what follows, whenever we refer to a term t, we mean a term built over $N \uplus T$, but not over T^{det}. The terminals op_F are excluded because they do not occur in the game, they are only introduced in the determinized scheme.

We will demonstrate a strategy for $\Diamond$ that maintains the invariant that the current (variable-free) term t reached is such that $\mathcal{M}^C[\![t]\!]\ \sigma_{\mathcal{M}^C}$ is satisfied by $\mathcal{L}(A)$. All plays are infinite or generate a word w. Since we maintain $\mathcal{M}^C[\![t]\!]\ \sigma_{\mathcal{M}^C}$ is satisfied by $\mathcal{L}(A)$, if t represents a word w, we know w is accepted by A and Player $\Diamond$ wins the game.

Initially, we have $\mathcal{M}^C[\![S]\!]\ \sigma_{\mathcal{M}^C} = \sigma_{\mathcal{M}^C}(S)$ which is satisfied by $\mathcal{L}(A)$ by assumption. Thus, suppose play reaches a term t such that $\mathcal{M}^C[\![t]\!]\ \sigma_{\mathcal{M}^C}$ is satisfied by $\mathcal{L}(A)$. There are two cases.

In the first case $t = a_1\ (\cdots\ (a_n\ \$))$ and let $w = a_1 \dots a_n$. Since

$$\mathcal{M}^C[\![a_1(\cdots(a_n(\$)))]\!]\ \sigma_{\mathcal{M}^C} = \mathsf{prepend}_{a_1 \dots a_n}(\varepsilon) = w$$

and w is satisfied by $\mathcal{L}(A)$, we know $w \in \mathcal{L}(A)$ and Player $\Diamond$ has won the game.

In the second case, we have $t = a_1(\cdots(a_n(F\ t_1 \cdots t_m)))$. By assumption, we know

$$\begin{aligned}&\mathcal{M}^C[\![a_1(\cdots(a_n(F\ t_1 \cdots t_m)))]\!]\ \sigma_{\mathcal{M}^C} = \\ &\qquad \mathsf{prepend}_{a_1 \dots a_n}((\mathcal{M}^C[\![F]\!]\ \sigma_{\mathcal{M}^C})\ (\mathcal{M}^C[\![t_1]\!]\ \sigma_{\mathcal{M}^C})\ \cdots\ (\mathcal{M}^C[\![t_m]\!]\ \sigma_{\mathcal{M}^C}))\end{aligned}$$

is satisfied by $\mathcal{L}(A)$. Let $F = e_1, \dots, F = e_\ell$ be the rewrite rules for F. There are two subcases.

1. If F is owned by $\Diamond$, then since $\mathcal{M}^C[\![F]\!] = \mathcal{M}^C[\![e_1]\!] \vee \cdots \vee \mathcal{M}^C[\![e_\ell]\!]$ there must exist some i such that

   $$\mathsf{prepend}_{a_1 \dots a_n}((\mathcal{M}^C[\![e_i]\!]\ \sigma_{\mathcal{M}^C})\ (\mathcal{M}^C[\![t_1]\!]\ \sigma_{\mathcal{M}^C})\ \cdots\ (\mathcal{M}^C[\![t_m]\!]\ \sigma_{\mathcal{M}^C}))$$

   is satisfied by $\mathcal{L}(A)$. The strategy of Player $\Diamond$ is to choose the i^{th} rewrite rule.
   We need to show the invariant is maintained. Let $e_i = \lambda x_1, \dots, x_m.e$. We have (using the substitution lemma, Lemma 3),

   $$\begin{aligned}&\mathsf{prepend}_{a_1 \dots a_n}((\mathcal{M}^C[\![\lambda x_1, \dots, x_m.e]\!]\ \sigma_{\mathcal{M}^C})\ (\mathcal{M}^C[\![t_1]\!]\ \sigma_{\mathcal{M}^C})\ \cdots\ (\mathcal{M}^C[\![t_m]\!]\ \sigma_{\mathcal{M}^C})) \\ =\ &\mathsf{prepend}_{a_1 \dots a_n}(\mathcal{M}^C[\![(\lambda x_1, \dots, x_m.e)\ t_1\ \dots\ t_m]\!]\ \sigma_{\mathcal{M}^C}) \\ =\ &\mathsf{prepend}_{a_1 \dots a_n}(\mathcal{M}^C[\![e[x_1 \mapsto t_1, \dots, x_m \mapsto t_m]]\!]\ \sigma_{\mathcal{M}^C}) \\ =\ &\mathcal{M}^C[\![a_1(\cdots(a_n(e[x_1 \mapsto t_1, \dots, x_m \mapsto t_m])))]\!]\ \sigma_{\mathcal{M}^C}\ .\end{aligned}$$

   Note that the term $a_1(\cdots(a_n(e[x_1 \mapsto t_1, \dots, x_m \mapsto t_m])))$ is the result of Player $\Diamond$ rewriting F via $F = e_i$. Since the satisfaction by $\mathcal{L}(A)$ passes through the equalities, Player $\Diamond$'s move maintains the invariant as required.
2. If F is owned by $\Box$ the argument proceeds as in the previous case. The key difference is that we have to show satisfaction is maintained no matter which move $\Box$ chooses. However, since in this case $\mathcal{M}^C[\![F]\!] = \mathcal{M}^C[\![e_1]\!] \wedge \cdots \wedge \mathcal{M}^C[\![e_\ell]\!]$ then for all i we have

   $$\mathsf{prepend}_{a_1 \dots a_n}((\mathcal{M}^C[\![e_i]\!]\ \sigma_{\mathcal{M}^C})\ (\mathcal{M}^C[\![t_1]\!]\ \sigma_{\mathcal{M}^C})\ \cdots\ (\mathcal{M}^C[\![t_m]\!]\ \sigma_{\mathcal{M}^C}))$$

   is satisfied by $\mathcal{L}(A)$. The remainder of the argument is identical.

◀

▶ **Lemma 26** (Player $\Box$). *If $\sigma_{\mathcal{M}^C}(S)$ is not satisfied by $\mathcal{L}(A)$ there is a winning strategy for $\Box$.*

Proof. In what follows, whenever we refer to a term t, we mean a term built over $N \uplus T$, but not over T^{det}. The terminals op_F are excluded because they do not occur in the game, they are only introduced in the determinized scheme.

For $\phi \in \mathcal{D}^C(o)$ and a variable-closed term t of kind o, we define *ϕ to be sound for t*, denoted $\phi \vdash t$, if for all $w \in T^*$ such that $\mathsf{prepend}_w(\phi)$ is not satisfied by $\mathcal{L}(A)$, Player $\Box$ has a winning strategy from term $w(t)$. For $w = \varepsilon$, we set $\mathsf{prepend}_\varepsilon(\phi) = \phi$ and let $\varepsilon(t) = t$. We can now restate the lemma as

$$\sigma_{\mathcal{M}^C}(S) \vdash S\ . \tag{1}$$

In particular, since $\sigma_{\mathcal{M}^C}(S)$ is not satisfied by $\mathcal{L}(A)$ it is the case that $\mathsf{prepend}_\varepsilon(\sigma_{\mathcal{M}^C}(S))$ is not satisfied. This means Player □ has a winning strategy from $\varepsilon(S) = S$.

In general, for $\Xi \in \mathcal{D}^C(\kappa_1 \to \kappa_2)$, we will also define $\Xi \vdash t$ for terms of kind $\kappa_1 \to \kappa_2$. That is, for a variable-closed term t of kind $\kappa_1 \to \kappa_2$ and a function $\Xi \in \mathcal{D}^C(\kappa_1 \to \kappa_2)$, we define $\Xi \vdash t$ to hold whenever for all variable-closed terms t' of kind κ_1 and $\Xi' \in \mathcal{D}^C(\kappa_1)$ such that $\Xi' \vdash t'$ we have $\Xi\ \Xi' \vdash t\ t'$:

$$\Xi \vdash t, \text{ if } \forall\ \Xi', t' \text{ such that } \Xi' \vdash t', \text{ we have } \Xi\ \Xi' \vdash t\ t'\ .$$

Similarly, we need to extend $\vdash$ to terms t with free variables $\vec{x} = x_1 \dots x_m$. Here, we make the free variables explicit and write $t(\vec{x})$. We define for $\Xi : (V \twoheadrightarrow \mathcal{D}^C) \to \mathcal{D}^C$ that $\Xi \vdash t(\vec{x})$ by requiring that for any variable-closed terms $t_1, \dots, t_m$ and any $\Xi_1, \dots, \Xi_m \in \mathcal{D}^C$ with $\Xi_j \vdash t_j$ for all $1 \leq j \leq m$, we have $\Xi\ \nu \vdash t[\forall j\colon x_j \mapsto t_j]$, where ν maps x_j to Ξ_j.

