Model Checking Liveness Properties of Higher-Order Functional Programs

M. M. Lester R. P. Neatherway C.-H. L. Ong S. J. Ramsay

Oxford University Computing Laboratory

Abstract. Recent advances in the model checking of recursion schemes have opened the prospect of a model checking approach to the verification of higher-order functional programs. We formulate the Resource Usage Verification Problem in a general (liveness) setting, where good behaviours are specified by alternating parity (word) automata; and we give a sound and complete decision procedure by reduction to the problem of model checking *higher-order recursion schemes* (HORS) against alternating parity tree automata. Extending Kobayashi's type-inference approach, we present an efficient algorithm for deciding a restriction of the model checking problem in which properties are expressed by *alternating weak tree automata* (and hence all CTL formulas). We have constructed a model checker, THORS, that implements our algorithm and a number of optimisations. Despite the hugely challenging worst-case time complexity, THORS performs remarkably well on small examples, even up to order 5. To our knowledge, this is the first model checker for HORS which allows for the specification of tree automata with a non-trivial acceptance condition, including all CTL properties.

1 Introduction

In the past decade, huge strides have been made in the development of finite-state and pushdown model checking for the verification of computer programs. Though highly effective when applied to first-order imperative programs such as C, these techniques are much less useful for higher-order functional programs. In contrast, the two standard approaches to the verification of higher-order programs are *type-based program analysis* on the one hand, and *theorem-proving* and *dependent types* on the other. The former is sound, but often imprecise; the latter typically requires human intervention.

In a POPL'09 paper [1], Kobayashi introduced a novel approach to the verification of higher-order functional programs by reduction to a model checking problem for higher-order recursion schemes. A form of simply-typed lambda calculus with recursion and (uninterpreted) first-order symbols, *higher-order recursion schemes* (HORS) are generators of infinite trees. Building on recent advances in the model checking problem for HORS [2], Kobayashi developed a type-based algorithm which is sound and complete for model checking HORS against trivial automata (i.e. Büchi tree automata with a trivial acceptance condition). This method has been successfully applied to the resource safety verification problem [3] for RUL, a simply-typed functional language with dynamic resource creation and access primitives. There are transforms which,

given a functional program M and a correctness property φ (e.g. an open file is eventually closed, and not read from or written to after it is closed), reduce the verification problem to one of deciding whether a trivial automaton $\mathcal{A}_\varphi$ accepts the tree generated by a recursion scheme R_M.

In this paper, we formulate the verification problem of *resource usage in accord with* φ, where φ is a specification of good resource-usage behaviour represented as an alternating parity automaton (equivalently a linear-time modal mu-calculus formula). We show that the problem can be reduced to the problem of model checking HORS against alternating parity tree automata. We illustrate the approach using liveness specifications for resource usage and discuss appropriate fairness assumptions.

After formulating the general problem and its solution, we turn our attention to a particular restriction which we believe will be relevant to practice. We develop an efficient algorithm for model checking HORS against *alternating weak tree automata* (AWT), which are an automata-theoretic characterisation of the *alternation-free modal mu-calculus* (AFMC). AFMC maintains a delicate balance between expressivity and algorithmics: it embeds all of CTL whilst remaining amenable to efficient methods (AFMC can be evaluated in time linear in the structure) [4, 5].

Given an AWT $\mathcal{A}$ and a HORS G, we consider the *acceptance problem* of whether $\mathcal{A}$ accepts the tree defined by G, denoted $[\![G]\!]$. By adapting Kobayashi and Ong's result [6], given an AWT $\mathcal{A}$, we construct an intersection type system $\boldsymbol{\lambda}^{\mathcal{A}}_{\wedge}$ such that $\mathcal{A}$ accepts $[\![G]\!]$ if, and only if, Éloïse has a winning strategy in a derived *weak* Büchi game $\mathbb{G}(G, \boldsymbol{\lambda}^{\mathcal{A}}_{\wedge})$. Intuitively Éloïse aims to prove that the HORS G is well-typed in $\boldsymbol{\lambda}^{\mathcal{A}}_{\wedge}$, and Abelard aims to disprove it. We present a semi-algorithm that is guaranteed to terminate in the case of a YES instance, and show how it can be used to form a decision procedure. We also discuss specialisations of the algorithm for less expressive automata, including deterministic AWT and automata with trivial and co-trivial acceptance conditions.

Because of the inherent complexity of the model checking problem (Theorem 5), building and interrogating a representation of the game naïvely would be a hopeless task. Fortunately, to determine a solution, we need only consider the reachable part of the underlying graph, which corresponds to those types that describe the computation of run-trees of $\mathcal{A}$ over $[\![G]\!]$. The algorithm consists of a two stage loop. In the first stage, the recursion scheme G is partially evaluated in order to extract those computationally relevant type bindings. If there is no run-tree of $\mathcal{A}$ over $[\![G]\!]$, then stage 1 will fail and the algorithm concludes that $[\![G]\!]$ is not accepted by $\mathcal{A}$. In the second stage a *weak* Büchi game is built from the bindings collected in the first, if Éloïse has a winning strategy in this game then the algorithm concludes that there is an accepting run-tree of $\mathcal{A}$ over $[\![G]\!]$. We present a linear-time algorithm for solving such games.

We have implemented the algorithm in a prototype tool, called THORS (Types for Higher-Order Recursion Schemes). We have evaluated the tool on a number of examples, some derived from programs, others from the literature. The experiments we have performed show that the model checker is remarkably fast, despite the n-EXPTIME complete worst-case time-complexity.

Related work. Kobayashi and Ong [6] have shown that, given an alternating parity tree automaton $\mathcal{A}$, there is a type theory $\mathcal{T}_{\mathcal{A}}$ that characterises it, in the sense that the tree $[\![G]\!]$ generated by an arbitrary HORS G is accepted by $\mathcal{A}$ if, and only if, Éloïse has a winning

strategy in a certain type-inference parity game. Their result provides the theoretical basis of our algorithm. On the implementation side, the precursor of our work is TReCS, Kobayashi's tool [7] that checks if a given deterministic trivial automaton accepts the tree generated by a HORS. Stage 1 of our algorithm builds on and extends Kobayashi's method of extracting computationally relevant type bindings from configuration graphs, which are representations of runs of the automaton on partial evaluation of the HORS. In [8] a new algorithm for trivial automata model checking of HORS is given, which runs in time linear in the size of the HORS, assuming that the automata and the largest order and arity of functions are fixed. To our knowledge, THORS is the first implementation of *non-trivial* automata (more generally, AWT / CTL) model checking of HORS.

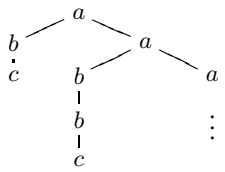
Outline. The basic notions such as HORS, AWT and the associated parity games are defined in Section 2. Section 3 introduces the resource liveness verification problem. Section 4 introduces the intersection type system that characterises AWT, the two-stage model checking algorithms, and presents relevant complexity results. In Section 5, we briefly discuss correctness, complexity and optimisation techniques. The implementation and experimental results are presented in Section 6.

2 Technical preliminaries

Higher-order recursion schemes *Kinds*[1] are expressions defined by the grammar $A ::= o \mid A \to B$. We define the *order* of a kind: $ord(o) := 0$ and $ord(A \to B) := \max(ord(A) + 1, ord(B))$. Assume a countably infinite set Var of kinded variables. A *higher-order recursion scheme* is a tuple $G = \langle \Sigma, \mathcal{N}, \mathcal{R}, S \rangle$ where (i) Σ is a *ranked alphabet* i.e. each *terminal* $f \in \Sigma$ has an arity $ar(f) \geq 0$; (ii) $\mathcal{N}$ is a set of kinded *non-terminals*; $S \in \mathcal{N}$ is a distinguished *start symbol* of kind o; and (iii) $\mathcal{R}$ is a finite set of rewrite rules of the form $F\, x_1 \cdots x_n \to e$, where $F : A_1 \to \cdots \to A_n \to o$ and each $x_i : A_i \in Var$, and $e : o$ is an applicative term generated from $\Sigma \cup \mathcal{N} \cup \{x_1, \cdots, x_n\}$; we define $\mathcal{R}(F) := \lambda x_1 \cdots x_n.e$. The *order* of a recursion scheme is the highest order (of the kind) of its non-terminals. We use *deterministic* recursion schemes (i.e. one rule for each non-terminal) to define possibly-infinite trees.

Example 1 (An order-2 recursion scheme G_1). Take the ranked alphabet Σ of symbols a, b, c of arities $2, 1, 0$ respectively and the set $\mathcal{N} = \{S : o, F : (o \to o) \to o, G : (o \to o) \to (o \to o) \to o \to o\}$ of non-terminals. Consider the order-2 recursion scheme G_1 with rewrite rules:

$$\begin{aligned} S &\to F\, b \\ F\, x &\to a\, (x\, c)\, (F\, (G\, b\, x)) \\ G\, x\, y\, z &\to x\, (y\, z) \end{aligned}$$



Unfolding from S, we have

$$S \to F\, b \to a\, (b\, c)\, (F\, (G\, b\, b)) \to a\, (b\, c)\, (a\, (G\, b\, b\, c)\, (F\, (G\, b\, (G\, b\, b)))) \to \cdots$$

[1] We use the word *kind* here instead of the more usual *type* because the latter is reserved for *intersection type*, to be introduced in the sequel.

thus generating the infinite term $a\,(b\,c)\,(a\,(b\,(b\,c))\,(\cdots))$. The tree generated by G_1, $[\![G_1]\!]$, is the abstract syntax tree of the infinite term, as shown above.

Formally the rewrite relation $\to_G$ is defined by induction over the following rules:

$$\frac{Fx_1 \cdots x_n \to e \text{ is a } \mathcal{R}\text{-rule}}{Ft_1 \cdots t_n \to_G e[t_1/x_1, \cdots, t_n/x_n]} \qquad \frac{t \to_G t'}{s\,t \to_G s\,t'} \qquad \frac{t \to_G t'}{t\,s \to_G t'\,s}$$

We write $\to_G^*$ for the reflexive, transitive closure of $\to_G$.

Henceforth fix a ranked alphabet Σ; set $m := \max\{ar(f) \mid f \in \Sigma\}$. A *$\Sigma$-labelled tree* is a partial function t from $\{1, \cdots, m\}^*$ to Σ such that $dom(t)$ is prefix-closed. A (possibly infinite) sequence π over $\{1, \cdots, m\}$ is a *path* of t if every finite prefix of π is in $dom(t)$. Given a term t, we define a (finite) tree $t^\perp$ by: $(F\,t_1 \cdots t_n)^\perp := \perp$ and $(f\,t_1 \cdots t_n)^\perp := f\,{t_1}^\perp \cdots {t_n}^\perp$ (where $n \geq 0$). E.g. $(f\ (F\ a)\ b)^\perp = f\ \perp\ b$. Let $\sqsubseteq$ be the least partial order on $\Sigma \cup \{\perp\}$ defined by $\forall a \in \Sigma. \perp \sqsubseteq a$. We extend $\sqsubseteq$ to a partial order on trees by: $t \sqsubseteq s$ iff $\forall w \in dom(t).(w \in dom(s) \wedge t(w) \sqsubseteq s(w))$. E.g. $\perp \sqsubseteq f\ \perp\ \perp \sqsubseteq f\ \perp\ b \sqsubseteq f\ a\ b$. For a directed set T of trees, we write $\bigsqcup T$ for the least upper bound of elements of T with respect to $\sqsubseteq$. We define the *tree generated by* G, or the *value tree* of G, $[\![G]\!] := \bigsqcup\{t^\perp \mid S \to_G^* t\}$. By construction, $[\![G]\!]$ is a possibly infinite, ranked $(\Sigma \cup \{\perp\})$-labelled tree.[2]

Alternating parity tree automata (APT) Given a finite set X, the set $\mathsf{B}^+(X)$ of *positive Boolean formulas* over X is defined by the grammar: $\mathsf{B}^+(X) \ni \theta ::= \mathsf{t} \mid \mathsf{f} \mid x \mid \theta \wedge \theta \mid \theta \vee \theta$, where x ranges over X. We say that a subset Y of X *satisfies* θ if assigning true to elements in Y and false to elements in $X \setminus Y$ makes θ true. An *alternating parity tree automaton* (or APT for short) over Σ-labelled trees is a tuple $\mathcal{A} = \langle \Sigma, Q, \delta, q_I, \Omega \rangle$ where (i) Σ is a ranked alphabet; let m be the largest arity of the terminal symbols; (ii) Q is a finite set of states, and $q_I \in Q$ is the initial state; (iii) $\delta : Q \times \Sigma \longrightarrow \mathsf{B}^+(\{1, \cdots, m\} \times Q)$ is the transition function where, for each $f \in \Sigma$ and $q \in Q$, $\delta(q, f) \in \mathsf{B}^+(\{1, \cdots, ar(f)\} \times Q)$; (iv) $\Omega : Q \to \{0, \cdots, m-1\}$ is the priority function. We say that an APT is *deterministic* just if its transition function δ is deterministic i.e. for each $q \in Q$ and $f \in \Sigma$, $\delta(q, f)$ is t or f or has the shape $\bigwedge_{i=1}^{ar(f)}(i, q_i)$; it is *conjunctive* just if δ maps every pair to a disjunction-free formula.

For any pair (q, f), the RHS of δ can be thought of as a set of satisfying assignments (explicitly so through conversion to DNF), each of which are a potential continuation of the run-tree. A satisfying assignment consists of a set of pairs (i, q') ($i \in \{1, \cdots, ar(f)\}$, $q' \in Q$) where each such pair corresponds to sending a copy of the automaton in state q' to child i of the current position in the tree. In Example 2 below, we can see that $\mathcal{A}_1$ has a single satisfying assignment for each transition and is therefore deterministic.

An *alternating Büchi tree automaton* (ABT), $\mathcal{A} = \langle \Sigma, Q, \delta, q_I, \Omega \rangle$, is an APT with at most two priorities (i.e. $\Omega : Q \to \{0, 1\}$); states with priority 0 (and 1) are called *accepting* (and *rejecting* respectively). An *alternating weak tree automaton* (AWT) $\mathcal{A}$ is an ABT that satisfies *weakness*: there is a partial order $\leq$ over a partition $\{Q_1, \cdots, Q_n\}$

[2] W.l.o.g. (see e.g. [6, Remark 2.1]) we consider HORS G whose tree $[\![G]\!]$ does not contain $\perp$.

of Q such that (i) for each i, either every state in Q_i is accepting, or none is; (ii) for every $q \in Q_i$ and $q' \in Q_j$ for which q' occurs in $\delta(q, f)$, for some $f \in \Sigma$, we have $Q_j \leq Q_i$. It follows that every infinite path of an $\mathcal{A}$-run ultimately gets trapped within some Q_j. A *deterministic weak tree automaton* (DWT) is an AWT that is deterministic.

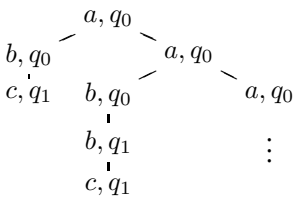
A *trivial automaton* is an APT that has only one priority, namely, 0 (equivalently [9, 1], it is a non-deterministic Büchi automaton all of whose states are accepting). It follows from the definition that a tree is accepted by a trivial automaton just if there is a run-tree over it (the adjective "trivial" refers to the absence of any acceptance condition on infinite paths of the run-tree). We define a *co-trivial automaton* to be an APT all of whose states have priority 1. Thus a tree is accepted by a co-trivial automaton if (and only if) there is a run-tree that has no infinite paths.

Example 2 (A DWT $\mathcal{A}_1$). Take the ranked alphabet Σ of Example 1; $[\![G_1]\!]$ is accepted by $\mathcal{A}_1 = \langle \Sigma, \{q_0, q_1\}, \delta, q_0, \{q_0 \mapsto 0, q_1 \mapsto 1\}\rangle$ with $\{q_1\} \leq \{q_0\}$, where δ is as follows (omitting all that δ maps to f): let $q \in \{q_0, q_1\}$

$$(q_0, a) \mapsto (1, q_0) \wedge (2, q_0),\ (q, b) \mapsto (1, q_1),\ (q, c) \mapsto \mathsf{t}.$$

Thus $\mathcal{A}_1$ accepts a Σ-labelled tree t just if in every path, if b occurs, then b occurs until c occurs. Note that $\mathcal{A}_1$ is actually a DWT. Because b and c have arities 1 and 0 respectively, the property defined by $\mathcal{A}_1$ can be described by the CTL formula $AG\,(b \rightarrow A(b\ U\ c))$.

A *run-tree* of an APT $\mathcal{A}$ over a Σ-labelled tree t is a $(dom(t) \times Q)$-labelled (unranked) tree r (see right for the run-tree of $\mathcal{A}_1$ over $[\![G_1]\!]$) satisfying: (i) $\epsilon \in dom(r)$ and $r(\epsilon) = (\epsilon, q_I)$; (ii) for every $\beta \in dom(r)$ with $r(\beta) = (\alpha, q)$, there is a (possibly empty) set S that satisfies $\delta(q, t(\alpha))$; and for each $(i, q') \in S$, there is some j such that $\beta\, j \in dom(r)$ and $r(\beta\, j) = (\alpha\, i, q')$.



Let $\pi = \pi_1\, \pi_2 \cdots$ be an infinite path in r; for each $i \geq 0$, let the state label of the node $\pi_1 \cdots \pi_i$ be q_{n_i} where $q_{n_0} = q_I$. We say that π satisfies the *parity* condition if the least priority that occurs infinitely often in $\Omega(q_{n_0})\, \Omega(q_{n_1})\, \Omega(q_{n_2}) \cdots$ is even. A run-tree r is *accepting* if every infinite path in it satisfies the parity condition. We say that a tree t is *accepted* by $\mathcal{A}$ just if there is an accepting run-tree of $\mathcal{A}$ over t.

Ong [2] showed that there is a procedure that, given a recursion scheme G and an APT $\mathcal{A}$, decides whether $\mathcal{A}$ accepts $[\![G]\!]$.

Theorem 1 (Ong [2]). *Let G be a recursion scheme of order n, and $\mathcal{A}$ be an APT. The problem of checking whether $\mathcal{A}$ accepts $[\![G]\!]$ is n-EXPTIME-complete.*

The correspondence between parity games and modal mu-calculus (equivalently APT [10]) specialises to one between weak Büchi games and alternation-free mu-calculus (equivalently AWT [11]). (For parity games and alternation-free mu-calculus, see Appendix A.) Recall that a *Büchi game* is just a parity game $\langle V_A, V_E, v_0, E, \Omega\rangle$ that has at most two priorities, say[3], 0 and 1; priority-0 nodes are called *accepting*, and

[3] In a weak Büchi game, every path has at most one infinitely-occurring priority. Hence it does not matter whether the range of Ω is $\{0, 1\}$ or $\{1, 2\}$. (We choose the former.)

priority-1 nodes are called *rejecting*. A *Büchi game* is said to be *weak* if there is a partial order $\leq$ over a partition $\{V_1, \cdots, V_n\}$ of the node-set such that (i) for each i, either every node in V_i is accepting, or none is; and (ii) for each $(u, v) \in E$, if $u \in V_i$ and $v \in V_j$ then $V_j \leq V_i$. It follows that in a weak Büchi game, every maximal play is eventually "trapped" in V_i for some i, and it is winning (for Éloïse) if, and only if, V_i is accepting. (Henceforth we assume that parity games are finite.)

3 Verifying resource usage liveness properties

We approach the verification of higher-order functional programs by reduction to the model checking of HORS. Take the verification problem: does the functional program P satisfy temporal specification φ?

1. The program P is first transformed to a recursion scheme $\widetilde{P}$ that generates a tree $[\![\widetilde{P}]\!]$ representing all possible event sequences in the computation of P. The property φ is also suitably transformed to a property $\widetilde{\varphi}$ of infinite trees which may be a tree automaton or another temporal formula.
2. The tree $[\![\widetilde{P}]\!]$ is then model checked against the transformed property $\widetilde{\varphi}$, such that P satisfies φ if, and only if, $[\![\widetilde{P}]\!]$ satisfies $\widetilde{\varphi}$.

This method is *fully automatic*, *sound* and *complete* for the Resource Safety Verification Problem as studied by Igarashi and Kobayashi [12, 1].

