A cartesian-closed category for higher-order model checking

Martin Hofmann
LMU Munich
Email: martin.hofmann@ifi.lmu.de

Jérémy Ledent
LIX, École Polytechnique
Email: jeremy.ledent@lix.polytechnique.fr

***Abstract*—In previous work we have described the construction of an abstract lattice from a given Büchi automaton. The abstract lattice is finite and has the following key properties. (i) There is a Galois insertion between it and the lattice of languages of finite and infinite words over a given alphabet. (ii) The abstraction is faithful with respect to acceptance by the automaton. (iii) Least fixpoints and ω-iterations (but not in general greatest fixpoints) can be computed on the level of the abstract lattice.**

This allows one to decide whether finite and infinite traces of first-order recursive boolean programs are accepted by the automaton and can further be used to derive a type-and-effect system for infinitary properties.

In this paper, we show how to derive from the abstract lattice a cartesian-closed category with fixpoint operator in such a way that the interpretation of a *higher-order* recursive program yields precisely the abstraction of its set of finite and infinite traces and thus provides a new algorithm for the higher-order model checking problem for trace properties.

All previous algorithms for higher-order model checking [2], [16] work inherently on arbitrary tree properties and no apparent simplification appears when instantiating them with trace properties. The algorithm presented here, while necessarily having the same asymptotic complexity, is considerably simpler since it merely involves the interpretation of the program in a cartesian-closed category.

The construction of the cartesian closed category from a lattice is new as well and may be of independent interest.

I. Introduction

We consider here the following scenario. We are given a program e whose evaluation emits a finite or infinite trace of events from a fixed alphabet Σ. The program e may involve higher-order functions and general recursive definitions even of higher-order functions.

Assuming that conditionals are abstracted as nondeterministic choice, decide whether every trace emitted belongs to an ω-regular language specified by a Büchi automaton, a temporal logic formula or similar.

Notice that if an abstracted program passes this test then so does the original one with "real" conditionals and full data dependency. The converse does of course not hold for bad behaviours in the abstracted program might not correspond to actual ones.

It is possible to treat these problems using higher-order model checking [2], [16], [20]. In this case, one first translates the program to a term in $\lambda\mathbf{Y}$, i.e., simply-typed lambda calculus with fixpoint combinator. One then introduces unary function symbols for the events and a binary function symbol for the conditional. The desired policy must then be translated into a path property of Böhm trees using μ-calculus. Indeed, higher-order model checking is capable of deciding whether the Böhm tree of a given term satisfies a prescribed μ-calculus formula. In the course of the verification there also arise genuine tree properties that must hold on (higher-order) subterms of the toplevel program even though at the end of the day we are interested in a path property only.

Since many properties relevant in practice are in fact path properties there arises the question whether there might not be a direct procedure that only operates on path properties. While for first-order programs such procedures are available [4], [27], [14], no such route was available for genuinely higher-order programs and indeed it was held as a belief in the community[1] that the detour via genuine tree properties is unavoidable for higher-order model checking just as complex numbers (or trigonometric functions) are needed to solve third order equations even if one only cares about real solutions. In a similar vein Walukiewicz and Salvati write about [14] in [24] that "their construction is restricted only to first-order $\lambda\mathbf{Y}$-terms. They use in an elegant way Wilke algebras that are an algebraic notion of recognizer for languages of infinite words. One of the problems we are facing here is that there does not exist equally satisfying notion of an algebraic recognizer for infinite trees. Even if we wanted to stay with properties of paths, it is not clear how to extend Wilke algebras to higher orders, the problem being to find an admissible class of fixpoint operations." Indeed, the contribution of this paper can be seen as an answer to this question. Starting from an ω-semigroup $\mathfrak{V}$ which is a known and mild generalization of a Wilke-algebra, we construct a cartesian-closed category $\mathbf{AFF}_{\mathfrak{V}}$ with fixpoints such that the interpretation of a $\lambda\mathbf{Y}$ term of ground type in this category allows one to check whether its traces are recognized by the underlying ω-semigroup and hence, if the ω-semigroup is constructed from a policy automaton, whether its traces are accepted by it.

Unlike in the model constructions by Salvati and Walukiewicz [24], [25] and the related works by Grellois and Melliès [13], [12], [11] no detour via parity tree automata and the resulting complexity is needed.

Our starting point is the abstract lattice construction from

978-1-5090-3018-7/17/$31.00 ©2017 IEEE

[1]Private communication with I. Walukiewicz and S. Salvati

[14] which from a given pair L_+, L_ω of regular and ω-regular languages constructs a finite ω-semigroup $\mathfrak{M} = (\mathfrak{M}_+, \mathfrak{M}_\omega)$ (in [14] it was only used as a Wilke algebra) which additionally carries a complete lattice structure (written "squarely" $\sqsubseteq$). In addition there is a Galois insertion α, γ between the ω-semigroup $(\mathfrak{L}_+, \mathfrak{L}_\omega)$ of (ω-)languages over Σ and $\mathfrak{M}$, the finite one. We remark that the abstraction is based on Büchi's original approach to complementation of Büchi automata [5] using equivalence relations which also forms the bases of the modern Ramseyian approach to the study of ω-regular language and automata [9], [1].

The abstraction is faithful in that $L \subseteq L_{+/\omega}$ iff $\alpha(L) \sqsubseteq U_{+/\omega}$ in the abstract lattice for some fixed elements $U_+ \in \mathfrak{M}_+$ and $U_\omega \in \mathfrak{M}_\omega$ independent from L.

Thus, in order to know whether the traces of a first-order program e fall into $L_{+/\omega}$ it suffices to interpret it (in the sense of abstract interpretation) in the abstract lattice using least fixpoints and ω-iterations to interpret recursion. This was essentially the contribution of [14]. In addition, it was shown there how the elements of the abstract lattice can be regarded as *effects* in a generic type-and-effect system like [26], [3], [10] and thus yield a type-based analysis capable of dealing with objects and higher-order functions, however with recursion restricted to first-order and with objects being abstracted to regions thus losing some precision, see [14] for details.

In order to generalize this to higher-order functions we make the following crucial observation. Suppose we have an ω-semigroup $\mathfrak{V} = (\mathfrak{V}_+, \mathfrak{V}_\omega)$ and a function $F : \mathfrak{V}_+ \times \mathfrak{V}_\omega \to \mathfrak{V}_\omega$ *which arises as the denotation of a second-order* $\lambda\mathbf{Y}$ *term*. While the dependency of $F(X_+, X_\omega)$ on the finitary argument X_+ may be rather arbitrary, the dependency on the infinitary argument X_ω is very restricted: there must exist monotone functions $f_\gamma : \mathfrak{V}_+ \to \mathfrak{V}_\omega$ ("constant") and $f_\pi : \mathfrak{V}_+ \to \mathfrak{V}_*$ ("prefix") such that

$$F(X_+, X_\omega) = f_\gamma(X_+) \sqcup f_\pi(X_+) X_\omega$$

Intuitively, once a $\lambda\mathbf{Y}$ term calls an infinitary argument, it can never interrupt it to do some other computation afterwards.

Recall at this point that an ω semigroup $\mathfrak{V} = (\mathfrak{V}_+, \mathfrak{V}_\omega)$ supports concatenation operations $\mathfrak{V}_+ \times \mathfrak{V}_+ \to \mathfrak{V}_+$ and $\mathfrak{V}_+ \times \mathfrak{V}_\omega \to \mathfrak{V}_\omega$ written multiplicatively as well as infinitary concatenation $\pi : \mathfrak{V}_+^{\mathbb{N}} \to \mathfrak{V}_\omega$. In addition, in our lattice-theoretic framework, both $\mathfrak{V}_+, \mathfrak{V}_\omega$ are complete lattices and all operations are monotone. The lattice $\mathfrak{V}_*$ appearing above is the extension of $\mathfrak{V}_+$ with a neutral element. The prototypical example is $\mathfrak{L} = (\mathfrak{L}_+, \mathfrak{L}_\omega)$ with $\mathfrak{L}_+ = \mathcal{P}(\Sigma^+)$ and $\mathfrak{L}_\omega = \mathcal{P}(\Sigma^\omega)$. Finite ω-semigroups arise via abstraction as described above.

Coming back to the above formula, we see that an argument $X_\omega \in \mathfrak{V}_\omega$ can, depending on X_+ either be ignored (by setting $f_\pi(X_+) = 0$, the least element in the lattice) or be prefixed with some element $A \in \mathfrak{V}_+$ (by setting $f_\pi(X_+) = A$) or just pushed through by setting $f_\pi(X_+) = \varepsilon$. In addition, there may be a constant summand $f_\gamma(X_+)$. We have thus seen that functions with $\mathfrak{V}_\omega$ in their domain that arise as denotations can be simplified to functions that only depend on $\mathfrak{V}_+$. While there may be (in)equality tests on some elements of $\mathfrak{V}_+$ no such tests are available for $\mathfrak{V}_\omega$. Once you start running an infinite computation you are stuck with it and there is no turning back.

Slightly more generally, consider that we have a function $F : P \times \mathfrak{V}_\omega^Q \to \mathfrak{V}_\omega$ where P, Q are arbitrary posets, typically $\mathfrak{V}_+$ or powers, products, coproducts thereof. Here $\mathfrak{V}_\omega^Q$ is the monotone function space. Again, if such a function is definable in $\lambda\mathbf{Y}$, i.e., arises as the denotation of such a $\lambda\mathbf{Y}$-term in an appropriate sense, then there must exist monotone functions $f_\gamma : P \to \mathfrak{V}_\omega$ and $f_\pi : P \times Q^{\mathrm{op}} \to \mathfrak{V}_*$ such that

$$F(p, X) = f_\gamma(p) \sqcup \bigsqcup_{q \in Q} f_\pi(p, q) X(q)$$

Here P^{op} is the poset P with the order reversed. In order to show that indeed the denotations of all $\lambda\mathbf{Y}$-terms fall into this rather rigid format we show how the functions that do so can be organised into a cartesian-closed category, the aforementioned category $\mathbf{AFF}_{\mathfrak{V}}$. It merely confirms the intuition that there is no way to further process or undo an infinitely running computation.

Having done that it remains to show how general fixpoints can be resolved. The gist of the construction can be seen as follows. Suppose that $P = Q$ in previous example so that $F : P \times \mathfrak{V}_\omega^P \to \mathfrak{V}_\omega$ and F is induced by f_γ and f_π where additionally $f_\pi(p, p') \in \mathfrak{V}_+$ rather than $\mathfrak{V}_*$. We can then define a fixpoint $U : P \to \mathfrak{V}_\omega$ by

$$\begin{aligned} U(p) = & \bigsqcup_{p_0, \ldots, p_n \in P} f_\pi(p, p_0) f_\pi(p_0, p_1) \ldots f_\pi(p_{n-1}, p_n) f_\gamma(p_n) \\ & \sqcup \bigsqcup_{p_0, \ldots \in P} \pi(f_\pi(p, p_0), f_\pi(p_0, p_1), f_\pi(p_1, p_2) \ldots) \end{aligned}$$

The first summand models computations that eventually use the constant branch f_γ whereas the second one models those that keep adding prefixes.

We remark that this construction of fixpoints of affine maps as a finite/infinite product already appears in [8] albeit not in a higher-order setting. The fixpoint construction in [12] is also related but more distantly so since it takes parity conditions into account.

Surprisingly, it turns out that due to the way function spaces are constructed in $\mathbf{AFF}_{\mathfrak{V}}$ this pattern already covers fixpoints at all higher types. The only complication, namely the necessity to pass from ω-iteration (Wilke algebras) to infinitary products (ω-semigroups) already arises at second order. Thereafter, nothing new happens, provided of course, one confines attention as we do to path properties.

The rest of this paper is organised as follows. The preliminary section II defines our language and its operational semantics. In section III, we show how it is related to $\lambda\mathbf{Y}$. We then define in section IV a first cartesian-closed category $\mathbf{GFP}$ in which we can interpret our language, and we relate it to the operational semantics. However this category uses general greatest fixpoints to interpret recursion, which are not necessarily preserved by the abstraction function α. The idea is then to refine this interpretation in order to make precise the fact that all the greatest fixpoints that we actually need can be expressed as infinite products. In section V we define Büchi algebras, and in section VI we recall the construction from [14] of a finite Büchi algebra $\mathfrak{M}$ from a Büchi automaton. Finally in section VII, we define the cartesian-closed category $\mathbf{AFF}_{\mathfrak{V}}$, which is parameterized by a Büchi algebra $\mathfrak{V}$. We can then interpret our language in both $\mathbf{AFF}_{\mathfrak{L}}$ and $\mathbf{AFF}_{\mathfrak{M}}$, and relate these with the interpretation in **GFP**. The main result of the paper is stated in Corollary 2.