We now show the following. For every number of iterations i in the fixed-point calculation, we have $\mathcal{M}^C[\![t]\!]\ \sigma^i_{\mathcal{M}^C} \vdash t$, for all terms t built over the terminals and non-terminals in the scheme of interest. After the induction, we will show that the result holds for the greatest fixed point. Note that we have a nested induction: the outer induction is along i, the inner is along the structure of terms.

Since we are inducting over non-closed terms, we will have to extend $\sigma^i_{\mathcal{M}^C}$ to assign valuations to free-variables. Thus we will write ν^i to denote a valuation such that $\nu^i(F) = \sigma^i_{\mathcal{M}^C}(F)$ for any non-terminal F.

Base case i.

In the base case, we have $i = 0$ and $\nu^i = \top$ for all non-terminals. We proceed by induction on the structure of terms. We will emphasize if an argumentation is independent of the iteration count. This is the case for all terms except non-terminals.

Base case t.

The base cases of the inner induction that are independent of the iteration count are the following.

1. **Case $t = \$$.**
   For all i, we have $\mathcal{M}^C[\![\$]\!]\ \nu^i = \varepsilon$. Take any word w such that $\mathsf{prepend}_w(\varepsilon)$ is not satisfied by $\mathcal{L}(A)$. No moves can be made from $w(\varepsilon)$ and Player □ has won the game.
2. **Case $t = a$.**
   We again reason over all i and show that

   $$\mathcal{M}^C[\![a]\!]\ \nu^i\ \Xi = \mathsf{prepend}_a(\Xi) \vdash a(t)$$

   for any variable-closed term $t : o$ and any Ξ so that $\Xi \vdash t$. Take any word w such that $\mathsf{prepend}_w(\mathsf{prepend}_a(\Xi))$ is not satisfied by $\mathcal{L}(A)$. It follows that $\mathsf{prepend}_{wa}(\Xi)$ is also not satisfied by $\mathcal{L}(A)$. From $\Xi \vdash t$, Player □ has a winning strategy from $wa(t)$. Since $wa(t) = w(a(t))$ we are done.
3. **Case $t = x$.**
   For all i and all extensions ν^i of $\sigma^i_{\mathcal{M}^C}$, we have

   $$\mathcal{M}^C[\![x]\!]\ \nu^i\ = \nu^i(x).$$

   Take any $\nu^i(x) = \Xi$ and any variable-closed term t' with $\Xi \vdash t'$. Then $\nu^i(x) \vdash x[x \mapsto t']$ is immediate.

The only base case of the inner induction that depends on the iteration count is $t = F$. Let F take m arguments and consider variable-closed terms $t_1, \ldots, t_m$ with corresponding Ξ_j such that $\Xi_j \vdash t_j$. We have

$$\mathcal{M}^C[\![F]\!]\ \nu^0\ \Xi_1\ \ldots\ \Xi_m = \sigma_{\mathcal{M}^C}(F)^0\ \Xi_1\ \ldots\ \Xi_m = \mathsf{true}.$$

Thus, trivially $\mathcal{M}^C[\![F]\!]\ \nu^0 \vdash F$ since true is never unsatisfied.

Step case t.

In both cases, the argumentation is independent of the actual iteration count. Therefore, we give it for a general i rather than for 0.

1. **Case $t = t'\ t''$.**

   Assume we already know that

   $$\mathcal{M}^C[\![t']\!]\ \nu^i \vdash t' \quad \text{and} \quad \mathcal{M}^C[\![t'']\!]\ \nu^i \vdash t''\ .$$

   Our task is to show that

   $$\mathcal{M}^C[\![t'\ t'']\!]\ \nu^i = (\mathcal{M}^C[\![t']\!]\ \nu^i)\ (\mathcal{M}^C[\![t'']\!]\ \nu^i) \vdash t'\ t''\ .$$

   Let the free variables be $x_1, \ldots, x_n$ and consider $\Xi_1 \vdash t_1$ to $\Xi_n \vdash t_n$. Let ν^i map x_j to Ξ_j for all $1 \leq j \leq n$. By the definition of $\vdash$ for terms with free variables, we have $\mathcal{M}^C[\![t']\!]\ \nu^i \vdash t'[\forall j : x_j \mapsto t_j]$ and $\mathcal{M}^C[\![t'']\!]\ \nu^i \vdash t''[\forall j : x_j \mapsto t_j]$. Then, by the definition of $\vdash$ for functions, we obtain

   $$\begin{aligned}\mathcal{M}^C[\![t'\ t'']\!]\ \nu^i &= (\mathcal{M}^C[\![t']\!]\ \nu^i)\ (\mathcal{M}^C[\![t'']\!]\ \nu^i)\\ &\vdash (t'[\forall j : x_j \mapsto t_j])\ (t''[\forall j : x_j \mapsto t_j]) = (t'\ t'')[\forall j : x_j \mapsto t_j]\ .\end{aligned}$$

   This means $\mathcal{M}^C[\![t'\ t'']\!]\ \nu^i \vdash t'\ t''$ as required.

2. **Case $t = \lambda x.e$.**

   Let the free variables of e be $x, x_1, \ldots, x_n$. For $\mathcal{M}^C[\![\lambda x.e]\!]\ \nu^i \vdash \lambda x.e$, we have to argue that for any $\Xi_1 \vdash t_1$ to $\Xi_n \vdash t_n$ with ν^i mapping x_i to Ξ_i for all $1 \leq i \leq n$, we get

   $$\mathcal{M}^C[\![\lambda x.e]\!]\ \nu^i \vdash (\lambda x.e)[\forall j : x_j \mapsto t_j]\ .$$

   This in turn means that for any $\Xi \vdash t$, we have to show

   $$(\mathcal{M}^C[\![\lambda x.e]\!]\ \nu^i)\ \Xi \vdash ((\lambda x.e)[\forall j : x_j \mapsto t_j])\ t\ .$$

   By the definition of the semantics, we have

   $$(\mathcal{M}^C[\![\lambda x.e]\!]\ \nu^i)\ \Xi = \mathcal{M}^C[\![e]\!]\ \nu^i[x \mapsto \Xi]\ .$$

   Moreover, since the t_j are variable-closed, they in particular are not affected by replacing x and we get

   $$((\lambda x.e)[\forall j : x_j \mapsto t_j])\ t = (\lambda x.(e[\forall j : x_j \mapsto t_j]))\ t.$$

   In the game, λ-redexes of the form $(\lambda x.e)\ t$ do not occur at all: When a non-terminal F is rewritten to its right-hand side $\lambda x.e$, this yields $e[x \mapsto t]$ within a single step. This means the game equates $(\lambda x.(e[\forall j : x_j \mapsto t_j]))\ t$ with $e[x \mapsto t, \forall j : x_j \mapsto t_j]$. Hence, all that remains to be shown is

   $$\mathcal{M}^C[\![e]\!]\ \nu^i[x \mapsto \Xi] \vdash e[x \mapsto t, \forall j : x_j \mapsto t_j]\ .$$

   This holds by the hypothesis of the inner induction, showing $\mathcal{M}^C[\![e]\!]\ \nu^i \vdash e$.

Step case i.
We again do an induction along the structure of terms. The only case that has not been treated in full generality is F. We now show that $\mathcal{M}^C[\![F]\!]\ \nu^{i+1} \vdash F$. Let F take m arguments and consider $\Xi_1 \vdash t_1$ to $\Xi_m \vdash t_m$. The task is to prove $\mathcal{M}^C[\![F]\!]\ \nu^{i+1}\ \Xi_1\ \dots\ \Xi_m \vdash F\ t_1\ \dots\ t_m$. To ease the notation, assume there are two right hand sides e_1', e_2 for F, i.e. we have the rules $F = \lambda x_1 \dots \lambda x_m.e_1$ and $F = \lambda x_1 \dots \lambda x_m.e_2$. This means the right-hand side in the determinised scheme is $F = \lambda x_1 \dots \lambda x_m.(op_F\ e_1\ e_2)$. Then,

$$\begin{aligned}
\mathcal{M}^C[\![F]\!]\ \nu^{i+1} &= \nu^{i+1}(F)\\
&= \mathcal{M}^C[\![\lambda x_1 \dots \lambda x_m.(op_F\ e_1\ e_2)]\!]\ \nu^i\\
&= v_1, \dots, v_m \mapsto \mathcal{M}^C[\![(op_F\ e_1\ e_2)[x_1 \mapsto v_1, \dots, x_m \mapsto v_m]]\!]\ \nu^i\\
&= v_1, \dots, v_m \mapsto \mathcal{M}^C[\![(op_F\ e_1[\vec{x} \mapsto \vec{v}]\ e_2[\vec{x} \mapsto \vec{v}])]\!]\ \nu^i\\
&= v_1, \dots, v_m \mapsto \mathcal{I}^C(op_F)\ \big(\mathcal{M}^C[\![e_1[\vec{x} \mapsto \vec{v}]]\!]\ \nu^i\big)\ \big(\mathcal{M}^C[\![e_2[\vec{x} \mapsto \vec{v}])]\!]\ \nu^i\big)
\end{aligned}$$