The goal of Igarashi and Kobayashi's research in resource safety verification [12] is to check statically if the manner in which a given program accesses resources (which model stateful objects such as files, memory cells and locks) satisfies a given safety property φ. Our aim here is to extend φ to all linear-time modal mu-calculus properties, including both safety and liveness properties. Consider a simple functional program that opens a file `foo`, reads it repeatedly until the end-of-file character is read, and then closes it. An abstraction of such a program is presented in an ML-like syntax as follows.

```
let rec g x = if b then close(x)
                   else read(x) ; g(x) in
let s = open_in "foo" in g(s)
```

Does the program access the file `foo` in accord with a correctness specification φ = "An opened file is eventually closed, after which it is not read"? The property φ may be described as a regular expression (e.g. $\mathtt{r}^*\,\mathtt{c}$) or an automaton. Are such verification questions decidable?

Igarashi and Kobayashi formalised the problem for a simply-typed call-by-value lambda-calculus with recursion and primitives for dynamically creating and allocating resources. Following [1], we consider a call-by-name version (for consistency with HORS) which we call RUL.

Resource usage language, RUL We first fix a finite set Q_I of resource sorts and a set L of resource access primitives. A RUL *program* D is a set of function definitions $\{F_1\,\overline{x_1} = e_1, \cdots, F_n\,\overline{x_n} = e_n\}$, where each F_i is a defined function symbol, and e_i is an expression which is defined by the grammar

$$e \;::=\; \star \mid x \mid F \mid e_1\,e_2 \mid \mathbf{if}{*}\,e_1\,e_2 \mid \mathbf{new}^q\,e \mid \mathbf{acc}_a\,x\,e$$

where q and a range over Q_I and L respectively. The program contains exactly one function definition of the form $S = e$, where S, a distinguished defined function symbol, is the "main" function; by convention $F_1 = S$.

We consider only well-typed programs. The set of types is defined by the grammar $\tau \ ::= \ \mathbf{R} \mid \mathbf{unit} \mid \tau_1 \to \tau_2$, where $\mathbf{R}$ is the type of resources and **unit** is the type of the unit value $\star$. A type environment, Γ, is a finite map from variables (including function names F_i) to types. Valid type judgements $\Gamma \vdash e : \tau$ are defined by induction over the following set of rules.

$$\Gamma \vdash \star : \mathbf{unit} \qquad \Gamma, x : \tau \vdash x : \tau \qquad \frac{\Gamma \vdash e_1 : \tau_1 \to \tau_2 \quad \Gamma \vdash e_1 : \tau_1}{\Gamma \vdash e_1\, e_2 : \tau}$$

$$\frac{\Gamma \vdash e_1 : \mathbf{unit} \quad \Gamma \vdash e_2 : \mathbf{unit}}{\Gamma \vdash \mathbf{if}{*}\, e_1\, e_2 : \mathbf{unit}} \qquad \frac{\Gamma \vdash e : \mathbf{R} \to \mathbf{unit}}{\Gamma \vdash \mathbf{new}^q\, e : \mathbf{unit}} \qquad \frac{\Gamma \vdash e_1 : \mathbf{R} \quad \Gamma \vdash e_2 : \mathbf{unit}}{\Gamma \vdash \mathbf{acc}_a\, e_1\, e_2 : \mathbf{unit}}$$

A program D is *well-typed under* Γ just if $\Gamma, \overline{x_i} : \overline{\tau_i} \vdash e_i : \mathbf{unit}$ is valid for each i, where $\Gamma = \{F_1 : \overline{x_1} \to \mathbf{unit}, \cdots, F_n : \overline{x_n} \to \mathbf{unit}\}$, $\overline{x_i} : \overline{\tau_i}$ abbreviates the set $\{x_{i1} : \tau_{i1}, \cdots, x_{ir_i} : \tau_{ir_i}\}$, and $\overline{x_i} \to \mathbf{unit}$ means $x_{i1} \to \cdots \to x_{ir_i} \to \mathbf{unit}$.

Example 3 (RUL program D_1). The following program is obtained from the preceding ML program by lambda-lifting followed by a standard CPS transformation.

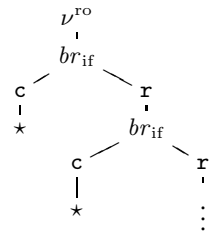
$$D_1 \ : \ \begin{cases} S = \mathbf{new}^{\mathrm{readonly}}\ (G\ \star) \\ G\, k\, x = \mathbf{if}{*}\ (\mathbf{acc}_{\mathrm{close}}\ x\ k)\ (\mathbf{acc}_{\mathrm{read}}\ x\ (G\, k\ x)) \end{cases}$$

Operational semantics. The small-step semantics of RUL is given by a binary relation $\to$ between pairs of the form $(R,\, e)$ where R is a set of resource names that have been created thus far in the computation (which is not garbage-collected). We define $\to$ by a set of rewrite rules as follows. Note that $\to$ is non-deterministic.

$$\begin{aligned} (R,\, F\,\overline{e}) &\to (R,\, e'[\overline{e}/\overline{y}]) \quad \text{if } F\,\overline{y} = e' \text{ is a } D\text{-equation} \\ (R,\, \mathbf{if}{*}\, e_1\, e_2) &\to (R,\, e_i) \quad i = 1, 2 \\ (R,\, \mathbf{new}^q\, e) &\to (R \cup \{x\},\, e\, x) \quad \text{if } x \notin R \\ (R \cup \{x\},\, \mathbf{acc}_a\, x\, e) &\to (R \cup \{x\},\, e) \end{aligned}$$

Example 4. A reduction sequence of D_1, and its tree of access sequence (definition to follow).

$$\begin{aligned} & (\varnothing,\, S) \\ \to\ & (\varnothing,\, \mathbf{new}^{\mathrm{readonly}}\ (G\ \star)) \\ \to\ & (\{x\},\, G\ \star\ x) \\ \to\ & (\{x\},\, \mathbf{if}{*}\ (\mathbf{acc}_{\mathrm{close}}\ x\ \star)\ (\mathbf{acc}_{\mathrm{read}}\ x\ (G\ \star\ x))) \\ \to\ & (\{x\},\, \mathbf{acc}_{\mathrm{read}}\ x\ (G\ \star\ x)) \\ \to\ & (\{x\},\, G\ \star\ x) \\ \to\ & \cdots \end{aligned}$$



Reasoning about resource usage is difficult because a program D may create infinitely many resources. To verify that D uses resources correctly, we need to do so *resource-wise* i.e. check that D uses *every* copy of every *sort* of resource (that has been created) in accord with the desired property.

Example 5 (Program D_2). D_2 creates infinitely many copies of "readonly" resource.

$$D_2 \ : \ \begin{cases} S = \mathbf{new}^{\mathrm{readonly}}\ (G\ \star) \\ G\ k\ x = \mathbf{if}{*}\ (\mathbf{acc}_{\mathrm{close}}\ x\ (\mathbf{new}^{\mathrm{readonly}}\ (G\ \star)))\ (\mathbf{acc}_{\mathrm{read}}\ x\ (G\ k\ x)) \end{cases}$$

It is possible to describe the correctness property either as a temporal logic formula or an automaton. Here we choose the latter. A *parity resource automaton* W is an alternating parity (word) automaton $\langle Q, L, \delta, Q_I, \Omega\rangle$, where Q is a set of states, $Q_I \subseteq Q$ is a set of initial states (we deliberately conflate initial states with *resource sorts* introduced earlier), L is a set of resource access primitives, $\delta : Q \times L \to 2^{2^Q}$ is a transition[4] function, $q_0 \in Q_I$ is an initial state, and $\Omega : Q \to \{0, \cdots, p\}$ is a priority function.

Definition 1. Fix a parity resource automaton $W = \langle Q, L, \delta, Q_I, \Omega\rangle$. Let x be a resource. A *configuration of resource* x (or simply x*-configuration*) is a pair, written $(H,\ e)_x$, where e is an expression and $H = \{x_1 : q_1, \cdots, x_n : q_n\}$ is a finite function that maps a resource x_i to its state $q_i \in \{q_{un}\} \cup Q$, with q_{un} meaning that the monitoring of resource x has not yet begun. The binary relation $\xrightarrow{d}_{D,W,x}$ over x-configurations, where $d \in \{1, 2, \epsilon\} \cup Q$ is defined by induction over the following rules. We shall omit the subscripts from $\xrightarrow{d}_{D,W,x}$ whenever they can readily be determined from the context. Let $x = x_{q_0}$ be a resource of sort $q_0 \in Q_I$. We define $\mathcal{T}_x$, the transition graph of x-configurations, as follows.

$$\begin{array}{ll}
(H,\ F\ \overline{e})_x \xrightarrow{\epsilon} (H,\ e'[\overline{e}/\overline{y}])_x & \text{where } F\ \overline{y} = e' \text{ is a } D\text{-equation} \\
(H,\ \mathbf{if}{*}\ e_1\ e_2)_x \xrightarrow{i} (H,\ e_i)_x & i = 1, 2 \\
(H,\ \mathbf{new}^{q_0}\ e)_x \xrightarrow{2} (H \cup \{x : q_0\},\ e\ x)_x & x \notin dom(H) \\
(H,\ \mathbf{new}^{q}\ e)_x \xrightarrow{1} (H \cup \{y : q\},\ e\ y)_x & y \notin dom(H) \\
(H \cup \{x : q\},\ \mathbf{new}^{q'}\ e)_x \xrightarrow{1} (H \cup \{x : q, y : q'\},\ e\ y)_x & y \notin dom(H) \\
(H \cup \{x : q\},\ \mathbf{acc}_a\ x\ e)_x \xrightarrow{q'} (H \cup \{x : q'\},\ e)_x & \text{if } q' \in X \in \delta(q, a), \text{ some } X \\
(H \cup \{y : q\},\ \mathbf{acc}_a\ y\ e)_x \xrightarrow{\epsilon} (H \cup \{y : q\},\ e)_x & \\
(H,\ \star)_x \xrightarrow{\epsilon} (H,\ \star)_x &
\end{array}$$

In the fourth clause above, q may or may not be q_0; in the penultimate clause, the state of y is unchanged. The *state* of $(H,\ e)_x$ is defined as $state(H,\ e)_x := H(x)$ if $x \in dom(H)$, and q_{un} otherwise. Note that if a configuration can do a ϵ-transition, its state is preserved by the transition, which is the unique transition from the configuration.

A parity game $\mathcal{G}(D, W)$. Let D be a RUL program and W a parity resource automaton. We construct a (parity) game $\mathcal{G}(D, W)$ between Éloïse (Verifier) and Abelard (Refuter) as follows.

- The initial position is the configuration set $\{(\varnothing,\ S)_{x_1}, \cdots, (\varnothing,\ S)_{x_n}\}$, where each x_i is a resource of sort q_i with $Q_I = \{q_1, \cdots, q_n\}$

[4] The RHS of the transition function can be presented equivalently as a positive Boolean formula. Thus, assuming $\delta(q, a) = \{S_1, \cdots, S_l\}$, we can write $\delta(q, a)$ as $\bigvee_i \bigwedge_{q \in S_i} (1, q)$.

- If the current position is a configuration set C, then Abelard is to move. He does so by choosing some configuration $c \in C$ to be the new position.
- If the current position is a configuration $(H,\, e)_x$, then Éloïse is to move. There are two cases. If $e = \mathbf{acc}_a\, x\, e'$ and $state(H,\, e)_x = q \in Q$, then Éloïse chooses some $T \in \delta(q, a)$, and the new position is the set $\{(H \cup \{x : q'\},\, e')_x \mid q' \in T\}$. In all other cases, the new position is $\{c \mid (H,\, e)_x \xrightarrow{d} c\}$.

Winning condition. If a player is unable to move at any position, then the other player wins. Suppose an infinite play $C_0\ c_0\ C_1\ \cdots$ ensues, where C_i ranges over configuration-sets, and c_i over configurations. Let $state(c_i) = q_i$; if the least priority that occurs infinitely often in the sequence $\Omega(q_0)\ \Omega(q_1)\ \Omega(q_2)\ \cdots$ is even, then Éloïse wins; otherwise Abelard wins. (We extend the priority map Ω by mapping q_{un} to the largest priority.) We say that D *satisfies* W just if Éloïse wins the game $\mathcal{G}(D, W)$.

Note that (i) if a play reaches a configuration of the shape $(H \cup \{x : q\},\, \mathbf{acc}_a\, x\, e)_x$ where $\delta(q, a) = \varnothing$ then Éloïse loses as she is stuck; (ii) if a play reaches $(H,\, \star)_x$ with state q then Éloïse wins if $\Omega(q)$ is even.

Example 6. Consider $W_\theta := \langle \{q_{ro}, q_c\}, \{\mathtt{r}, \mathtt{c}\}, \delta, \{q_{ro}\}, \{(q_{ro}, \theta), (q_c, 0)\}\rangle$ with $\delta : (q_{ro}, \mathtt{r}) \mapsto \{\{q_{ro}\}\}, (q_{ro}, \mathtt{c}) \mapsto \{\{q_c\}\}$. Then D_1 satisfies the resource (safety) automaton W_0, but not the (liveness) automaton W_1 because of the infinite path that always takes the right branch of **if**$*$. This path may be disregarded under suitable fairness assumptions, on which more anon.

Transformation to recursion schemes. Following [1], we reduce the resource usage verification problem to the APT model checking problem for HORS.

Definition 2. Given a pair (D, W) as before, we construct a HORS $G_D = \langle \Sigma, \mathcal{N}, \mathcal{R}\rangle$ and an APT $\mathcal{A}_{D,W}$ such that D satisfies W if and only if $[\![G_D]\!]$ is accepted by $\mathcal{A}_{D,W}$. We define $G_D := \langle \Sigma, \mathcal{N}, \mathcal{R}\rangle$ where

$$
\begin{aligned}
\Sigma &:= \{a : 1 \mid a \in L\} \cup \{br_{\mathrm{if}} : 2, br_{\mathrm{new}} : 2, \star : 0\} \cup \{\nu^q : 1 \mid q \in Q_I\}\\
\mathcal{N} &:= \{F : (\Gamma(F))^\sharp \mid F \in dom(\Gamma)\} \cup \{\mathbf{new}^q : (((o \to o) \to o \to o) \to o) \to o \mid q \in Q_I\}\\
&\quad \cup\ \{\mathbf{acc}_a \mid ((o \to o) \to o \to o) \to o \to o \mid a \in L\}\\
&\quad \cup\ \{\mathbf{if}* : o \to o \to o, I : (o \to o) \to o \to o, K : (o \to o) \to o \to o\}
\end{aligned}
$$

with Γ witnessing the well-typedness of D; and $\mathcal{R}$ consisting of the following rules:

$$
\begin{array}{ll}
\mathbf{if}*\, x\, y \to br_{\mathrm{if}}\, x\, y & F\, \overline{x} \to e \quad \text{for each } D\text{-equation } F\, \overline{x} = e\\
I\, x\, k \to x\, k & \mathbf{new}^q\, k \to br_{\mathrm{new}}\, (\nu^q\, (k\, I))\, (k\, K) \quad \text{for each } q \in Q_I\\
K\, x\, k \to k & \mathbf{acc}_a\, x\, k \to x\, a\, k \quad \text{for each } a \in L
\end{array}
$$

where $(-)^\sharp$ is a translation of types of RUL to kinds defined by

$$\mathbf{R}^\sharp = (o \to o) \to o \to o \qquad \mathbf{unit}^\sharp = o \qquad (\tau_1 \to \tau_2)^\sharp = \tau_1^\sharp \to \tau_2^\sharp$$

We define APT $\mathcal{A}_{D,W} := \langle \Sigma, Q', \delta', q_{un}, \Omega'\rangle$ where Σ is as before, $Q' = Q \cup \{q_{un}\}$, Ω' maps q_{un} to 0, and $\Omega' \restriction Q := \Omega$; and δ' is defined as follows:

$$
\begin{array}{ll}
(q, br_{\mathrm{if}}) \mapsto (1, q) \wedge (2, q) & (q, br_{\mathrm{new}}) \mapsto (1, q) \wedge (2, q)\\
(q, \nu^{q'}) \mapsto \begin{cases} (1, q') & \text{if } q = q_{un}\\ \mathsf{t} & \text{otherwise} \end{cases} & (q, \star) \mapsto \begin{cases} \mathsf{t} & \text{if } \Omega'(q) \text{ is even}\\ \mathsf{f} & \text{otherwise} \end{cases}\\
(q, a) \mapsto \bigvee_{1 \le i \le k} \bigwedge_{q' \in Q_i} (1, q') & \text{if } q \in Q \text{ and } \delta(q, a) = \{Q_1, \cdots, Q_k\}
\end{array}
$$

Theorem 2 (Reduction). *Let D be a RUL program and W a parity resource automaton. Then D satisfies W if and only if $[\![G_D]\!]$ is accepted by the APT $\mathcal{A}_{D,W}$.*

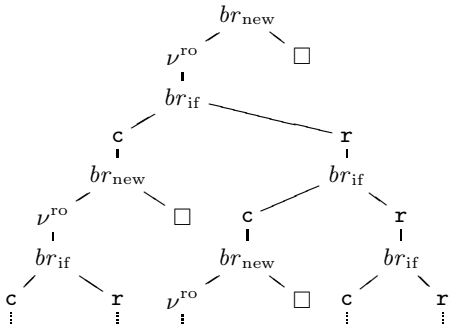
See Appendix B for a proof. We develop a non-standard notion of weak bisimulation between labelled transition systems (as game graphs), and prove (Theorem 7) that Éloïse winning strategies of the parity game over one graph determine the Éloïse winning strategies of the parity game over the other graph. Theorem 2 is then a corollary.

Liveness and fairness Liveness properties (under certain conditions, some "good" configurations are eventually reachable) are harder to check than safety properties (under certain conditions, no "bad" configuration is ever reachable). In practice, however, liveness is as important as safety. *Fairness* (one version says: under certain conditions, some events must occur infinitely often) is a property often associated with liveness checking. It is known that liveness of a system can be reduced to termination of a modified system with fairness assumptions. Another use of fairness is as an *assumption* of a liveness property. For example in the dining philosophers' problem, a path that ignores a philosopher's request indefinitely is unfair, and it can violate a desired liveness property. When constructing abstract models of programs, conditionals are often transformed to non-deterministic branches; a common fairness assumption is to deem an (infinite) path *unfair* if it always takes the left branch, or always takes the right branch.

Example 7 (Running example: G_{D_2} and CWT $\mathcal{A}_2$). See E.g. 5 (for D_2) and Def. 2.

The tree $[\![G_{D_2}]\!]$ is shown on the right. The box $\Box$ represents possible access sequences obtained by keeping track of different occurrences of the resource. We say an access sequence is *unfair* if, from some point onwards, it *only* takes the right branch of br_{if} (intuitively because it corresponds to reading an infinite "readonly" resource). Set φ_2 to be the CTL formula

$$AG\,(\mathtt{r} \Rightarrow A\,((\mathtt{r} \vee br_{\mathrm{if}})\,U\,\mathtt{c})).$$



When restricted to *fair*[5] paths, $[\![G_{D_2}]\!]$ satisfies φ_2. We encode φ_2 together with the fairness assumption in the *conjunctive* weak tree automaton (CWT) $\mathcal{A}_2$ with the following transition function.

$q_0, br_{\mathrm{new}} \mapsto (1, q_0) \wedge (2, q_0)$
$q_0, \nu^{\mathrm{ro}} \mapsto (1, q_r) \wedge (1, q_0)$
$q_0, br_{\mathrm{if}} \mapsto (1, q_0) \wedge (2, q_0)$
$q_0, \mathtt{r} \mapsto (1, q_0)$
$q_0, \mathtt{c} \mapsto (1, q_0)$
$q_0, \star \mapsto \mathtt{t}$

$q_r, br_{\mathrm{if}} \mapsto (1, q_l) \wedge (2, q_r)$
$q_r, \mathtt{r} \mapsto (1, q_r)$
$q_r, \mathtt{c} \mapsto \mathtt{t}$
$q_r, \star \mapsto \mathtt{t}$

$q_l, \mathtt{r} \mapsto (1, q_l)$
$q_l, br_{\mathrm{if}} \mapsto (1, q_l) \wedge (2, q_l)$
$q_l, \mathtt{c} \mapsto \mathtt{t}$
$q_l, \star \mapsto \mathtt{t}$

[5] Fairness assumptions can take various (not necessarily equivalent) forms. Alternatively we can say that a path is *fair* if it satisfies the *compassion constraint* $\{(\mathtt{r}, \mathtt{c})\}$ in the sense of Pnueli [13] i.e. if r occurs infinitely often, so does c.

The state q_r (respectively q_l) signals a node reachable by a path that, from some point onwards, *only* takes (respectively, does *not* only take) the right branch of br_{if}. Weakness is given by $\{q_0, q_r\}$ (accepting) $> \{q_l\}$ (rejecting).

