# Reachability games and game semantics: comparing nondeterministic programs*

ANDRZEJ S. MURAWSKI
Oxford University Computing Laboratory, Wolfson Building, Parks Road, Oxford OX1 3QD, UK

## Abstract

*We investigate the notions of may- and must-approximation in Erratic Idealized Algol (a nondeterministic extension of Idealized Algol), and give explicit characterizations of both inside its game model. Notably, must-approximation is captured by a novel preorder on nondeterministic strategies, whose definition is formulated in terms of winning regions in a reachability game. The game is played on traces of one of the strategies and its objective is reaching a complete position without encountering any divergences.*

*The concrete accounts of may- and must-approximation make it possible to derive tight complexity bounds for the corresponding decision problems in the finitary (finite datatypes) variant* $\mathsf{EIA}_f$ *of Erratic Idealized Algol. In fact we give a complete classification of the complexity of may- and must-approximation for fragments of* $\mathsf{EIA}_f$ *of bounded type order (for terms in* $\beta$*-normal form). The complexity of the decidable cases ranges from PSPACE to 2-EXPTIME for may-approximation and from EXPSPACE to 3-EXPTIME for must-approximation.*

*Our decidability results rely on a representation theorem for nondeterministic strategies which, for a given term, yields a single (finite or visibly pushdown) automaton capturing both traces and divergences of the corresponding strategy with two distinct sets of final states. The decision procedures producing optimal bounds incorporate numerous automata-theoretic techniques: complementation, determinization, computation of winning regions in reachability games over finite and pushdown graphs as well as product constructions.*

*We see our work as a starting point of research that relates game semantics with other game-based theories.*

## 1. Introduction

Game semantics [3, 20] has emerged as a versatile methodology for giving semantics to programming languages. In particular, its concrete flavour has quickly inspired applications to program analysis [23, 24, 25]. In recent years a subarea, called *algorithmic game semantics* [1], was born, which seeks to establish connections between game semantics and models of computation with a view to applying them to software verification. The insights obtained this way have already contributed several verification tools [1, 11, 14, 7, 22] and helped to prove new decidability/undecidability results in program verification [13, 32, 27, 31, 29, 30, 33].

A significant number of papers in this spirit concerned Idealized Algol, a "golden standard" for investigating higher-order imperative computation and a good starting point to an automata-theoretic analysis of game semantics. They have exploited full abstraction of game models in order to provide effective means for analyzing contextual *approximation* and *equivalence*. Intuitively, one program is said to *approximate* another iff, in any context, the latter behaves at least as well as the former. Two programs are *equivalent* iff their behaviour is indistinguishable in any context. Of course, these definitions crucially depend on what "behaviour" means. In a deterministic setting, such as Idealized Algol, one typically defines "behaviour" as termination. In many game models contextual approximation and equivalence can be recast in a more explicit way, opening up the way to using games to reason about these concepts. Indeed, in recent years a complete classification of complexity bounds was obtained for the problems of deciding approximation and equivalence in finitary fragments of Idealized Algol, restricted by type order and the availability of recursion [32, 27, 28, 31, 29]. The table below summarizes the bounds[1] for equivalence; they apply to $\beta$-normalized terms. Those for approximation are the same except that "decidable" should be replaced with "undecidable".

| order | pure | $+\mathbf{while}$ | $+^2\mu_0$ | $+\mu_1$ |
|---|---|---|---|---|
| 1 | co-NP | PSPACE | dec. | dec. |
| 2 | PSPACE | PSPACE | dec. | undec. |
| 3 | EXPTIME | EXPTIME | dec. | undec. |
| 4 | undec. | undec. | undec. | undec. |

*A long version of this paper is available online [26].

[1] Whenever a complexity class is mentioned, completeness has been proved. "dec." stands for "decidable"; more precisely, "reducible to the language equivalence problem for deterministic pushdown automata (of exponentially larger size)" [36].