Acknowledgement: We acknowledge support by the German Research Foundation (DFG) through the grant GuideForce. We also thank the NUS-IMS, Singapore, for an invitation to the programme on Automata, Logic and Games, Sep. 2016, where a preliminary version of this work was presented. The discussions with the participants were very fruitful and helped us bring this work into its present form.

II. Preliminaries

Let Σ be a finite alphabet containing a special symbol $\checkmark$.

We write Σ^+ for the set of finite *nonempty* words over Σ and Σ^ω for the set of infinite words. Henceforth, we call the elements of Σ^+ just *finite words* without the qualification of nonemptiness. Finite words can be concatenated with finite or infinite words as usual and we note concatenation by juxtaposition or "$\cdot$". We write $\mathfrak{L}_+ := \mathcal{P}(\Sigma^+)$ for the set of languages of finite words and $\mathfrak{L}_\omega := \mathcal{P}(\Sigma^\omega)$ for the set of languages of infinite words.

If $(L_i)_{i\in\mathbb{N}}$ is an $\mathbb{N}$-indexed family of languages of finite words then we can form their ω-product $\prod_{i=0}^{\infty} L_i \in \mathfrak{L}_\omega$ which consists of all the infinite words of the form $w_0w_1w_2\dots$ with $w_i \in L_i$. Notice that this is well-defined since the empty words does not belong to any of the L_i. This operation endows the pair $\mathfrak{L}_+, \mathfrak{L}_\omega$ with the structure of an ω-semigroup in the sense of [6]. Of course, Σ^+, Σ^ω also forms an ω-semigroup.

We consider a simply-typed lambda calculus over one base type **comm** representing commands. The syntax of terms is given by

$$e ::= x \mid a \mid e_1; e_2 \mid e_1 + e_2 \mid \text{fix} \mid \lambda x.\, e \mid e_1\ e_2$$

Herein, x ranges over variables (to be bound by λ), a ranges over letters from $\Sigma \setminus \{\checkmark\}$, and $e_1; e_2$ represents sequential composition, and $e_1 + e_2$ stands for nondeterministic choice which is generally used as an abstraction of a conditional "if b then e_1 else e_2". The types are

$$\tau ::= \mathbf{comm} \mid \tau_1 {\to} \tau_2$$

A context Γ is a finite function from variables to types. The typing judgement $\Gamma \vdash e : \tau$ is defined in the usual way by the following, entirely standard, typing rules.

$$\frac{}{\Gamma \vdash x : \Gamma(x)} \qquad \frac{\Gamma \vdash e_1 : \tau_1 \to \tau_2 \quad \Gamma \vdash e_2 : \tau_1}{\Gamma \vdash e_1\ e_2 : \tau_2}$$

$$\frac{\Gamma, x : \tau_1 \vdash e : \tau_2}{\Gamma \vdash \lambda x.e : \tau_1 \to \tau_2} \qquad \frac{\Gamma \vdash e : \tau \to \tau}{\Gamma \vdash \text{fix}\ e : \tau} \qquad \frac{}{\Gamma \vdash a : \mathbf{comm}}$$

$$\frac{\Gamma \vdash e_1 : \mathbf{comm} \quad \Gamma \vdash e_2 : \mathbf{comm}}{\Gamma \vdash e_1 + e_2 : \mathbf{comm}} \qquad \frac{\Gamma \vdash e_1 : \mathbf{comm} \quad \Gamma \vdash e_2 : \mathbf{comm}}{\Gamma \vdash e_1; e_2 : \mathbf{comm}}$$

Let e be a closed term of type **comm** and e' also be such a term or the special symbol **skip**. Let $a \in \Sigma$, we write $e \xrightarrow{a} e'$ to mean that e evaluates in one step to e' issuing event a. The definition of this reduction relation is again completely standard and is based on "call-by-name"; the only peculiarity is that unfolding a fixpoint leads to a $\checkmark$-event being issued. This is done so that nonterminating programs always leave an infinite trace which simplifies the subsequent technical development. See [14] for a way to get rid of the $\checkmark$-events. The inductive definition of the one-step reduction is then as follows:

$$\frac{}{a \xrightarrow{a} \mathbf{skip}} \qquad \frac{e[e'/x]\ \vec{e} \xrightarrow{a} e''}{(\lambda x.\, e)\ e'\ \vec{e} \xrightarrow{a} e''} \qquad \frac{}{\text{fix}\ e\ \vec{e} \xrightarrow{\checkmark} e\ (\text{fix}\ e)\ \vec{e}}$$

$$\frac{e_1 \xrightarrow{a} e_1' \neq \mathbf{skip}}{e_1; e_2 \xrightarrow{a} e_1'; e_2} \quad \frac{e_1 \xrightarrow{a} \mathbf{skip}}{e_1; e_2 \xrightarrow{a} e_2} \quad \frac{e_1 \xrightarrow{a} e_1'}{e_1 + e_2 \xrightarrow{a} e_1'} \quad \frac{e_2 \xrightarrow{a} e_2'}{e_1 + e_2 \xrightarrow{a} e_2'}$$

We henceforth call *program* a closed term of type **comm**.

Remark. A program is necessarily of one of the following forms: a, $e_1 + e_2$, $e_1; e_2$, fix $e\ \vec{e}$, or $(\lambda x.\, e)\ e'\ \vec{e}$, where $\vec{e}$ is a possibly empty chain of applications.

Let e be a program. We define $L_+(e)$ as the set of those words $a_1 \dots a_n \in \Sigma^+$ such that

$$e \xrightarrow{a_1} e_1 \xrightarrow{a_2} e_2 \dots e_{n-1} \xrightarrow{a_n} \mathbf{skip}$$

Notice that $L_+(e) \subseteq \Sigma^+$ for each program e.

We define $L_\omega(e)$ as the set of those infinite words $w \in \Sigma^\omega$ such that there exist programs $e_1, e_2 \dots$ with $e_1 = e$ and $e_i \xrightarrow{a_i} e_{i+1}$ for $i = 1, 2, \dots$ and $w = a_1, a_2, \dots$.

Thus, $L_+(e)$ records the traces of terminating executions of e whereas $L_\omega(e)$ records the traces of nonterminating executions. We thus have that at most one of $L_+(e)$ and $L_\omega(e)$ is the empty set.

Examples. Let $e_1 := \text{fix}(\lambda x.\, a; x)$. We have $L_+(e_1) = \varnothing$ and $L_\omega(e_1) = (\checkmark\ a)^\omega$.

Let $e_2 := \text{fix}(\lambda x.\, (a; x) + b)$. We have $L_\omega(e_2) = L_\omega(e_1)$, but $L_+(e_2) = (\checkmark\ a)^*\checkmark\ b$.

Let $e_3 := \text{fix}(\lambda x.\, (a; x; b) + c)$. We have $L_\omega(e_3) = L_\omega(e_1)$, but $L_+(e_3) = \{(\checkmark\ a)^n \checkmark\ c\, b^n \mid n \geq 0\}$.

Let $e_4 := \text{fix}(\lambda f.\lambda x.\, (a; f(b; x; c)) {+} x)$. We have $L_+(e_4\ d) = \{(\checkmark\ a)^n \checkmark\ b^n\, d\, c^n \mid n \geq 0\}$ and $L_\omega(e_4\ d) = (\checkmark\ a)^\omega$.

Let $e_5 := \text{fix}(\lambda x.\, (e_4\ d); x)$. We have $L_+(e_5) = \varnothing$ and $L_\omega(e_5) = (\checkmark\ L_+(e_4\ d))^\omega \cup (\checkmark\ L_+(e_4\ d))^*(\checkmark\ a)^\omega$.

Let $e_6 := (\lambda x.\, a; x)\ b$. We have $L_+(e_6) = \{ab\}$.

It is well known that if within e recursion (fix) is used at type **comm** only, then $L_+(e)$ is context-free and so is $L_\omega(e)$ in a suitably extended sense, whereas in the presence of higher-type recursion (cf. Examples e_4 and e_5) non context-free languages may appear. Example e_6 illustrates that our semantics is call-by-name, i.e., arguments are not completely evaluated before plugging them in for formal parameters.

III. Relationship with $\lambda\mathbf{Y}$

In [24] the calculus $\lambda\mathbf{Y}$ is studied which has none of $a, \mathbf{skip}, ;, +$ but instead is parametrized over a set of first-order constants, i.e., unary, binary, ternary, etc.

We can encode our calculus into $\lambda\mathbf{Y}$ (with base type written o) by introducing a unary constant $a : o \to o$ for each $a \in \Sigma$ and accordingly defining $\mathbf{comm} := o \to o$. We further define sequential composition $e_1; e_2 := \lambda x.\, e_1(e_2\ x)$. For nondeterministic choice we introduce a single binary constant $+$. Our traces then correspond to the branches (finite and infinite) of the Böhm-tree associated with the encoding of a term.

We can also achieve a converse translation by mapping a first-order $\lambda\mathbf{Y}$-constant c of arity $n \geq 1$ to $\lambda x_1 \dots \lambda x_n.\bar{c}; (x_1 + \cdots + x_n)$ where $\bar{c}$ is an event corresponding to the constant c. Since we are in this paper only interested in branches, i.e., trace properties and not global tree properties, we prefer our formulation over $\lambda\mathbf{Y}$ since it is closer to programming practice. Since, however, both calculi are so close, we use the name $\lambda\mathbf{Y}$ for ours in the rest of this paper as well.

IV. Denotational semantics

We also have a denotational characterisation of L_+ and L_ω as follows

Proposition 1. *L_+ and L_ω are the least, resp. greatest solutions of the following equations:*

$$\begin{array}{rcl}
L_+(a) & = & \{a\} \\
L_\omega(a) & = & \varnothing \\
L_+(e_1; e_2) & = & L_+(e_1)L_+(e_2) \\
L_\omega(e_1; e_2) & = & L_\omega(e_1) \cup L_+(e_1)L_\omega(e_2) \\
L_{+/\omega}(e_1 + e_2) & = & L_{+/\omega}(e_1) \cup L_{+/\omega}(e_2) \\
L_{+/\omega}(\text{fix } e\ \vec{e}) & = & \checkmark L_{+/\omega(e\ (\text{fix } e)\ \vec{e})} \\
L_{+/\omega}((\lambda x.\, e)\ e'\ \vec{e}) & = & L_{+/\omega}(e[e'/x]\ \vec{e})
\end{array}$$

Proof. This is standard. Denote L'_+ and L'_ω the least, resp. greatest solution of the above system which exist by Knaster-Tarski.

It is not hard to see that L_+ and L_ω *are solutions* which implies $L'_+ \subseteq L_+$ and $L_\omega \subseteq L'_\omega$ (pointwise). To show that $L_+ \subseteq L'_+$ we show that $w \in L_+(e) \Rightarrow w \in L'_+(e)$ by induction on the sum of the sizes of the derivations that appear in the hypothesis $w \in L_+(e)$, and case analysis on e. Thus, $L_+ = L'_+$. Finally, if $w \in L'_\omega(e)$ then we can produce a sequence of triples (e_i, u_i, w_i) where e_i is a program, $u_i \in \Sigma^+$ and $w_i \in L'_\omega(e_i)$, with $e \xrightarrow{u_i}_* e_i$ and $w = u_i w_i$, which establishes that $w \in L_\omega(e)$. For example, if $e = \text{fix } e'\ \vec{e}$ then w must be of the form $\checkmark w_1$ and we can put $e_1 = e'\ (\text{fix } e)\ \vec{e}$ and $u_1 = \checkmark$. This case covers everything for if $L'_\omega(e) \neq \varnothing$, there exists $u \in \Sigma^+$ such that $e \xrightarrow{u}_* \text{fix } e'\ \vec{e}$ for some $e', \vec{e}$. □

This characterisation suffers from the defect that lambda-abstraction is modelled by mimicking beta-reduction on the semantic level rather than using mathematical functions. In order to achieve the latter we can organise $\mathfrak{L}_+$ and $\mathfrak{L}_\omega$ into a cartesian-closed category, i.e., a denotational model structure for the simply-typed lambda calculus.

For partially ordered sets (posets) A, B we denote by $A \times B$ their cartesian product with the componentwise ordering and we denote $A + B$ their disjoint union with canonical injections inl and inr. We denote B^A or $A \Rightarrow B$ the poset of monotone functions from A to B ordered pointwise. Either notation may be used depending on readability. If A, B are complete lattices so is $A \times B$. If B is a complete lattice then B^A is a complete lattice even if A is not. The category of posets with monotone maps is a bicartesian closed category with products, coproducts and exponentials given by the above. We also use the notation P^{op} for the opposite of P, i.e., the poset P with the ordering reversed. We tend to treat the standard isomorphisms $A \times (B \times C) \simeq (A \times B) \times C$ and $C^{A \times B} \simeq (C^B)^A$ (currying) and $C^{A+B} \simeq C^A \times C^B$ as identities to simplify notation.

We use the symbols $\sqsubseteq, \sqcap, \sqcup$ to refer to the generic structure of partial orders and complete lattices.