Example 8. Using the same terminals as the preceding example, consider the CTL formula $\varphi_3 := AG\,(\nu^{\mathrm{ro}} \Rightarrow EX(EX\ \mathtt{c}))$. Intuitively φ_3 says that the input program processes empty files ("readonly" resource) correctly. E.g. $[\![G_{D_2}]\!]$ satisfies φ_3. We can encode φ_3 in the following non-conjunctive AWT $\mathcal{A}_3$ with a trivial acceptance condition.

$$
\begin{array}{lll}
q_m, br_{\mathrm{new}} \mapsto (1,q_m) \wedge (2,q_m) & q_m, \nu^{\mathrm{ro}} \mapsto (1,q_m) \wedge (1,q_0) & q_m, \mathtt{r} \mapsto (1,q_m) \\
q_m, \mathtt{c} \mapsto (1,q_m) & q_m, br_{\mathrm{if}} \mapsto (1,q_m) \wedge (2,q_m) & q_m, \star \mapsto \mathtt{t} \\
q_0, br_{\mathrm{if}} \mapsto (1,q_1) \vee (2,q_1) & q_1, \mathtt{c} \mapsto \mathtt{t} &
\end{array}
$$

4 AWT model-checking by type inference

An intersection type system for AWT Fix an AWT $\mathcal{A} = \langle \Sigma, Q, \delta, q_I, \Omega \rangle$. We construct an intersection type system $\boldsymbol{\lambda}^{\mathcal{A}}_{\wedge}$ parameterised by $\mathcal{A}$ and introduce a game-playing notion of *typability* for recursion schemes. The type system characterises the automaton $\mathcal{A}$ in that a recursion scheme G is typable in $\boldsymbol{\lambda}^{\mathcal{A}}_{\wedge}$ if, and only if, $[\![G]\!]$ is accepted by $\mathcal{A}$. Let $q \in Q$. We define

$$
\begin{array}{llll}
\textit{Types} & \theta & ::= & q \mid \tau \to \theta \\
\textit{Conjunctive Types} & \tau & ::= & \bigwedge\{\theta_1, \cdots, \theta_k\} \quad (k \geq 0)
\end{array}
$$

Each type can be written uniquely as $\theta = \tau_1 \to \cdots \to \tau_n \to q$ for some $n \geq 0$; we call q the *state* of θ, written $state(\theta)$. We write $\bigwedge_{i=1}^{k} \theta_i$ for $\bigwedge\{\theta_1, \cdots, \theta_k\}$, and $\top$ for $\bigwedge \varnothing$. Given a priority map $\Omega : Q \to \mathbb{N}$ on Q, we extend it to all types by $\Omega(\tau \to \theta) := \Omega(\theta)$. We introduce a notion of *well-kindedness* of type. We define the relations $\tau ::_{\mathrm{ct}} A$ and $\theta :: A$, which should be read "τ is a conjunctive type of kind A" and "θ is a type of kind A" respectively, by induction over the following rules:

$$
\frac{}{q_i :: o} \qquad \frac{\tau ::_{\mathrm{ct}} A \quad \theta :: B}{\tau \to \theta \ :: \ A \to B} \qquad \frac{\theta_i :: A \quad \text{for each } i \in \{1, \cdots, k\}}{\bigwedge_{i=1}^{k} \theta_i \ ::_{\mathrm{ct}} \ A}
$$

Note that there are only finitely many well-kinded types of each kind.

Intuitively, a term $\lambda x.s$ of type $(q_1 \wedge q_2) \to q$ is a function that takes a tree-argument t which can be accepted from the states q_1 and q_2, and returns a tree $s[t/x]$ which can be accepted from state q. By abuse of notation, we write $q \leq q'$ to mean $q \in Q_i$ and $q' \in Q_j$ and $Q_i \leq Q_j$. An AWT processing the tree $s[t/x]$ will read the root with state q before reading the respective roots of subtrees with states q_1 and q_2 respectively. It follows that $q_1 \leq q$ and $q_2 \leq q$. This motivates a notion of *consistency* of type which is defined as follows. Each $q \in Q$ is *consistent*; $\bigwedge_{i=1}^{k} \theta_i \to \theta$ is *consistent* just if θ is consistent, and for each i, θ_i is consistent, and $state(\theta_i) \leq state(\theta)$. We say that a type θ is *well-formed* if (i) θ is *well-kinded* i.e. $\theta :: A$ for some kind A, and (ii) θ is consistent.

A *type judgement* of the system $\boldsymbol{\lambda}^{\mathcal{A}}_{\wedge}$ has the form $\Gamma \vdash t : \theta$, where t is a λ-term (we treat non-terminals as variables), and Γ, called a *type environment*, is a set of bindings of the form $x : \theta$. Note that Γ may contain multiple bindings of the same variable. We

$$\frac{\theta \text{ is well-formed}}{x : \theta \vdash x : \theta} \quad \text{(VAR)}$$

$$\frac{\{(i, q_{ij}) \mid 1 \leq i \leq n, 1 \leq j \leq k_i\} \text{ satisfies } \delta(q, f)}{\varnothing \vdash f : \bigwedge_{j=1}^{k_1} q_{1j} \to \cdots \to \bigwedge_{j=1}^{k_n} q_{nj} \to q} \quad \text{(TERM)}$$

$$\frac{\Gamma_0 \vdash s : \bigwedge_{i=1}^{k} \theta_i \to \theta \qquad \Gamma_i \vdash t : \theta_i \text{ for each } i \in \{1, \cdots, k\}}{\Gamma_0 \cup \Gamma_1 \cup \cdots \cup \Gamma_k \vdash s\, t : \theta} \quad \text{(APP)}$$

$$\frac{\Gamma, x : \bigwedge_{i \in I} \theta_i \vdash t : \theta \qquad I \subseteq J \qquad \text{for } j \in J \setminus I,\ state(\theta_j) \leq state(\theta), \text{ and } \theta_j \text{ is well-formed}}{\Gamma \vdash \lambda x.t : \bigwedge_{j \in J} \theta_j \to \theta} \quad \text{(ABS)}$$

Fig. 1. Rules defining valid judgements of $\boldsymbol{\lambda}^{\mathcal{A}}_{\wedge}$.

write $\Gamma, x : \bigwedge_{i=1}^{k} \theta_i$ as a shorthand for $\Gamma \cup \{x : \theta_1, \cdots, x : \theta_k\}$ where $k \geq 0$ and x is assumed *not* to occur in Γ. (Thus $\Delta = \Gamma, x : \bigwedge \varnothing$ means that Δ contains no bindings of x.) Valid type judgements of $\boldsymbol{\lambda}^{\mathcal{A}}_{\wedge}$ are defined by induction over the rules in Figure 4.

Remark 1. (i) (ABS) is the only place where weakening may be introduced. (ii) The rule (APP) is not multiplicative: it is possible that $\Gamma_i \cap \Gamma_j \neq \varnothing$. Thus there is implicit contraction in the type system.

Lemma 1. *For every valid type judgement $\Gamma \vdash s : \theta$, (i) θ is well-formed, and (ii) for each binding $x : \theta'$ in Γ, θ' is well-formed, and $state(\theta') \leq state(\theta)$.*

Typing a recursion scheme in $\boldsymbol{\lambda}^{\mathcal{A}}_{\wedge}$. Following Kobayashi and Ong [6], we derive a weak Büchi game from the type system $\boldsymbol{\lambda}^{\mathcal{A}}_{\wedge}$. Let $G = \langle \Sigma, \mathcal{N}, \mathcal{R}, S \rangle$ be a recursion scheme. We say that a binding $F : \theta$ is *G-consistent* if $F \in \mathcal{N}$ and for some kind A, $F : A$ and $\theta :: A$. A type environment Γ is *G-consistent* if every binding in it is G-consistent.

Definition 3. (i) Given an AWT $\mathcal{A} = \langle \Sigma, Q, \delta, q_I, \Omega \rangle$ with a partial order $\leq$ over a partition $\{Q_1, \cdots, Q_n\}$ of Q, and a recursion scheme $G = \langle \Sigma, \mathcal{N}, \mathcal{R}, S \rangle$, we define the associated (weak Büchi) game $\mathbb{G}(G, \boldsymbol{\lambda}^{\mathcal{A}}_{\wedge}) = \langle V_A, V_E, (S : q_I), E, \Omega' \rangle$ where

$$\begin{aligned} V_E &:= \{(F : \theta) \mid (F : \theta) \text{ is } G\text{-consistent}\} \\ V_A &:= \{(\Gamma, q) \mid \exists\, (F : \theta)\,.\, \Gamma \vdash \mathcal{R}(F) : \theta \ \wedge\ q = state(\theta)\} \\ E &:= \{((F : \theta), (\Gamma, state(\theta))) \mid \Gamma \vdash \mathcal{R}(F) : \theta\} \cup \{((\Gamma, q), (F : \theta)) \mid (F : \theta) \in \Gamma\} \end{aligned}$$

$(S : q_I) \in V_E$ is the initial node, and the priority function Ω' maps $(F : \theta)$ to $\Omega(state(\theta))$, and (Γ, q) to $\Omega(q)$. The underlying directed graph has node-set $V_A \cup V_E$ and edge-set E.

(ii) We say that G is *well-typed* in $\boldsymbol{\lambda}^{\mathcal{A}}_{\wedge}$ if Éloïse has a winning strategy in $\mathbb{G}(G, \boldsymbol{\lambda}^{\mathcal{A}}_{\wedge})$.

Lemma 2. *The construction in Definition 3(i) yields a weak Büchi game.*

Remark 2. The second component q of an A-node (Γ, q) is introduced so that the resultant edge relation is monotone w.r.t. $\leq$.

The game $\mathbb{G}(G, \boldsymbol{\lambda}^{\mathcal{A}}_{\wedge})$ may be understood intuitively as follows. The game begins by Abelard challenging Éloïse to prove $S : q_I$ i.e. S has type q_I. As the opening move, Éloïse produces an environment Γ (precisely, a move (Γ, q), for some q) such that $\Gamma \vdash \mathcal{R}(S) : q_I$. Abelard responds by choosing a binding from Γ, say, $F : \theta$, and challenges Éloïse to prove $F : \theta$. Éloïse then responds with an environment Γ' such that $\Gamma' \vdash \mathcal{R}(F) : \theta$. The play either alternates between Éloïse and Abelard indefinitely, or ends when one of the players is unable to move. Éloïse wins a play if at some point, it produces the empty type environment as a move (so that Abelard is unable to pick a binding), or if the play is infinite, and 0 is an infinitely-occurring priority (i.e. it is trapped in an accepting Q_i). See Appendix C for a linear-time algorithm for solving weak Büchi games.

By adapting the proof of the main result of Kobayashi and Ong [6], we obtain the following characterisation.

Theorem 3. *Given a recursion scheme G and an AWT $\mathcal{A}$, $[\![G]\!]$ is accepted by $\mathcal{A}$ if, and only if, G is well-typed in $\boldsymbol{\lambda}^{\mathcal{A}}_{\wedge}$ (i.e. Éloïse has a winning strategy in $\mathbb{G}(G, \boldsymbol{\lambda}^{\mathcal{A}}_{\wedge})$).*

The algorithms $\boldsymbol{\Xi_{\mathcal{A}}}$ and $\boldsymbol{\widehat{\Xi}_{\mathcal{A}}}$ Fix an instance of the model checking problem: a recursion scheme G and an AWT $\mathcal{A}$. Based on Theorem 3, we present a type-inference approach to decide whether $\mathcal{A}$ accepts $[\![G]\!]$. Our task is to construct, and then solve, the weak Büchi game $\mathbb{G}(G, \boldsymbol{\lambda}^{\mathcal{A}}_{\wedge})$. Building the underlying game graph naïvely (e.g. by constructing the edge-set explicitly) would be prohibitively expensive. The number of Éloïse-nodes in tandem with types is subject to an n-tall tower-of-exponential growth, where n is the order of the recursion scheme. Our solution stems from the observation that to solve a given game $\mathbb{G}(G, \boldsymbol{\lambda}^{\mathcal{A}}_{\wedge})$, we need only consider a small, reachable part (from the start node) of the underlying game graph, namely, a certain subgraph restricted to nodes that describe the computation determined by the problem instance $(G, \mathcal{A})$.

We present a semi-algorithm $\Xi_{\mathcal{A}}$ (part of the decision procedure $\widehat{\Xi}_{\mathcal{A}}$), which returns "YES" if $\mathcal{A}$ accepts $[\![G]\!]$. $\Xi_{\mathcal{A}}$ comprises two stages:

- *Stage 1*. Build a representation of all run-trees of $\mathcal{A}$ over $[\![G]\!]$; if there is none, we conclude that $[\![G]\!]$ is not accepted by $\mathcal{A}$. Stage 1 is in turn organised into three substages, which are similar in outline to Kobayashi's algorithm [7]; we point out the differences in the following.
- *Stage 2*. Derive a game, which will be considerably smaller than $\mathbb{G}(G, \boldsymbol{\lambda}^{\mathcal{A}}_{\wedge})$, in which Éloïse can play only those strategies that correspond to *well-defined* run-trees, so that Éloïse has a winning strategy in this game if, and only if, one of these run-trees is accepting.

A pseudocode presentation of algorithm $\Xi_{\mathcal{A}}$ is given in Figure 2.

Write $\overline{\mathcal{A}}$ as the complement automaton of $\mathcal{A}$ i.e. $L(\overline{A}) = \overline{L(\mathcal{A})}$. We define $\widehat{\Xi}_{\mathcal{A}}$ as the algorithm that, given input G, dovetails the computation of $\Xi_{\mathcal{A}}(G)$ and $\Xi_{\overline{\mathcal{A}}}(G)$, returning the outcome of the former, but the *negation* of the outcome of the latter, whichever terminates first. In the following we explain the various stages of $\Xi_{\mathcal{A}}$.

Initialisation $\mathcal{C} :=$ initial configuration graph
Stage_1.1% ***Expand configuration graph*** $\mathcal{C}$
  $count := 0$;
  while ($count < MAX$ & $\mathcal{C}$ has open node)
    do $N :=$ an open node;
      try { $\mathcal{C} := expand(\mathcal{C}, N)$} *catch* (NO_RUNTREE) { *return* NO };
Stage_1.2% ***Extract environment*** Γ ***from*** $\mathcal{C}$
  $\Gamma := ElimTE(\Gamma_{\mathcal{C}})$;
Stage_1.3% ***Compute*** $\mathcal{F}_{\mathrm{nw}}$***-fixpoint from*** Γ
  while $\Gamma \neq \mathcal{F}_{\mathrm{nw}}(\Gamma)$ *do* $\Gamma := \mathcal{F}_{\mathrm{nw}}(\Gamma)$;
  if $S : q_I \notin \Gamma$ *then goto* Stage_1.1
Stage_2% ***Solve weak Büchi game*** $\mathbb{G}^-(\Gamma)$
  if Eloïse has a winning strategy in $\mathbb{G}^-(\Gamma)$ *then return* YES *else goto* **Stage_1.1**

Fig. 2. The Semi-Algorithm $\Xi_{\mathcal{A}}$

Stage 1.1: Expansion of the configuration graph The key insight behind Kobayashi's algorithm is that the desired type environment may be obtained by repeating a fixpoint computation, using progressively larger seed values, all of which are *small* because they contain only (over-approximations of) bindings relevant to the problem instance $(G, \mathcal{A})$. These bindings (of non-terminals) are extracted from *configuration graphs*, which are a representation of an appropriate superposition of the unique[6] run-tree of $\mathcal{A}$ on a finite (and in general partial) evaluation trace of the recursion scheme G.

We generalise Kobayashi's configuration graph [7] to the setting of alternating automata $\mathcal{A}$. Formally a configuration graph is a directed graph, whose nodes are labelled by triples of the form $\langle t, q, g\rangle$ where t is a ground-kind applicative term generated from the terminals and non-terminals, q is a state of the automaton and the flag g indicates whether the node is *open* or *closed*. Edges of a graph are labelled by either 0 or pairs of numbers. The initial configuration graph is a singleton graph whose node, the *root node*, is labelled by $\langle S, q_I, open\rangle$.

Procedure $expand(\mathcal{C}, N)$. Suppose N is an open node, with label $\langle t, q, open\rangle$, of a configuration graph $\mathcal{C}$. The procedure $expand(\mathcal{C}, N)$ constructs the *expansion* of $\mathcal{C}$ at N, which is the graph obtained from $\mathcal{C}$ by replacing the flag of N by $closed$, and adding nodes and edges as follows.

(1) *Case*: $t = f\, t_1 \cdots t_m$. Suppose the set of minimal sets satisfying $\delta(f, q)$ is

$$\{\{(j, q^i_{jk}) \mid 1 \leq j \leq m, 1 \leq k \leq r_j\} \mid 1 \leq i \leq l\}$$

where $l \geq 0$. We assume that each node N, whose label contains a term headed by a terminal symbol – we call such a node *terminal-headed* – has a counter, $N.c$, which is initialized to l. If $\delta(f, q)$ is false (i.e. $l = 0$), then do $backPropagate(N)$; otherwise, for each $1 \leq i \leq l$, $1 \leq j \leq m$ and $1 \leq k \leq r_j$, let N' be the node labelled by $\langle t_j, q^i_{jk}, g'\rangle$ (we add such a node to $\mathcal{C}$ if it does not exist); add an edge from N to N' labelled by

[6] because $\mathcal{A}$ is assumed to be deterministic in [7]

(i, j), where i indicates the non-deterministic choice, and j refers to the j-th argument of the terminal f.

(2) *Case*: $t = F\,t_1 \cdots t_m$ and $\mathcal{R}(F) = \lambda\overline{x}.s$. Let N' be the node that is labelled by $\langle s[\overline{t}/\overline{x}], q, g'\rangle$ (we add such a node to $\mathcal{C}$ if it does not exist). Add a 0-labelled edge from N to N'.

Procedure $backPropagate(N)$
if there is a path[7] from the root to N that does not meet a terminal-headed node
then throw NO_RUNTREE
for each i' and each terminal-headed node N' that connects to N via a path whose trace matches $(i', -)\,0^*$ *do*
{ $N'.c := N'.c - 1$;
for each j and (i', j)-successor N'' of N' *do* delete (i', j)-edge from N' to N'';
if $N'.c = 0$ *then* $backPropagate(N')$ }

A configuration graph is said to be *closed* if all its nodes are (flagged as) closed.

Stage 1.2: Extraction of type environment $\Gamma_{\mathcal{C}}$ For each node N with label $\langle t, q, g\rangle$, and for each prefix u of t, we define a set $\boldsymbol{\tau}_{u,N}$ of types by induction on the kind of u as follows.

(1) *Case*: u has kind o. Then $t = u$; set $\boldsymbol{\tau}_{u,N} := \{q\}$.

(2) *Case*: u has kind $A \to B$. Then $t = u\,v\,\overline{v}$, where v and $u\,v$ have kinds A and B respectively. Let $\{N_1, \cdots, N_p\}$ be the set of nodes that are reachable from N in $\mathcal{C}$, and have labels of the form $\langle v\,\overline{w}, q', g'\rangle$ where v "originates" from the copy of v in N. Let $S_1, \cdots, S_l \subseteq \{N_1, \cdots, N_p\}$ be the maximally *compatible* subsets, in the sense that two nodes N_i and $N_{i'}$, reachable from N by paths (see footnote 7) π and π' respectively, are *compatible* just if whenever there exists π_1 such that $\pi_1\,(i, j) \leq \pi$ and $\pi_1\,(i', j') \leq \pi'$ then $i = i'$. I.e. each S_i consists of nodes reachable from N following compatible Éloïse choices. Then $\boldsymbol{\tau}_{u,N}$ consists of types θ defined as follows: for each $1 \leq i \leq l$ where $S_i = \{N_{i1}, \cdots, N_{ir_i}\}$, for each $\theta_{ij} \in \boldsymbol{\tau}_{v,N_{ij}}$ and $\theta' \in \boldsymbol{\tau}_{u\,v,N}$,

$$\theta \;:=\; \bigwedge\{\theta_{i1}, \cdots, \theta_{ir_i}, \alpha\} \to \theta'$$

where α is a fresh type variable, if u occurs in an open node (including N) reachable from N via a path compatible with S_i; otherwise set θ to $\bigwedge\{\theta_{i1}, \cdots, \theta_{ir_i}\} \to \theta'$. Given a configuration graph $\mathcal{C}$, we define its type environment $\Gamma_{\mathcal{C}}$ as

$$\Gamma_{\mathcal{C}} := \{(F : \tau) \mid \text{some } \mathcal{C}\text{-node } N \text{ has label } \langle F\,t_1 \cdots t_m, q, g\rangle, \tau \in \boldsymbol{\tau}_{F,N}\}$$

Type variables are subsequently eliminated using Kobayashi's $ElimTE$ [7]. Even though a closed configuration graph $\mathcal{C}$ may be infinite, $\Gamma_{\mathcal{C}}$ is necessarily finite.