[2] $+\mu_i$ signifies the availability of recursion at order $i$, i.e. the outcome

The goal of this paper is to contribute an equally comprehensive understanding of Erratic Idealized Algol, i.e. Idealized Algol extended with (finite) nondeterminism. In the nondeterministic case, one traditionally distinguishes two notions of "behaviour" inducing two complementary notions of contextual approximation [10]: the possibility of termination (may-terminate) leading to may-approximation and the guarantee of termination (must-terminate) giving rise to must-approximation. As in the deterministic case, both are problematic to reason about directly because of quantification over all contexts.

The two preorders have been studied for Erratic Idealized Algol by Harmer and McCusker in [17, 16] through game semantics. In order to capture must-approximation they extended the deterministic definition of strategies (solely based on traces) to a notion of nondeterministic strategy, which comprises two components: traces and divergences, the latter being inspired by process-algebraic semantics [18]. Although they show how to capture both may- and must-approximation in this framework, the corresponding full abstraction results rely on quotienting with respect to the so-called *intrinsic* preorder, which, similarly to the notions of contextual approximation and equivalence, involves quantification over all strategies mimicking contexts. The first two of our results (Propositions 3.1 and 3.8) show how to avoid the intrinsic preorder when characterizing may- and must-approximation.

As in other settings, may-approximation turns out easier to account for, because – as in the deterministic case – it can be captured through (complete) traces. Consequently, may-approximation corresponds to the containment problem for the sets of induced complete plays. This, in conjunction with a representation theorem (Theorem 4.5) for terms of finitary Idealized Algol ($\mathsf{EIA}_f$), allows us to derive decidability results for may-approximation via the language containment problem for the corresponding automata. Moreover, we can also give exact complexity bounds for may-approximation between terms in $\beta$-normal form. The optimality of the bounds is confirmed by reductions via computation histories. The flavour of the hardness proofs is similar to hardness proofs for universality problems in formal language theory: one finds a term whose game semantics corresponds to strings *not* representing a successful computation. We summarize the bounds for may-approximation below in a table analogous to the previous one. The bounds for may-equivalence are exactly the same.

| order | pure | $+\mathbf{while}$ | $+\mu_0$ |
|---|---|---|---|
| 1 | PSPACE | EXPSPACE | undecidable |
| 2 | EXPSPACE | EXPSPACE | undecidable |
| 3 | 2-EXPTIME | 2-EXPTIME | undecidable |
| 4 | undecidable | undecidable | undecidable |

of a recursive definition can be a term whose type is of order (at most) $i$.

Must-approximation in $\mathsf{EIA}_f$ presents a more difficult challenge, as it does not seem to correspond to any explicit preorder on traces and divergences known to date [37]. Thus we see our explicit characterization of must-approximation given in Proposition 3.8 as arguably the most interesting contribution of the paper. The new preorder we introduce in Definition 3.7 is based on concepts borrowed from the theory of reachability games, as used in program verification [15]. In short, we view a nondeterministic strategy as a reachability game, in which the player O tries to complete a play without encountering any divergences. This perspective allows us to ask whether a given play is winning for either of the players, which forms the basis of our new preorder: one strategy improves upon another if any difference between the former and the latter (trace or divergence) is compensated by a winning position for the player P in the reachability game associated with the latter.