Here, $\mathcal{I}^C(op_F)$ is a conjunction or disjunction, depending on the owner of F. Recall that the conjunction and disjunction of functions are defined by evaluating the argument functions separately and combining the results. This means

$$\begin{aligned}
\mathcal{M}^C[\![F]\!]\ \nu^{i+1} &= v_1, \dots, v_m \mapsto \mathcal{I}^C(op_F)\ \big(\mathcal{M}^C[\![e_1[\vec{x} \mapsto \vec{v}]]\!]\ \nu^i\big)\ \big(\mathcal{M}^C[\![e_2[\vec{x} \mapsto \vec{v}])]\!]\ \nu^i\big)\\
&= v_1, \dots, v_m \mapsto \big(\mathcal{M}^C[\![e_1[\vec{x} \mapsto \vec{v}]]\!]\ \nu^i\big)\ (\vee/\wedge)\ \big(\mathcal{M}^C[\![e_2[\vec{x} \mapsto \vec{v}])]\!]\ \nu^i\big)\\
&= \big(v_1, \dots, v_m \mapsto \mathcal{M}^C[\![e_1[\vec{x} \mapsto \vec{v}]]\!]\ \nu^i\big)\ (\vee/\wedge)\ \big(v_1, \dots, v_m \mapsto \mathcal{M}^C[\![e_2[\vec{x} \mapsto \vec{v}])]\!]\ \nu^i\big)\\
&= \big(\mathcal{M}^C[\![\lambda x_1 \dots \lambda x_m.e_1]\!]\ \nu^i\big)\ (\vee/\wedge)\ \big(\mathcal{M}^C[\![\lambda x_1 \dots \lambda x_m.e_2]\!]\ \nu^i\big)\\
&= \big(\mathcal{M}^C[\![e_1']\!]\ \nu^i\big)\ (\vee/\wedge)\ \big(\mathcal{M}^C[\![e_2']\!]\ \nu^i\big)\ .
\end{aligned}$$

With the same reasoning, we obtain

$$\begin{aligned}
&\mathcal{M}^C[\![F]\!]\ \nu^{i+1}\ \Xi_1\ \dots\ \Xi_m\\
&\quad = (\mathcal{M}^C[\![e_1']\!]\ \nu^i\ \Xi_1\ \dots\ \Xi_m)\ (\vee/\wedge)\ (\mathcal{M}^C[\![e_2']\!]\ \nu^i\ \Xi_1\ \dots\ \Xi_m).
\end{aligned}$$

We have to prove that for any $w \in T^*$, if $\mathcal{L}(A)$ does not satisfy the formula

$$\begin{aligned}
&\mathsf{prepend}_w((\mathcal{M}^C[\![e_1']\!]\ \nu^i\ \Xi_1\ \dots\ \Xi_m)\ (\vee/\wedge)\ (\mathcal{M}^C[\![e_2']\!]\ \nu^i\ \Xi_1\ \dots\ \Xi_m))\\
&= \mathsf{prepend}_w(\mathcal{M}^C[\![e_1']\!]\ \nu^i\ \Xi_1\ \dots\ \Xi_m)\ (\vee/\wedge)\ \mathsf{prepend}_w(\mathcal{M}^C[\![e_2']\!]\ \nu^i\ \Xi_1\ \dots\ \Xi_m),
\end{aligned}$$

then Player □ has a winning strategy from $w(F\ t_1 \dots\ t_m)$.

Assume Player ◇ owns F and the formula is not satisfied. If Player □ owns F, the reasoning is similar. Since we have a disjunction for Player ◇, $\mathsf{prepend}_w(\mathcal{M}^C[\![e_1']\!]\ \nu^i\ \Xi_1\ \dots\ \Xi_m)$ is not satisfied. By the hypothesis of the outer induction, we obtain $\mathcal{M}^C[\![e_1']\!]\ \nu^i\ \vdash e_1$ and thus $\mathcal{M}^C[\![e_1']\!]\ \nu^i\ \Xi_1\ \dots\ \Xi_m \vdash e_1'\ t_1\ \dots\ t_m$. As in the case of λ-abstraction above, we use that the game identifies $e_1'\ t_1\ \dots\ t_m$ and $e_1[x_1 \mapsto t_1, \dots e_m \mapsto t_m]$. Hence, Player □ has a winning strategy from $w(e_1[x_1 \mapsto t_1, \dots e_m \mapsto t_m])$. The same argumentation applies to $\mathsf{prepend}_w(\mathcal{M}^C[\![e_2]\!]\ \nu^i\ \Xi_1\ \dots\ \Xi_m)$. Consequently, whichever move Player ◇ makes at $w(F\ t_1\ \dots\ t_m)$, Player □ has a winning strategy.

This finishes the outer induction, proving that $\mathcal{M}^C[\![t]\!]\ \sigma^i_{\mathcal{M}^C} \vdash t$ for all terms t and all $i \in \mathbb{N}$. We would like to conclude $\mathcal{M}^C[\![t]\!]\ \sigma_{\mathcal{M}^C} \vdash t$. Since the cppo under consideration is not finite, this needs to be proven separately.

Limit case.
We have shown $\mathcal{M}^C[\![t]\!]\ \sigma^i_{\mathcal{M}^C} \vdash t$ for all $i \in \mathbb{N}$; we now show $\mathcal{M}^C[\![t]\!]\ \sigma_{\mathcal{M}^C} \vdash t$ noting by Kleene that $\sigma_{\mathcal{M}^C} = \bigsqcap_{i\in\mathbb{N}} \sigma^i_{\mathcal{M}^C}$. Once we have this we have $\sigma_{\mathcal{M}^C}(S) \vdash S$ which proves the lemma.

We formulate a slightly more general induction hypothesis for induction over kinds: Given a descending sequence of Ξ_i for all $i \in \mathbb{N}$ such that each $\Xi_i \vdash t$, we have $\bigsqcap_{i\in\mathbb{N}} \Xi_i \vdash t$. In the base case we have t is of kind o and we assume $\Xi_i \vdash t$. We now argue $\bigsqcap_{i\in\mathbb{N}} \Xi_i \vdash t$.

Take any w and suppose $\mathsf{prepend}_w(\bigsqcap_{i\in\mathbb{N}} \Xi_i)$ is not satisfied, then we need to show by the definition of $\vdash$ that Player $\Box$ has a winning strategy. Since $\sqcap$ is conjunction, if

$$\mathsf{prepend}_w(\bigsqcap_{i\in\mathbb{N}} \Xi_i) = \bigsqcap_{i\in\mathbb{N}} \mathsf{prepend}_w(\Xi_i)$$

is not satisfied, it must be the case that for some i we have $\mathsf{prepend}_w(\Xi_i)$ is not satisfied. In this case, we have $\Xi_i \vdash t$ by assumption and thus by the definition of $\vdash$ that Player $\Box$ has a winning strategy from $w(t)$. This proves $\bigsqcap_{i\in\mathbb{N}} \Xi_i \vdash t$.

If t is of kind $\kappa_1 \to \kappa_2$ we need to show for all $\Xi \vdash t'$ that $(\bigsqcap_{i\in\mathbb{N}} \Xi_i)\ \Xi \vdash t\ t'$. We have by the definition of $\sqcap$ over functions

$$(\bigsqcap_{i\in\mathbb{N}} \Xi_i)\ \Xi = \bigsqcap_{i\in\mathbb{N}} (\Xi_i\ \Xi)$$

Since by assumption on Ξ_i and definition of $\vdash$ for function kinds, we have $\Xi_i\ \Xi \vdash t\ t'$ for each i. By the induction on the kind, we obtain $\bigsqcap_{i\in\mathbb{N}}(\Xi_i\ \Xi) \vdash t\ t'$. Since $(\bigsqcap_{i\in\mathbb{N}} \Xi_i)\ \sigma = \bigsqcap_{i\in\mathbb{N}}(\Xi_i\ \sigma)$ we establish the desired statement that finishes the induction.