Definition. The category **GFP** has for objects pairs $A = (A_+, A_\omega)$ of complete lattices. A morphism from A to B is a pair of monotone functions $f = (f_+, f_\omega)$ where $f_+ : A_+ \to B_+$ and $f_\omega : A_+ \times A_\omega \to B_\omega$.

Composition of f and g is given by $h = gf$ where $h_+(a_+) = g_+(f_+(a_+))$ and $h_\omega(a_+, a_\omega) = g_\omega(f_+(a_+), f_\omega(a_+, a_\omega))$. Notice that a_+, a_ω are just decorated variables here.

Cartesian products and function spaces are given by $(A \times B)_{+/\omega} = A_{+/\omega} \times B_{+/\omega}$ and $(A \Rightarrow B)_+ = B_+^{A_+}$ and $(A \Rightarrow B)_\omega = B_\omega^{A_+ \times A_\omega}$.

The following is a direct consequence of the folklore fact that the category of complete lattices and monotone functions is cartesian-closed which in itself follows by straightforward calculation.

Lemma 1. *The category **GFP** is indeed a cartesian-closed category.*

Definition. For every object A in **GFP** we define the fixpoint combinator as the following morphism:

$$\text{fix}_A : (A \Rightarrow A) \to A$$

where $(\text{fix}_A)_+(f_+) = \text{lfp}(f_+)$ and $(\text{fix}_A)_\omega(f_+, f_\omega) = \text{gfp}(\lambda a_\omega.\, f_\omega(\text{lfp}(f_+), a_\omega))$. Herein lfp and gfp denote the least, resp. greatest fixpoint of a monotone map.

It is easy to see that the fixpoint combinator indeed yields fixpoints in the sense that $f(\text{fix}_A(f)) = \text{fix}_A(f)$ holds in the internal language of **GFP**. Formally:

$$\text{app} \circ \langle \text{id}_{A \Rightarrow A}, \text{fix}_A \rangle = \text{fix}_A : (A \Rightarrow A) \to A$$

where $\text{app} : (A \Rightarrow A) \times A \to A$ is the application map. We also recall that the least / greatest fixpoints of a monotone map $f : A \to A$ between complete lattices are given by $\text{lfp}(f) = \bigsqcup_{x : f(x) \sqsubseteq x} x$ and $\text{gfp}(f) = \bigsqcap_{x : x \sqsubseteq f(x)} x$, respectively.

If the lattice A is finite we can compute fixpoints more efficiently by iteration: $\mathrm{lfp}(f) = f^k(\bot)$ where k is the least number so that $f^k(\bot) = f^{k+1}(\bot)$. Likewise, $\mathrm{gfp}(f) = f^k(\top)$ where k is the least number so that $f^k(\top) = f^{k+1}(\top)$. Here $\top$ and $\bot$ refer to the least and greatest elements.

We interpret the base type as the object $[\![\mathbf{comm}]\!]^{\mathbf{GFP}} = (\mathfrak{L}_+, \mathfrak{L}_\omega)$ and a base term $a \in \Sigma$ as the morphism $[\![a]\!]^{\mathbf{GFP}} : 1 \to [\![\mathbf{comm}]\!]^{\mathbf{GFP}}$ given by $(\{a\}, \varnothing)$. As usual, 1 stands for the terminal object here. Nondeterministic choice and sequential composition are interpreted as union and concatenation:

$$\begin{aligned}
[\![+]\!]^{\mathbf{GFP}}_+(X_+, Y_+) &= X_+ \cup Y_+ \\
[\![+]\!]^{\mathbf{GFP}}_\omega(X_+, Y_+, X_\omega, Y_\omega) &= X_\omega \cup Y_\omega \\
[\![\,;]\!]^{\mathbf{GFP}}_+(X_+, Y_+) &= X_+ Y_+ \\
[\![\,;]\!]^{\mathbf{GFP}}_\omega(X_+, Y_+, X_\omega, Y_\omega) &= X_\omega \cup X_+ Y_\omega
\end{aligned}$$

Notice that $[\![+]\!]^{\mathbf{GFP}}, [\![\,;]\!]^{\mathbf{GFP}} : [\![\mathbf{comm}]\!]^{\mathbf{GFP}} \times [\![\mathbf{comm}]\!]^{\mathbf{GFP}} \to [\![\mathbf{comm}]\!]^{\mathbf{GFP}}$.

Using the cartesian-closed structure for abstraction and application and the fixpoint operator for recursion this then extends to an interpretation of all types $[\![\tau]\!]^{\mathbf{GFP}}$, contexts $[\![\Gamma]\!]^{\mathbf{GFP}} = \prod_{x \in dom(\Gamma)} [\![\Gamma(x)]\!]^{\mathbf{GFP}}$ and finally terms $\Gamma \vdash t : \tau$ as morphisms $[\![t]\!]^{\mathbf{GFP}} : [\![\Gamma]\!]^{\mathbf{GFP}} \to [\![\tau]\!]^{\mathbf{GFP}}$.

We give the semantic clause for fixpoints explicitly, since it requires the somewhat artificial insertion of a $\checkmark$-symbol:

For each type τ we define a morphism $\checkmark_\tau : [\![\tau]\!]^{\mathbf{GFP}} \to [\![\tau]\!]^{\mathbf{GFP}}$ by

$$\begin{aligned}
\checkmark_{\mathbf{comm},+}(X_+) &= \checkmark X_+ \\
\checkmark_{\mathbf{comm},\omega}(X_+, X_\omega) &= \checkmark X_\omega \\
\checkmark_{\tau \to \tau'} &= [\![\tau]\!]^{\mathbf{GFP}} \Rightarrow \checkmark_{\tau'}
\end{aligned}$$

The right hand side of the last clause refers to postcomposition with $\checkmark_{\tau'}$.

Now, the semantic clause for recursion becomes

$$[\![\mathrm{fix}_\tau]\!]^{\mathbf{GFP}} = \mathrm{fix}_{[\![\tau]\!]^{\mathbf{GFP}}} \circ \checkmark_{\tau \to \tau}$$

We can show that the interpretation in **GFP** coincides with the operational semantics at ground types.

Theorem 1. *Let e be a program and $[\![e]\!]^{\mathbf{GFP}} = (L_+, L_\omega)$ be its interpretation in **GFP** where $L_+ \in \mathfrak{L}_+$ and $L_\omega \in \mathfrak{L}_\omega$. We have $L_+ = L_+(e)$ and $L_\omega = L_\omega(e)$.*

Proof. The proof follows the pattern set out by Plotkin [23]: one direction uses the fact that all the steps of the operational semantics are identities at the level of the denotational semantics. The other direction uses a logical relation and approximations to the fixpoints.

The directions $L_+(e) \subseteq L_+$ and $L_\omega(e) \supseteq L_\omega$ follow from the fact that the sets $[\![e]\!]^{\mathbf{GFP}}_{+/\omega}$ for e closed and of ground type satisfy the system of equations that $L_{+/\omega}(e)$ is the least/greatest fixpoint of. For example, we have

$$[\![\mathrm{fix}\ e\ \vec{e}]\!]^{\mathbf{GFP}}_\omega = \checkmark [\![e\ (\mathrm{fix}\ e)\ \vec{e}]\!]^{\mathbf{GFP}}_\omega$$

For the converse we use a logical relation as announced. Let $\mathrm{Tm}(\tau)$ stand for the set of closed terms of type τ and let $\mathrm{Gl}(\tau)$ stand for the set of global elements of $[\![\tau]\!]^{\mathbf{GFP}}$, i.e., for pairs of elements $x_+ \in X_+, x_\omega \in X_\omega$ when $[\![\tau]\!]^{\mathbf{GFP}} = (X_+, X_\omega)$. Note that, formally, a global element is a morphism from the terminal object to $[\![\tau]\!]^{\mathbf{GFP}}$.

For each type τ we define a relation $\sim_\tau \subseteq \mathrm{Tm}(\tau) \times \mathrm{Gl}(\tau)$ by

- $e \sim_{\mathbf{comm}} (v_+, v_\omega)$ if $L_+(e) \supseteq v_+$ and $L_\omega(e) \subseteq v_\omega$.
- $e \sim_{\tau \to \tau'} (f_+, f_\omega)$ if whenever $e' \sim_\tau (v_+, v_\omega)$ then $e\ e' \sim_{\tau'} (f_+(v_+), f_\omega(v_+, v_\omega))$

The following is shown by induction on types:

Lemma 2. *If $e \sim_\tau (v_+, v_\omega)$ and $v'_+ \sqsubseteq v_+$ and $v'_\omega \sqsupseteq v_\omega$ then $e \sim_\tau (v'_+, v'_\omega)$, too.*

If $(v_{+,i})_{i \geq 0}$ and $(v_{\omega,i})_{i \geq 0}$ form an ascending, resp. descending chain in $[\![\tau]\!]^{\mathbf{GFP}}_{+/\omega}$ and $e \sim_\tau (v_{+,i}, v_{\omega,i})$ holds for all i then $e \sim_\tau (\bigsqcup_i v_{+,i}, \bigsqcap_i v_{\omega,i})$, too.

For any closed term $e \in \mathrm{Tm}(\tau)$ we have $e \sim_\tau (\bot, \top)$.

Now, suppose that $\Gamma \vdash e : \tau$ and that $\eta(x) \in \mathrm{Tm}(\Gamma(x))$ and $\rho(x) \in \mathrm{Gl}(\Gamma(x))$ and $\eta(x) \sim_{\Gamma(x)} \rho(x)$ for each $x \in dom(\Gamma)$. We show by induction on typing derivations that $e[\eta] \sim_\tau [\![e]\!]^{\mathbf{GFP}} \circ \rho$. The Theorem is a special case of this.

The cases of basic terms including $+$ and ; are direct from the semantic definitions which mimic the syntactic evaluation rules. The cases of abstraction and application follow from the definition of $\sim$ as a logical relation. The only interesting case is thus recursion: here we have to show $\mathrm{fix}_\tau \sim_{(\tau\to\tau)\to\tau} [\![\mathrm{fix}_\tau]\!]^{\mathbf{GFP}}$ holds for all τ.

Thus, assume that $\vdash e : \tau \to \tau$ and $v_{+/\omega} \in [\![\tau \to \tau]\!]^{\mathbf{GFP}}_{+/\omega}$ and $e \sim_\tau (v_+, v_\omega)$. We define an ascending chain $(v_{+,i})_i$ in $[\![\tau]\!]^{\mathbf{GFP}}_+$ and a descending chain $(v_{\omega,i})_i$ in $[\![\tau]\!]^{\mathbf{GFP}}_\omega$ inductively as follows.

- $v_{+,0} = \bot$ and $v_{\omega,0} = \top$
- $v_{i+1,+} = \checkmark v_+(v_{i,+})$ and $v_{\infty,+} = \bigsqcup_i v_{i,+} = [\![\mathrm{fix}]\!]^{\mathbf{GFP}}_+(v_+)$.
- $v_{i+1,\omega} = \checkmark v_\omega(v_{\infty,+}, v_{i,\omega})$ and $v_{\infty,\omega} = \bigsqcap_i v_{i,\omega} = [\![\mathrm{fix}]\!]^{\mathbf{GFP}}_\omega(v_+, v_\omega)$

Induction on i using the Lemma now shows that for all i it holds that fix $e \sim_\tau (v_{i,+}, v_{i,\omega})$ and thus fix $e \sim_\tau (v_{\infty,+}, v_{\infty,\omega})$.

But $[\![\mathrm{fix}_\tau]\!]^{\mathbf{GFP}}(v_+, v_\omega) = (v_{\infty,+}, v_{\infty,\omega})$, so we are done. □

V. Büchi algebras

In this section we introduce an ordered version of ω-semigroups which encompasses both the standard language-theoretic model, i.e., $\mathfrak{L}_+, \mathfrak{L}_\omega$, and finite abstractions thereof based on Büchi automata as described in [14]. Since, in contrast to loc.cit., we work with nonempty finite words and properly infinite words here, we can use the framework of ω-semigroups introduced by Perrin and Pin [22], [21] out of the box.