Example 9. Recall Example 7. After expanding 26 nodes, our implementation THORS computes a $\Gamma_{\mathcal{C}}$ containing the following bindings for the non-terminal G:

$$\top \to (\top \to q_0 \to q_0) \to q_0 \quad \top \to ((q_0 \to q_0) \to q_0 \to q_0) \to q_0$$
$$\top \to ((\top \to q_l) \to \top \to q_l) \wedge ((q_r \to q_r) \to q_r \to q_r) \to q_r$$

[7] A *path* (in a configuration graph) is a sequence that matches the regular expression $\langle node\rangle\,(\langle edge\_label\rangle\,\langle node\rangle)^*$; its *trace* is the subsequence of edge labels.

Stage 1.3: Computation of $\mathcal{F}_{\mathrm{nw}}$-fixpoint from seed Γ Henceforth let Γ and Δ range over G-consistent type environments. We define a monotone function on type environments

$$\mathcal{F}_{\mathrm{nw}}(\Gamma) := \{(F : \theta) \in \Gamma \mid \exists \Delta \subseteq \Gamma \,.\, \Delta \vdash \mathcal{R}(F) : \theta\}$$

(The subscript of $\mathcal{F}_{\mathrm{nw}}$ emphasises the "non-weakening" nature of the system $\boldsymbol{\lambda}^{\mathcal{A}}_{\wedge}$.) The following is a straightforward consequence of the Knaster-Tarski Fixpoint Theorem.

Lemma 3. *Let Γ be a G-consistent type environment. Writing $\mathcal{F}^i_{\mathrm{nw}}(\Gamma)$ to mean the i-fold application of $\mathcal{F}_{\mathrm{nw}}$ to Γ, $\mathcal{F}_{\mathrm{nw}}(\cdots(\mathcal{F}_{\mathrm{nw}}(\Gamma)))$, we have a descending sequence*

$$\mathcal{F}^0_{\mathrm{nw}}(\Gamma) := \Gamma \supseteq \mathcal{F}^1_{\mathrm{nw}}(\Gamma) \supseteq \mathcal{F}^2_{\mathrm{nw}}(\Gamma) \supseteq \cdots \supseteq \mathcal{F}^i_{\mathrm{nw}}(\Gamma) \supseteq \mathcal{F}^{i+1}_{\mathrm{nw}}(\Gamma) \supseteq \cdots.$$

There exists an $n \geq 0$ such that $\mathcal{F}^n_{\mathrm{nw}}(\Gamma) = \mathcal{F}^{n+1}_{\mathrm{nw}}(\Gamma)$; let κ be the least such n. Then $\mathcal{F}^\kappa_{\mathrm{nw}}(\Gamma)$—written $Fix_\Gamma(\mathcal{F}_{\mathrm{nw}})$—is the largest fixpoint of $\mathcal{F}_{\mathrm{nw}}$ contained in Γ i.e. $Fix_\Gamma(\mathcal{F}_{\mathrm{nw}})$ is the largest Γ' such that $\Gamma' \subseteq \Gamma$, and for every $(F : \theta) \in \Gamma'$ there exists $\Delta \subseteq \Gamma'$ such that $\Delta \vdash \mathcal{R}(F) : \theta$.

Example 10. Recall again Example 7. THORS computes $\Gamma = Fix_{\Gamma_C}(\mathcal{F}_{\mathrm{nw}})$ with bindings of G exactly as in Example 9. Furthermore, the fixpoint contains $S : q_0$, so it witnesses the existence of a run tree of $\mathcal{A}_2$ over $[\![G_{D_2}]\!]$.

Stage 2: Construction and solution of the game $\mathbb{G}^-(\Gamma)$ The weak Büchi game $\mathbb{G}^-(\Gamma) = \langle V_A, V_E, E, v_0, \Omega' \rangle$ is constructed using the fixpoint Γ obtained from Stage 1.3 and an auxiliary function $TEnv_\Gamma$, whose domain of definition is Γ, which maps $F : \theta$ to $\{\Delta \subseteq \Gamma \mid \Delta \vdash \mathcal{R}(F) : \theta\}$. Recall Definition 3. As shown in Lemma 2, the game $\mathbb{G}^-(\Gamma)$ will inherit the weakness of the specification automaton so that the nodes of the game graph may be partitioned according to a linear order. The game may be solved in linear time by proceeding up the ordering. Within each partition V_i, each terminal node v is marked $\mathbf{t}$ or $\mathbf{f}$ according to whether $v \in V_A$ or V_E. The effect of this is then propagated such that any predecessor v that is now fully determined[8] are marked and propagated in turn. After this stage, remaining nodes are marked according to the acceptance condition of V_i, and the effect propagated as before. Processing of V_i is now complete. Éloïse has a winning strategy from v_0 just if v_0 is marked $\mathbf{t}$ after processing of all V_i. See Appendix C for a presentation of the linear-time algorithm.

Remark 3. (i) In case $\mathcal{A}$ is *conjunctive*, Stage 2 of $\Xi_{\mathcal{A}}$ may be modified such that the "*else goto* **Stage_1.1**" branch is replaced by "*return* NO" (this indicating that the single possible run-tree is not accepting). Furthermore this modified version is guaranteed to terminate [7] without running $\Xi_{\mathcal{A}}$ and $\Xi_{\overline{\mathcal{A}}}$ in parallel.

(ii) In case $\mathcal{A}$ is a co-trivial automaton, $[\![G]\!]$ is accepted by $\mathcal{A}$ if, and only if, Éloïse has a strategy that forces every play to be finite, because, by construction, every node of the game graph is rejecting.

(iii) In case $\mathcal{A}$ is a trivial automaton, every node of the corresponding weak Büchi game $\mathbb{G}^-(\Gamma)$ is accepting. Thus Stage 2 is redundant; it can be replaced by the single command "*return* YES".

[8] $v \in V_E$ (resp. V_A) with *any* child $\mathbf{t}$ (resp. $\mathbf{f}$) or *all* children $\mathbf{f}$ (resp. $\mathbf{t}$)

5 Correctness, complexity and optimisations

Correctness Fix an AWT $\mathcal{A}$ and a HORS G. We show the following in Appendix D:

Theorem 4 (Correctness).

(i) If $\mathcal{A}$ accepts $[\![G]\!]$, then $\Xi_{\mathcal{A}}$ terminates on input G and returns YES.
(ii) If $\mathcal{A}$ rejects $[\![G]\!]$, then if $\Xi_{\mathcal{A}}$ terminates on input G then it returns NO.

Hence $\widehat{\Xi}_{\mathcal{A}}$ is a procedure for deciding if $\mathcal{A}$ accepts $[\![G]\!]$.

Example 11. Recall once more Example 7. The game graph for the resulting game $\mathbb{G}^{-}(\Gamma)$, as computed by THORS, is shown in Figure 3. Éloïse nodes are drawn as diamonds and Abelard nodes as squares. Environments and types are omitted for brevity. There are two partitions: an accepting partition, where all types end in state q_0 or q_r, and a rejecting partition, where all types end in state q_l. Starting from S, the play is either infinite within the accepting partition, or finite and ends with Abelard unable to play from the empty environment. In either case, Éloïse wins.

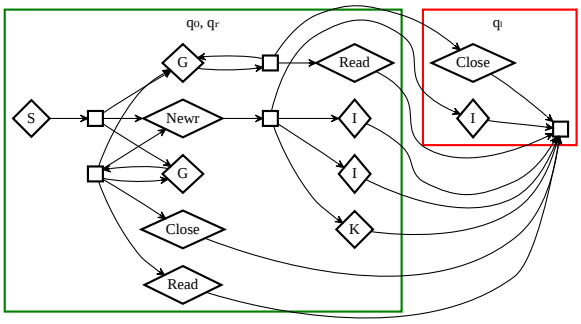


Fig. 3. The game $\mathbb{G}^{-}(\Gamma)$ resulting from Example 7.

Theorem 5 (Complexity). *Let G be an order-n recursion scheme, and $\mathcal{A}$ be an APT. We consider the problem of whether $[\![G]\!]$ is accepted by $\mathcal{A}$.*

(i) In case $\mathcal{A}$ is alternating weak, or alternating trivial, or alternating co-trivial, the problem is n-EXPTIME complete.
(ii) In case $\mathcal{A}$ is deterministic weak, or deterministic trivial, or deterministic co-trivial, the problem is $(n-1)$-EXPTIME decidable; it is also $(n-1)$-EXPTIME hard in the first two cases.

The cases of AWT and DWT are due to Kobayashi and Ong [14] and Kobayashi [7] respectively. See Appendix E for a proof of the Theorem.

Remark 4. The decision problem FINITE (Is the tree generated by a given order-n recursion scheme finite?) is in $(n-1)$-EXPTIME, and is conjectured to be $(n-1)$-EXPTIME hard [14]. It follows from König's Lemma that FINITE reduces to the deterministic co-trivial case of the acceptance problem of Theorem 5.

Optimisation	*Memory*	*Time for* $\mathbb{T}_\Gamma$	*Game Nodes*	*Time Solving*	*Total Time*
None	>4GB	—	—	—	—
Canonisation	2.3GB	4168	359	.100	4169
Bitsets	301MB	200	359	.107	201
Environment Minimisation	4MB	.049	53	.001	.121
Subtype Minimisation	4MB	.049	53	.001	.123
Subtype but not Environment Minimisation	4MB	.088	91	.002	.162

Table 1. The effects of various optimisations on $\mathbb{T}_\Gamma$ calculation. Times are in seconds.

Optimisation Given a type environment Γ and a term s, we use an efficient type-inference scheme (see Appendix F) to compute the set of valid type judgements for s under subsets of Γ, which we denote $\mathbb{T}_\Gamma(s)$. Our initial implementation of $\mathbb{T}_\Gamma$ was slow. However, we discovered a number of optimisations that improved performance considerably in the examples we tested. The improvement on one example (order5-v-dwt) can be seen from the results in Table 1, which shows the *cumulative (orders of magnitude)* improvement in memory usage and speed of THORS with different optimisations enabled, as well as the size of the resulting game graph and time taken to solve it. See Appendix G for a full discussion of the optimisation techniques.

6 Implementation and experiments

We have constructed THORS (Types for Higher-Order Recursion Schemes), a model checker for recursion schemes, which can be tested via a web-interface at `https://mjolnir.comlab.ox.ac.uk/thors/`. THORS implements the 2-stage algorithm presented in Section 4. A number of features are worth mentioning.

(i) When expanding the configuration graph in Stage 1.1, a heuristic is used when determining which node should be chosen to be expanded. Open nodes are selected according to a breadth-first strategy. However, since we have found it most productive to search deeper in the graph, for each node selected in this way, a bounded (above and below) number of open descendants are expanded according to a depth-first search.

(ii) We have implemented the two optimisations of Kobayashi [7, Section 3.3] (use of canonical types and efficient computation of $Elim$).

(iii) The computation of the $\mathcal{F}_{\mathrm{nw}}$-fixpoints in Stage 1.2 uses the optimised type system with subtyping $\boldsymbol{\lambda}^{\mathcal{A}}_{\wedge,\leq}$ of Appendix F.

(iv) We have implemented the collapsing optimisation for the co-trivial case and the optimised computation of $\mathbb{T}_\Gamma$ described in Section 5; see Appendix G for details.

We have evaluated the tool using eight example programs which are presented in Table 2. The columns “O”, “R” and “Q” indicate the order of the recursion scheme, the number of rules in the scheme and the number of states in the property automaton respectively. The columns “Time”, “Nodes”, “Game” record the elapsed time until termination (in milliseconds), the number of nodes in the configuration graph and the number of nodes in the game graph respectively. The column “Result” records whether

the property was satisfied (Y) or unsatisfied (N). The final column indicates the classification of the property automaton.

Example	*O*	*R*	*Q*	*Time*	*Nodes*	*Game*	*Result*	*Property*
D1	4	7	2	1	19	16	Y	Deterministic Weak
D2	4	7	3	1	26	17	Y	Conjunctive Weak
D2-ex	4	7	3	1	26	-	Y	Alternating Trivial
intercept	4	15	2	35	200	31	Y	Conjunctive Weak
imperative	3	6	3	129	200	17	Y	Deterministic Weak
boolean2	2	15	1	1	13	-	Y	Deterministic Trivial
order5-2	5	9	4	19	200	37	N	Deterministic Co-trivial
lock1	4	12	3	2	32	32	Y	Deterministic Co-trivial
order5-v-dwt	5	11	4	163	400	53	Y	Deterministic Weak
lock2	4	11	4	109	800	-	Y	Deterministic Trivial
example2-1	1	2	2	190	200	-	Y	Deterministic Trivial

Table 2. Experimental data for AWT model checking.

D1, D2 and D2-ex These three examples are detailed in Examples 3, 7 and 8 respectively. The "D2-ex" example has an empty entry for "Game" due to the fact that although the automaton is alternating (and non-conjunctive), it requires only a trivial acceptance condition and hence it is sufficient to find any run-tree.

intercept This example is a model of an OCaml program which takes input on one incoming network socket and mirrors this data to a second output socket. The desired behaviour is that whenever the first socket is closed, the second socket must eventually close. More details can be found in Appendix H.

imperative This example is a translation of the motivating example in Cook, Koskinen and Vardi [15]. It is a simple C-program containing two integer variables x and n. The property of interest is given in CTL as $AG[(x = 1) \Rightarrow AF(x = 0)]$. Although C-programs are, by definition, first-order, the transformation process (in particular the CPS-transform) artificially raises the order by 3. However, it is notable that the transformation is considerably shorter than that required by [15] and the raising of the order does not seem to hinder the tool. More details can be found in Appendix H.

boolean2 This example is a translation of a Boolean program which is obtained from a C program by predicate abstraction using 3 predicates. We check a reachability property. More details can be found in Appendix H.

order5-2 and lock1 These examples are both taken taken from [7] but, rather than checking the trivial properties detailed there, we instead check the corresponding co-trivial property. The former and also 'order5-v-dwt' are designed to evaluate the performance of the tool on HORS which are, according to the worst-case time-complexity, the most likely to cause efficiency problems.

order5-v-dwt This example is due to Kobayashi[9]. The scheme generates resources, non-deterministically reading and closing them in a (hopefully) sensible fashion. We require that if a file is ever read, then it is certainly closed.

lock2 and example2-1 These examples are taken from [7] and have been provided to give an indication of performance relative to the TRecS tool introduced therein. Since we do not know of any other tools that can handle alternating or conjunctive

[9] Personal communication (7 April 2010)

automata with possibly non-trivial acceptence conditions, this sample of deterministic trivial properties is the only comparison we are able to provide.

Further directions and conclusions We have formulated the resource usage verification problem (in accord with a parity resource automaton) for RUL and shown that it reduces to the APT model checking problem for HORS. We have developed an algorithm to automate its solution for the practically-relevant case of AWT. Our implementation shows the algorithm to perform surprisingly well on small examples despite the inherent worst-case complexity. In future work, we plan to extend our tool to handle full Büchi games, which will allow for specifications in CTL* and perform a detailed study of the efficiency of our implementation.

References

1. Kobayashi, N.: Types and higher-order recursion schemes for verification of higher-order programs. In: Proceedings of POPL 2009, ACM Press (2009) 416–428
2. Ong, C.H.L.: On model-checking trees generated by higher-order recursion schemes. In: LICS'06, Computer Society Press (2006) 81–90 Long version (55 pp.) downloadable at `users.comlab.ox.ac.uk/luke.ong/`.
3. Igarashi, A., Kobayashi, N.: Resource usage analysis. In: POPL. (2002) 331–342
4. Fischer, M.J., Ladner, R.E.: Propositional dynamic logic of regular programs. JCSS **18** (1979) 194–211
5. Kupferman, O., Vardi, M.Y., Wolper, P.: An automata-theoretic approach to branching-time model checking. J. ACM **47**(2) (2000) 312–360
6. Kobayashi, N., Ong, C.H.L.: A type theory equivalent to the modal mu-calculus model checking of higher-order recursion schemes. In: Proceedings of LICS 2009, IEEE Computer Society (2009)
7. Kobayashi, N.: Model-checking higher-order functions. In: PPDP'09, ACM (2009) 25–36
8. Kobayashi, N.: A practical linear-time algorithm for trivial automata model checking of higher order recursion schemes. Submitted to FoSSaCS'11 (2010)
9. Aehlig, K.: A finite semantics of simply-typed lambda terms for infinite runs of automata. Logical Methods in Computer Science **3** (2007) 1–23
10. Emerson, E.A., Jutla, C.S.: Tree automata, mu-calculus and determinacy. In: Proceedings of FOCS'91. (1991) 368–377
11. Kupferman, O., Vardi, M.Y.: Freedom, weakness, and determinism: From linear-time to branching-time. In: LICS. (1998) 81–92
12. Igarashi, A., Kobayashi, N.: Resource usage analysis. ACM Trans. Program. Lang. Syst. **27**(2) (2005) 264–313
13. Kesten, Y., Pnueli, A., Vardi, M.Y.: Verification by augmented abstraction: The automata-theoretic view. J. Comput. Syst. Sci. **62**(4) (2001) 668–690
14. Kobayashi, N., Ong, C.H.L.: Complexity of model checking recursion schemes for fragments of the modal mu-calculus. In: Proceedings of ICALP 2009, Springer-Verlag (2009)
15. Cook, B., Koskinen, E., Vardi, M.: Branching-time reasoning for infinite-state systems. Unpublished preprint. (2010)
16. Niwinski, D.: On fixed-point clones (extended abstract). In: ICALP. (1986) 464–473
17. Kupferman, O., Vardi, M.Y.: $\Pi_2 \cap \Sigma_2 \equiv$ AFMC. In: ICALP. (2003) 697–713
18. Kobayashi, N.: Model checking higher-order programs. preprint (2010)

19. Engelfriet, J.: Interated stack automata and complexity classes. Information and Computation **95** (1991) 21–75
20. Knapik, T., Niwiński, D., Urzyczyn, P.: Higher-order pushdown trees are easy. In: FOSSACS'02, Springer (2002) 205–222 LNCS Vol. 2303.
21. Durak, B.: `http://abaababa.ouvaton.org/caml/` (2010)

A Parity games, weak Büchi games and AFMC

A *parity game* is a tuple $\langle V_A, V_E, v_0, E, \Omega\rangle$ such that $E \subseteq V \times V$ is the edge relation of a directed graph whose node-set V is the disjoint union of V_A and V_E (A-nodes and E-nodes respectively); $v_0 \in V$ is the start node; and $\Omega : V \to \{0, \cdots, M-1\}$ assigns a priority to each node. A play consists in the players, Abelard and Éloïse, taking turns to move a token along the edges of the graph. During the play, if the token is on a node $v \in V_A$ (respectively $v \in V_E$), then Abelard (respectively Éloïse) chooses an edge $(v, v') \in E$ and moves the token onto v'. At the start of a play, the token is placed on v_0. Thus we define a *play* to be a finite or infinite path $\pi = v_0\, v_{n_1}\, v_{n_2} \cdots$ in the graph that starts from v_0. Suppose π is a maximal play. The winner of π is determined as follows.

- If π is finite, and it ends in a E-node (respectively A-node), then Abelard (respectively Éloïse) wins.
- If π is infinite, then Éloïse wins if π satisfies the *parity condition* i.e. the least priority that occurs infinitely often in the sequence $\Omega(v_0)\, \Omega(v_{n_1})\, \Omega(v_{n_2}) \cdots$ is even; otherwise Abelard wins.

An Éloïse-*strategy* (or *strategy*, for short) σ is a map from plays that end in a E-node to a node that extends the play. We say that a strategy σ is *winning* if Éloïse wins every (maximal) play π that *conforms* with the strategy (i.e. for every prefix π_0 of π that ends in a V_E-node, $\pi_0\ \sigma(\pi_0)$ is a prefix of π). Finally a strategy σ is *memoryless* (or history-free) if σ's action is determined by the last node of the play; formally, for all plays π_1 and π_2 that are consistent with σ, if their respective last nodes are the same V_E-node, then $\sigma(\pi_1) = \sigma(\pi_2)$. We say that a parity game is *solvable* (from v_0) if there is a winning strategy (for Éloïse) from v_0. It is known that if there is a winning strategy for a parity game, then there is also a memoryless winning strategy for the game.