Finally, since $\mathcal{M}^C[\![t]\!]\ \sigma^i_{\mathcal{M}^C}$ satisfies the conditions of the above induction hypothesis and because we have already shown $\mathcal{M}^C[\![t]\!]\ \sigma^i_{\mathcal{M}^C} \vdash t$ for all t, we obtain

$$\bigsqcap_{i\in\mathbb{N}} (\mathcal{M}^C[\![t]\!]\ \sigma^i_{\mathcal{M}^C}) \vdash t\ .$$

Then, since using continuity of $\mathcal{M}^C[\![t]\!]$ we have

$$\mathcal{M}^C[\![t]\!]\ \sigma_{\mathcal{M}^C} = \mathcal{M}^C[\![t]\!]\ (\bigsqcap_{i\in\mathbb{N}} \sigma^i_{\mathcal{M}^C}) = \bigsqcap_{i\in\mathbb{N}} (\mathcal{M}^C[\![t]\!]\ \sigma^i_{\mathcal{M}^C})$$

we obtain the lemma as required. ◀

D Proofs for Section 5

D.1 Proof of Lemma 9

Proof. We show that, if (P1) holds, then for every $\kappa \in K$ and every $v_r \in \mathcal{D}_r(\kappa)$ there is a compatible $v_l \in \mathcal{D}_l(\kappa)$ with $\alpha(v_l) = v_r$. We proceed by induction on kinds. The base case is given by the assumption (P1) and the fact that every ground element is compatible. Assume we have the required surjectivity of α for κ_1 and κ_2 and consider $f_r \in \mathcal{D}_r(\kappa_1 \to \kappa_2)$. The task is to find a compatible function f_l so that $\alpha(f_l) = f_r$. Assume $f_r\ v_r = v'_r$. By surjectivity for κ_1, there are compatible elements in $\alpha^{-1}(v_r)$, and similar for v'_r. Let v'_l be a compatible element that is mapped to v'_r by α. We define $f_l\ v_l = v'_l$ for all compatible $v_l \in \alpha^{-1}(v_r)$. Since α is total on $\mathcal{D}(\kappa_1)$, this assigns a value to all compatible v_l. We do not impose any requirements on how to map elements that are not compatible.

We argue that f_l is compatible. To this end, consider compatible v^1_l and v^2_l with $\alpha(v^1_l) = \alpha(v^2_l)$. By definition, both are mapped identically by f_l, $f_l\ v^1_l = f_l\ v^2_l$. Hence, in particular the abstractions coincide. Moreover, given a compatible v_l, we defined $f_l\ v_l = v_l'$ to be a compatible element.

Concerning the equality of the functions, we have $\alpha(f_l)\ v_r = \alpha(f_l\ v_l) = \alpha(v_l') = v_r'$. The first equality is the definition of abstraction for functions and the fact that $\alpha^{-1}(v_r)$ contains compatible elements, one of them being v_l, the second is the fact that v_l is mapped to v_l', and the last is by $v_l' \in \alpha^{-1}(v_r')$. ◀

D.2 Proof of Lemma 10

Proof. We proceed by induction on kinds to show that, if (P1) and (P2) hold, then for all kinds $\kappa \in K$ and for all descending chains of compatible values $f_1, f_2, \ldots \in \mathcal{D}(\kappa)$, we have $\bigsqcap_{i\in\mathbb{N}} f_i$ again compatible and $\alpha(\bigsqcap_{i\in\mathbb{N}} f_i) = \bigsqcap_{i\in\mathbb{N}} \alpha(f_i)$. The base case is the assumption.

In the induction step, let $\kappa = \kappa_1 \to \kappa_2$ and $f_1, f_2, \ldots \in \mathcal{D}_l(\kappa)$ be a descending chain of compatible elements. Let $v_l \in \mathcal{D}_l(\kappa_1)$ be compatible. The following equalities will be helpful:

$$\alpha((\bigsqcap_{i\in\mathbb{N}} f_i)\ v_l) = \alpha(\bigsqcap_{i\in\mathbb{N}} (f_i\ v_l)) = \bigsqcap_{i\in\mathbb{N}} \alpha(f_i\ v_l) = \bigsqcap_{i\in\mathbb{N}} (\alpha(f_i)\ \alpha(v_l)) = (\bigsqcap_{i\in\mathbb{N}} \alpha(f_i))\ \alpha(v_l).$$

The first equality is the definition of $\sqcap$ on functions, the second is the induction hypothesis for κ_2, the third is compatibility of the f_i and v_l, the last is again $\sqcap$ on functions.

To show compatibility, note that the above implies $\alpha((\bigsqcap_{i\in\mathbb{N}} f_i)\ v_l) = \alpha((\bigsqcap_{i\in\mathbb{N}} f_i)\ v_l')$ as long as $\alpha(v_l) = \alpha(v_l')$, for all compatible $v_l, v_l' \in \mathcal{D}_l(\kappa_1)$. For compatibility of $(\bigsqcap_{i\in\mathbb{N}} f_i)\ v_l$ with $v_l \in \mathcal{D}_l(\kappa_1)$ compatible, note that $(\bigsqcap_{i\in\mathbb{N}} f_i)\ v_l = \bigsqcap_{i\in\mathbb{N}}(f_i\ v_l)$. The latter is the meet over a descending chain of compatible elements in κ_2. By the induction hypothesis on κ_2, it is again compatible.

For $\sqcap$-continuity, consider a value $v_r \in \mathcal{D}_r(\kappa_1)$. By Lemma 9, there is a compatible $v_l \in \mathcal{D}_l(\kappa_1)$ with $\alpha(v_l) = v_r$. We have

$$\alpha(\bigsqcap_{i\in\mathbb{N}} f_i)\ v_r = \alpha((\bigsqcap_{i\in\mathbb{N}} f_i)\ v_l) = (\bigsqcap_{i\in\mathbb{N}} \alpha(f_i))\ \alpha(v_l) = (\bigsqcap_{i\in\mathbb{N}} \alpha(f_i))\ v_r.$$

The first equality is the definition of abstraction on functions. Note that we need here the fact that $\bigsqcap_{i\in\mathbb{N}} f_i$ is compatible by the induction hypothesis. The second equality is the auxiliary one from above. The last equality is by $\alpha(v_l) = v_r$. ◀

D.3 Proof of Lemma 11

Proof. We show that, if (P3) holds, then $\alpha(\top^l_\kappa) = \top^r_\kappa$ for all $\kappa \in K$. We proceed by induction on kinds. The base case is given by the assumption (P3). Assume for κ_2, we have $\alpha(\top^l_{\kappa_2}) = \top^r_{\kappa_2}$. Consider function $\top^l_{\kappa_1\to\kappa_2} \in \mathcal{D}_l(\kappa_1 \to \kappa_2)$. We have to show $\alpha(\top^l_{\kappa_1\to\kappa_2}) = \top^r_{\kappa_1\to\kappa_2}$. If the given top element is not compatible, this holds. Assume it is. For $v_r \in \mathcal{D}_r(\kappa_1)$, there are two cases. If there is no compatible $v_l \in \mathcal{D}_l(\kappa_1)$ with $\alpha(v_l) = v_r$, we have

$$\alpha(\top^l_{\kappa_1\to\kappa_2})\ v_r = \top^r_{\kappa_2} = \top^r_{\kappa_1\to\kappa_2}\ v_r.$$

If there is such a v_l, we obtain

$$\alpha(\top^l_{\kappa_1\to\kappa_2})\ v_r = \alpha(\top^l_{\kappa_1\to\kappa_2}\ v_l) = \alpha(\top^l_{\kappa_2}) = \top^r_{\kappa_2} = \top^r_{\kappa_1\to\kappa_2}\ v_r.$$

The first equality is the definition of abstraction for functions, the next is the fact that $\top^l_{\kappa_1\to\kappa_2}$ maps every element $v_l \in \mathcal{D}_l(\kappa_1)$ to $\top^l_{\kappa_2}$. The image of $\top^l_{\kappa_2}$ is $\top^r_{\kappa_2}$ by the induction hypothesis. The last equality is the definition of $\top^r_{\kappa_1\to\kappa_2}$. ◀

D.4 Proof of Lemma 12

Proof. Assume (P1), (P4), and (P5) hold. We show, for all terms t and all compatible ν, $\mathcal{M}_l[\![t]\!]\ \nu$ is compatible and $\alpha(\mathcal{M}_l[\![t]\!]\ \nu) = \mathcal{M}_r[\![t]\!]\ \alpha(\nu)$. We proceed by structural induction on t.

1. **Case F, x.**
   By the assumption, $\mathcal{M}_l[\![F]\!]\ \nu = \nu(F)$ is compatible. Moreover,

   $$\alpha(\mathcal{M}_l[\![F]\!]\ \nu) = \alpha(\nu(F)) = \alpha(\nu)(F) = \mathcal{M}_r[\![F]\!]\ \alpha(\nu)$$

   holds. For $x \in V$, the reasoning is similar.

2. **Case terminal s.**
   Note that $\mathcal{M}_l[\![s]\!]\ \nu = \mathcal{I}_l(s)$. If s is ground, the claim holds by (P4). Let $s : \kappa_1 \to \kappa_2$. For compatibility, consider $v_l, v_l' \in \mathcal{D}(\kappa_1)$ compatible with $\alpha(v_l) = \alpha(v_l')$. Then

   $$\alpha(\mathcal{I}_l(s)\ v_l) = \mathcal{I}_r(s)\ \alpha(v_l) = \mathcal{I}_r(s)\ \alpha(v_l') = \alpha(\mathcal{I}_l(s)\ v_l').$$

   The first equality is (P4), the next is $\alpha(v_l) = \alpha(v_l')$, and the last is again (P4). The second requirement on compatibility is satisfied by (P5).