This more concrete account of must-approximation makes it possible to use techniques for solving reachability games on finite and pushdown graphs in order to decide must-approximation in $\mathsf{EIA}_f$. Indeed we demonstrate how this can be done through our representation theorem (Theorem 4.5), which shows how, given a suitably restricted $\mathsf{EIA}_f + \mathbf{while}$ term, one can construct a corresponding automaton with two sets of final states that capture traces and divergences respectively. Then we can decide must-approximation with a decision procedure that integrates determinization theorems for finite and visibly pushdown automata, algorithms for calculating winning regions in reachability games over finite and pushdown graphs, backward determinization of alternating finite automata and product constructions. The complexity bounds obtained this way are given below. They turn out tight as confirmed by hardness proofs in which, in each case, we construct a term of relevant order such that the associated reachability game corresponds to building up a valid (alternating) machine run of the desired complexity (and divergences correspond to possible errors). Bounds for must-equivalence are the same as those given below, as are bounds for may&must-approximation and may&must-equivalence.

| | pure | $+\mathbf{while}$ | $+\mu_0$ |
|---|---|---|---|
| 1 | PSPACE | 2-EXPTIME | undecidable |
| 2 | 2-EXPTIME | 2-EXPTIME | undecidable |
| 3 | 3-EXPTIME | 3-EXPTIME | undecidable |
| 4 | undecidable | undecidable | undecidable |

We view our paper as establishing a happy marriage between two game-based theories that were developed independently and for different purposes. We hope further fruitful interplay will be possible. For game semantics this would have to entail a shift of emphasis to winning during play. This aspect was already present in one of the early game models [2], based on strategies that were winning in

an abstract sense, but later on turned out dispensable in research on full abstraction. Fortunately, it seems reasonable to expect that attempts to apply game semantics to program specification and verification should create new opportunities for interaction between game semantics and other kinds of games, possibly including those with more complicated objectives than reachability.

## 2. Erratic Idealized Algol and its game model

Erratic Idealized Algol [17] is a nondeterministic variant of Reynolds's Idealized Algol [34]. Its types are generated by the grammar $\theta ::= \beta \,|\, \theta \to \theta$, where $\beta$ ranges over the *ground* types $\mathsf{com}$, $\mathsf{exp}$, $\mathsf{var}$ of commands, expressions and variables respectively. The order $\mathsf{ord}(\theta)$ of a type is defined by: $\mathsf{ord}(\beta) = 0$ and $\mathsf{ord}(\theta_1 \to \theta_2) = \max(\mathsf{ord}(\theta_1) + 1, \mathsf{ord}(\theta_2))$. We shall consider a finitary version $\mathsf{EIA}_f$ of Erratic Idealized Algol, intended to make the original language more amenable to automata-theoretic analysis. The ground types of $\mathsf{EIA}_f$ are finite and recursion is forbidden. The syntax is given in Figure 1. $\Gamma$ consists of typed free identifiers $x : \theta$. The $\Omega_\beta$ constant represents divergence. In what follows we shall consider some extensions of $\mathsf{EIA}_f$, notably, with while-loops and fixed points:

$$\frac{\Gamma \vdash M : \mathsf{exp} \quad \Gamma \vdash N : \mathsf{com}}{\Gamma \vdash \mathbf{while}\, M\, \mathbf{do}\, N : \mathsf{com}} \qquad \frac{\Gamma, x : \theta \vdash M : \theta}{\Gamma \vdash \mu x^\theta.M : \theta}.$$

We call a term-in-context $\Gamma \vdash M : \theta$ an *ith-order term* provided its typing derivation uses exclusively judgments of the shape $\Gamma' \vdash M' : \theta'$, where the types of the free identifiers in $\Gamma'$ are of order less than $i$ and $\mathsf{ord}(\theta') \leq i$. The collection of $i$th-order $\mathsf{EIA}_f$ terms will be denoted by $\mathsf{EIA}_i$. Similarly we can define $i$th-order fragments of $\mathsf{EIA}_f + \mathbf{while}$ and $\mathsf{EIA}_f + \mu$, which we refer to as $\mathsf{EIA}_i + \mathbf{while}$ and $\mathsf{EIA}_i + \mu$ respectively. For a finer analysis of fixed points we also distinguish subfragments $\mathsf{EIA}_i + \mu_j$ $(j < i)$ of $\mathsf{EIA}_i + \mu$ in which the fixed-point rule has been applied to types $\theta$ with $\mathsf{ord}(\theta) \leq j$.