Definition (ω-semigroup)**.** An ω-semigroup is a two-sorted algebra $\mathfrak{V} = (\mathfrak{V}_+, \mathfrak{V}_\omega)$ equipped with the following operations:

- a binary operation on $\mathfrak{V}_+$ written multiplicatively

- a mapping $\mathfrak{V}_+ \times \mathfrak{V}_\omega \to \mathfrak{V}_\omega$ called *mixed product* and also written multiplicatively
- a mapping $\pi : \mathfrak{V}_+^{\mathbb{N}} \to \mathfrak{V}_\omega$ called *infinite product*

such that

- $\mathfrak{V}_+$ with the binary operation is a semigroup, i.e., the binary operation is associative
- for each $s, t \in \mathfrak{V}_+$ and $u \in \mathfrak{V}_\omega$, $s(tu) = (st)u$,
- for every increasing sequence $(k_n)_n \in \mathbb{N}^{\mathbb{N}}$ and $(s_n)_n \in \mathfrak{V}_+^{\mathbb{N}}$ one has $\pi((s_n)_n) = \pi((t_n)_n)$ where $t_0 = s_0 s_1 \dots s_{k_0}$ and $t_{n+1} = s_{k_n+1} \dots s_{k_{n+1}}$,
- $s \cdot \pi(s_0, s_1, s_2, \dots) = \pi(s, s_0, s_1, s_2, \dots)$

Given ω-semigroups $\mathfrak{V} = (\mathfrak{V}_+, \mathfrak{V}_\omega)$ and $\mathfrak{V}' = (\mathfrak{V}'_+, \mathfrak{V}'_\omega)$ a morphism f from $\mathfrak{V}$ to $\mathfrak{V}'$ consists of two functions $f_+ : \mathfrak{V}_+ \to \mathfrak{V}'_+$ and $f_\omega : \mathfrak{V}_\omega \to \mathfrak{V}'_\omega$ which are compatible with the algebraic structure, i.e., $f_+(st) = f_+(s)f_+(t)$ and $f_+(s)f_\omega(u) = f_\omega(su)$ and $f_\omega(\pi((s_n)_n)) = \pi((f_+(s_n))_n)$ hold for appropriately sorted arguments.

A *Wilke algebra* has instead of the infinitary product operation merely a power operation $(-)^\omega : \mathfrak{M}_+ \to \mathfrak{M}_\omega$ where $a^\omega = \pi(a\ a\ a\ \dots)$.

Notice that $\mathfrak{L} = (\mathfrak{L}_+, \mathfrak{L}_\omega)$ forms an ω-semigroup with binary and infinitary concatenation *of languages* for operations, i.e., in particular, $\pi((X_n)_n) = \prod_{n=0}^{\infty} X_n$. Similarly, $(\Sigma^+, \Sigma^\omega)$ forms an ω-semigroup with (infinitary) concatenation *of words*.

Definition (Büchi algebra)**.** A *Büchi algebra* is an ω-semigroup $\mathfrak{V} = (\mathfrak{V}_+, \mathfrak{V}_\omega)$ such that both $\mathfrak{V}_+, \mathfrak{V}_\omega$ are complete lattices and the ω-semigroup operations are monotone. Moreover, we require that $0.s = 0$ holds for all $s \in \mathfrak{V}_+ \cup \mathfrak{V}_\omega$ with 0 denoting the least element of $\mathfrak{V}_+$ or $\mathfrak{V}_\omega$.

A similar structure also appears in Ésik and Kuich's work [8] under the name *bi-inductive semiring-semimodule pairs*.

Recall that a Galois insertion between complete lattices A, C consists of monotone maps $\alpha : C \to A$ and $\gamma : A \to C$ such that $\alpha(\gamma(a)) = a$ and $\gamma(\alpha(c)) \sqsupseteq c$. The lattice C typically represents concrete values whereas A represents abstract values. The function α furnishes for each concrete value its abstraction, whereas $\gamma(a)$ is the denotation (*concretisation*) of an arbitrary abstract value. The abstraction function α applied to c thus furnishes the smallest abstract value whose concretisation lies above c [7].

It is well-known and easy to see that $\alpha(\bigsqcup_{i \in I} c_i) = \bigsqcup_{i \in I} \alpha(c_i)$. Furthermore, if $f : C \to C$ and $g : A \to A$ are monotone maps satisfying $\alpha f = g \alpha$ then $\alpha(\mathrm{lfp}(f)) = \mathrm{lfp}(g)$.

A *Galois insertion* between Büchi algebras $(\mathfrak{V}_+, \mathfrak{V}_\omega)$ and $(\mathfrak{V}'_+, \mathfrak{V}'_\omega)$ consists of a pair of Galois insertions α_+, γ_+ and $\alpha_\omega, \gamma_\omega$ between the underlying complete lattices such that α_+, α_ω form a morphism between ω-semigroups.

Notice that γ_+, γ_ω are not required to be morphisms.

Definition (Büchi abstraction)**.** A *Büchi abstraction* is a finite Büchi algebra (both lattices finite) related to the Büchi algebra $(\mathfrak{L}_+, \mathfrak{L}_\omega)$ via a Galois insertion.

In other words, a Büchi abstraction is a finite ω-semigroup carrying a lattice structure and related to the ω-semigroup $(\mathfrak{L}_+, \mathfrak{L}_\omega)$ via a structure-preserving Galois insertion.

For technical reasons we need to extend the semigroup component of a Büchi algebra to a monoid.

Definition (monoid completion)**.** Let $\mathfrak{V} = (\mathfrak{V}_+, \mathfrak{V}_\omega)$ be a Büchi algebra. We define the complete lattice $\mathfrak{V}_* = \mathfrak{V}_+ \times 2$ where $2 = \{0, 1\}$ is the Sierpinski lattice, i.e., such that $0 < 1$. The intuition is that $(x, 0)$ represents $x \in \mathfrak{V}_+ \subseteq \mathfrak{V}_*$, whereas $(x, 1)$ represents x augmented with the empty word.

Motivated by this intuition, we extend the multiplication operations as follows:

$$\begin{array}{lcl}
(x,0)(y,0) & = & (xy, 0) \\
(x,0)(y,1) & = & (xy \sqcup x, 0) \\
(x,1)(y,0) & = & (xy \sqcup y, 0) \\
(x,1)(y,1) & = & (xy \sqcup x \sqcup y, 1) \\
(x,0)w & = & xw \\
(x,1)w & = & xw \sqcup w
\end{array}$$

We consider $\mathfrak{V}_+$ a subset of $\mathfrak{V}_*$ via the embedding $x \mapsto (x, 0)$.

Notice that if we write $\varepsilon := (0, 1)$ then $\varepsilon x = x = x\varepsilon$ and $\varepsilon w = w$. Moreover, $(x, 1) = x \sqcup \varepsilon$. Also, $\mathfrak{V}_*$ is a monoid extending the semigroup $\mathfrak{V}_+$. The infinitary product, however, remains reserved for sequences in $\mathfrak{V}_+$ although it could be formally extended to sequences in $\mathfrak{V}_*$ with infinitely many elements from $\mathfrak{V}_+$.

For example, $\mathfrak{L}_*$ is (isomorphic to) the set of languages of possibly empty finite words. The element $\varepsilon = (\varnothing, 1)$ corresponds to the language containing only the empty word.

Notation. If ϕ is a predicate, we write $[\phi]$ for 0 if ϕ is false and ε if ϕ is true.

VI. Finite abstractions from Büchi automata

The following subsection essentially recapitulates results from [14] albeit with different notation and also with the distinction that we now work with finite, nonempty words and properly infinite words as opposed to finite words and at most infinite words. We thus mostly state results or give intuitions. For more detailed proofs the reader may consult loc.cit. We also remark that the parts concerning combinatorics of finite and infinite words and semigroups are already contained in Büchi's work and in [6]. What was new in [14] was the link to Galois insertions and hence abstract interpretation.

Let $\mathbf{A} = (\Sigma, Q, \delta, q_I, F)$ be a nondeterministic Büchi automaton (NBA) over our alphabet Σ.

The data of an NBA are the same as for finite automata, i.e., Q is a finite set of states, $q_I \in Q$ is the initial state, $F \subseteq Q$ is the set of final states and $\delta \subseteq Q \times \Sigma \times Q$ is the transition relation.

For a non-empty finite word $u \in \Sigma^+$ and states q, q' of $\mathbf{A}$ write $u : q \to q'$ to mean that q' can be reached from q while consuming u and write $u : q \to_\omega q'$ to mean that q' can be reached from q while consuming u and in such a way that a final state is visited on the way.

We define $L_+(\mathbf{A}) = \{u \in \Sigma^+ \mid \exists q \in F.\ u : q_I \to q\}$ thus running $\mathbf{A}$ like an NFA. One then defines, following Büchi, $L_\omega(\mathbf{A})$ as the set of all words $w \in L_\omega$ that admit a decomposition $w = uv_1v_2v_3\dots$ such that $u : q_I \to q_1$ and $v_i : q_i \to_\omega q_{i+1}$ holds for some sequence $(q_i)_i$ of states. Notice that w.l.o.g. this sequence may be assumed constant.

Define $V_+(u) = \{(q, q') \mid u : q \to q'\}$ and $V_\omega(u) = \{(q, q') \mid u : q \to_\omega q'\}$. Notice that $V_+(uv) = V_+(u)V_+(v)$ and $V_\omega(uv) = V_+(u)V_\omega(v) \cup V_\omega(u)V_+(v)$ where juxtaposition is relational composition.

Now define $u \sim v$ if $V_+(u) = V_+(v)$ and $V_\omega(u) = V_\omega(v)$. Clearly, $\sim$ is an equivalence relation on Σ^+, which is moreover of finite index.

If $[u], [u']$ are $\sim$-classes represented by u, u' we put $[u] \cdot [u'] = [uu']$ which is well-defined.

We now construct a Büchi abstraction $(\mathfrak{M}_+, \mathfrak{M}_\omega)$ (and α, γ), depending of course on the NBA $\mathbf{A}$, as follows. Its finitary part $\mathfrak{M}_+$ contains sets of $\sim$-classes and we put

$$\gamma_+(X) = \bigcup_{U \in X} U \qquad \alpha_+(L) = \{U \mid U \cap L \neq \varnothing\}$$

Also, $X \cdot Y = \{U \cdot V \mid U \in X, V \in Y\}$. It is easy to see that α_+, γ_+ form a structure-preserving Galois insertion between the semigroups $\mathfrak{M}_+$ and $\mathfrak{L}_+$.

A *patch* is a pair (U, V) where U and V are classes. A patch (U, V) represents the set $\gamma_\omega(U, V) = UV^\omega \in \mathfrak{L}_\omega$. We extend γ_ω to sets of patches by $\gamma_\omega(Y) = \bigcup_{(U,V) \in Y} \gamma_\omega(U, V)$. Such a set of patches Y is closed if whenever $\gamma_\omega(U, V) \cap \gamma_\omega(Y) \neq \varnothing$ then $(U, V) \in Y$. We write $Y^\dagger$ for the least closed set containing Y and define the infinitary part $\mathfrak{M}_\omega$ of the Büchi abstraction as the closed sets of patches ordered by inclusion. Concatenation is defined on patches by $U \cdot (U', V) = (UU', V)$ and then extended to $\mathfrak{M}_\omega$ elementwise followed by taking a closure. If $L \subseteq \Sigma^\omega$ we define $\alpha_\omega(L) = \{(U, V) \mid UV^\omega \cap L \neq \varnothing\}^\dagger$. If $X_i \in \mathfrak{M}_+$ we define

$$\pi((X_i)_i) = \alpha_\omega(\prod_{i=0}^{\infty} \gamma_+(X_i))$$

Proposition 2. *Let $L_i \subseteq \Sigma^+$ be a family of languages. We have*

$$\alpha_\omega(\prod_{i=0}^{\infty} L_i) = \pi((\alpha_+(L_i))_i)$$

Proof. We have to show

$$\alpha_\omega(\prod_{i=0}^{\infty} L_i) = \alpha_\omega(\prod_{i=0}^{\infty} \gamma_+(\alpha_+(L_i)))$$

The direction $\subseteq$ follows from monotonicity of α_ω and of the infinite product, and the fact that $L_i \subseteq \gamma_+(\alpha_+(L_i))$. For the other direction pick a patch (U, V) in the RHS. Performing induction on the closure process, we can assume w.l.o.g. that $UV^\omega \cap \prod_{i=0}^\infty \gamma_+(\alpha_+(L_i)) \neq \varnothing$, so we can find $w = v_0v_1v_2\dots$ such that $[v_i] \cap L_i \neq \varnothing$ for each i and $w \in UV^\omega$. For $i < j$ define $c(i, j) = [v_i \dots v_{j-1}]_\sim$. By Ramsey's theorem there exists an infinite sequence $i_0 < i_1 < i_2 < \dots$ and a $\sim$-class V' such that $c(i_j, i_k) = V'$ for all $j < k$. Putting $U' = [v_0 \dots v_{i_0-1}]_\sim$ we then have $w \in U'V'^\omega$. Now, pick $v'_i \in L_i \cap [v_i]$, hence $v'_i \sim v_i$ and $v'_i \in L_i$. Thus, $w' = v'_0v'_1 \dots \in \prod_{i=0}^\infty L_i$. But now $v'_{i_j} \dots v'_{i_{j+1}-1} \in V'$ and $v'_0 \dots v'_{i_0-1} \in U'$ so that $U'V'^\omega \cap \prod_{i=0}^\infty L_i \neq \varnothing$. Thus, $(U', V') \in \alpha_\omega(\prod_{i=0}^\infty L_i)$ and, since $w \in UV^\omega \cap U'V'^\omega$ we also have $(U, V) \in \alpha_\omega(\prod_{i=0}^\infty L_i)$ by closure, as required. $\square$

We notice that, in particular, we have an operation $(-)^\omega : \mathfrak{M}_+ \to \mathfrak{M}_\omega$ such that $\alpha(L^\omega) = \alpha(L)^\omega$ and we used this operation in [14] in order to compute $\alpha(L_+(e))$ and $\alpha(L_\omega(e))$ for a *first-order* term e.