   To show $\alpha(\mathcal{M}_l[\![s]\!]\ \nu) = \mathcal{M}_r[\![s]\!]\ \alpha(\nu)$, consider a value $v_r \in \mathcal{D}_r(\kappa_1)$. By Lemma 9, there is some compatible $v_l \in \mathcal{D}_l(\kappa_1)$ with $\alpha(v_l) = v_r$. We have

   $$\alpha(\mathcal{I}_l(s))\ v_r = \alpha(\mathcal{I}_l(s)\ v_l) = \mathcal{I}_r(s)\ \alpha(v_l) = \mathcal{I}_r(s)\ v_r.$$

   The first equality is compatibility of $\mathcal{I}_l(s)$ and the definition of function abstraction. The next equality is (P4). The last is $\alpha(v_l) = v_r$.

For the induction step, assume the claim holds for t_1 and t_2.

1. **Case $t_1\ t_2$.**
   For compatibility, observe that $\mathcal{M}_l[\![t_1\ t_2]\!]\ \nu = (\mathcal{M}_l[\![t_1]\!]\ \nu)\ (\mathcal{M}_l[\![t_2]\!]\ \nu)$. Moreover, $\mathcal{M}_l[\![t_1]\!]\ \nu$ and $\mathcal{M}_l[\![t_2]\!]\ \nu$ are both compatible by the induction hypothesis. By definition of compatibility, applying a compatible function to a compatible argument yields a compatible value. Hence, $\mathcal{M}_l[\![t_1\ t_2]\!]\ \nu$ is compatible.

   For the equality, note that

   $$\mathcal{M}_r[\![t_1\ t_2]\!]\ \alpha(\nu) = (\mathcal{M}_r[\![t_1]\!]\ \alpha(\nu))\ (\mathcal{M}_r[\![t_2]\!]\ \alpha(\nu)) = \alpha(\mathcal{M}_l[\![t_1]\!]\ \nu)\ \alpha(\mathcal{M}_l[\![t_2]\!]\ \nu).$$

   The first equality is by the definition of the semantics, the second is the induction hypothesis. Compatibility justifies the first of the following equalities. The second is again the definition of the semantics:

   $$\alpha(\mathcal{M}_l[\![t_1]\!]\ \nu)\ \alpha(\mathcal{M}_l[\![t_2]\!]\ \nu) = \alpha((\mathcal{M}_l[\![t_1]\!]\ \nu)\ (\mathcal{M}_l[\![t_2]\!]\ \nu)) = \alpha(\mathcal{M}_l[\![t_1\ t_2]\!]\ \nu).$$

2. **Case $\lambda x : \kappa.t_1$.**
   We argue for compatibility. Consider compatible v_l and v_l' with $\alpha(v_l) = \alpha(v_l')$. By definition of the semantics and the induction hypothesis, we have

   $$\alpha((\mathcal{M}_l[\![\lambda x.t_1]\!]\ \nu)\ v_l) = \alpha(\mathcal{M}_l[\![t_1]\!]\ \nu[x \mapsto v_l]) = \mathcal{M}_r[\![t_1]\!]\ \alpha(\nu[x \mapsto v_l])\ .$$

For v_l', the reasoning is similar. Since $\alpha(v_l) = \alpha(v_l')$, we have $\alpha(\nu[x \mapsto v_l]) = \alpha(\nu[x \mapsto v_l'])$. Hence, $\mathcal{M}_r[\![t_1]\!]\ \alpha(\nu[x \mapsto v_l]) = \mathcal{M}_r[\![t_1]\!]\ \alpha(\nu[x \mapsto v_l'])$. We conclude the desired equality. For the second requirement in compatibility, let v_l be compatible. By definition of the semantics, $(\mathcal{M}_l[\![\lambda x.t_1]\!]\ \nu)\ v_l = \mathcal{M}_l[\![t_1]\!]\ \nu[x \mapsto v_l]$. Since ν and v_l are compatible, $\nu[x \mapsto v_l]$ is compatible. Hence, $\mathcal{M}_l[\![t_1]\!]\ \nu[x \mapsto v_l]$ is compatible by the induction hypothesis.

To prove $\mathcal{M}_r[\![\lambda x.t_1]\!]\ \alpha(\nu) = \alpha(\mathcal{M}_l[\![\lambda x.t_1]\!]\ \nu)$, consider an arbitrary value $v_r \in \mathcal{D}_r(\kappa)$. Let $v_l \in \mathcal{D}_l(\kappa_1)$ be compatible with $\alpha(v_l) = v_r$, which exists by Lemma 9. We have:

$$(\mathcal{M}_r[\![\lambda x.t_1]\!]\ \alpha(\nu))\ v_r = \mathcal{M}_r[\![t_1]\!]\ \alpha(\nu)[x \mapsto v_r] = \mathcal{M}_r[\![t_1]\!]\ \alpha(\nu[x \mapsto v_l])\ .$$

We showed above that $\mathcal{M}_l[\![\lambda x.t_1]\!]\ \nu$ is compatible. Using the definition of abstraction for functions and the definition of the semantics, the other function yields

$$\alpha(\mathcal{M}_l[\![\lambda x.t_1]\!]\ \nu)\ v_r = \alpha((\mathcal{M}_l[\![\lambda x.t_1]\!]\ \nu)\ v_l) = \alpha(\mathcal{M}_l[\![t_1]\!]\ \nu[x \mapsto v_l])\ .$$

With the induction hypothesis, $\alpha(\mathcal{M}_l[\![t_1]\!]\ \nu[x \mapsto v_l]) = \mathcal{M}_r[\![t_1]\!]\ \alpha(\nu[x \mapsto v_l])$. ◀

D.5 Proof of Theorem 13

Proof. Recall σ_l^0 and σ_r^0 are the greatest elements of the respective domains. We have

$$\alpha(\sigma_l) = \alpha(\bigsqcap_{i\in\mathbb{N}} rhs^i_{\mathcal{M}_l}(\sigma_l^0)) = \bigsqcap_{i\in\mathbb{N}} \alpha(rhs^i_{\mathcal{M}_l}(\sigma_l^0)) = \bigsqcap_{i\in\mathbb{N}} rhs^i_{\mathcal{M}_r}(\sigma_r^0) = \sigma_r.$$

The first equality is Kleene's theorem. The second equality uses the fact that each $rhs^i_{\mathcal{M}_l}(\sigma_l^0)$ is compatible and that they form a descending chain (both by induction on i), and then applies Lemma 10. The third equality also relies on compatibility of the $rhs^i_{\mathcal{M}_l}(\sigma_l^0)$ and invokes Lemma 12. Moreover, it needs $\alpha(\sigma_l^0) = \sigma_r^0$ by Lemma 11. The last equality is again Kleene's theorem. ◀

E Proofs for Section 6

E.1 Proof of Lemma 14

Proof. Observe $\mathcal{I}^A(\$) = Q_f = \mathsf{acc}(\varepsilon)$. Given a formula $\phi \in \mathsf{PBool}(\mathsf{acc}(T^*))$, we have to show that $\mathsf{pre}_a(\phi) \in \mathsf{PBool}(\mathsf{acc}(T^*))$. Since pre_a distributes over conjunction and disjunction, it is sufficient to show the requirement for atomic propositions. Consider $Q = \mathsf{acc}(w)$. We have $\mathcal{I}^A(a)\ \mathsf{acc}(w) = \mathsf{pre}_a(\mathsf{acc}(w)) = \mathsf{acc}(a.w)$. Finally, $\mathcal{I}^A(op_F)$ with $F \in N$ is conjunction or disjunction, and there is nothing to do as the formula structure is not modified. ◀

E.2 Proof of Lemma 15

Proof. We require, for all terminals s, $\mathcal{I}^A(s)$ is $\sqcap$-continuous over the respective lattices. We remark that the case $s = op_F$ is identical to Lemma 5. Hence, we show the case $s = a \in \Gamma$. Given a descending chain $(x_i)_{i\in\mathbb{N}}$, we have to show $\mathcal{I}(a)\ (\bigsqcap_{i\in\mathbb{N}} x_i) = \bigsqcap_{i\in\mathbb{N}}(\mathcal{I}(a)\ x_i)$. Recall that the meet of formulas is conjunction, and that we are in a finite domain. The latter means that the infinite conjunction is really the conjunction of finitely many formulas. Now pre_a is defined to distribute over finite conjunctions. We have

$$\mathcal{I}(a)\ (\bigsqcap_{i\in\mathbb{N}} x_i) = \mathsf{pre}_a\left(\bigwedge_{i \text{ finite}} x_i\right) = \bigwedge_{i \text{ finite}} \mathsf{pre}_a(x_i) = \bigsqcap_{i\in\mathbb{N}}(\mathcal{I}(a)\ x_i)$$

as required. ◀

E.3 Proof of Proposition 17

Proof. To show α is precise, we have to show (P1) to (P5). For (P1), it is sufficient to argue that for every set of states $Q \in \mathsf{acc}(T^*)$ there is a word that is mapped to it — which holds by definition. For formulas, note that $\alpha = \mathsf{acc}$ distributes over conjunction and disjunction, which means we can take the same connectives in the concrete as in the abstract and replace the leaves appropriately. Note that we only need a set consisting of one formula.