Alternation-free modal mu-calculus (AFMC) and weak Büchi game

The *alternation depth* of a modal mu-calculus formula is the maximal depth of a chain of alternating least and greatest fixpoint operators. Since alternation depth is the major determinant of complexity, it is appropriate to use it to classify mu-calculus formulas. For $i \geq 0$, Σ_i (respectively Π_i) consists of formulas of alternation depth i in which the outermost fixpoint operator in the nested chain is least (respectively greatest) fixpoint. (See [16] for a definition.) Thus formulas of (fixpoint free) modal logic are $\Pi_0 = \Sigma_0$. The *alternation-free modal mu-calculus* (AFMC) consists of formulas with no alternation between least and greatest fixpoint operators. So AFMC is a natural closure of $\Sigma_1 \cup \Pi_1$, which is contained in $\Sigma_2 \cap \Pi_2$; in fact $\mathrm{AFMC} \equiv \Sigma_2 \cap \Pi_2$ (see [17]). The

well-known equivalence [10] between APT and modal mu-calculus specialises to one between AWT and AFMC.

Theorem 6 (Kupferman and Vardi [11]). *AWT and AFMC are equi-expressive, and there are linear translations between them.*

B Proof of Theorem 2 (Reduction)

In this section we develop a somewhat non-standard notion of weak bisimulation between labelled transition systems (viewed as game graphs), and prove a general result (Theorem 7) to the effect that Éloïse winning strategies of the parity game over one graph determine the Éloïse winning strategies of the parity game over the other graph. We then show that Theorem 2 is a corollary of Theorem 7.

Remark 5. Comparison with [18, Theorem 3.10] and the proof therein. Theorem 2 extends [18, Theorem 3.10] from safety properties (trivial automata) to all properties expressible in the modal mu-calculus (equivalently alternating parity automata). Our proof of the Theorem is both a simplification (we avoid the intermediate transition system of *extended run-time states* – see [18, Appendix A]) and an extension of Kobayashi's proof of Theorem 3.10; in particular Theorem 7 is a theorem of some generality.

Labelled transition system compatible with an alternating parity automaton

Fix a set Q of states and a set Dir of directions (or actions). A *labelled transition system over* (Q, Dir) (or (Q, Dir)-LTS, or simply LTS) is a tuple

$$\mathcal{T} \;=\; \langle C, \langle \xrightarrow{d} \mid d \in Dir \cup \{\epsilon\}\rangle, I, state, \rho\rangle$$

where C is a set of configurations, $\xrightarrow{d} \subseteq C \times C$ is a labelled transition relation with d ranging over $Dir \cup \{\epsilon\}$, $I \subseteq C$ is a set of start configurations, and $state : C \longrightarrow Q$ and $\rho : C \longrightarrow \Sigma$ are respectively the state and node-labelling maps. For convenience, we sometimes write elements of C as pairs (q, c) where $q = state(c)$; we refer to q as the *state*, and c the *code*.

Henceforth we assume that LTSs are *deterministic*, meaning that they are

(i) *deterministic qua* edge-labelled directed graphs: for every $d \in Dir \cup \{\epsilon\}$, if $s \xrightarrow{d} s_1$ and $s \xrightarrow{d} s_2$ then $s_1 = s_2$, and
(i) ϵ*-deterministic*: if $s \xrightarrow{\epsilon} s'$ then for every $d \in Dir \cup \{\epsilon\}$, if $s \xrightarrow{d} s''$ then $d = \epsilon$ and $s' = s''$. Further $state(s) = state(s')$. We call such an s a *silent configuration*.

Thus, for each d, the transition relation $\xrightarrow{d}$ is *functional*. Further, we assume that the transition relation is *directional*, meaning that for each $d \in Dir \cup \{\epsilon\}$, there is a (partial) function, $succ_d$ (say), such that for every q and c, if $(q, c) \xrightarrow{d} (q', c')$ then $c' = succ_d(c)$. I.e. the code-component of the configuration returned by the transition function $\xrightarrow{d}$ depends only on the code-component of the argument.

Let $\mathcal{A} = \langle \Sigma, Q, \delta, q_I, \Omega \rangle$ be an alternating parity automaton. We say that a (Q, Dir)-LTS $\mathcal{T}$ is *compatible* with $\mathcal{A}$ just if (i) the state of every initial configuration is the initial state q_I; (ii) for each configuration (q, c), and for each $d \in Dir \cup \{\epsilon\}$, there is a $\mathcal{T}$-transition $(q, c) \xrightarrow{d} (q', succ_d(c))$ iff there is a (minimal) S that satisfies $\delta(q, \rho(c))$, and $(q', d) \in S$. Intuitively such an LTS is the underlying graph of a parity game that characterises the decision problem: is $\mathcal{T}$ accepted by $\mathcal{A}$.

We say that a configuration (q, c) has a *unique out-transition* just if $\{(q', c') \mid (q, c) \xrightarrow{d} (q', c'), d \in Dir \cup \{\epsilon\}\}$ is a singleton set.

Given a (Q, Dir)-LTS $\mathcal{T} = \langle C, \langle \xrightarrow{d} \mid d \in Dir \cup \{\epsilon\}\rangle, I, state, \rho\rangle$ compatible with an alternating parity automaton $\mathcal{A} = \langle \Sigma, Q, \delta, q_I, \Omega\rangle$, we define a (parity) game $\mathbb{G}(\mathcal{T}, \mathcal{A})$ as follows.

- The start position is the set I.
- If the current position is a configuration (q, c), then it is Éloïse's turn to move. Éloïse chooses a minimal set $X = \{(q_1, d_1), \cdots, (q_k, d_k)\}$ that satisfies $\delta(q, \rho(c))$, and the new position is the set $succ_X(c) := \{(q_i, succ_{d_i}(c)) \mid 1 \leq i \leq k\}$.
- If the current position is a configuration set B, then it is Abelard's turn to move. He chooses some element $(q, c) \in B$, which becomes the new position.

The winning condition is defined as follows. If a player is unable to move at a position, then the other player wins. Suppose an infinite play ensues

$$(q_0, c_0)\ C_0\ (q_1, c_1)\ C_1\ \cdots$$

if the least priority that occurs infinitely often in the sequence $\Omega(q_0)\ \Omega(q_1)\ \Omega(q_2)\ \cdots$ is even, then Éloïse wins; otherwise Abelard wins.

Set $Dir = \{1, 2\} \cup Q$, where Q is a fixed set of states, and $Q' = Q \cup \{q_{un}\}$. We consider two instances of (Q', Dir)-LTS compatible with an alternating parity automaton.

LTS determined by D and W. Fix a RUL program D and a parity resource automaton $W = \langle Q, L, \delta, Q_I, \Omega\rangle$. Then, using the notation in the preceding, the transition system $\bigcup_{i=1}^{n} \mathcal{T}_{x_i}$ (i.e. disjoint union of $\mathcal{T}_{x_i}$ of Definition 1, where each x_i is a resource of sort $q_i \in Q_I = \{q_1, \cdots, q_n\}$) is a (Q', Dir)-LTS $\mathcal{T}_1 = \langle C_1, \langle \xrightarrow{d} \mid d \in Dir \cup \{\epsilon\}\rangle, I_1, state_1, \rho_1\rangle$, compatible with the alternating parity automaton $\mathcal{A}_W = \langle Q', \Sigma_1, \delta_1, q_{un}, \Omega_1\rangle$ where

$$\begin{array}{rcl} \Sigma_1 & = & \{F_i : 1 \mid 1 \leq i \leq n\} \cup \{\mathbf{new}_j^{q_i} : 1 \mid 1 \leq i, j \leq n\} \\ & \cup & \{\mathbf{acc}_a^{+} : 1,\ \mathbf{acc}_a^{-} : 1 \mid a \in L\} \cup \{\mathbf{if}* : 2,\ \star : 1\} \end{array}$$

and δ_1 is the map:

$$\begin{array}{rcll} (q, F_i) & \mapsto & (\epsilon, q) & \\ (q, \mathbf{if}*) & \mapsto & (1, q) \wedge (2, q) & \\ (q, \mathbf{new}_j^{q_i}) & \mapsto & \begin{cases} (1, q_{un}) \wedge (2, q_i) & \text{if } q = q_{un} \text{ and } i = j \\ (1, q) & \text{otherwise} \end{cases} & \\ (q, \mathbf{acc}_a^{+}) & \mapsto & \bigvee_i \bigwedge_{q' \in Q_i} (1, q') & \text{if } q \in Q \text{ and } \delta(q, a) = \{Q_1, \cdots, Q_n\} \\ (q, \mathbf{acc}_a^{-}) & \mapsto & (\epsilon, q) & \\ (q, \star) & \mapsto & (\epsilon, q) & \end{array}$$

The node-labelling map $\rho_1 : C_1 \to \Sigma_1$ does the obvious thing, except mapping $(H, \mathbf{acc}_a\, y\, e)_x$ to $\mathbf{acc}_a^+$ if $x = y$, and to $\mathbf{acc}_a^-$ otherwise; and mapping $(H, \mathbf{new}^{q_i}\, e)_{x_j}$ to $\mathbf{new}_j^{q_i}$. The priority map Ω_1 extends Ω by mapping q_{un} to the largest priority.

Lemma 4. *The game $\mathcal{G}(D, W)$ is just $\mathbb{G}(\mathcal{T}_1, \mathcal{A}_W)$.*

LTS determined by HORS G_D and APT $\mathcal{A}_{D,W}$. From the transform, HORS $G_D = \langle \Sigma, \mathcal{N}, \mathcal{R} \rangle$ and APT $\mathcal{A}_{D,W} = \langle \Sigma, Q'', \delta', \Omega' \rangle$, we define a (Q', Dir)-LTS $\mathcal{T}_2$, compatible with $\mathcal{A}_{\mathcal{D},\mathcal{W}}$, whose transition function is defined as follows:

$$
\begin{aligned}
(q, F\,\overline{e}) &\xrightarrow{\epsilon} (q, e'[\overline{e}/\overline{y}]) \\
(q, br_{\mathrm{if}}\, e_1\, e_2) &\xrightarrow{i} (q, e_i) \quad i = 1, 2 \\
(q, br_{\mathrm{new}}\, e_1\, e_2) &\xrightarrow{i} (q, e_i) \quad i = 1, 2 \\
(q, \nu^{q'}\, e) &\xrightarrow{1} \begin{cases} (q', e) & \text{if } q = q_{un} \\ (q_{any}, e) & \text{otherwise} \end{cases} \\
(q_{any}, a\, e) &\xrightarrow{1} (q_{any}, e) \\
(q, a\, e) &\xrightarrow{q'} (q', e) \quad \text{if } q \in Q, \text{ and } q' \in S \text{ satisfies } \delta(q, a) \text{ for some minimal } S \\
(q, \star) &\xrightarrow{\epsilon} (q, \star)
\end{aligned}
$$

and the priority function Ω' is that of the automaton. The following straightforward lemma is just the standard restatement of accepting run-trees as winning strategies.

Lemma 5. *Let G_D and $\mathcal{A}_{D,W}$ be as before. Then $[\![G_D]\!]$ is accepted by $\mathcal{A}_{D,W}$ if and only if Éloïse has a winning strategy for the game $\mathbb{G}(\mathcal{T}_2, \mathcal{A}_{D,W})$.*

Winning-strategy bisimulation between parity games

A notion of weak bisimulation. Let $\langle C_i, \langle \xrightarrow{d} \mid d \in Dir \cup \{\epsilon\} \rangle, I, state_i, \rho_i \rangle$ be a (Q, Dir)-LTS, for $i = 1, 2$; and let $s \in C_1$ and $t \in C_2$. We define

(i) $s \sim t$ iff $state_1(s) = state_2(t)$, and there exist s', t' such that $s \xrightarrow{\epsilon}\!\!\!\!\rightarrow s'$ and $t \xrightarrow{\epsilon}\!\!\!\!\rightarrow t'$ and $s' \approx t'$

(ii) $s \approx t$ iff

- for every $d \in Dir$ and s', if $s \xrightarrow{d} s'$ then there exist s'', t' such that $t \xrightarrow{d} \xrightarrow{\overline{d_1}}\!\!\!\!\rightarrow t'$ and $s' \xrightarrow{\overline{d_2}}\!\!\!\!\rightarrow s''$ and each configuration in the sequences $\xrightarrow{\overline{d_1}}\!\!\!\!\rightarrow$ and $\xrightarrow{\overline{d_2}}\!\!\!\!\rightarrow$ (except the last) has a unique out-transition, and $s'' \sim t'$, and
- for every $d \in Dir$ and t', if $t \xrightarrow{d} t'$ then there exist t'', s' such that $s \xrightarrow{d} \xrightarrow{\overline{d_1}}\!\!\!\!\rightarrow s'$ and $t' \xrightarrow{\overline{d_2}}\!\!\!\!\rightarrow t''$ and each configuration in the sequences $\xrightarrow{\overline{d_1}}\!\!\!\!\rightarrow$ and $\xrightarrow{\overline{d_2}}\!\!\!\!\rightarrow$ (except the last) has a unique out-transition, and $s'' \sim t'$

Theorem 7 (Winning-strategy bisimulation). *For $i = 1, 2$, let $\mathcal{T}_i = \langle C_i, \langle \xrightarrow{d} \mid d \in Dir \cup \{\epsilon\} \rangle, I_i, state_i, \rho_i \rangle$ be a (Q, Dir)-LTS, compatible with an alternating parity automaton $\mathcal{A}_i = \langle \Sigma_i, Q, \delta_i, \Omega_i \rangle$, such that*

(i) for each initial $(q, c_1) \in I_1$*, there is some initial* $(q, c_2) \in I_2$ *such that* $(q, c_1) \sim (q, c_2)$*; conversely for each initial* $(q, c_2) \in I_2$*, there is some initial* $(q, c_1) \in I_1$ *such that* $(q, c_1) \sim (q, c_2)$

(ii) whenever $(q_1, c_1) \approx (q_2, c_2)$*, then* $\delta_1(q_1, \rho_1(c_1)) = \delta_2(q_2, \rho_2(c_2))$

(iii) $\Omega_1 = \Omega_2$*.*

Then Éloïse has a winning strategy in $\mathbb{G}(\mathcal{T}_1, \mathcal{A}_1)$ *if and only if he has a winning strategy in* $\mathbb{G}(\mathcal{T}_2, \mathcal{A}_2)$*.*

Proof. (Sketch) Suppose τ is a Éloïse winning strategy for $\mathbb{G}(\mathcal{T}_2, \mathcal{A}_2)$. We construct a winning strategy σ for $\mathbb{G}(\mathcal{T}_1, \mathcal{A}_1)$ by imitating τ, and we do so round by round. Suppose Abelard picks $(q, c_1) \in I_1$ with $(q, c_1) \sim (q, c_2)$. It follows from the definition of $\sim$ that for some c_1' and c_2', we have $(q, c_1') \approx (q, c_2')$. Because of ϵ-determinacy and $\mathcal{A}_i$-compatibility, there is no choice for σ at each configuration in the sequence $(q, c_1) \xrightarrow{\epsilon}\!\!\!\!\rightarrow (q, c_1')$, exclusive of (q, c_1'). At the configuration (q, c_1'), Éloïse copies the strategy τ's action at (q, c_2'), which is a valid move because of assumption (ii). Now, suppose Abelard chooses the transition $(q, c_1') \xrightarrow{d} \gamma$, it follows from the definition of $\approx$ that there are configurations γ_1, γ_2 such that $(q, c_1') \xrightarrow{d} \xrightarrow{\overline{d_1}}\!\!\!\!\rightarrow \gamma_1$ and $(q, c_2') \xrightarrow{d} \xrightarrow{\overline{d_2}}\!\!\!\!\rightarrow \gamma_2$ with $\gamma_1 \sim \gamma_2$. Since each configuration in the sequence $(q, c_1') \xrightarrow{d} \xrightarrow{d_1}\!\!\!\!\rightarrow \gamma_1$, except the last, has the unique out-transition property and is $\mathcal{A}_1$-compatible, σ has no choice at each configuration in the sequence. Thus we have completed one round of strategy simulation. We argue inductively in a similar fashion. The satisfaction of parity follows from assumption (iii).

Relating the two LTSs

We relate the reduction of a RUL program and its recursion-scheme translate. We write $(H,\ e)_x \simeq (q, t)$ to mean $state\,(H,\ e)_x = q$, and

(i) if $x \notin dom(H)$ with $H = \{y_1 : q_1, \cdots, y_n : q_n\}$ then $t = e[\overline{K/y_i}]$ (and necessarily $q = q_{un}$)

(ii) if $x \in dom(H)$ then $H = \{y_1 : q_1, \cdots, y_n : q_n, x : q\}$ and $t = e[\overline{K/y_i}, I/x]$.

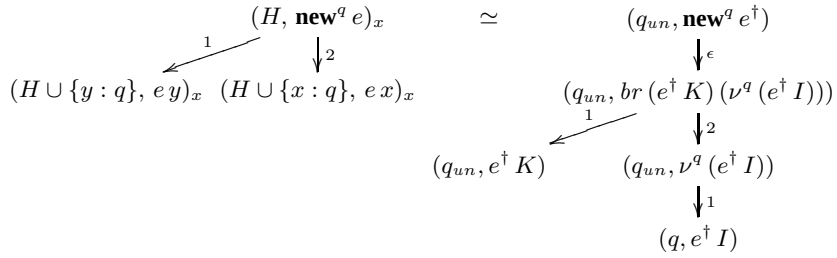
Henceforth we shall write $(-)\dagger$ to mean the substitution $(-)[\overline{K/y_i}, I/x]$ or $(-)[\overline{K/y_i}]$, according to whether $x \in dom(H)$ or not.

In the following we show that the two (Q', Dir)-LTSs, $\mathcal{T}_1$ and $\mathcal{T}_2$, satisfy the assumptions of Theorem 7 by a case analysis. Theorem 2 then follows from Lemma 4 and Lemma 5. First we remark that there is no harm in ignoring transitions in $\mathcal{T}_2$ of the form

$$(q,\, br\,(e^\dagger\, K)\,(\nu^{q'}\,(e^\dagger\, I))) \xrightarrow{2} (q, \nu^{q'}\,(e^\dagger\, I))$$

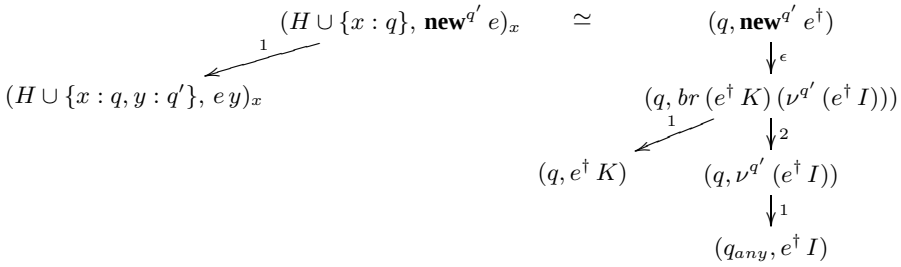
(and hence all transition thereafter, and so we may regard $\mathcal{T}_2$ as a (Q', Dir)-LTS) because the transition that follows is necessarily $(q, \nu^{q'}\,(e^\dagger\, I)) \xrightarrow{1} (q_{any}, e^\dagger\, I)$, and it is easy to see that Éloïse has a winning strategy starting from a configuration of state q_{any}.

1. Assume: $x \notin dom(H)$.



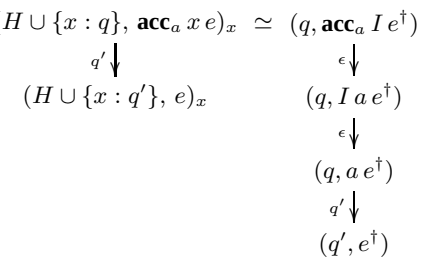
We have $(H \cup \{y : q\},\ e\ y)_x \simeq (q_{un}, e^\dagger\ K)$ and $(H \cup \{x : q\},\ e\ x)_x \simeq (q, e^\dagger\ I)$.

2. Assume: $x \in dom(H)$ and so $q \in Q$.



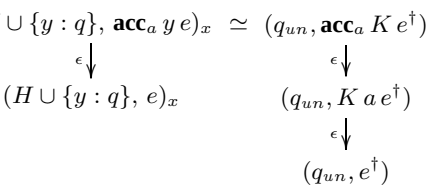
We have $(H \cup \{x : q, y : q'\},\ e\ y)_x \simeq (q, e^\dagger\ K)$. As remarked in the preceding, we ignore the $\xrightarrow{2}$-transition on the RHS and hence all transitions thereafter.