We have that $\gamma_{+/\omega}(\alpha_{+/\omega}(L_{+/\omega}(\mathbf{A}))) = L_{+/\omega}(\mathbf{A})$ and also $\gamma_{+/\omega}(\alpha_{+/\omega}(\overline{L_{+/\omega}(\mathbf{A})})) = \overline{L_{+/\omega}(\mathbf{A})}$. As a result for all $L_{+/\omega} \subseteq \Sigma^{+/\omega}$ we have $L_{+/\omega} \subseteq L_{+/\omega}(\mathbf{A})$ iff $\alpha(L_{+/\omega}) \subseteq \alpha(L_{+/\omega}(\mathbf{A}))$, and likewise for $\overline{L_{+/\omega}(\mathbf{A})}$. So, the abstraction loses no precision if we are interested in adherence to policies specified by $\mathbf{A}$ or its complement.

We remark that the Büchi algebra $(\mathfrak{M}_+, \mathfrak{M}_\omega)$ is *not* the same as the ω-semigroup constructed from an NBA in [21], even if we disregard the lattice structure. Indeed, in our notation the ω-semigroup defined there is $(\mathfrak{P}_+, \mathfrak{P}_\omega)$ with $\mathfrak{P}_+ = \Sigma^+/\sim$ and $\mathfrak{P}_\omega = \mathcal{P}(Q)$, i.e., sets of states of $\mathbf{A}$. One then puts $[u]_\sim[v]_\sim = [uv]_\sim$ and $[u]_\sim H = \{q \mid \exists q'.u : q \to q' \land q' \in H\}$ and $[v_0]_\sim[v_1]_\sim[v_2]_\sim \dots$ as the set of states q such that there exists a successful run of $\mathbf{A}$ on $v_0v_1 \dots$ starting from q rather than from q_I.

We have a surjective morphism $f : (\Sigma^+, \Sigma^\omega) \to \mathfrak{P}$ given by $f_+(u) = [u]_\sim$ and $f_\omega(a_0a_1 \dots) = [a_0]_\sim[a_1]_\sim \dots$. It is clear that there are subsets $U_{+/\omega} \subseteq \mathfrak{P}_{+/\omega}$ such that $w \in L_{+/\omega}(\mathbf{A})$ iff $f_{+/\omega}(w) \in U_{+/\omega}$ thus $\mathfrak{P}$ "recognizes" $L_{+/\omega}(\mathbf{A})$ in the sense of [21]. Our Büchi algebra $\mathfrak{M}$ is a kind of power structure of Perrin and Pin's $\mathfrak{P}$ in the sense that it "recognizes" languages rather than words: we have *elements* $U_{+/\omega} \in \mathfrak{M}_{+/\omega}$ such that $L \subseteq L_{+/\omega}(\mathbf{A})$ iff $\alpha_{+/\omega}(L) \subseteq U_{+/\omega}$, namely $U_{+/\omega} = \alpha(L_{+/\omega}(\mathbf{A}))$.

VII. Affine denotational semantics

The crucial idea in [14] was that the particular greatest fixpoints needed in the definition $L_\omega(e)$ can—in the case of a first-order term e always be expressed in terms of concatenation, union, and ω-powers and can thus be abstracted. We notice that in general α does not preserve greatest fixpoints (as is well-known it does preserve least fixpoints). See loc.cit. for a concrete counterexample.

The rest of this paper is devoted to showing that the greatest fixpoints needed in the definition of $L_\omega(e)$ for e of arbitrary order can be expressed as infinite products and are thus also amenable to abstraction by the above proposition.

Our plan is to extend a given Büchi abstraction to all types including higher-order functions. The higher-order functions per se are unproblematic; given two interpretations $[\![\tau]\!]_1^{\mathbf{GFP}}$ and $[\![\tau]\!]_2^{\mathbf{GFP}}$, e.g., one built on $(\mathfrak{L}_+, \mathfrak{L}_\omega)$ (the **GFP**-semantics) and another built in an analogous fashion upon $(\mathfrak{M}_+, \mathfrak{M}_\omega)$ then we can lift α, γ to all types by putting $\alpha_+(f_+)(x_+) = \alpha_+(f_+(\gamma_+(x_+)))$ and $\alpha_\omega(f_\omega)(x_+, x_\omega) = \alpha_\omega(f_\omega(\gamma_+(x_+), \gamma_\omega(x_\omega))))$ and similarly for γ_+, γ_ω.

As in the first-order case the difficulty lies in the use of greatest fixpoints in the semantics of recursion. We thus have to characterise the actually occurring greatest fixpoints in terms of constructions that may be abstracted, in particular infinite products. This is what we shall do in this section.

Given a Büchi algebra $\mathfrak{V} = (\mathfrak{V}_+, \mathfrak{V}_\omega)$ we construct a cartesian-closed category $\mathbf{AFF}_{\mathfrak{V}}$ which admits a fixpoint combinator defined in terms of least fixpoints and infinite products rather than general greatest fixpoints.

The crucial idea behind this construction is the fact that the functions denoted by $\lambda\mathbf{Y}$-terms are *affine*, in the sense that once an infinitary computation is called, it cannot be stopped. Some intuitive explanations behind this idea are given in the introduction; the reader might want to keep them in mind while reading the rest of this section.

We will then establish logical relations between **GFP** and $\mathbf{AFF}_{\mathfrak{L}}$ on the one hand and between $\mathbf{AFF}_{\mathfrak{L}}$ and $\mathbf{AFF}_{\mathfrak{M}}$ with $\mathfrak{M}$ the finite Büchi algebra induced by our policy automaton $\mathbf{A}$. As a result, we will be able to determine whether or not $L_{+/\omega}(e) \subseteq L_{+/\omega}(\mathbf{A})$ or $L_{+/\omega}(e) \cap L_{+/\omega}(\mathbf{A}) = \varnothing$ solely on the basis of the interpretation of e in the category $\mathbf{AFF}_{\mathfrak{M}}$ which only involves finite lattices and thus can be done effectively by tabulation and finite fixpoint iteration.

A note about our terminology. Traditionally, an "affine function" is a function f of the form $f(x) = a + bx$. What we call an affine map below is actually the data (a, b) of its coefficients, which we call "constant" and "prefix" since our product is concatenation. We then define the *extension* $\mathrm{ext}(a, b)$ of an affine map, which is the actual function represented by these coefficients.

Definition (Affine polynomial)**.** Let $\mathfrak{V} = (\mathfrak{V}_+, \mathfrak{V}_\omega)$ be a Büchi algebra and P be a poset. An *affine polynomial* over $\mathfrak{V}$ and indexed by P is a pair (U, C) where $U \in \mathfrak{V}_\omega$ and $C : P^{\mathrm{op}} \to \mathfrak{V}_*$ is a monotone function. It induces a monotone function $\mathrm{ext}(U, C) : \mathfrak{V}_\omega^P \to \mathfrak{V}_\omega$ by

$$\mathrm{ext}(U, C)(X) = U \sqcup \bigsqcup_{p \in P} C(p) X(p)$$

The affine polynomials over $\mathfrak{V}$ and indexed by P form a complete lattice with the component- and pointwise ordering which we call $\mathrm{AfP}_{\mathfrak{V}}(P)$.

Affine polynomials thus represent certain monotone functions on $\mathfrak{V}_\omega^P$ and have been introduced for that purpose. Under additional assumptions on absence of redundancies one can ensure that each monotone function is represented by at most one affine polynomial. Since these assumptions would, however, further complicate notations and definitions we accept the fact that affine polynomials carry some intensional information.

The use of the opposite poset restricts the number of possible affine polynomials without sacrificing any generality. Suppose that $C : |P| \to \mathfrak{V}_*$ is an arbitrary function (and $U \in \mathfrak{V}_\omega$). We then have, by monotonicity of X,

$$\begin{aligned}\mathrm{ext}(U, C)(X) &= U \sqcup \bigsqcup_{p \in P} C(p) X(p) \\ &= U \sqcup \bigsqcup_{p \in P} \bigsqcup_{p' : p' \sqsupseteq p} C(p') X(p) \\ &= \mathrm{ext}(U, C')(X)\end{aligned}$$

where $C'(p) = \bigsqcup_{p' \sqsupseteq p} C(p')$ and $C' : P^{\mathrm{op}} \to \mathfrak{V}_*$.

Definition (Affine Map)**.** Let $\mathfrak{V} = (\mathfrak{V}_+, \mathfrak{V}_\omega)$ be a Büchi algebra and P_1, P_2 be posets. An *affine map* from $\mathfrak{V}_\omega^{P_1}$ to $\mathfrak{V}_\omega^{P_2}$ (over $\mathfrak{V}$) is a monotone function $f : P_2 \to \mathrm{AfP}_{\mathfrak{V}}(P_1)$. We write $f \in \mathrm{AfM}(\mathfrak{V}_\omega^{P_1}, \mathfrak{V}_\omega^{P_2})$ in this case. We are aware of the fact that the notation and nomenclature are slightly inaccurate because P_1 might not be uniquely determined from $\mathfrak{V}_\omega^{P_1}$.

For $f \in \mathrm{AfM}(\mathfrak{V}_\omega^{P_1}, \mathfrak{V}_\omega^{P_2})$, we define $\mathrm{ext}(f) : \mathfrak{V}_\omega^{P_1} \to \mathfrak{V}_\omega^{P_2}$ by $\mathrm{ext}(f)(X) = \lambda p_2.\mathrm{ext}(f(p_2))(X)$.

Given $f \in \mathrm{AfM}(\mathfrak{V}_\omega^{P_1}, \mathfrak{V}_\omega^{P_2})$ and $g \in \mathrm{AfM}(\mathfrak{V}_\omega^{P_2}, \mathfrak{V}_\omega^{P_3})$, we define their composition $h = g \circ f \in \mathrm{AfM}(\mathfrak{V}_\omega^{P_1}, \mathfrak{V}_\omega^{P_3})$ as follows. The idea is to mimic composition on the extensional level, that is, we want the following relation to be satisfied:

$$\mathrm{ext}(g \circ f) = \mathrm{ext}(g) \circ \mathrm{ext}(f)$$

Write $f(p_2) = (U_{p_2}, C_{p_2})$ and $g(p_3) = (V_{p_3}, D_{p_3})$. Then put $h(p_3) = (W_{p_3}, E_{p_3})$ with

$$\begin{aligned}W_{p_3} &:= V_{p_3} \sqcup \bigsqcup_{p_2 \in P_2} D_{p_3}(p_2) \cdot U_{p_2} \\ E_{p_3}(p_1) &:= \bigsqcup_{p_2 \in P_2} D_{p_3}(p_2) \cdot C_{p_2}(p_1)\end{aligned}$$

It is clear that composition of affine maps is associative. Moreover, we have the identity affine map given by

$$\mathrm{id}_P = \lambda p.\, (0, \lambda p'.\, [p' \sqsubseteq p])$$

where $[\cdot]$ is the notation for predicates introduced previously. The above condition should be understood as meaning $[p' = p]$, but since it needs to be monotonous, we put a (harmless) inequality. This trick will be used several times later.

Notice that the affine maps over $\mathfrak{V}$ form a category $\mathbf{AfM}_{\mathfrak{V}}$ (with posets as objects), and that the assignment $P \mapsto \mathfrak{V}_\omega^P$ and $f \mapsto \mathrm{ext}(f)$ is a functor from that category to the category of complete lattices and monotone functions.

This category has cartesian products given by the disjoint union: $\mathfrak{V}_\omega^{P_1} \times \mathfrak{V}_\omega^{P_2} = \mathfrak{V}_\omega^{P_1 + P_2}$, with first projection $\mathrm{pr}_1 = \lambda p_1.\, (0, \lambda p'.\, [p' \sqsubseteq \mathrm{inl}\, p_1])$ and similarly for pr_2. Notice that $\mathrm{ext}(\mathrm{pr}_i) : \mathfrak{V}_\omega^{P_1} \times \mathfrak{V}_\omega^{P_2} \to \mathfrak{V}_\omega^{P_i}$ is the usual projection and, more generally, the functor ext preserves products.

Notation. Given $f \in \mathrm{AfM}(\mathfrak{V}_\omega^P, \mathfrak{V}_\omega^Q)$, by definition we have $f : Q \to \mathfrak{V}_\omega \times \mathfrak{V}_*^{P^{\mathrm{op}}}$. We sometimes split f in two components $f_\gamma : Q \to \mathfrak{V}_\omega$ ("constant") and $f_\pi : Q \times P^{\mathrm{op}} \to \mathfrak{V}_*$ ("prefix"). Formally, $f_\gamma(q) = f(q).1$ and $f_\pi(q, p) = f(q).2(p)$. Thus, in this notation, $\mathrm{ext}(f)(X, q) = f_\gamma(q) \sqcup \bigsqcup_{p \in P} f_\pi(q, p) \cdot X(p)$.