(P2) is satisfied by the concrete meet being the union of sets of formulas and α being defined by an element-wise application.

For (P3), note that the greatest elements are $\{\mathsf{true}\}$ for $\mathcal{D}^C(o)$ and true for $\mathcal{D}^A(o)$. By definition, $\alpha(\{\mathsf{true}\}) = \alpha(\mathsf{true}) = \mathsf{true}$.

For (P4), consider \$. We have $\alpha(\mathcal{I}^C(\$)) = \alpha(\{\varepsilon\}) = \mathsf{acc}(\varepsilon) = Q_f = \mathcal{I}^A(\$)$. The first equality is by definition of the concrete interpretation, the second is the definition of α, the third uses the fact that ε is accepted precisely from the final states, and the last equality is the interpretation of the \$ in the abstract domain.

For a letter a and a word $w \subseteq T^*$, we have

$$\alpha(\mathcal{I}^C(a)\ w) = \alpha(\mathsf{prepend}_a(w)) = \alpha(a.w) = \mathsf{acc}(a.w) = \mathsf{pre}_a(\mathsf{acc}(w)) = \mathcal{I}^A(a)\ \alpha(w).$$

The first equality is the interpretation of a in the concrete, the second is the definition of prepending a letter, the third is the definition of the abstraction, the next is how taking predecessors changes the set of states from which a word is accepted, and the last equality is the interpretation of a in the abstract domain and the definition of the abstraction function. The relation generalizes to formulas by noting that both the concrete interpretation and the abstract interpretation of a distribute over conjunction and disjunction. It also generalizes to sets of formulas by noting that $\mathsf{prepend}_a$ is applied to all elements in the set and, in the abstract domain, pre_a distributes over conjunction.

Let F be a non-terminal owned by $\Box$. To simplify the notation, let the associated operation be binary, $op_F : o \to o \to o$. Let $\Phi_1, \Phi_2 \in \mathcal{D}^C(o)$ be sets of formulas. We have

$$\begin{aligned}\alpha(\mathcal{I}^C(op_F)\ \Phi_1\ \Phi_2) = \alpha(\Phi_1 \cup \Phi_2) &= \bigwedge_{\phi \in \Phi_1 \cup \Phi_2} \alpha(\phi) \\ &= \bigwedge_{\phi \in \Phi_1} \alpha(\phi)\ \wedge \bigwedge_{\phi \in \Phi_2} \alpha(\phi) = \mathcal{I}^A(op_F)(\alpha(\Phi_1)\ \alpha(\Phi_2)).\end{aligned}$$

The first equality is the concrete interpretation of op_F. The second is the definition of the abstraction function. The third equality holds as we work up to logical equivalence. The last is the abstract interpretation of op_F and again the definition of the abstraction.

Assume F is owned by $\Diamond$ and op_F is again binary. Consider $\Phi_1, \Phi_2 \in \mathcal{D}^C(o)$. It will be convenient to denote $\{\phi_1 \vee \phi_2 \mid \phi_1 \in \Phi_1, \phi_2 \in \Phi_2\}$ by Φ. We have

$$\begin{aligned}\alpha(\mathcal{I}^C(op_F)\ \Phi_1\ \Phi_2) = \alpha(\Phi) &= \bigwedge_{\phi_1 \vee \phi_2 \in \Phi} \alpha(\phi_1 \vee \phi_2) \\ &= \bigwedge_{\phi_1 \in \Phi_1, \phi_2 \in \Phi_2} (\alpha(\phi_1) \vee \alpha(\phi_2)) \\ &= (\bigwedge_{\phi_1 \in \Phi_1} \alpha(\phi_1)) \vee (\bigwedge_{\phi_2 \in \Phi_2} \alpha(\phi_2)) = \mathcal{I}^A(op_F)(\alpha(\Phi_1)\ \alpha(\Phi_2)).\end{aligned}$$

The first equality is the concrete interpretation of op_F, the second is the definition of α on sets of formulas. The third equality is the fact that α distributes over disjunctions and rewrites the iteration over the elements of Φ. The following equality is distributivity of conjunction over disjunction, and the fact that we work up to logical equivalence. The last is the abstract interpretation of op_F and the definition of the abstraction function.

It remains to show (P5). For $\mathcal{I}^C(\$)$ and $\mathcal{I}^C(a)$, there is nothing to do as all ground values are compatible. Assume F is owned by $\Box$ and op_F is binary. The proof for $\Diamond$ is similar. We show that, given a set of formulas Φ, the function $\Phi \cup -$ is compatible. An inspection of the proof of (P4) shows that for any set of formulas ϕ_1, we have

$$\alpha(\Phi \cup \Phi_1) = \alpha(\Phi) \wedge \alpha(\Phi_1).$$

Hence, if $\alpha(\Phi_1) = \alpha(\Phi_2)$, then $\alpha(\Phi \cup \Phi_1) = \alpha(\Phi \cup \Phi_2)$. That $\Phi \cup \Phi_1$ is compatible holds as the element is ground. ◀

E.4 Proof of Lemma 18

Proof. v We have $\mathcal{I}^O(\$) = \bigvee Q_f = \alpha(Q_f) = \alpha(\mathsf{acc}(\varepsilon))$. For $\mathcal{I}^O(a)$, we note that both the abstract and the optimized interpretation distribute over conjunctions and disjunctions. Hence, it remains to consider whether the application to leaves results in a disjunction that is the image of an abstract set. Let $Q = \mathsf{acc}(w)$. We have

$$\begin{aligned}\mathcal{I}^O(a)\ \alpha(\mathsf{acc}(w)) = \mathcal{I}^O(a)\ (\bigvee Q) &= \bigvee_{q \in Q} \mathcal{I}^O(a)\ q \\ &= \bigvee_{q \in Q} \bigvee \mathsf{pre}_a(\{q\}) \\ &= \bigvee \mathsf{pre}_a(Q) \\ &= \alpha(\mathsf{pre}_a(Q)) = \alpha(\mathsf{pre}_a(\mathsf{acc}(w))) = \alpha(\mathsf{acc}(a.w)).\end{aligned}$$

The first equality is the definition of the abstraction function. Then we apply distributivity of the optimized interpretation of a over disjunctions. The following equality is the actual interpretation of a in the optimized model. The next equality uses $\mathsf{pre}_a(Q) = \bigcup_{q \in Q} \mathsf{pre}_a(q)$. The following is again the definition of the abstraction function. Then we replace Q by its definition. Finally, we note the interplay between pre_a and $\mathsf{acc}(-)$.

For conjunction and disjunction, which are used as the interpretation of op_F depending on the player, we note that α distributes to the arguments. Hence, if the arguments are $\alpha(\phi_1)$ and ϕ_2, we have $\alpha(\phi_1) \wedge \alpha(\phi_2) = \alpha(\phi_1 \wedge \phi_2)$. ◀

E.5 Proof of Lemma 19

Proof. We need, for all terminals s, $\mathcal{I}^O(s)$ is $\sqcap$-continuous over the respective lattices. We remark that the case $s = op_F$ is identical to Lemma 5. The case $s = a \in \Gamma$ follows from distributivity of $\mathcal{I}^O(a)$ as in the proof of Lemma 15. ◀

E.6 Proof of Proposition 20

Proof. We show the optimised abstraction is precise. Surjectivity in (P1) holds by definition as does (P3). Also $\sqcap$-continuity in (P2) is by the fact that the meets over the concrete domain are finite, and hence the definition of α already yields continuity. We argue for (P4).

For \$, Lemma 18 yields $\mathcal{I}^O(\$) = \alpha(Q_f)$, which is $\alpha(\mathcal{I}^A(\$))$ as required. For a, the same lemma shows $\mathcal{I}^O(a)\ \alpha(\mathsf{acc}(w)) = \alpha(\mathsf{pre}_a(\mathsf{acc}(w)))$, which is $\alpha(\mathcal{I}^A(a)\ \mathsf{acc}(w))$. The equality generalizes to formulas as both, the abstraction function and the interpretations distribute over conjunctions and disjunctions. For op_F, assume it is a binary conjunction. We have

$$\mathcal{I}^O(op_F)\ \alpha(\phi_1)\ \alpha(\phi_2) = \alpha(\phi_1) \wedge \alpha(\phi_2) = \alpha(\phi_1 \wedge \phi_2) = \alpha(\mathcal{I}^A(op_F)\ \phi_1\ \phi_2).$$

The first equality is the definition of the interpretation in the optimized model, the next is distributivity of α over conjunction. Finally, we have the interpretation of op_F in the abstract model.