3.



We have $(H \cup \{x : q'\},\ e)_x \simeq (q', e^\dagger)$.

4. Assume: $x \notin dom(H)$



We have $(H \cup \{y : q\},\ e)_x \simeq (q_{un}, e^\dagger)$.

5.

$$\begin{array}{ccc} (H \cup \{x : q, y : q'\},\ \mathbf{acc}_a\ y\ e)_x & \simeq & (q, \mathbf{acc}_a\ K\ e^\dagger) \\ \epsilon\downarrow & & \epsilon\downarrow \\ (H \cup \{x : q, y : q'\},\ e)_x & & (q, K\ a\ e^\dagger) \\ & & \epsilon\downarrow \\ & & (q, e^\dagger) \end{array}$$

We have $(H \cup \{x : q, y : q'\},\ e)_x \simeq (q, e^\dagger)$.

6.

$$\begin{array}{ccc} (H,\ F\ \overline{e})_x & \simeq & (q, F\overline{e^\dagger}) \\ \epsilon\downarrow & & \epsilon\downarrow \\ (H,\ e'[\overline{e}/\overline{x}])_x & \simeq & (q, e'^\dagger[\overline{e^\dagger}/\overline{x}]) \end{array}$$

Note that a special case is $(\varnothing,\ S)_{x_i} \simeq (q_{un}, S)$ for each i.

7.

$$\begin{array}{ccccc} & (H,\ \mathbf{if}{*}\ e_1\ e_2)_x & & \simeq & (q, \mathbf{if}{*}\ e_1^\dagger\ e_2^\dagger) \\ {\scriptstyle 1}\swarrow & & {\scriptstyle 2}\downarrow & & {\scriptstyle 1}\swarrow\ {\scriptstyle 2}\downarrow \\ (H,\ e_1)_x & & (H,\ e_2)_x & & (q, e_1^\dagger) \qquad (q, e_2^\dagger) \end{array}$$

We have, for $i = 1, 2$, $(H,\ e_i)_x \simeq (q, e_i^\dagger)$.

Thus we can conclude that whenever $(H,\ e)_x \simeq (q, e^\dagger)$, then $(H,\ e)_x \sim (q, e^\dagger)$. In particular, satisfaction of condition (i) of Theorem 2 follows from Case 6. It is straightforward to check that condition (ii) is satisfied by inspecting each of the above cases.

C Linear-time algorithm for solving weak Büchi games

The weak Büchi game $\mathbb{G}^-(\Gamma) = \langle V_A, V_E, E, v_0, \Omega' \rangle$ is constructed using the fixpoint Γ obtained from Stage 1.3 and an auxiliary function $TEnv_\Gamma$, whose domain of definition is Γ, which maps $F : \theta$ to $\{\Delta \subseteq \Gamma \mid \Delta \vdash \mathcal{R}(F) : \theta\}$. The underlying directed graph has node-set $V_A \cup V_E$, with start node $v_0 := (S : q_I)$. We set:

$$\begin{array}{l} V_A := \{(\Delta, q) \mid \exists\, (F : \theta)\,.\, \Delta \in TEnv_\Gamma(F : \theta)\ \wedge\ q = state(\theta)\} \\ E := \{((F : \theta), (\Delta, state(\theta))) \mid \Delta \vdash \mathcal{R}(F) : \theta\} \cup \{((\Delta, q), (F : \theta)) \mid (F : \theta) \in \Delta\} \end{array}$$

The priority function Ω' maps $(F : \theta)$ to $\Omega(state(\theta))$, and (Γ, q) to $\Omega(q)$, where Ω is the priority function of $\mathcal{A}$. $\mathbb{G}^-(\Gamma)$ inherits a partial order $\leq$, over a partition $\{V_1, \cdots, V_n\}$ (say) of $V_A \cup V_E$, from the AWT $\mathcal{A}$, as in Lemma 2.

Solving weak Büchi game $\langle V_A, V_E, E, v_0, \Omega \rangle$ in linear time. We describe a solver for an (arbitrary) weak Büchi game. First construct a linear ordering $\leqslant$ of the partition of $V_A \cup V_E$ such that $\leq\ \subseteq\ \leqslant$; w.l.o.g. assume $V_1 \leqslant V_2 \leqslant \cdots \leqslant V_n$. The algorithm proceeds by rounds: at the end of round i, all nodes in V_i are marked by $\mathbf{t}$ or $\mathbf{f}$. The idea

is that Éloïse has a winning strategy from a node if, and only if, it is marked $\mathbf{t}$. We label each node v by a pair $\langle num\_left, pred\_list \rangle$ where num_left is a counter (initialized to v's out-degree) that maintains the number of successor nodes yet to be processed (before v's mark can be determined), and $pred\_list$ is a list of v's predecessors. The algorithm maintains a number i (initialized to 1) indicating the current round, and two stacks $Stack_{\mathbf{t}}$ and $Stack_{\mathbf{f}}$. The stacks contain nodes which have been marked $\mathbf{t}$ and $\mathbf{f}$ respectively but have not yet propagated their marks further. Starting from round 1, the algorithm proceeds as follows.

As soon as a node is marked $\mathbf{t}$ (resp. $\mathbf{f}$) (and if it is not already on a stack), it is pushed onto $Stack_{\mathbf{t}}$ (resp. $Stack_{\mathbf{f}}$). As long as $Stack_{\mathbf{t}}$ or $Stack_{\mathbf{f}}$ is non-empty, a node v is popped from a stack, and we process every (unmarked) node v' in the $pred\_list$ of v using the following rules.

- *Case*: v is marked $\mathbf{t}$. If $v' \in V_E$ then mark v' with $\mathbf{t}$ else $\{$if $num\_left = 1$ then mark v' with $\mathbf{t}$ else decrement num_left $\}$.
- *Case*: v is marked $\mathbf{f}$. If $v' \in V_A$ then mark v' with $\mathbf{f}$ else $\{$if $num\_left = 1$ then mark v' with $\mathbf{f}$ else decrement num_left $\}$.

At the start, when both $Stack_{\mathbf{t}}$ and $Stack_{\mathbf{f}}$ are empty, mark all terminal nodes $v \in V_i$ (i.e. those with out-degree 0) with $\mathbf{t}$ or $\mathbf{f}$ according to whether $v \in V_A$ or V_E. The stacks are then processed as outlined above and the marks propagated as far as possible. When the stacks are again empty, all remaining nodes $v \in V_i$ are marked in accord with the type of V_i (i.e. the mark is $\mathbf{t}$ or $\mathbf{f}$ according to whether V_i is accepting or rejecting) and the stacks are processed again; i is then incremented. In this way, every node in V_i is marked and its effect fully propagated at the end of round i. Éloïse has a winning strategy from v_0 just if v_0 is marked $\mathbf{t}$.

Since each node is pushed onto a stack only once, and since the processing of a node popped from a stack involves a constant number of operations on each of its (as yet unmarked) predecessor node, the overall time complexity is linear in the size of G.

D Proof of Theorem 4 (Correctness)

Proof. (i) Suppose, for a contradiction, $[\![G]\!]$ is accepted by $\mathcal{A}$ but $\Xi_{\mathcal{A}}$ does not terminate on input G. Let $\mathcal{C}_i$ be the configuration graph that is constructed at the end of Stage 1.1 in the i-th iteration of the loop, and let $\mathcal{C}$ be the union of all the $\mathcal{C}_i$'s. Then, since the expansion process is assumed to be fair, $\mathcal{C}$ is the closed configuration graph. It follows from Lemma 8 that there exists i such that $\Gamma_{\mathcal{C}} \subseteq ElimTE(\Gamma_{\mathcal{C}_i})$. By Lemma 3, at the i-th iteration, stage 1.3 constructs the largest type environment Γ that is a fixpoint of $\mathcal{F}_{\mathrm{nw}}$ and $\Gamma \subseteq ElimTE(\Gamma_{\mathcal{C}_i})$. Now it follows from Theorem 8 that $\Gamma_{\mathcal{C}}$ is a fixpoint of $\mathcal{F}_{\mathrm{nw}}$ and $S : q_0 \in \Gamma_{\mathcal{C}}$. Thus we have $S : q_0 \in \Gamma_{\mathcal{C}} \subseteq \Gamma$. It follows from Theorem 9 that Éloïse has a winning strategy in $\mathbb{G}^-(\Gamma)$. Hence the algorithm $\Xi_{\mathcal{A}}$ on input G terminates by the i-th iteration, returning YES.

(ii) Suppose $[\![G]\!]$ is rejected by $\mathcal{A}$ and the algorithm $\Xi_{\mathcal{A}}$ terminates on input G. Either there is no run-tree of $\mathcal{A}$ over $[\![G]\!]$, or there are but none is accepting. If the former, then after some finite iterations, the algorithm exits and returns NO, as desired. If the latter, then it follows from Theorem 9 that Éloïse does not have a winning strategy for

$\mathbb{G}^-(\Gamma)$, for every type environment Γ that ever reaches Stage 2 during the computation (for if Éloïse had a winning strategy in $\mathbb{G}^-(\Gamma)$ then the same strategy would be winning in $\mathbb{G}^-(\Gamma_{\mathcal{C}})$, contradicting Theorem 9). Hence the computation will not terminate. Thus this case is impossible.

The rest of the subsection is concerned with Lemma 8 and Theorem 8. The proofs are obtained by extending Kobayashi's arguments to the case where $\mathcal{A}$ is an AWT. Let π be a sequence over $\{0, 1, 2, \cdots, m\}$ where m is the largest arity of terminals. We write $\mathcal{C}(\pi)$ for the node N such that a path from the root to N is labelled by π.

Let $\mathcal{C}$ be the closed configuration graph, and $\mathcal{C}'$ be a finitely expanded graph. Suppose that the node $\mathcal{C}(\pi)$ is labelled by $[t\,\overline{s}, q]$. We define the relation $\mathcal{C}' \preceq_{\pi,t} \mathcal{C}$ by structural induction on the sort of t as follows.

(i) If t has sort o and $\boldsymbol{\tau}_{t,\mathcal{C}'(\pi)} = \{q\}$ then $\mathcal{C}' \preceq_{\pi,t} \mathcal{C}$.
(ii) Suppose t has sort $\kappa_1 \to \kappa_2$; it follows that $\overline{s}$ has the form $s_0\,\overline{s'}$. Then $\mathcal{C}' \preceq_{\pi,t} \mathcal{C}$ just if the following conditions hold:
   (a) $\mathcal{C}' \preceq_{\pi,t\,s_0} \mathcal{C}$
   (b) for every element of $\boldsymbol{\tau}_{t,\mathcal{C}(\pi)}$—which must have the form $\bigwedge_{i=1}^m \tau_i \to \tau'$ with $\tau' \in \boldsymbol{\tau}_{t\,s_0,C(\pi)}$, for each i, there exists a path π_i such that $\tau_i \in \boldsymbol{\tau}_{s_0,C(\pi\,\pi_i)}$ and $\mathcal{C}' \preceq_{\pi\,\pi_i,s_0} \mathcal{C}$.

$$\begin{aligned} \mathit{Elim}(q) &:= \{q\} \\ \mathit{Elim}(\textstyle\bigwedge_{i=1}^m \tau_i \to \tau) &:= \{\textstyle\bigwedge_{i=1}^m \theta_i \to \theta \mid \theta_i \in \mathit{Elim}'(\tau_i), \theta \in \mathit{Elim}(\tau)\} \end{aligned}$$

$$\mathit{Elim}'(\tau) := \begin{cases} \mathit{Elim}(\tau) \cup \{\top\} & \text{if } \tau \text{ contains a type variable} \\ \mathit{Elim}(\tau) & \text{otherwise} \end{cases}$$

By abuse of notation, we use the same notation for the pointwise extension of Elim to a set of types; e.g. $\mathit{Elim}(\boldsymbol{\tau}_{t,\mathcal{C}(\pi)})$. We define ElimTE as the pointwise extension of Elim i.e.

$$\mathit{ElimTE}(\Gamma) := \{F : \theta \mid F : \tau \in \Gamma, \theta \in \mathit{Elim}(\tau)\}$$

Lemma 6. *Let $\mathcal{C}$ be the closed configuration graph. If $\mathcal{C}' \preceq_{\pi,t} \mathcal{C}$ then the following holds:*

(I) If $\mathcal{C}''$ is obtained from expansions of $\mathcal{C}'$ then $\mathcal{C}'' \preceq_{\pi,t} \mathcal{C}$
(II) $\boldsymbol{\tau}_{t,\mathcal{C}(\pi)} \subseteq \mathit{Elim}(\boldsymbol{\tau}_{t,\mathcal{C}'(\pi)})$.

Proof. Let the label of $\mathcal{C}(\pi)$ be $[t\,\overline{s}, q]$. The proof proceeds by induction on the sort of t. In case the sort of t is o, then by definition of $\mathcal{C}' \preceq_{\pi,t} \mathcal{C}$, we have $\boldsymbol{\tau}_{t,\mathcal{C}'(\pi)} = \{q\}$. By the definitions of expansion and of $\boldsymbol{\tau}_{t,N}$, we have $\boldsymbol{\tau}_{t,\mathcal{C}'(\pi)} = \boldsymbol{\tau}_{t,\mathcal{C}''(\pi)} = \boldsymbol{\tau}_{t,\mathcal{C}(\pi)} = \{q\}$. Thus we have $\mathcal{C}'' \preceq_{\pi,t} \mathcal{C}$ and $\boldsymbol{\tau}_{t,\mathcal{C}(\pi)} = \{q\} \subseteq \{q\} = \mathit{Elim}(\boldsymbol{\tau}_{t,\mathcal{C}'(\pi)})$ as required.

If the sort of t is $\kappa_1 \to \kappa_2$ then it follows from the assumption $\mathcal{C}' \preceq_{\pi,t} \mathcal{C}$ that (i) $\overline{s} = s_0\,\overline{s'}$, (ii) every type in $\boldsymbol{\tau}_{t,\mathcal{C}(\pi)}$ has the form $\bigwedge_{i=1}^m \tau_i \to \tau$, (iii) $\mathcal{C}' \preceq_{\pi,t\,s_0} \mathcal{C}$, and (iv) for each τ_i there exists π_i such that $\tau_i \in \boldsymbol{\tau}_{s_0,\mathcal{C}(\pi\,\pi_i)}$ and $\mathcal{C}' \preceq_{\pi\,\pi_i,s_0} \mathcal{C}$. By the induction hypothesis (I), we have $\mathcal{C}'' \preceq_{\pi,t\,s_0} \mathcal{C}$ and $\mathcal{C}'' \preceq_{\pi\,\pi_i,s_0} \mathcal{C}$, which implies (I). By the induction hypothesis (II), we also have $\boldsymbol{\tau}_{t\,s_0,\mathcal{C}(\pi)} \subseteq \mathit{Elim}(\boldsymbol{\tau}_{t\,s_0,\mathcal{C}'(\pi)})$ and

$\boldsymbol{\tau}_{s_0,\mathcal{C}(\pi\,\pi_i)} \subseteq \mathit{Elim}(\boldsymbol{\tau}_{s_0,\mathcal{C}'(\pi\,\pi_i)})$. By the definition of $\boldsymbol{\tau}_{t,\mathcal{C}'(\pi)}$, an element of which has the form

$$\bigwedge_{i=1}^{m} \tau_i \wedge \bigwedge_{j=1}^{k} \sigma_j \to \tau$$

where $\tau_i \in \boldsymbol{\tau}_{s_0,\mathcal{C}'(\pi\,\pi_i)}$ and $\tau \in \boldsymbol{\tau}_{t\,s_0,\mathcal{C}'(\pi)}$. By definition of Elim, we can construct each element of $\boldsymbol{\tau}_{t,\mathcal{C}(\pi)}$ as an element $\bigwedge_{i=1}^{m} \alpha_i \wedge \bigwedge_{j=1}^{k} \beta_j \to \tau$ of $\mathit{Elim}(\boldsymbol{\tau}_{t,\mathcal{C}'(\pi)})$ as follows: (i) choose α_i to be τ_i; (ii) from $\mathit{Elim}(\sigma_j)$, choose β_j to be $\top$ if σ_j contains a type variable, otherwise σ_j is an element of $\boldsymbol{\tau}_{s_0,\mathcal{C}'(\pi\,\pi_i)}$ for some i, and so, choose σ_j to be τ_i instead.

Lemma 7. *Let $N = \mathcal{C}(\pi)$ be a node of a closed configuration graph $\mathcal{C}$, and suppose that N is labelled with $[t\,\overline{s}, q]$. Then there exists a finitely-expanded graph $\mathcal{C}'$ such that $\mathcal{C}' \preceq_{\pi,t} \mathcal{C}$.*

Proof. The proof is by induction on the sort of t. If the sort is o then the result follows immediately: just expand the graph until N is expanded, and let $\mathcal{C}'$ be the resulting graph.

If the sort is $\kappa_1 \to \kappa_2$ then $\overline{s} = s_0\,\overline{s'}$ and let $\boldsymbol{\tau}_{t,N} = \{\bigwedge_{j=1}^{r_j} \tau_{ij} \to \tau_i \mid 1 \leq i \leq n\}$. By the induction hypothesis, there exists a finitely expanded graph $\mathcal{C}'_0$ such that $\mathcal{C}'_0 \preceq_{\pi,t\,s_0} \mathcal{C}$. By definition of $\boldsymbol{\tau}_{t,N}$, for each i and each j, there exists π_{ij} such that $\tau_{ij} \in \boldsymbol{\tau}_{s_0,\mathcal{C}(\pi\,\pi_{ij})}$. By the induction hypothesis, there exists a finitely expanded graph $\mathcal{C}'_{ij}$ such that $\mathcal{C}'_{ij} \preceq_{\pi\,\pi_{ij},s_0} \mathcal{C}$. Thus the union of $\mathcal{C}'_0, \mathcal{C}'_{11}, \cdots, \mathcal{C}'_{1r_1}, \cdots, \mathcal{C}'_{n1}, \cdots, \mathcal{C}'_{nr_n}$, satisfies the required condition by Lemma 6(I).

Lemma 8 (Key). *Suppose that $[\![G]\!]$ is accepted by $\mathcal{A}$, and let $\mathcal{C}$ be the closed configuration graph for G and $\mathcal{A}$. Then there exists a finitely expanded configuration graph $\mathcal{C}'$ such that $\Gamma_{\mathcal{C}} \subseteq \mathit{ElimTE}(\Gamma_{\mathcal{C}''})$ for every finite expansion $\mathcal{C}''$ of $\mathcal{C}'$.*

Proof. For each $F_i : \tau_{ij} \in \Gamma_{\mathcal{C}}$, pick a node $N_{ij} = \mathcal{C}(\pi_{ij})$ such that $\tau_{ij} \in \boldsymbol{\tau}_{F_i,N_{ij}}$. By Lemma 7, there exists a finitely expanded graph $\mathcal{C}_{ij}$ such that $\mathcal{C}_{ij} \preceq_{\pi\,\pi_{ij},F_i} \mathcal{C}$. Let $\mathcal{C}'$ be the union of the $\mathcal{C}_{ij}$'s, and $\mathcal{C}''$ be a finite expansion of it. By Lemma 6 (I), $\mathcal{C}'' \preceq_{\pi\,\pi_{ij},F_i} \mathcal{C}$ for every i and j. It then follows from Lemma 6 (II) that $\Gamma_{\mathcal{C}} \subseteq \mathit{ElimTE}(\mathcal{C}'')$ as required.

Theorem 8. *Assume $[\![G]\!]$ is accepted by AWT $\mathcal{A}$. If $\mathcal{C}$ is a closed configuration graph of G over $\mathcal{A}$, then G is well-typed under $\Gamma_{\mathcal{C}}$ i.e. for all $(F : \theta) \in \Gamma_{\mathcal{C}}$, there exists $\Gamma \subseteq \Gamma_{\mathcal{C}}$ such that $\Gamma \vdash \mathcal{R}(F) : \theta$, and $(S : q_0) \in \Gamma$.*

Proof. For each t that occurs in a head position in a node N of the configuration graph $\mathcal{C}$ (i.e. the label of N has the form $\langle t\,\overline{s}, q, \mathit{closed}\rangle$), and for each $\tau \in \boldsymbol{\tau}_{t,N}$, we first construct a type derivation tree $\Pi_{t,\tau,N}$ of $\Gamma \vdash t : \tau$ for some $\Gamma \subseteq \Gamma_{\mathcal{C}}$.