Here and in the sequel, we use .1 and .2 to refer to components of a set-theoretic pair. If $z = (x, y)$ then $x = z.1$ and $y = z.2$.

Definition (Fixpoint in $\mathbf{AfM}_{\mathfrak{V}}$)**.** Let $f \in \mathrm{AfM}(\mathfrak{V}_\omega^{(Q+P)}, \mathfrak{V}_\omega^P)$ (equivalently, $f : Q \times P \to P$ in $\mathbf{AfM}_{\mathfrak{V}}$). Moreover, assume that for all p, p', $f_\pi(p, \mathrm{inr}(p')) \in \mathfrak{V}_+$ (as opposed to $\mathfrak{V}_*$).

In the following, we drop the injections inl and inr for convenience: $f_\pi(p, p')$ and $f_\pi(p, q)$ denote respectively $f_\pi(p, \mathrm{inr}(p'))$ and $f_\pi(p, \mathrm{inl}(q))$.

For $p, p' \in P$ and a nonempty sequence $\vec{p} = (p_0, \dots, p_n) \in P^+$ write $\vec{p} : p \rightsquigarrow p'$ to mean that $p_0 = p$ and $p_n = p'$. Furthermore, write $f_\pi(\vec{p})$ for $f_\pi(p_0, p_1) f_\pi(p_1, p_2) \dots f_\pi(p_{n-1}, p_n)$ and $f_\pi((p)) = \varepsilon$ (case $n = 0$).

For an infinite sequence $\vec{p} = (p_0, p_1, \dots) \in P^\omega$ write $\vec{p} : p \rightsquigarrow$ to mean $p_0 = p$ and write $f_\pi(\vec{p}) = \pi((f_\pi(p_n, p_{n+1}))_{n \geq 0})$. Notice that this is well-defined since $f_\pi(p', p'') \in \mathfrak{V}_+$. Armed with this notation we define $\mathrm{fix}(f) \in \mathrm{AfM}(\mathfrak{V}_\omega^Q, \mathfrak{V}_\omega^P)$ as follows:

$$\mathrm{fix}(f)_\gamma(p) = \bigsqcup_{\substack{p' \in P, \vec{p} \in P^+ \\ \vec{p} : p \rightsquigarrow p'}} f_\pi(\vec{p}) \cdot f_\gamma(p') \quad \sqcup \quad \bigsqcup_{\substack{\vec{p} \in P^\omega \\ \vec{p} : p \rightsquigarrow}} f_\pi(\vec{p})$$

and

$$\mathrm{fix}(f)_\pi(p, q) = \bigsqcup_{\substack{p' \in P, \vec{p} \in P \\ \vec{p} : p \rightsquigarrow p'}} f_\pi(\vec{p}) \cdot f_\pi(p', q)$$

Proposition 3. *The affine map* $\mathrm{fix}(f)$ *satisfies the following fixpoint equation:*

$$\mathrm{fix}(f) = f \circ \langle \mathrm{id}_Q, \mathrm{fix}(f) \rangle$$

Proof. This follows by straightforward calculation.

The function $h = f \circ \langle \mathrm{id}_Q, \mathrm{fix}(f) \rangle$ is given by

$$\begin{aligned} h_\gamma(p) = \ & f_\gamma(p) \ \sqcup \ \cancel{\bigsqcup_{q \in Q} f_\pi(p, \mathrm{inl}\, q) \cdot \mathrm{id}_{Q\gamma}(q)} \\ & \sqcup \bigsqcup_{p \in P} f_\pi(p, \mathrm{inr}\, p') \cdot \mathrm{fix}(f)_\gamma(p') \end{aligned}$$

where the second term cancels as indicated since $\mathrm{id}_{Q\gamma}(q) = 0$, and

$$\begin{aligned} h_\pi(p, q) = & \bigsqcup_{q' \in Q} f_\pi(p, \mathrm{inl}\, q') \cdot \mathrm{id}_{Q\pi}(q', q) \\ & \sqcup \bigsqcup_{p \in P} f_\pi(p, \mathrm{inr}\, p') \cdot \mathrm{fix}(f)_\pi(p', q) \end{aligned}$$

where the first term is $f_\pi(p, \mathrm{inl}\, q)$ since $\mathrm{id}_{Q\pi}(q', q) = [q' \sqsubseteq q]$ (and by monotonicity of f_π). □

Proposition 4. $\mathrm{ext}(\mathrm{fix}(f))$ *is the greatest fixpoint of* $\mathrm{ext}(f)$*.*

Proof. Given the definition of $\mathrm{ext}(f)$ the fixpoint equation $F(Y) = \mathrm{ext}(f)(Y, F(Y))$ (in the unknown $F : \mathfrak{V}_\omega^Q \to \mathfrak{V}_\omega^P$) becomes

$$\begin{aligned} F(Y)(p) = f_\gamma(p) \ \sqcup & \bigsqcup_{q \in Q} f_\pi(p, \mathrm{inl}\, q) \cdot Y(q) \\ \sqcup & \bigsqcup_{p' \in P} f_\pi(p, \mathrm{inr}\, p') \cdot F(Y)(p') \end{aligned}$$

The greatest solution of this equation is

$$\begin{aligned} F(Y)(p) = & \bigsqcup_{\substack{p' \in P, \vec{p} \in P^+ \\ \vec{p} : p \rightsquigarrow p'}} f_\pi(\vec{p}) (f_\gamma(p') \sqcup \bigsqcup_{q \in Q} f_\pi(p', q) Y(q)) \\ & \sqcup \bigsqcup_{\substack{\vec{p} \in P^\omega \\ \vec{p} : p \rightsquigarrow}} f_\pi(\vec{p}) \end{aligned}$$

Intuitively, we can think of it as executing computationally the fixpoint equation and looping with a new argument p' whenever we reach $F(Y)(p')$. The first line corresponds to the finite computation paths, either ending with a call to f_γ or to Y. In order to get the greatest fixpoint, we also include the infinite paths which correspond to the second line.

But now, this explicit solution is precisely $\mathrm{ext}(\mathrm{fix}(f))$. □

We now amalgamate the category of affine maps with the category of complete lattices just as in the construction of **GFP**:

Definition (Category $\mathbf{AFF}_{\mathfrak{V}}$)**.** Let $\mathfrak{V} = (\mathfrak{V}_+, \mathfrak{V}_\omega)$ be a Büchi algebra. The category $\mathbf{AFF}_{\mathfrak{V}}$ has for objects pairs $X = (X_+, X_{\mathrm{arg}})$ where X_+ is a complete lattice and X_{arg} is a poset.

A morphism from $X = (X_+, X_{\mathrm{arg}})$ to $Y = (Y_+, Y_{\mathrm{arg}})$ consists of a pair of monotone functions $f_+ : X_+ \to Y_+$ and

$$f_{\mathrm{arg}} : X_+ \to \mathrm{AfM}(\mathfrak{V}_\omega^{X_{\mathrm{arg}}}, \mathfrak{V}_\omega^{Y_{\mathrm{arg}}})$$

The composition $h = g \circ f$ of maps $f : X \to Y$ and $g : Y \to Z$ is given by $h_+ = g_+ \circ f_+$ and

$$h_{\mathrm{arg}}(x_+) = g_{\mathrm{arg}}(f_+(x_+)) \circ f_{\mathrm{arg}}(x_+)$$

It is easy to see that this forms indeed a category.

Definition (Functor Ext : $\mathbf{AFF}_{\mathfrak{V}} \to$ **GFP**)**.** We extend ext to a functor Ext : $\mathbf{AFF}_{\mathfrak{V}} \to$ **GFP**. On objects, $\mathrm{Ext}(X_+, X_{\mathrm{arg}}) = (X_+, \mathfrak{V}_\omega^{X_{\mathrm{arg}}})$ and on morphisms, $\mathrm{Ext}(f_+, f_{\mathrm{arg}}) = (f_+, \mathrm{ext}(f_{\mathrm{arg}}))$, where $\mathrm{ext}(f_{\mathrm{arg}}) : X_+ \times \mathfrak{V}_\omega^{X_{\mathrm{arg}}} \to \mathfrak{V}_\omega^{Y_{\mathrm{arg}}}$ is defined as $\mathrm{ext}(f_{\mathrm{arg}})(x_+, x_\omega) = \mathrm{ext}(f_{\mathrm{arg}}(x_+))(x_\omega)$.

Thus, the functor Ext takes $\mathbf{AFF}_{\mathfrak{V}}$ to a subcategory of **GFP** such that the infinitary lattice X_ω is actually of the form $\mathfrak{V}_\omega^{X_{\mathrm{arg}}}$, and the morphisms are affine with respect to their infinitary component.

Notation. If X is an object of $\mathbf{AFF}_{\mathfrak{V}}$ we use the notation X_+ and X_{arg} to refer to its two components. We use (possibly decorated versions) of x resp. y to range over elements of X_+ resp. Y_+ and (possibly decorated versions) of ξ and η to range over X_{arg} and Y_{arg}. Similar conventions apply to other letters.

If $f : X \to Y$ and $x \in X_+$ we write $f(x)$ for $f_+(x)$. If $x \in X_+$ and $\eta \in Y_{\mathrm{arg}}$ we write $f(x, \eta)$ for $f_{\mathrm{arg}}(x, \eta)$. We also write $f_\gamma(x, \eta)$ ("constant") for $f_{\mathrm{arg}}(x, \eta).1 \in \mathfrak{V}_\omega$ and $f_\pi(x, \eta, \xi)$ ("prefix") for $f_{\mathrm{arg}}(x, \eta).2(\xi) \in \mathfrak{V}_*$.

Theorem 2 ($\mathbf{AFF}_{\mathfrak{V}}$ cartesian-closed)**.** *The category* $\mathbf{AFF}_{\mathfrak{V}}$ *for* $\mathfrak{V}$ *a Büchi algebra is cartesian closed. The product* $X \times Y$ *of*

X and Y is given by $(X\times Y)_+ = X_+\times Y_+$ and $(X\times Y)_{\mathrm{arg}} = X_{\mathrm{arg}} + Y_{\mathrm{arg}}$. The function space $X \Rightarrow Y$ is given by

$$\begin{array}{lcl}(X\Rightarrow Y)_+ &=& X_+ \Rightarrow (Y_+ \times \mathfrak{V}_*^{Y_{\mathrm{arg}}\times X_{\mathrm{arg}}^{\mathrm{op}}})\\ (X\Rightarrow Y)_{\mathrm{arg}} &=& X_+ \times Y_{\mathrm{arg}}\end{array}$$

Notice that the $\times$ symbols above refer to cartesian products of posets (not coproducts).

The canonical isomorphism

$$\boldsymbol{AFF}_{\mathfrak{V}}(Z\times X, Y) \simeq \boldsymbol{AFF}_{\mathfrak{V}}(Z, X\Rightarrow Y)$$

sends $f : Z\times X \to Y$ to $\lambda(f) : Z \to X{\Rightarrow}Y$ where

$$\begin{array}{ll}\lambda(f)_+(z) &= \lambda x.(f(z,x), \lambda(\eta,\xi).f_\pi((z,x),\eta,\mathrm{inr}\,\xi))\\ \lambda(f)_\gamma(z,(x,\eta)) &= f_\gamma((z,x),\eta)\\ \lambda(f)_\pi(z,(x,\eta),\zeta) &= f_\pi((z,x),\eta,\mathrm{inl}\,\zeta)\end{array}$$

The application morphism

$$\mathrm{app} : (X\Rightarrow Y)\times X \to Y$$

is defined as

$$\begin{array}{lcl}\mathrm{app}_+(f,x) &=& f(x).1\\ \mathrm{app}_\gamma(f,x,\eta) &=& 0\\ \mathrm{app}_\pi(f,x,\eta,\mathrm{inl}\,(x',\eta')) &=& [(x',\eta')\sqsubseteq(x,\eta)]\\ \mathrm{app}_\pi(f,x,\eta,\mathrm{inr}\,\xi) &=& f(x).2(\xi,\eta)\end{array}$$

Proof. Once the definitions are in place as they are the verifications amount to mechanical type checking and equational reasoning. □

The definition of the function space while amenable to straightforward verification may seem mysterious at first. To understand it, consider that a morphism from $Z\times X$ to Y comprises $f_+ : Z_+\times X_+ \to Y_+$ and

$$f_{\mathrm{arg}} : (Z_+\times X_+)\times Y_{\mathrm{arg}} \to \mathfrak{V}_\omega \times \mathfrak{V}_*^{(Z_{\mathrm{arg}}+X_{\mathrm{arg}})^{\mathrm{op}}}$$

Now notice that

$$(Z_+\times X_+ \to Y_+) \quad\simeq\quad (Z_+ \to Y_+^{X_+}) \tag{1}$$

and

$$\begin{array}{l}(Z_+\times X_+\times Y_{\mathrm{arg}} \to \mathfrak{V}_\omega\times\mathfrak{V}_*^{(Z_{\mathrm{arg}}+X_{\mathrm{arg}})^{\mathrm{op}}})\\ \simeq\quad (Z_+\times(X_+\times Y_{\mathrm{arg}}) \to \mathfrak{V}_\omega\times\mathfrak{V}_*^{Z_{\mathrm{arg}}^{\mathrm{op}}})\\ \qquad\times(Z_+ \to (\mathfrak{V}_*^{X_{\mathrm{arg}}^{\mathrm{op}}\times Y_{\mathrm{arg}}})^{X_+})\end{array} \tag{2}$$

The right-hand side of equation (1) together with the second component of the right-hand side of equation (2) make up $\lambda(f)_+$, whereas the first component of the right-hand side of equation (2) accounts for $\lambda(f)_{\mathrm{arg}}$.