For (P5), there is nothing to do for $\mathcal{I}^C(\$)$ and $\mathcal{I}^C(a)$, as all ground values are compatible. We consider the conjunctions and disjunctions used to resolve the non-determinism. Consider a formula ϕ. The task is to show that the function $\phi \wedge -$ is compatible. Consider ϕ_1 and ϕ_2 with $\alpha(\phi_1) = \alpha(\phi_2)$. Then

$$\alpha(\phi \wedge \phi_1) = \alpha(\phi) \wedge \alpha(\phi_1) = \alpha(\phi) \wedge \alpha(\phi_2) = \alpha(\phi \wedge \phi_2).$$

The first equality is distributivity of the abstraction function over conjunctions. The next is the assumed equality. The third is again distributivity. Compatibility of $\phi \wedge \phi_1$ holds as ground values are always compatible. ◀

E.7 Proof of Corollary 24

To show the complexity, we argue the upper and lower bounds separately.

Proof of Proposition 22. We need to argue that $\sigma_{\mathcal{M}^O}$ can be computed in $(k+1)$-times exponential time. We have that $\sigma_{\mathcal{M}^O} = \bigsqcap_{i \in \mathbb{N}} rhs^i_{\mathcal{M}^O}(\sigma^0_l)$. Since the domains $\mathcal{D}^O(\kappa)$ are finite for all kinds κ, there is an index $i_0 \in \mathbb{N}$ such that $\sigma_{\mathcal{M}^O} = \bigsqcap_{i=0}^{i_0} rhs^i_{\mathcal{M}^O}(\sigma^0_l) = rhs^{i_0}_{\mathcal{M}^O}(\sigma^0_l)$. In the following, we will see that the number of iterations, i.e. the index i_0 is at most $(k+1)$-times exponential, and that one iteration can be executed in $(k+1)$-times exponentially many steps.

First, we reason about the number of iterations. For a partial order $\mathcal{D}$, we define its *height* $h(\mathcal{D})$ as the length of the longest strictly descending chain, i.e. the height is m if the longest such chain is of the shape

$$x_0 > x_1 > \ldots > x_k\ .$$

The height of the domain is an upper bound for i_0 by its definition: If for some index i_1 we have $rhs^{i_1}_{\mathcal{M}^O}(\sigma^0_l) = rhs^{i_1+1}_{\mathcal{M}^O}(\sigma^0_l)$, we know $\bigsqcap_{i=0}^{i_0} rhs^i_{\mathcal{M}^O}(\sigma^0_l) = rhs^{i_0 1}_{\mathcal{M}^O}(\sigma^0_l)$ and thus $i_1 = i_0$. Such an index i_1 has to exist and has to be smaller than the height of the domain, otherwise the sequence of the $rhs^i_{\mathcal{M}^O}(\sigma^0_l)$ would form a chain that is strictly longer than the height, a contradiction to the definition.

It remains to see what the height of our optimised domain is. Recall that $rhs_{\mathcal{M}^O}$ has the type signature $(N \to \mathcal{D}^O) \to (N \to \mathcal{D}^O)$. Our goal in the following is to determine $h(N \to \mathcal{D}^O)$. We can identify $N \to \mathcal{D}^O$ with $\mathcal{D}^O(F_1) \times \ldots \times \mathcal{D}^O(F_\ell)$, where $F_1, \ldots, F_\ell$ are the non-terminals of the scheme. The height of this product domain is the sum of its height. We are done if we show that even the domain $\mathcal{D}^O(F)$ with the maximal height is $(k+1)$-times exponentially high, since the number of non-terminals is polynomial in the input scheme.

In the following we prove: If kind κ is of order k', then $\mathcal{D}^O(\kappa)$ has $(k'+1)$-times exponential height. For the induction step, we also need to consider the cardinality of $\mathcal{D}^O(\kappa)$, therefore, we strengthen the statement and also prove that the cardinality $\left|\mathcal{D}^O(\kappa)\right|$ is $(k'+2)$-times exponential.

We proceed by induction on k'.

In the base case $k' = 0$, we necessarily have $\kappa = o$, and indeed the domain $\alpha(\mathsf{PBool}(\mathsf{acc}(T^*))) \subseteq \mathsf{PBool}(Q_{NFA})$ is singly exponentially high. To see that this is the case, consider a strictly decreasing chain $(\phi_j)_{j\in\mathbb{N}}$ of positive boolean formulas over Q_{NFA}, i.e. a chain where each formula is strictly implied by the next. To each formula, ϕ_j, we assign the set $\mathcal{Q}_j = \{Q \subseteq Q_{NFA} \mid Q \text{ satisfies } \phi_j\}$ of assignments under which ϕ_j evaluates to true. That ϕ_j is strictly implied by ϕ_{j+1} translates to the fact that $\mathcal{Q}_j$ is a strict subset of $\mathcal{Q}_{j+1}$. This gives us that the sets $\mathcal{Q}_j$ themselves form a strictly ascending chain in $\mathcal{P}(\mathcal{P}(Q_{NFA}))$, and it is easy to see that such a chain has length at most $|\mathcal{P}(Q_{NFA})| = 2^{|Q_{NFA}|}$.

Furthermore, we can represent each equivalence class of formulas in $\mathsf{PBool}(Q_{NFA})$ by a representative in conjunctive normal form, i.e. by an element of $\mathcal{P}(\mathcal{P}(Q_{NFA}))$. This shows that the cardinality of the domain is indeed bounded by $|\mathcal{P}(\mathcal{P}(Q_{NFA}))| = 2^{|\mathcal{P}(Q_{NFA})|} = 2^{2^{|Q_{NFA}|}}$.

Now assume the statement holds for k', and consider κ of order $k'+1$. We need an inner induction on the arity m of κ.

Since o is the only kind of arity 0, and does not have order $k'+1$ for any k', there is nothing to do in the base case.

Now assume that $\kappa = \kappa_1 \to \kappa_2$. By the definitions of arity and order, we know that κ_1 is of order at most k', therefore we now by the outer induction that the height of $\mathcal{D}^O(\kappa_1)$ is at most $(k'+1)$-times exponential. The order of κ_2 is at most $(k'+1)$, but the arity of κ_2 is strictly less than the arity of κ, thus we get by the inner induction that the height of $\mathcal{D}^O(\kappa_2)$ is at most $(k'+2)$-times exponential.

The domain $\mathcal{D}^O(\kappa_1 \to \kappa_2) = \mathit{Cont}(\mathcal{D}^O(\kappa_1), \mathcal{D}^O(\kappa_2))$ is a subset of all functions from $\mathcal{D}^O(\kappa_1)$ to $\mathcal{D}^O(\kappa_2)$. Let us reason about the height of this more general function domain. We know that its height is the height of the target times the size of the source, i.e. $h(\mathcal{D}^O(\kappa_2)) \cdot \left|\mathcal{D}^O(\kappa_1)\right|$. The induction completes the proof, as both $h(\mathcal{D}^O(\kappa_2))$ and $\left|\mathcal{D}^O(\kappa_1)\right|$ are at most $(k'+2)$-times exponential.

It remains to argue that each iteration can be implemented in at most $(k+1)$-times exponentially many steps. To this end, we argue that each element of $\mathcal{D}^O(\kappa)$ can be represented by an object of size $(k'+1)$-times exponential, where k' is the order of κ. It is easy to see that all operations that need to be executed on these objects, namely evaluation, conjunction, disjunction, and predecessor computation can be implemented in polynomial time in the size of the objects.

Let $k' = 0$, i.e. $\kappa = o$. We again represent each element of $\mathcal{D}^O(o)$ by a formula over Q_{NFA} in conjunctive normal form, i.e. as an element of $\mathcal{P}(\mathcal{P}(Q_{NFA}))$. In the worst case, one single formula ϕ contains everyone of the $2^{Q_{NFA}}$ many clauses, each clause having size at most $|Q_{NFA}|$. This means that one formula needs at most singly exponential space.

For the induction step, consider κ of order $k+1$. As above, we need an inner induction on the arity of κ, for which the base case is trivial.

Let $\kappa = \kappa_1 \to \kappa_2$. An element of $\mathcal{D}^O(\kappa)$ is a function that assigns to each of the $\left|\mathcal{D}^O(\kappa_1)\right|$-many elements of $\mathcal{D}^O(\kappa_1)$ an element of $\mathcal{D}^O(\kappa_2)$. In the previous part of the proof, we have argued, that $\left|\mathcal{D}^O(\kappa_1)\right|$ is at most $(k+1)$ times exponential. By the induction on the arity, we know that each object in $\mathcal{D}^O(\kappa_2)$ can be represented in at most $(k+2)$-times exponential space. This shows that objects of $\mathcal{D}^O(\kappa)$ can be represented using $(k+2)$-times exponential space, and finishes the proof. ◀

We show that determining the winner in a higher-order word game is $(k+1)$EXP-hard for an order-k recursion scheme.