Now, suppose $F : \theta \in \Gamma_{\mathcal{C}}$ and $\mathcal{R}(F) = \lambda x_1 \cdots x_m.s$. We need to show that there exists $\Gamma \subseteq \Gamma_{\mathcal{C}}$ such that $\Gamma \vdash \lambda x_1 \cdots x_m.s : \theta$. By construction of $\mathcal{C}$, we have $\theta \in \boldsymbol{\tau}_{F,N}$, for some node N of $\mathcal{C}$. The node N must be labelled with $[F\,t_1 \cdots t_m, q]$ and has a single outgoing edge to a node N' with label $[s[t_1/x_1, \cdots, t_m/x_m], q]$. By construction, θ has the shape

$$\bigwedge\{\tau_{N_1} \mid N_1 \in S_1\} \to \cdots \to \bigwedge\{\tau_{N_m} \mid N_m \in S_m\} \to q$$

where each τ_{N_i} is some type in $\boldsymbol{\tau}_{t_i,N_i}$, S_i is a set of compatible nodes where t_i occurs in a head position, and $S_1, \cdots, S_m$ are mutually compatible. (The intuition is that the type θ is extracted from a *single* run-tree that "contains" N.)

For each i, let S'_i be the set of nodes N_i in S_i such that for some $\Gamma \subseteq \Gamma_{\mathcal{C}}$, $\Gamma \vdash t_i : \tau_{N_i}$ occurs in $\Pi_{s[t_1/x_1,\cdots,t_m/x_m],\tau,N'}$. From the derivation tree $\Pi_{s[t_1/x_1,\cdots,t_m/x_m],\tau,N'}$, we can obtain a derivation for

$$\Delta \cup \bigcup_{i=1}^{m} \{x_i : \tau_{N_i} \mid N_i \in S'_i\} \vdash s : q$$

for some $\Delta \subseteq \Gamma_{\mathcal{C}}$. Since $S'_i \subseteq S_i$, we get $\Delta \vdash \lambda x_1 \cdots x_m.s : \theta$ by (ABS) as required.

The following key theorem follows from the proof of the completeness theorem of Kobayashi and Ong [6].

Theorem 9. *Let G be a recursion scheme and $\mathcal{A}$ an AWT. Then $[\![G]\!]$ is accepted by $\mathcal{A}$ if, and only if, the closed configuration graph $\mathcal{C}$ exists, and Éloïse has a winning strategy in the derived weak Büchi game $\mathbb{G}^-(\Gamma_{\mathcal{C}})$. (Note that $\Gamma_{\mathcal{C}}$ is necessarily finite, even though $\mathcal{C}$ may be infinite.)*

E Proof of Theorem 5 (Complexity)

(i). The case of alternating trivial automata is proved in [14]. Here we deal with the other two. The upper bound follows from the n-EXPTIME completeness of modal mu-calculus / APT model checking [2]. The lower bound for alternating weak automata follows from that for alternating trivial automata, since it subsumes the latter. In the following we establish the n-EXPTIME hardness for the alternating co-trivial case.

Engelfriet [19] proved a hierarchy theorem for higher-order pushdown alternating word automata (AWA):

Theorem 10 (Engelfriet 1991). *For each $k \geq 0$, the class of finite-word languages recognized by order-k pushdown AWA is $\bigcup_{d>1} DTIME(\mathsf{exp}_k\, d\, n)$.*

We use the Theorem to show that the acceptance problem of alternating co-trivial automata for trees generated by order-k safe (and hence arbitrary) recursion schemes is k-EXPTIME hard.

Fix an input alphabet A. An *order-k pushdown AWA* is given by a 7-tuple

$$\mathcal{A} = \langle P, \lambda, p_0, \Gamma, A, \Delta, F \rangle$$

where the labelling function $\lambda : P \to \{E, A\}$ partitions the state-set P into E-states and A-states, $p_0 \in P$ is the start state, Γ is the stack alphabet, $\Delta \subseteq P \times \Gamma \times (A \cup \{\epsilon\}) \times P \times Op_k$ is the transition relation, Op_k is the set of order-k stack actions, and $F \subseteq P$ is the set of final states. We assume that for every p and γ, if there are some q and θ such that $(p, \gamma, q, \epsilon, \theta) \in \Delta$, then it is *not* the case that there are q', θ' and $a \in A$ such that $(p, \gamma, q', a, \theta') \in \Delta$. I.e. no configuration can have both an A-transition and an ϵ-transition. Recall that a word $w \in A^*$ is *accepted* by an AWA $\mathcal{A}$ if there is a finite run-tree over w such that every leaf is a final state.

For our purpose it is convenient to define the acceptance of a word by an order-k pushdown AWA $\mathcal{A}$ in a game setting. Take a word $w = w_1 \cdots w_{|w|} \in A^*$ (with each $w_i \in A$), we define the *acceptance parity game* $Acc(w, \mathcal{A})$ as follows. The *E-nodes* are elements of the set $P_E \times [1..|w|] \times \mathsf{Stack}_k(\Gamma)$ where $\mathsf{Stack}_k(\Gamma)$ is the set of order-k stacks over the stack alphabet Γ; similarly for the *A-nodes*. The edge-set is defined as follows: Take a node (p, i, s).

- For each $(p, top_1\, s, a, p', \theta) \in \Delta$ where $a = w_i \in A$, there is an edge from (p, i, s) to $(p', i + 1, \theta(s))$.
- For each $(p, top_1\, s, \epsilon, p', \theta) \in \Delta$, there is an edge from (p, i, s) to $(p', i, \theta(s))$.

Further we designate every node of the form $(p, |w|, s)$ where $p \in F$ as an *accept* node i.e. winning for Éloïse, and every terminal node (i.e. no outgoing edges) that is not an accept state as a *reject* node i.e. losing for Éloïse. The priority function Ω is the constantly-1 function. (Thus Éloïse loses every infinite play.) We say that a word w is *accepted* by the pushdown AWA $\mathcal{A}$ just if Éloïse has a winning strategy in the game $Acc(w, \mathcal{A})$ from the start node $(p_0, 1, \bot_k)$ where $\bot_k$ is the empty k-stack.

We can express the game $Acc(w, \mathcal{A})$ as a parity game $G = \langle \lambda', \Omega' \rangle$ defined on the configuration graph of an order-k pushdown system (PDS)

$$\mathcal{R}_{w,\mathcal{A}} \;=\; \langle P \times [1..|w|], (p_0, 1), \Gamma, \Delta' \rangle$$

where the transition relation $\Delta' \subseteq (P \times [1..|w|]) \times \Gamma \times (P \times [1..|w|]) \times Op_k$ is defined as follows: let i, p, p' and θ range over the appropriate sets

$$\begin{array}{rcl} ((p, i), \gamma, (p', i + 1), \theta) \in \Delta' & \iff & (p, \gamma, w_i, p', \theta) \in \Delta \\ ((p, i), \gamma, (p', i), \theta) \in \Delta' & \iff & (p, \gamma, \epsilon, p', \theta) \in \Delta \end{array}$$

The (immediate) accept and reject nodes remain the same. The labelling map λ' of the parity game G is given by the map λ restricted to the first component of the control state (p, i), and the priority function Ω' is the constantly-1 function.

Thanks to Emerson and Jutla [10], we have the following reduction:

Theorem 11. *There is a (polynomial) reduction of* **P1** *to* **P2**.

P1. *Given a parity game G over a directed graph, does Éloïse have a winning strategy from the start node?*
P2. *Given an APT $\mathcal{A}_G$ and a directed graph, is the unravelling of the graph accepted by the automaton?*

A useful fact is that the unravelling of the configuration graph of an order-n PDS is a ranked tree generated by an order-n PDA; one only has to note that an appropriate labelling of the edges makes the order-n PDS graph *deterministic*.

Lemma 9. *Suppose $\mathcal{R} = \langle \Gamma, Q, \Delta, q_0 \rangle$ is an order-n PDS. Let t be the tree obtained by unravelling the configuration graph of $\mathcal{R}$ and by labelling every node by the control state and the top_1 stack symbol of the corresponding configuration. Then t is generated by an order-n PDA $\widetilde{\mathcal{R}}$ of size polynomial in the size of $\mathcal{R}$.*

Proof. Consider the following order-n PDA $\widetilde{\mathcal{R}} = \langle \Sigma, \Gamma, Q', \delta, q_0 \rangle$ where we set:

- $Trans = \{(q, \theta) \mid \exists p \in Q. \exists a \in \Gamma\,.\,(p, a, q, \theta) \in \Delta\}$ is the set of all transitions that can be applied in $\mathcal{R}$.
- $Q' = Q \cup Trans$
- $\Sigma = Q \times \Gamma$ is the set of shapes (we ignore the link in the case of CPDA) and the arity of $(q, a) \in \Sigma$ is $|\{(q', \theta) \mid (q, a, q', \theta) \in \Delta\}|$.
- For every $q \in Q$, and every $a \in \Gamma$, $\delta(q, a) = ((q, a); (q_1, \theta_1), \cdots, (q_k, \theta_k))$ where $\{(q_1, \theta_1), \cdots, (q_k, \theta_k)\} = \{(q', \theta) \mid (q, a, q', \theta) \in \Delta\}$.
- For every $(q, \theta) \in Trans$, and every $a \in \Gamma$, $\delta((q, \theta), a) = (q, \theta)$.

Then one easily checks that $\widetilde{\mathcal{R}}$ generates t.

Theorem 12 (Knapik et al. [20]). *Fix a ranked alphabet, and let $k \geq 0$. Order-k safe recursion schemes and order-k pushdown tree automata generate the same class of ranked trees. Further there are polynomial inter-translations between the formalisms.*

We can now prove the desired result.

Theorem 13. *Let G be an order-k recursion scheme, and $\mathcal{A}$ be an alternating co-trivial automaton. The problem of whether $[\![G]\!]$ is accepted by $\mathcal{A}$ is k-EXPTIME hard.*

Proof. Fix a $k \geq 1$ and a $d > 1$. By Theorem 10, there is an order-k pushdown AWA $\mathcal{A}$ and some $w \in A^*$ such that it takes time at least $\mathsf{exp}_k d|w|$ to decide whether Éloïse has a winning strategy in the acceptance game $Acc(w, \mathcal{A})$, which can be viewed as a parity game G (say) over the configuration graph of the order-k PDS $\mathcal{R}_{w,\mathcal{A}}$. By Lemma 9 and Theorem 11, the solvability of G over $\mathcal{R}_{w,\mathcal{A}}$ is reducible to the question of whether the APT $\mathcal{A}_G$ accepts the tree generated by the order-k PDA $\widetilde{\mathcal{R}_{w,\mathcal{A}}}$. By Theorem 12, let $S_{w,\mathcal{A}}$ be the order-k safe recursion scheme that generates the same tree as $\widetilde{\mathcal{R}_{w,\mathcal{A}}}$. Note that $\mathcal{A}_G$ has a trivial acceptance condition – every node has priority 1. Hence, deciding whether $\mathcal{A}_G$ accepts the tree generated by the order-k safe recursion scheme $S_{w,\mathcal{A}}$ takes time at least $\mathsf{exp}_k d|w|$.

(ii). If $\mathcal{A}$ is an APT, then the complement of the language $\mathcal{L}(\mathcal{A})$ is also recognized by an APT which is written $\overline{\mathcal{A}}$. Kobayashi observed that in case $\mathcal{A}$ is deterministic trivial, $\overline{\mathcal{A}}$ is a disjunctive APT [7]. The $(n-1)$-EXPTIME decidability then follows from the $(n-1)$-EXPTIME completeness of the disjunctive APT acceptance problem for trees generated by order-n recursion schemes [14]. The same argument works for the case of deterministic weak automata (and hence also for deterministic co-trivial). Take a deterministic weak automaton $\mathcal{A}$ with a priority function Ω. In $\overline{\mathcal{A}}$, which is a disjunctive APT, the priority map $\overline{\Omega}$ is the inversion of Ω. Thus $\overline{\mathcal{A}}$ accepts a tree t just if there is no run-tree of $\mathcal{A}$ over t (which is equivalent to the existence of an "error path" w.r.t. $\mathcal{A}$), or the run-tree of $\mathcal{A}$ over the t has an infinite path in which 1 is the infinitely occurring priority. The $(n-1)$-EXPTIME hardness for deterministic weak and deterministic trivial automata follow from the $(n-1)$-EXPTIME completeness of the Reachability Problem [14].

F An efficient intersection type system with subtyping $\boldsymbol{\lambda}^{\mathcal{A}}_{\wedge,\leq}$

Kobayashi has shown in [7] that to make verification of recursion schemes at higher-orders tractable, even for trivial automata checking, it is essential to apply two type-based optimisations during inference. The first optimisation is a kind of symmetry reduction of the search space, in which only intersection types in a subtype-canonical form (with respect to the standard notion of subtyping on intersection types) are visited by the search. The optimisation carries its own cost: in order to type the canonical forms arising from the algorithm, the system $\boldsymbol{\lambda}^{\mathcal{A}}_{\wedge}$ must be extended to allow subtype reasoning.

Whilst we desire the additional flexibility of subtyping in order to perform symmetry reduction, we also wish to minimise the size (and computation time) of $TEnv_{\Gamma}(F : \tau)$ for each binding $F : \tau$. In order to strike the right balance, we present a new type system $\boldsymbol{\lambda}^{\mathcal{A}}_{\wedge,\leq}$, which consists of replacing the rules (APP) and (ABS) of $\boldsymbol{\lambda}^{\mathcal{A}}_{\wedge}$ with the following:

$$\frac{\begin{array}{c}\bigwedge_{j\in I_i}\theta_{ij} \leq \bigwedge S_i \quad \text{for each } i \in \{1,\cdots,n\} \\ \Gamma_{ij} \vdash t_i : \theta_{ij} \quad \text{for each } i \in \{1,\cdots,n\},\, j \in I_i\end{array}}{\{\xi : \bigwedge S_1 \to \cdots \to \bigwedge S_n \to \theta\} \cup \bigcup_{i=1}^{n}\bigcup_{j\in I_i}\Gamma_{ij} \vdash \xi\, t_1 \cdots t_n : \theta} \text{ (L-APP)}$$

$$\frac{\Gamma, x_{i_1} : \bigwedge S_{i_1}, \cdots, x_{i_r} : \bigwedge S_{i_r} \vdash t : \theta \quad FVar(t) = \{x_{i_1},\cdots,x_{i_r}\}}{\Gamma \vdash \lambda x_1 \cdots x_n.t : \bigwedge S_1 \to \cdots \to \bigwedge S_n \to \theta} \text{ (L-ABS)}$$

in which $\sigma \leq \tau$ asserts the usual intersection subtype relation between types σ and τ.

The system is a compromise between allowing sufficient subtype reasoning to type the canonical forms arising from the model checking algorithm and constraining the structure of type derivations so that they are more amenable to proof search. The key to the latter is that the system possesses a strong form of the subformula property. We say that a type σ is a *suffix* of a type τ just if τ is of the form $\bigwedge S_1 \to \cdots \to \bigwedge S_k \to \sigma$ for some $k \geq 0$.

Lemma 10 ($\boldsymbol{\lambda}^{\mathcal{A}}_{\wedge,\leq}$ Suffix Property). *If the judgement $\Gamma \vdash \mathcal{R}(F) : \tau$ is derivable for some $F : \tau$ in Γ where $\mathcal{R}(F) = \lambda\overline{x} : \overline{\theta}.t$, then, for every (proper) subderivation with some goal judgement $\Delta \vdash s : \sigma$, the type σ is a suffix of some type bound in $\Gamma \cup \{\overline{x} : \overline{\theta}\}$.*

A consequence of the property is that, given a type environment Γ and a term t, it is straightforward to compute the set of valid type judgements for s under subsets of Γ, which we denote $\mathbb{T}_{\Gamma}(s)$. That is, given type environment Δ and intersection type θ:

$$(\Delta, \theta) \in \mathbb{T}_{\Gamma}(s) \iff \Delta \subseteq \Gamma \ \wedge\ \Delta \vdash s : \theta$$

From $\mathbb{T}_{\Gamma}(s)$, it is easy to derive $\mathcal{F}_{\mathrm{nw}}(\Gamma)$ and $TEnv_{\Gamma}$ (letting $s = \mathcal{R}(F)$ for each non-terminal symbol F), which allow for the computation of the fixpoint and the construction of the game graph respectively.

Given a purely applicative term $s := \xi\, t_1 \cdots t_n$, we define $\mathbb{T}_{\Gamma}(s)$ by a recursive procedure as follows. For ease of exposition, we abuse notation to write $\xi : \theta \in \Gamma$ to

indicate that $\xi : \theta$ is a valid conclusion either by the (TERM) rule (in case ξ is a terminal symbol) or by the (VAR) rule (in case ξ is a non-terminal or variable symbol).

$$\begin{array}{l} \left(\{\xi \ : \ \bigwedge S_1 \to \cdots \to \bigwedge S_n \to \theta\} \cup \bigcup_{i=1}^{n} \bigcup_{j \in I_i} \Gamma_{ij},\ \theta\right) \ \in \ \mathbb{T}_\Gamma(\xi\, t_1 \cdots t_n) \\ \iff \\ \begin{cases} \xi \ : \ \bigwedge S_1 \to \cdots \to \bigwedge S_n \to \theta \ \in \ \Gamma \quad \wedge \\ \forall i \in \{1, \cdots, n\} \,.\, \{(\Gamma_{ij}, \theta_{ij}) \mid j \in I_i\} \subseteq \mathbb{T}_\Gamma(t_i) \ \wedge \ \bigwedge_{j \in I_i} \theta_{ij} \leq \bigwedge S_i. \end{cases} \end{array}$$

Observe that, owing to the suffix property, the computation of $\mathbb{T}_\Gamma(s)$ need only consider types in Γ and not all those types in the upward closure of Γ under $\leq$ (which would be disastrous for the construction of the game graph). All the types that occur on the right hand side of the defining equation are either derived from a recursive call or themselves belong to the environment Γ.

Typing judgement incorporating subtyping is necessary

Consider the higher order recursion scheme $G = (\Sigma, \mathcal{N}, \mathcal{R}, S)$ with terminals Σ and non-terminals $\mathcal{N}$ given by (left and right columns respectively):

$$\begin{array}{ll} \mathtt{a} : o \to o \to o & S : o \\ \mathtt{b} : o \to o & F : ((o \to o) \to o) \to o \\ \mathtt{c} : o \to o & G : (o \to o) \to o \\ \mathtt{i} : o & M : o \to o \end{array}$$

and production rules $\mathcal{R}$ given by:

$$\begin{array}{c} S \to F\, G \\ F\, f \to \mathtt{a}\, (f\, \mathtt{b})\, (f\, M) \\ G\, g \to g\, \mathtt{i} \\ M\, x \to \mathtt{a}\, (\mathtt{b}\, x)\, (\mathtt{c}\, x) \end{array}$$

Furthermore, fix the following deterministic trivial automaton $\mathcal{A} = (\Sigma, Q, \delta, q_0)$ whose states are $Q = \{q_0, q_1\}$ and whose transition function, δ, is given by:

$$q_0 \xrightarrow{\mathtt{a}} q_0\, q_0 \quad q_0 \xrightarrow{\mathtt{b}} q_0 \quad q_0 \xrightarrow{\mathtt{i}} \epsilon \quad q_0 \xrightarrow{\mathtt{c}} q_1 \quad q_1 \xrightarrow{\mathtt{t}} \epsilon$$

It should be clear that $\mathcal{A}$ accepts $[\![G]\!]$. Yet the algorithm will produce a type environment Γ containing $F : ((q_0 \wedge q_1 \to q_0) \to q_0) \to q_0$ as the only binding for non-terminal F. Consequently, G is not well typed under Γ in the type system *without* subtyping since, in particular, in this system it is not the case that:

$$\Gamma \vdash \mathcal{R}(F) : ((q_0 \wedge q_1 \to q_0) \to q_0) \to q_0$$

For such a statement to be derivable requires a derivable judgement

$$f : (q_0 \wedge q_1 \to q_0) \to q_0 \vdash f : (q_0 \to q_0) \to q_0$$

as a prerequisite.

G Optimisation

(i) *Environment Canonisation.* The *types* in our fixpoint Γ are already in a canonical form. We canonise the *environment* by finding, for each set of bindings to a non-terminal $F_i : \bigwedge S_i$, the canonical form of the conjunction of those types $\bigwedge S_i$. By Corollary 2 in Appendix G.1, the resulting environment remains a fixpoint. Although the reduction in number of bindings may be small, this can still have a significant effect on the speed and memory usage of the computation by reducing the branching factor.