To understand the *app* morphism, recall that we can decompose a morphism $f : X\to Y$ into three parts

$$\begin{array}{l}f_+ : X_+ \to Y_+\\ f_\gamma : X_+\times Y_{\mathrm{arg}} \to \mathfrak{V}_\omega\\ f_\pi : X_+\times Y_{\mathrm{arg}}\times X_{\mathrm{arg}}^{\mathrm{op}} \to \mathfrak{V}_*\end{array}$$

and remark that $(X\Rightarrow Y)_+ = X_+ \Rightarrow (Y_+\times\mathfrak{V}_*^{Y_{\mathrm{arg}}\times X_{\mathrm{arg}}^{\mathrm{op}}})$ is just the types of f_+ and f_π put together. Moreover, $(X\Rightarrow Y)_{\mathrm{arg}} = X_+\times Y_{\mathrm{arg}}$ is the input type of f_γ: indeed, the Ext functor sends the Z_{arg} part of an object to $\mathfrak{V}_\omega^{Z_{\mathrm{arg}}}$.

Now look at how a function $f : X\to Y$ is sent to $\lambda(f) : 1 \to (X\Rightarrow Y)$, where $1 = (\{\star\},\varnothing)$ is the terminal object.

$$\begin{array}{lcl}\lambda(f)_+(\star) &=& \lambda x\,(f_+(x), \lambda(\eta,\xi).\,f_\pi(x,\eta,\xi))\\ \lambda(f)_\gamma(\star,(x,\eta)) &=& f_\gamma(x,\eta)\\ \lambda(f)_\pi &=& \varnothing \qquad \text{(the empty function)}\end{array}$$

So in **GFP**, $\mathrm{Ext}(\lambda(f))$ gives an element of $(X\Rightarrow Y)_+$ obtained by amalgamating f_+ and f_π, and an element of $\mathfrak{V}_\omega^{(X\Rightarrow Y)_{\mathrm{arg}}}$ which is f_γ.

Notice that in $\mathbf{AFF}_{\mathfrak{V}}$, $g : (X\Rightarrow Y)\to Z$ has for second component $g_{\mathrm{arg}} : (X\Rightarrow Y)_+ \to \mathrm{AfM}(\mathfrak{V}_\omega^{(X\Rightarrow Y)_{\mathrm{arg}}}, \mathfrak{V}_\omega^{Z_{\mathrm{arg}}})$. Thus, it represents in **GFP** a map $\mathrm{ext}(g_{\mathrm{arg}}) : (X\Rightarrow Y)_+\times\mathfrak{V}_\omega^{(X\Rightarrow Y)_{\mathrm{arg}}} \to \mathfrak{V}_\omega^{Z_{\mathrm{arg}}}$, which is affine w.r.t. its second argument (which intuitively is the 'f_γ' part of the function given as argument).

Finally, here is what app does: $\mathrm{app}_+(f,x)$ just applies the f_+ part of f to x. Now, $\mathrm{ext}(f_{\mathrm{arg}}) : X_+\times\mathfrak{V}_\omega^{X_{\mathrm{arg}}} \to \mathfrak{V}_\omega^{Y_{\mathrm{arg}}}$ is the function

$$\mathrm{ext}(f_{\mathrm{arg}})(x,X,\eta) = f_\gamma(x,\eta) \sqcup \bigsqcup_{\xi\in X_{\mathrm{arg}}} f_\pi(x,\eta,\xi)\cdot X(\xi)$$

$\mathrm{app}_{\mathrm{arg}}$ gives the prefixes of that expression, and it must be affine w.r.t. both X *and* f_γ. Thus, there is no constant component; the prefix in front of f_γ is ε when the argument is (x,η) and 0 otherwise; and the prefix in front of X is given by f_π.

It may seem surprising that one does not have $(X\Rightarrow Y)_+ = X_+\Rightarrow Y_+$ as was the case with **GFP** and even more so that the unusual definition we made does indeed work. It would be interesting to know whether it is an instance of a more general construction or perhaps has been described before, but we could not find any source to that effect despite considerable search and conversations with category theory experts.

Definition (Fixpoint in $\mathbf{AFF}_{\mathfrak{V}}$)**.** Let $f : Z\times X\to X$ be such that $f_\pi((z,x),\xi,\mathrm{inr}(\xi')) \in \mathfrak{V}_+$. We define $\mathrm{fix}(f) : Z\to X$ as follows.

$$\begin{array}{c}\mathrm{fix}(f)_+(z) = \mathrm{lfp}(\lambda x.\, f_+(z,x))\\ \mathrm{fix}(f)_{\mathrm{arg}}(z) = \mathrm{fix}_{\mathbf{AfM}_{\mathfrak{V}}}(f_{\mathrm{arg}}(z,\mathrm{fix}(f)_+(z)))\end{array}$$

We would like to define a morphism $\mathrm{fix}_X : (X\Rightarrow X)\to X$ by taking $Z = (X\Rightarrow X)$ and $f = \mathrm{app}$, but app does not satisfy the condition on prefixes: app_π might return $\varepsilon\notin\mathfrak{V}_+$.

Instead, let us define an object $(X\Rightarrow_+ Y)$ for every X, Y, which is to be understood as the subspace of $(X\Rightarrow Y)$ consisting of functions which satisfy the prefix condition.

$$\begin{array}{lcl}(X\Rightarrow_+ Y)_+ &=& X_+ \to Y_+\times\mathfrak{V}_+^{Y_{\mathrm{arg}}\times X_{\mathrm{arg}}^{\mathrm{op}}}\\ (X\Rightarrow_+ Y)_{\mathrm{arg}} &=& (X\Rightarrow Y)_{\mathrm{arg}}\end{array}$$

That is, we replace $\mathfrak{V}_*$ by $\mathfrak{V}_+$ in the definition of the function space. The canonical injection $\iota : (X\Rightarrow_+ Y)\to(X\Rightarrow Y)$ is defined as $\iota_+(f) = f$ and $\iota_{\mathrm{arg}}(f) = \mathrm{id}_{(X\Rightarrow Y)_{\mathrm{arg}}}$.

Finally, we get the desired fixpoint operator $\mathrm{fix}'_X : (X \Rightarrow_+ X) \to X$ by precomposing with ι before applying:

$$\mathrm{fix}'_X = \mathrm{fix}(\mathrm{app} \circ (\iota \times \mathrm{id}_X))$$

Definition (Semantics in $\mathbf{AFF}_{\mathfrak{L}}/\mathbf{AFF}_{\mathfrak{M}}$)**.** We define an interpretation $[\![-]\!]^{\mathfrak{L}}$ of types and terms in $\mathbf{AFF}_{\mathfrak{L}}$ similarly to what we did in **GFP**, except that we have to use the affine map formalism. The interpretation $[\![-]\!]^{\mathfrak{M}}$ in $\mathbf{AFF}_{\mathfrak{M}}$ being very similar, we only write its definition when it differs.

We interpret the base type $\mathbf{comm}$ as $[\![\mathbf{comm}]\!]^{\mathfrak{L}} = (\mathfrak{L}_+, \{\star\})$. A term $a \in \Sigma$ is interpreted as the morphism $[\![a]\!]^{\mathfrak{L}} : 1 \to [\![\mathbf{comm}]\!]^{\mathfrak{L}}$ (recall that $1 = (\{\star\}, \varnothing)$) defined by $[\![a]\!]^{\mathfrak{L}}_+(\star) = \{a\}$ and $[\![a]\!]^{\mathfrak{L}}_{\mathrm{arg}}(\star) = (\varnothing, \varnothing)$. (In $\mathbf{AFF}_{\mathfrak{M}}$, we put $[\![a]\!]^{\mathfrak{M}}_+(\star) = [a]$, the equivalence class of a.)

We then define morphisms $[\![+]\!]^{\mathfrak{L}}, [\![\,;]\!]^{\mathfrak{L}} : [\![\mathbf{comm}]\!]^{\mathfrak{L}} \times [\![\mathbf{comm}]\!]^{\mathfrak{L}} \to [\![\mathbf{comm}]\!]^{\mathfrak{L}}$ as follows:

$$\begin{array}{lcl}
[\![+]\!]^{\mathfrak{L}}_+(L_1, L_2) & = & L_1 \cup L_2 \\
[\![+]\!]^{\mathfrak{L}}_{\mathrm{arg}}(L_1, L_2, \star) & = & (\varnothing, \lambda\eta.\, \varepsilon) \\
[\![\,;]\!]^{\mathfrak{L}}_+(L_1, L_2) & = & L_1 L_2 \\
[\![\,;]\!]^{\mathfrak{L}}_{\mathrm{arg}}(L_1, L_2, \star) & = & \left(\varnothing, \lambda\eta.\, \mathrm{case}(\eta) \begin{cases} \mathrm{inl}\,\star \mapsto \varepsilon \\ \mathrm{inr}\,\star \mapsto L_1 \end{cases}\right)
\end{array}$$

Abstraction and application are interpreted using the cartesian-closed structure. Finally for the fixpoint operator, we define as in **GFP** a morphism $\checkmark_\tau$ by induction on the types:

$$\begin{array}{lcl}
\checkmark_{\mathbf{comm},+}(L) & = & \checkmark \cdot L \\
\checkmark_{\mathbf{comm},\mathrm{arg}}(L) & = & (\varnothing, \lambda\eta.\, \checkmark) \\
\checkmark_{\tau \to \tau'} & = & [\![\tau]\!]^{\mathfrak{L}} \Rightarrow \checkmark_{\tau'}
\end{array}$$

(In $\mathbf{AFF}_{\mathfrak{M}}$, replace $\checkmark$ by $[\checkmark] \in \mathfrak{M}_+$.)

Notice that $\checkmark_{\tau\to\tau} : ([\![\tau]\!]^{\mathfrak{L}} \Rightarrow [\![\tau]\!]^{\mathfrak{L}}) \to ([\![\tau]\!]^{\mathfrak{L}} \Rightarrow_+ [\![\tau]\!]^{\mathfrak{L}})$. We can then define:

$$[\![\mathrm{fix}_\tau]\!]^{\mathfrak{L}} = \mathrm{fix}'_{[\![\tau]\!]} \circ \checkmark_{\tau\to\tau} : ([\![\tau]\!]^{\mathfrak{L}} \Rightarrow [\![\tau]\!]^{\mathfrak{L}}) \to [\![\tau]\!]^{\mathfrak{L}}$$

Remark. Notice that we have defined $[\![-]\!]^{\mathfrak{L}}$ so that $\mathrm{Ext}([\![\mathbf{comm}]\!]^{\mathfrak{L}}) = [\![\mathbf{comm}]\!]^{\mathbf{GFP}}$, $\mathrm{Ext}([\![a]\!]^{\mathfrak{L}}) = [\![a]\!]^{\mathbf{GFP}}$, $\mathrm{Ext}([\![+]\!]^{\mathfrak{L}}) = [\![+]\!]^{\mathbf{GFP}}$ and $\mathrm{Ext}([\![\,;]\!]^{\mathfrak{L}}) = [\![\,;]\!]^{\mathbf{GFP}}$.

However, we do not get those relations for higher-typed constants $\mathrm{fix}, \mathrm{app}$ and λ since Ext does not preserve exponentials. We therefore use a logical relation as in the proof of Theorem 1.

As before, we write $\mathrm{Gl}([\![X]\!]^{\mathbf{GFP}})$ and $\mathrm{Gl}([\![X]\!]^{\mathfrak{L}})$ for the set of global elements of X, that is, morphisms from the terminal object to X in the categories **GFP** and $\mathbf{AFF}_{\mathfrak{L}}$. Recall that in **GFP**, a global element of an object (X_+, X_ω) is a pair $(x_+, x_\omega) \in X_+ \times X_\omega$. In $\mathbf{AFF}_{\mathfrak{L}}$, a global element of (X_+, X_{arg}) is a pair $(x_+, x_\omega) \in X_+ \times {\mathfrak{L}_\omega}^{X_{\mathrm{arg}}}$. In particular, $\mathrm{Gl}([\![\mathbf{comm}]\!]^{\mathbf{GFP}})$ and $\mathrm{Gl}([\![\mathbf{comm}]\!]^{\mathfrak{L}})$ are isomorphic; their elements are (up to isomorphism) pairs $(L_+, L_\omega) \in \mathfrak{L}_+ \times \mathfrak{L}_\omega$. This isomorphism is actually the Ext functor; in the following, we treat it as an equality.