Proof of Proposition 23. We begin with a result due to Engelfriet [14] that shows alternating k-iterated pushdown automata with a polynomially bounded auxiliary work-tape (k-PDA$^+$) characterise the $(k+1)$EXP word languages. We fix any $(k+1)$EXP-hard language and its corresponding alternating k-PDA B. Let $\mathcal{L}(B)$ be the set of words accepted by B. Deciding if a given word w is in the language defined by B is $(k+1)$EXP-hard in the size of w (recall B is fixed). We show that this problem can be reduced in polynomial time to an inclusion problem $\mathcal{L}(B') \subseteq \mathcal{L}(A)$ for some k-iterated pushdown automaton (without work-tape) (k-PDA) B' and NFA A of size polynomial in the length of w. From B', we can construct in polynomial time an equivalent game over a scheme G. This will show the game language inclusion problem for order-k schemes is $(k+1)$EXP-hard.

In an alternating k-PDA$^+$, there are two Players $\diamond$ and $\Box$. When decided whether a word w is in the language of a k-PDA$^+$, $\diamond$ will attempt to prove the word is in the language, while $\Box$ will try to refute it.

We first describe how to obtain B' from B. Since the word w is fixed, we can force B to output the word w by forming a product of w with the states of B. Call this automaton $B \times w$. This reduces the word membership problem to the problem of determining whether $B \times w$ can reach an accepting state. Next, to remove the worktape from $B \times w$ (and form B') we replace the output of $B \times w$ (which will always be w or empty) with a series of guesses of the worktape. That is, a transition of $B \times w$ will be simulated by B' by first making a transition as expected, and then outputting a guess (consistent with the transition) of what the worktape of $B \times w$ should be. The automaton A will accept a guessed sequence of worktapes iff it is able to find an error in the sequence. The word w will be in the language of B if B' is able to reach a final state and produce a word w' that is correct; that is, w' is *not* in the language of A.

Note, here, the reversal of the roles of the Players. In B, control states are owned by $\diamond$ or $\Box$. When determining if $w \in \mathcal{L}(B)$ for some w, the first Player $\diamond$ tries to show the word is accepted, while the second Player $\Box$ tries to force a non-accepting run. In B', however, w is accepted iff the output of B' is not included in the language of A. Thus, $\Box$ will effectively be aiming to prove that $w \in \mathcal{L}(B)$.

In more detail, we take any $(k+1)$EXP-hard language and its equivalent (fixed) alternating k-PDA$^+$. Given a word w, deciding $w \in \mathcal{L}(B)$ is $(k+1)$EXP-hard. We define B' directly from B rather than going through the intermediate $B \times w$.

A transition (p, a, o, σ, p') of B means the following. From control state p, upon reading a character a from w, apply operation o to the work-tape (which may become stuck if not applicable) and operation σ to the stack (which may also become stuck if not applicable). Next, move to control state p', from which the remainder of w is to be read.

Let m be the polynomial bound on the size of the work-tape of A given the input word w. Let Σ be the alphabet of the work-tape. Let the set of work-tape operations $O = \{o_1, \ldots, o_n\}$ and work-tape positions $P = \{1, \ldots, m\}$ be disjoint from Σ. Also, let $\circ \in \Sigma$ be the initial symbol appearing in each cell of the initial work-tape. We will construct A' such that

$$\mathcal{L}(A') \subseteq \circ^m (PO\Sigma^m)^* \ .$$

That is, A' outputs a sequence of work-tape configurations separated by positions in P and operations in O. That is, A' will simulate a run of A over w.

For every control state p of A, we will have control states (p, w') of A', where w' is a suffix of w. We will also have (p, w', o) where o is a work-tape operation to be applied. Then for each transition (p, a, o, σ, p') of B we have a transition $((p, aw'), \varepsilon, \sigma, (p', w', o))$ of $B \times w$. From (p', w', o) the automaton B' will output some character from P (a guess at the work-tape head position), followed by o (to indicate the operation applied). It will then be able to output any word from Σ^m (a guess of the work-tape contents) before moving to (p', w') and continuing the simulation. Initially, B' will simply output $\circ^m$ and move to control state (p, w) where p is the initial control state of B.

The final step in defining B' is to assign ownership of the control states. Recall, we needed to switch the roles of the Players. Thus, we define $O((p, w)) = \Diamond$ whenever p belongs to $\Box$ in B. All other control states of B' are owned by $\Box$. We define the accepting control states to be those of the form (p, ε) where p is accepting in B. Observe these have no outgoing transitions.

Next we define the regular automaton A which detects mistakes in the work-tape. Such an error is either due to a poorly updated cell, or due to a poorly updated head position. The set of work-tape operations O is such that there is a mapping

$$\pi : (P \times O \to P) \cup (P \times \Sigma \times P \times O \to \Sigma \cup \{\bot\})$$

where $\bot \notin \Sigma$ and

- $\pi(i, o) = j$ means if the head is at position i, it is at position j after operation o, and
- $\pi(i, \alpha, j, o) = \beta$ means, if the head is at position i, α is the contents of the cell at position j, and operation o is applied, then β is the contents of the cell after applying o. If $\beta = \bot$ then o could not be applied to this work-tape and became stuck. (E.g. if $i = j$ and the operation required the head to read a character other than α.)

Thus, we require the following regular language, for which a polynomially-sized regular automaton is straightforward to construct. Let $\Gamma = \Sigma \cup P \cup O$.

$$\mathcal{L}(A) = \left(\Gamma^* \left(\bigcup_{\pi(i,o)\neq j} io\Sigma^m j\right)\Gamma^*\right) \cup \left(\Gamma^* \left(\bigcup_{\pi(i,\alpha,j,o)\neq\beta} io\Sigma^j \alpha\Gamma^{m+2}\beta\right)\Gamma^*\right) .$$

We have thus defined a k-PDA B' that produces some word w' not accepted by A iff w is accepted by B.

The final step is to produce a game over a scheme G that is equivalent to the game problem for k-iterated pushdown automata. This is in fact a straightforward adaptation of the techniques introduced by Knapik *et al.* [31]. However, we choose to complete the sketch using definitions from Hague *et al.* [23] as we believe these provide a clearer reference. In particular, we adapt their Definition 4.3.

The key to the reduction is a tight correspondence (given in op. cit.) between configurations (q, s) of a k-iterated pushdown automaton, and terms of the form[1] $F^a_q \vec{\Psi}_{k-1} \cdots \vec{\Psi}_0$. That is, every configuration is represented (in a precise sense) by such a term and every term of such a form represents a configuration. Moreover, for every transition (q, a, o, σ, q') of the pushdown automaton, when $o \neq \varepsilon$ we can associate a rewrite rule of the scheme

$$F^a_q = \lambda\vec{x}.o(e_{(q',\sigma)})$$

such that the term obtained by applying the rewrite rule to $F^a_q \vec{\Psi}_{k-1} \cdots \vec{\Psi}_0$ is a term $o(F^b_{q'} \vec{\Psi}'_{k-1} \cdots \vec{\Psi}'_0)$ where $F^b_{q'} \vec{\Psi}'_{k-1} \cdots \vec{\Psi}'_0$ represents the configuration reached by the transition. That is, $(q', \sigma(s))$. When $o = \varepsilon$ we simply omit o, that is

$$F^a_q = \lambda\vec{x}.e_{(q',\sigma)} .$$

[1] In fact, in op. cit. non-terminals had the form $F^{a,e}_q \vec{\Psi}\vec{\Psi}_{k-1} \cdots \vec{\Psi}_0$. where e and $\vec{\Psi}$ are used to handle *collapse links*, which we do not need here.

To each non-terminal, we assign $O(F_q^a) = \Diamond$ whenever q is a $\Diamond$ control state. Otherwise, $O(F_q^a) = \Box$. For every accepting control state q we introduce the additional rule

$$F_q^a = \lambda \vec{x}.\$ \ .$$

Finally, we have an initial rule

$$S = t$$

where t is the term representing the initial configuration.

Given the tight correspondence between configurations and transitions of the k-PDA and terms and rewrite steps of G, alongside the direct correspondence between the owner of a control state q and the owner of a non-terminal of G, it is straightforward to see, via induction over the length of an accepting run in one direction, or derivation sequence in the other, that B' is able to produce a word not in A iff a word not in A is derivable from S. Thus, we have reduced the word acceptance problem for some alternating k-PDA$^+$ to the game problem for language inclusion of a scheme. This shows the problem is $(k+1)$EXP-hard. ◀