(ii) *Bitset Environment Representation.* All non-terminal bindings occurring in the computation of $\mathbb{T}_\Gamma$ are drawn from the fixpoint Γ, which is usually small. Therefore we encode this portion of a type environment efficiently as a fixed-length bit string, each bit signalling the presence or absence of a particular binding. As well as reducing the average memory needed to store an environment, this also increases the speed of union and subset operations on environments by reducing them to simple bit-level operations.

(iii) *Environment Minimisation.* Whenever a single environment Γ_i is to be picked from a set of possible environments $\Gamma_1, \cdots, \Gamma_n$, it is safe to consider just a minimal subset; see Appendix G.2 for the justification. Here, a minimal subset of a set $\mathcal{K}$ of environments is the smallest $\mathcal{J} \subseteq \mathcal{K}$ such that, for every $\Gamma \in \mathcal{K}$, there is a $\Delta \in \mathcal{J}$ with $\Delta \subseteq \Gamma$. Such a subset can be found quickly when using a bitset representation.
In applying the rule (L-APP), we minimise the sets of environments in 4 places. This drastically reduces the explosion of environments within this rule.

(iv) *Subtype Minimisation.* When applying the rule (L-APP) and picking for each argument subtypes $\bigwedge_{j \in I_i} \theta_{ij} \leq \bigwedge S_i$, it is sufficient to consider a minimal set of permissible subtypes; see Appendix G.2 for a discussion. This optimisation does not remove any environments that are not removed by environment minimisation. However, it is still worthwhile as it reduces the explosion of environments at an intermediate stage in the computation. The final row of the table demonstrates this optimisation being applied without environment minimisation; this shows that it is fairly effective by itself.

(v) *Collapsing optimisation.* If the tree generated by a recursion scheme is accepted by a deterministic trivial automaton, and the tree is finite, then it is accepted by the corresponding co-trivial automaton. The single-state trivial automaton with transitions for every terminal in an alphabet accepts every tree labelled with that alphabet. If the corresponding co-trivial automaton accepts a tree, then it must be finite. So to determine if a co-trivial automaton accepts a tree, we show that a run-tree exists (as for a trivial automaton) and then check finiteness using the single-state co-trivial automaton, rewriting all states in types being considered to a single type, q_0.

G.1 Environment canonisation: soundness proof

We define $\Gamma' \leq \Gamma$ to mean $dom(\Gamma) = dom(\Gamma')$, and for every $F : \theta$ in Γ, there exists $F : \theta'$ in Γ' such that $\theta' \leq \theta$.

Lemma 11. *If $\Gamma \vdash \xi\, t_1 \cdots t_n : \tau$ (each t_i is assumed to be an applicative term) and $\Gamma' \leq \Gamma$ then $\Gamma' \vdash \xi\, t_1 \cdots t_n : \tau'$ for some $\tau' \leq \tau$.*

Proof. Let $\theta = \bigwedge S_1 \to \cdots \to \bigwedge S_n \to \tau$, and suppose $\xi : \theta$ is in Γ. Let $\theta' = \bigwedge S'_1 \to \cdots \to \bigwedge S'_n \to \tau'$, and suppose $\xi : \theta'$ is in Γ' with $\theta' \leq \theta$. I.e. $\tau' \leq \tau$, and for each i, $\bigwedge S_i \leq \bigwedge S'_i$. From the assumption, we have:

(i) for each i, for each $j \in I_i$, $\Gamma_{ij} \vdash t_i : \theta_{ij}$
(ii) for each i, $\bigwedge_{j \in I_i} \theta_{ij} \leq \bigwedge S_i$
(iii) $\Gamma = \{\xi : \theta\} \cup \bigcup_i \bigcup_{j \in I_i} \Gamma_{ij}$.

For each i and j, choose $\Gamma'_{ij} \subseteq \Gamma'$ such that $\Gamma'_{ij} \leq \Gamma_{ij}$. It follows from (i) and the Induction Hypothesis that for each i and j, $\Gamma'_{ij} \vdash t_i : \theta'_{ij}$, for some $\theta'_{ij} \leq \theta_{ij}$. It follows from (ii) that for each i, $\bigwedge_{j \in I_i} \theta'_{ij} \leq \bigwedge S'_i$. Set $\Gamma' := \{\xi : \theta'\} \cup \bigcup_i \bigcup_{j \in I_i} \Gamma'_{ij}$. Then, $\Gamma' \leq \Gamma$, and by (L-APP), we have $\Gamma' \vdash \xi\, t_1 \cdots t_n : \tau'$ as required.

Corollary 1. *If $\Gamma \vdash \mathcal{R}(H) : \sigma$ and $\Gamma' \leq \Gamma$ then $\Gamma' \vdash \mathcal{R}(H) : \sigma$.*

Proof. Suppose $\sigma = \bigwedge S_1 \to \cdots \to \bigwedge S_n \to q$, $\mathcal{R}(H) = \lambda x_1 \cdots x_n.t$ and $FVar(t) = \{x_{i_1}, \cdots, x_{i_k}\}$. Then,

$$\Gamma, x_{i_1} : \bigwedge S_{i_1}, \cdots, x_{i_k} : \bigwedge S_{i_k} \vdash t : q.$$

Since $\Gamma', x_{i_1} : \bigwedge S_{i_1}, \cdots, x_{i_k} : \bigwedge S_{i_k} \leq \Gamma, x_{i_1} : \bigwedge S_{i_1}, \cdots, x_{i_k} : \bigwedge S_{i_k}$, it follows from Lemma 11 that $\Gamma', x_{i_1} : \bigwedge S_{i_1}, \cdots, x_{i_k} : \bigwedge S_{i_k} \vdash t : q$, and so, $\Gamma' \vdash \mathcal{R}(H) : \sigma$.

Corollary 2. *If Γ is a $\mathcal{F}_{\mathrm{nw}}$-fixpoint and $\{F : \theta, F : \theta'\} \subseteq \Gamma$ with $\theta' \leq \theta$ then $\Gamma \setminus \{F : \theta\}$ is also a $\mathcal{F}_{\mathrm{nw}}$-fixpoint.*

Proof. Take a binding $G : \alpha$ in $\Gamma \setminus \{F : \theta\}$. Since Γ is a $\mathcal{F}_{\mathrm{nw}}$-fixpoint, $\Delta \vdash \mathcal{R}(H) : \alpha$ is provable for some $\Delta \subseteq \Gamma$. Let Δ' be obtained from Δ by replacing the binding $F : \theta$ by $F : \theta'$. Since $\Delta' \leq \Delta$, thanks to Corollary 1, we have $\Delta' \vdash \mathcal{R}(H) : \alpha$. As $\Delta' \subseteq (\Gamma \setminus \{F : \theta\})$, $\Gamma \setminus \{F : \theta\}$ is also a $\mathcal{F}_{\mathrm{nw}}$-fixpoint.

G.2 Optimising $\mathbb{T}_\Gamma$: subset minimalisation

The motivation for this optimisation stems from the observation that in the weak Büchi game derived from $\mathbb{T}_\Gamma$, certain edges from E-nodes can be safely removed. Namely, if there are edges from an E-node $(F : \theta)$ to Γ and Γ' with $\Gamma \subseteq \Gamma'$, then remove the edge to Γ'. (If Abelard can win from Γ, then he can also win from Γ', as it offers him more options for his next move. So there is never any reason for Éloïse to play Γ' when she can play Γ. Hence it is safe to remove the edge to Γ'.) We aim to use this idea to construct an optimised version of $\mathbb{T}_\Gamma$, $\mathbb{T}^{\mathrm{opt}}_\Gamma$.

Consider the computation of $\mathbb{T}_\Gamma(\xi\, t_1 \cdots t_n)$, looking specifically at a single argument term t_k. Let $\mathcal{U} = \{U_1, \ldots, U_m\}$ be the set of all sets U_i, such that $\bigwedge U_i \leq \bigwedge S_k$ and for each $\theta \in U_i$ there exists a $\Xi \subseteq \Gamma$ such that $(\Xi, \theta) \in \mathbb{T}_\Gamma(t_k)$. To define $\mathbb{T}^{\mathrm{opt}}_\Gamma$, we consider a subset $\overline{\mathcal{U}} = \{U_{x_1}, \cdots, U_{x_l}\}$ of $\mathcal{U}$ that is:

(i) *complete*: for every $i \in [1, m]$ there exists a $j \in [1, l]$ such that $U_{x_j} \subseteq U_i$;
(ii) *minimal*: for each $j \in [1, l]$, if for some $i \in [1, m]$ we have $U_i \subseteq U_{x_j}$ then $U_i = U_{x_j}$.

Note that if $U_i = \varnothing$ for some i then $\overline{\mathcal{U}} = \{\varnothing\}$.

To see the effect of the choice of $\overline{\mathcal{U}}$, consider the construction of the resulting environments. Suppose $U_y \in \mathcal{U} = \{\theta_1, \cdots, \theta_p\}$ with:

$$\begin{array}{cc} \{\Delta \mid (\Delta, \theta_1) \in \mathbb{T}_\Gamma(t_k)\} = \{\Delta_{11}, \cdots, \Delta_{1x_1}\} = D_1 \\ \vdots \qquad\qquad\qquad \vdots \\ \{\Delta \mid (\Delta, \theta_p) \in \mathbb{T}_\Gamma(t_k)\} = \{\Delta_{p1}, \cdots, \Delta_{1x_p}\} = D_p \end{array}$$

Set $\{\Delta_1, \cdots, \Delta_z\} = \{\bigcup_{j=1}^{p} \Theta_j \mid (\Theta_1, \cdots, \Theta_p) \in D_1 \times \cdots \times D_p\}$. Let $\{\Delta'_1, \cdots, \Delta'_{z'}\}$ be the corresponding set of type environments constructed in the same way from $U_z \in \overline{\mathcal{U}}$, with $U_z \subseteq U_y$ witnessing the completeness of $\overline{\mathcal{U}}$. As $U_z \subseteq U_y$, for each $j \in [1, z]$ there exists an $i \in [1, z']$ such that $\Delta'_i \subseteq \Delta_j$.

We show, by induction on purely applicative terms t, that for every t, θ and Δ, if $(\Delta, \theta) \in \mathbb{T}_\Gamma(t)$ then for some $\Delta' \subseteq \Delta$, $(\Delta', \theta) \in \mathbb{T}^{\mathrm{opt}}_\Gamma(t)$. The base case is obvious. Write $\tau = \bigwedge S_1 \to \cdots \to \bigwedge S_n \to \theta$. Suppose $(\{\xi : \tau\} \cup \bigcup_{i=1}^{n} \bigcup_{j \in I_i} \Gamma_{ij}, \theta) \in \mathbb{T}_\Gamma(\xi\, t_1 \cdots t_n)$. By definition, for each $i \in [1, n]$ there exists $\{\theta_{ij} \mid j \in I_i\}$ such that $\bigwedge_{j \in I_i} \theta_{ij} \leq \bigwedge S_i$, and for each i and j, $(\Gamma_{ij}, \theta_{ij}) \in \mathbb{T}_\Gamma(t_i)$. By the induction hypothesis, for each $i \in [1, n]$ and $j \in I_i$, there exists $\Gamma'_{ij} \subseteq \Gamma_{ij}$ such that $(\Gamma'_{ij}, \theta_{ij}) \in \mathbb{T}^{\mathrm{opt}}_\Gamma(t_i)$. Then, by definition of $\mathbb{T}^{\mathrm{opt}}_\Gamma$, for each i, there exists $I_i^- \subseteq I_i$ such that $\bigwedge_{j \in I_i^-} \theta_{ij} \leq \bigwedge S_i$. Thus $(\{\xi : \tau\} \cup \bigcup_{i=1}^{n} \bigcup_{j \in I_i^-} \Gamma'_{ij}, \theta) \in \mathbb{T}^{\mathrm{opt}}_\Gamma(\xi\, t_1 \cdots t_n)$. We note that $\{\xi : \tau\} \cup \bigcup_{i=1}^{n} \bigcup_{j \in I_i^-} \Gamma'_{ij} \subseteq \{\xi : \tau\} \cup \bigcup_{i=1}^{n} \bigcup_{j \in I_i} \Gamma_{ij}$ as required.

As an immediate consequence, we have soundness of the optimisation: for each $(F : \theta)$ in Γ, and for each $(\Delta, \theta) \in \mathbb{T}_\Gamma(\mathcal{R}(F))$, there exists $(\Delta', \theta) \in \mathbb{T}^{\mathrm{opt}}_\Gamma(\mathcal{R}(F))$ such that $\Delta' \subseteq \Delta$.

H Examples

H.1 Intercept

For this example, we take a network-oriented OCaml program. This program reads an arbitrary amount of data from a network socket into a queue and is then responsible for forwarding the data on to another socket. The full program can be found at [21] and an abstracted form in ML-like syntax as found elsewhere in the paper follows.

```
let rec g y n = for i in 1 to n do write(y) ; done ; close(y)
let rec f x y n = if b then read(x) ; f(x,y,n+1)
                       else close(x) ; g(y,n)
let t = open_out "socket2" in
let s = open_in "socket1" in f(s,t,0)
```

We can then construct a RUL program via lambda-lifting and CPS transformation.

$$
\begin{aligned}
S &= \mathbf{new}^{\mathrm{r}}\ C1 \\
C1\ x &= \mathbf{new}^{\mathrm{w}}\ (C2\ x) \\
C2\ x\ y &= F\ x\ y\ Zero\ \star \\
F\ x\ y\ n\ k &= \mathbf{if}*\ (\mathbf{acc}_{\mathrm{read}}\ x\ (F\ x\ y\ (Succ\ n)\ k))\ (\mathbf{acc}_{\mathrm{close}}\ x\ (G\ y\ n\ k) \\
G\ y\ n\ k &= n\ (\mathbf{acc}_{\mathrm{write}}\ y)\ (\mathbf{acc}_{\mathrm{close}}\ y\ k) \\
Zero\ f\ x &= x \\
Succ\ n\ f\ x &= f\ (n\ f\ x)
\end{aligned}
$$

For this program it would be useful to confirm that if the "in" socket stops transmitting data then the "out" socket is eventually closed ($AG\ close_{in} \Rightarrow AF\ close_{out}$). Given that this example property is slightly more complicated, requiring interplay *between* resources, a slightly altered transformation to HORS is required that differentiates between operations on the two resources, while also unconditionally instantiating both resources. Given this fixed number of resources it is possible to extend the alphabet in this way:

$$
\begin{aligned}
S &\to Newr\ C1 \\
C1\ x &\to Neww\ (C2\ x) \\
C2\ x\ y &\to F\ x\ y\ Zero\ end \\
F\ x\ y\ n\ k &\to br\ (Read\ x\ (F\ x\ y\ (Succ\ n)\ k)) \\
&\qquad (Closer\ x\ (G\ y\ n\ k) \\
G\ y\ n\ k &\to n\ (Write\ y)\ (Closew\ y\ k) \\
I\ x\ y &\to x\ y \\
K\ x\ y &\to y \\
Newr\ k &\to newr\ (k\ I) \\
Neww\ k &\to neww\ (k\ I) \\
Closer\ x\ k &\to x\ closer\ k \\
Closew\ x\ k &\to x\ closew\ k \\
Read\ x\ k &\to x\ read\ k \\
Write\ x\ k &\to x\ write\ k \\
Zero\ f\ x &\to x \\
Succ\ n\ f\ x &\to f\ (n\ f\ x)
\end{aligned}
$$

The property may then be specified as the AWT $\mathcal{A} = \langle \Sigma, \{q_0, q_f\}, \delta, q_0, \{q_0 \mapsto 0, q_f \mapsto 1\}\rangle$. δ is defined below (where $q \in \{q_0, q_f\}$ and $* \in \{newr, neww, read, write\}$):

$$
\begin{aligned}
(q, *) &\mapsto (1, q) \\
(q, br) &\mapsto (1, q) \wedge (2, q) \\
(q_0, closer) &\mapsto (1, q_f) \wedge (1, q_0) \\
(q_0, closew) &\mapsto (1, q_0) \\
(q_f, closer) &\mapsto (1, q_f) \\
(q_f, closew) &\mapsto \mathtt{t}
\end{aligned}
$$

We can see that the AG part of the property is handled by the "$(q_0, closer)$" transition, which spawns an additional copy of the automaton to check that $AF\ closew$ holds from this state.

H.2 Boolean

This example was chosen to demonstrate that model-checking of first-order boolean programs can be performed using this method. We take as input the following small C function:

```
#include <assert.h>
void foo(int x, int y, int z, int w)
{
  do {
    z = 0;
    x = y;
    if (w) {
      x++;
      z = 1;
    }
  } while (x!=y);

  assert (z==0);
}
```

Through standard predicate abstraction, taking the predicate set $\{x = y, w \neq 0, z = 0\}$, we obtain a boolean program:

```
decl x_eq_y;
decl w_neq_0;
decl z_eq_0;

void main() begin
  x_eq_y := *;
  w_neq_0 := *;
  z_eq_0 := *;
loop_start:
  z_eq_0 := 1;
  x_eq_y := 1;
  if(w_neq_0) then
    x_eq_y := 0;
    z_eq_0 := 1;
  fi;
  if(!x_eq_y) then
    goto loop_end;
  fi;
  goto loop_start;
loop_end:
  assert (z_eq_0);
end
```

We can perform a literal translation to HORS by modelling the state as a number of formal parameters passed from one line in the boolean program (or rule in the HORS) to the next:

$$
\begin{aligned}
S &\to L1\_1 \\
L1\_1 &\to br\,(L1\_2\ True)\,(L1\_2\ False) \\
L1\_2\ x &\to br\,(L1\_3\ x\ True)\,(L1\_3\ x\ False) \\
L1\_3\ x\ y &\to br\,(L2\ x\ y\ True)\,(L2\ x\ y\ False) \\
L2\ x\ y\ z &\to L3\ x\ y\ True \\
L3\ x\ y\ z &\to L4\ \ True\ y\ z \\
L4\ x\ y\ z &\to y\,(L5\ x\ y\ z)\,(L7\ x\ y\ z) \\
L5\ x\ y\ z &\to L6\ False\ y\ z \\
L6\ x\ y\ z &\to L7\ x\ y\ True \\
L7\ x\ y\ z &\to x\,(L9\ x\ y\ z)\,(L8\ x\ y\ z) \\
L8\ x\ y\ z &\to L10\ x\ y\ z \\
L9\ x\ y\ z &\to L2\ x\ y\ z \\
L10\ x\ y\ z &\to z\ end\ fail \\
True\ x\ y &\to x \\
False\ x\ y &\to y
\end{aligned}
$$

Here the *fail* terminal signals failure of the assertion, and therefore we only need to check that it is never encountered. This can be done using the trivial automaton $\mathcal{A} = \langle \Sigma, \{q_0\}, \{(q_0, br) \mapsto q_0\,q_0, (q_0, end) \mapsto \mathtt{t}\}, q_0, \{q_0 \mapsto 0\}\rangle$.

H.3 Imperative

This example allows us to show a different approach to verifying the following small imperative program used as running example by Cook *et al.* [15].

```
while(*) {
  x := 1;
  n := *;
  while(n>0) {
    n := n - 1;
  }
  x := 0;
}
while(1) {}
```

Using Church numerals as in Section H.1 we can model the *while* loop and obtain a fairly direct translation:

$$
\begin{aligned}
S &\to br\ EnterLoop\ end \\
EnterLoop &\to set\_x\_one\,(IncCounter\ Zero\,(Loop\ end)) \\
IncCounter\ n\ f &\to br\,(RunLoop\ n\ f)\,(IncCounter\,(Succ\ n)\ f) \\
RunLoop\ n\ f &\to n\ decr\,(set\_x\_zero\ f) \\
Zero\ f\ x &\to x \\
Succ\ n\ f\ x &\to f\,(n\ f\ x) \\
Loop\ k &\to loop\,(Loop\ k)
\end{aligned}
$$

The suggested property is $\varphi = AG[(x = 1) \Rightarrow AF(x = 0)]$, a CTL formula that we can check using the automaton $\mathcal{A} = \langle \Sigma, \{q_0, q_r, q_c\}, \delta, q_0, \{q_0 \mapsto 0, q_r \mapsto 1, q_c \mapsto 0\}\rangle$. δ is defined below (where $q \in \{q_0, q_r\}$):

$$
\begin{aligned}
(q_0, set\_x\_one) &\mapsto q_0 \\
(q_0, set\_x\_zero) &\mapsto q_c \\
(q_0, end) &\mapsto \mathtt{t} \\
(q_0, decr) &\mapsto q_r \\
(q, br) &\mapsto q\, q \\
(q_r, set\_x\_one) &\mapsto q_c \\
(q_r, decr) &\mapsto q_r \\
(q_c, loop) &\mapsto q_c
\end{aligned}
$$