We define a relation $\sim_\tau \subseteq \mathrm{Gl}([\![\tau]\!]^{\mathbf{GFP}}) \times \mathrm{Gl}([\![\tau]\!]^{\mathfrak{L}})$ for every type τ as follows.

- $(L_+, L_\omega) \sim_{\mathbf{comm}} (L'_+, L'_\omega)$ iff $L_+ = L'_+$ and $L_\omega = L'_\omega$.
- $(f_+, f_\omega) \sim_{\tau\to\tau'} f'$ iff whenever $(x_+, x_\omega) \sim_\tau x'$, then $(f_+(x_+), f_\omega(x_+, x_\omega)) \sim_{\tau'} \mathrm{app} \circ \langle f', x'\rangle$.

Lemma 3. *Suppose there is a derivation $\Gamma \vdash e : \tau$ and two valuations $\eta(x) \in Gl([\![\Gamma(x)]\!]^{\mathbf{GFP}})$ and $\eta'(x) \in Gl([\![\Gamma(x)]\!]^{\mathfrak{L}})$ such that for every $x \in dom(\Gamma)$, $\eta(x) \sim_{\Gamma(x)} \eta'(x)$. Then $[\![e]\!]^{\mathbf{GFP}} \circ \eta \sim_\tau [\![e]\!]^{\mathfrak{L}} \circ \eta'$.*

Proof. As usual by induction on derivations. The cases of basic terms follow from the above remark about Ext. Abstraction and application follow from the fact that both categories in question are cartesian-closed. The fix-case, finally, follows from the fact that both fixpoint operators yield, when applicable, least fixpoints in the first component and greatest fixpoints in the second one. ☐

Corollary 1. *For every program e (closed term of type* $\mathbf{comm}$*) we have, up to isomorphism, $[\![e]\!]^{\mathfrak{L}} = (L_+(e), L_\omega(e))$.*

Proof. From Lemma 3 we get $[\![e]\!]^{\mathfrak{L}} = [\![e]\!]^{\mathbf{GFP}}$, and Theorem 1 then yields the announced statement. ☐

Now consider the interpretation $[\![-]\!]^{\mathfrak{M}}$ with respect to some Büchi abstraction $\mathfrak{M} = (\mathfrak{M}_+, \mathfrak{M}_\omega)$ with Galois insertion α, γ.

Theorem 3. *Let e be a program. Then $\alpha_{+/\omega}([\![e]\!]^{\mathfrak{L}}) = [\![e]\!]^{\mathfrak{M}}$.*

Proof. We lift the abstraction α to products, coproducts, and monotone function spaces as described at the head of Section VII. It is then clear that it preserves the entire structure that participates in the definition of the semantics in a Büchi algebra: Least upper bounds, concatenation, infinite products, basic symbols, and the cartesian-closed structure. Thus, $\alpha([\![e]\!]^{\mathfrak{L}})$ can be rewritten to $[\![e]\!]^{\mathfrak{M}}$ and the result follows. ☐

Putting everything together we thus obtain:

Corollary 2 (Main result)**.** *Let e be a program and $U_{+/\omega} \in \mathfrak{M}_{+/\omega}$. We have $L_{+/\omega}(e) \subseteq \gamma_{+/\omega}(U_{+/\omega})$ iff $X_{+/\omega} \sqsubseteq U_{+/\omega}$ where $[\![e]\!]^{\mathfrak{M}} = (X_+, X_\omega)$. In particular, when $\mathbf{A}$ is the policy automaton, taking $U_{+/\omega} = \alpha_{+/\omega}(L_{+/\omega}(\mathbf{A}))$, we get $L_{+/\omega}(e) \subseteq L_{+/\omega}(\mathbf{A})$ iff $X_{+/\omega} \sqsubseteq \alpha_{+/\omega}(L_{+/\omega}(\mathbf{A}))$.*

Moreover, $[\![e]\!]^{\mathfrak{M}}$ is effectively computable by evaluation in the finite model $\mathfrak{M}$.

The last part of the corollary requires a little bit of explanation. The only difficulty is the computation of the fixpoints, which involve infinite supremums of infinite products. It amounts to the following task:

Given a finite poset P, elements $p, p' \in P$ and a finite function $f : P \times P \to \mathfrak{M}_+$, compute:

$$A(p,p') = \bigsqcup_{\substack{\vec{p} \in P^+ \\ \vec{p} : p \rightsquigarrow p'}} f(\vec{p}) \qquad \text{and} \qquad B(p) := \bigsqcup_{\substack{\vec{p} \in P^\omega \\ \vec{p} : p \rightsquigarrow}} f(\vec{p})$$

Write $A^n(p,p')$ to mean $A(p,p')$ where the supremum ranges over sequences $\vec{p}$ of length smaller than n. Then A^n is computable since it is a finite supremum of finite products, and moreover $A^{\mathrm{Card}(\mathfrak{M}_*)}(p,p') = A(p,p')$, so $A(p,p')$ is effectively computable. In practise, we can compute A^n

iteratively from A^{n-1}, and halt when it stops increasing using the same idea as in the Floyd-Warshall algorithm.

To compute B, we notice that

$$B(p) = \bigsqcup_{q \in P} A(p,q) \cdot A(q,q)^\omega$$

where $A(q,q)^\omega = \pi(A(q,q)A(q,q)\cdots)$.

Indeed, given any $\vec{p} : p \rightsquigarrow$, by the infinite pigeonhole principle, there is a $q \in P$ that appears infinitely many times in $\vec{p}$, and thus $f(\vec{p}) \sqsubseteq A(p,q)A(q,q)^\omega$.

Finally, the fact that $A(q,q)^\omega$ is computable was already present in [14]. Given $X \in \mathfrak{M}_+$, X^ω is by definition $\alpha_\omega(\gamma_+(X)^\omega)$. Since $\gamma_+(X)^\omega$ is regular, given a patch (C,D), we can decide whether $CD^\omega \cap \gamma_+(X)^\omega \neq \varnothing$ using Büchi nonemptiness and then compute the closure.

We remark that this algorithm shows that in the case of finite Büchi algebras the special case of ω-powers, cf. Wilke algebras, is sufficient.

VIII. Conclusion

We have shown how to construct a cartesian-closed category with fixpoints from a given Büchi automaton such that interpretation of a $\lambda\mathbf{Y}$-term in this category allows one to determine whether its finite and infinite traces are accepted by the automaton.

The interpretations of all types in this category are finite lattices; as a result the interpretation of a term is computable and thus leads to a new algorithm for higher-order model checking of trace properties and as such constitutes a new step in Salvati and Walukiewicz's programme of *Model Checking by Evaluating Effective Semantics* in the terminology of [20].

While we have not rigorously checked this, it appears that our algorithm fits the known [15] complexity bound $n-1$-EXPTIME for nth-order model checking of trace properties in $\lambda\mathbf{Y}$. Notice here that our type $\mathbf{comm}$ of order 0 corresponds in traditional $\lambda\mathbf{Y}$ to $o \to o$, of order 1 thus one level is "saved".

A natural next step would be to design a type-and-effect system based on our interpretation. Here, we would use elements of $[\![\tau]\!]^{\mathfrak{M}}$ to *refine* the higher-order type τ. In addition, we could use regions to abstract objects and pointers and thus obtain a refined type system for a higher-order language with objects such as modern versions of Java or Scala.

Also, it would be interesting to run some benchmarks in order to find out whether our procedure competes with or is (on path properties!) better than existing implementations of higher-order model checking such as [19], [18], [17].

References

[1] Parosh Aziz Abdulla et al Advanced Ramsey-based Büchi automata inclusion testing. Proc. *CONCUR*, pages 187–202. Springer, 2011.

[2] Klaus Aehlig, Jolie G. de Miranda, and C.-H. Luke Ong. The monadic second-order theory of trees given by arbitrary level-two recursion schemes is decidable. Proc. *TLCA*, pages 39–54. Springer, 2005.

[3] Lennart Beringer, Robert Grabowski, and Martin Hofmann. Verifying pointer and string analyses with region type systems. *Computer Languages, Systems & Structures*, 39(2):49–65, 2013.

[4] Ahmed Bouajjani, Javier Esparza, and Oded Maler. Reachability analysis of pushdown automata: Application to model-checking. Proc. CONCUR, pages 135–150. Springer, 1997.

[5] J. R. Büchi. On a decision method in restricted second order arithmetic. In *Proc. Congress on Logic, Method, and Philosophy of Science*, pages 1–12, Stanford, CA, USA, 1962. Stanford University Press.

[6] Olivier Carton, Dominique Perrin, and Jean-Eric Pin. Automata and semigroups recognizing infinite words. In Jörg Flum, Erich Grädel, and Thomas Wilke, editors, *Logic and Automata*, volume 2 of *Texts in Logic and Games*, pages 133–168. Amsterdam University Press, 2008.

[7] Patrick Cousot and Radhia Cousot. Abstract interpretation: A unified lattice model for static analysis of programs by construction or approximation of fixpoints. Proc. POPL, pages 238–252. ACM, 1977.

[8] Z. Esik and W. Kuich. "on iteration semiring-semimodule pairs". *Semigroup Forum*, 75:129–159, 2007.

[9] Seth Fogarty and Moshe Y. Vardi. Büchi complementation and size-change termination. *Logical Methods in Computer Science*, 8(1), 2012.

[10] Robert Grabowski, Martin Hofmann, and Keqin Li. Type-based enforcement of secure programming guidelines - code injection prevention at SAP. Proc. FAST, pages 182–197. Springer, 2011.

[11] Charles Grellois and Paul-André Melliès. Finitary semantics of linear logic and higher-order model-checking. Proc. MFCS, pages 256–268. Springer, 2015.

[12] Charles Grellois and Paul-André Melliès. An infinitary model of linear logic. Proc. FOSSACS, pages 41–55. Springer, 2015.

[13] Charles Grellois and Paul-André Melliès. Relational semantics of linear logic and higher-order model checking. Proc. CSL, pages 260–276, LiPICS, 2015.

[14] Martin Hofmann and Wei Chen. Abstract interpretation from Büchi automata. Proc. CSL-LICS, pages 51:1–51:10. ACM, 2014.

[15] Naoki Kobayashi and C.-H. Luke Ong. Complexity of model checking recursion schemes for fragments of the modal mu-calculus. Proc. ICALP, pages 223–234. Springer, 2009.

[16] Naoki Kobayashi and C.-H. Luke Ong. A type system equivalent to the modal mu-calculus model checking of higher-order recursion schemes. In *LICS*, pages 179–188. IEEE Computer Society, 2009.

[17] Naoki Kobayashi, Ryosuke Sato, and Hiroshi Unno. Proc. PLDI, pages 222–233. ACM, 2011.

[18] Akihiro Murase, Tachio Terauchi, Naoki Kobayashi, Ryosuke Sato, and Hiroshi Unno. Temporal verification of higher-order functional programs. Proc. POPL, pages 57–68. ACM, 2016.

[19] Robin P. Neatherway, Steven J. Ramsay, and C.-H. Luke Ong. A traversal-based algorithm for higher-order model checking. Proc. ICFP, pages 353–364. ACM, 2012.

[20] Luke Ong. Higher-order model checking: An overview. In Proc. LICS, pages 1–15. IEEE Computer Society, 2015.

[21] Dominique Perrin and Jean-Eric Pin. Semigroups and automata on infinite words. In J. Fountain, editor, *NATO Advanced Study Institute on Semigroups, Formal Languages and Groups*. Kluwer, 1995.

[22] Dominique Perrin and Jean-Eric Pin. *Infinite words : automata, semigroups, logic and games*. Pure and applied mathematics. Academic, London, San Diego (Calif.), 2004.

[23] Gordon Plotkin. A structural approach to operational semantics. Report DAIMI FN-19, Computer Science Department, Aarhus University, September 1981.

[24] Sylvain Salvati and Igor Walukiewicz. A model for behavioural properties of higher-order programs. Proc. CSL, LiPICS, 2015.

[25] Sylvain Salvati and Igor Walukiewicz. Using models to model-check recursive schemes. *Logical Methods in Computer Science*, 11(2), 2015.

[26] Christian Skalka, Scott F. Smith, and David Van Horn. Types and trace effects of higher order programs. *J. Funct. Program.*, 18(2):179–249, 2008.

[27] Igor Walukiewicz. Pushdown processes: Games and model checking. Proc. CAV, pages 62–74. Springer, 1996.