The *call-by-name* operational semantics of $\mathsf{EIA}_f + \mu$ is given in the Appendix (Figure 3). The judgments have the shape $s, M \Downarrow s', V$, where $s, s' : \{x_1, \cdots, x_n\} \to \{0, \cdots, max\}$ represent the store before and after reduction (we assume that variables are always initialized to 0).

Note that, due to nondeterminism introduced by the rule for $\mathbf{or}$, for any given $s$, we may have $s, M \Downarrow s', V$ for multiple pairs $s', V$. Hence $s, M \Downarrow s', V$ is best read as "M may converge to V". Using $\Downarrow$, one can define a notion of program approximation for *arbitrary* terms-in-context as follows. By convention $M \Downarrow V$ stands for $\emptyset, M \Downarrow \emptyset, V$.

**Definition 2.1.** Given two terms $\Gamma \vdash M_1 : \theta$ and $\Gamma \vdash M_2 : \theta$, we shall say that $\Gamma \vdash M_1 : \theta$ *may-approximates* $\Gamma \vdash M_2 : \theta$ iff, for any context $C[-]$ such that $C[M_1] : \mathsf{com}$ (equivalently $C[M_2] : \mathsf{com}$), whenever $C[M_1] \Downarrow$ we also have $C[M_2] \Downarrow$. We then write $\Gamma \vdash M_1 \sqsubseteq_{\mathrm{may}} M_2$. Two terms are *may-equivalent*, noted $\Gamma \vdash M_1 \cong_{\mathrm{may}} M_2$, if they may-approximate one another.

Intuitively, may-approximation measures capability for producing values without accounting for possible failures. For instance, $\mathbf{skip} \cong_{\mathrm{may}} \mathbf{skip}\,\mathbf{or}\,\Omega_{\mathsf{com}}$. A complementary notion of approximation is provided by changing the focus from potential termination to the possibility of divergence so that an approximant is required to be "at least as divergent" as the term it approximates. Typically, this is defined by stating the contrapositive with the help of a must-converge predicate $s, M \Downarrow_{\mathrm{must}}$. We give its exact definition in the Appendix (Figure 4). Informally, $s, M \Downarrow_{\mathrm{must}}$ is supposed to mean that, no matter how nondeterminism is resolved during the reduction of $M$ from state $s$, divergence will not occur.

**Definition 2.2.** Given two terms $\Gamma \vdash M_1 : \theta$ and $\Gamma \vdash M_2 : \theta$, we shall say that $\Gamma \vdash M_1 : \theta$ *must-approximates* $\Gamma \vdash M_2 : \theta$ iff, for any context $C[-]$ such that $C[M_1] : \mathsf{com}$ (equivalently $C[M_2] : \mathsf{com}$), whenever $C[M_1] \Downarrow_{\mathrm{must}}$ we also have $C[M_2] \Downarrow_{\mathrm{must}}$. We then write $\Gamma \vdash M_1 \sqsubseteq_{\mathrm{must}} M_2$. Two terms are *must-equivalent*, noted $\Gamma \vdash M_1 \cong_{\mathrm{must}} M_2$, if they must-approximate one another.

For instance, it turns out that $c : \mathsf{com} \vdash c; \Omega_{\mathsf{com}} \cong_{\mathrm{must}} \Omega_{\mathsf{com}}$. In fact they are both may- and must-equivalent. Combining may- and must-approximation (resp. equivalence) gives rise to may&must-approximation (resp. equivalence), which is a more comprehensive way of comparing nondeterministic programs. In what follows we shall first study may- and must-concepts in isolation. Bounds for may&must-approximation and equivalence will then turn out easy to derive.

**Example 2.3.** Let us write $[cond]$ for $\mathbf{if}\, cond\, \mathbf{then}\, \mathbf{skip}\, \mathbf{else}\, \Omega_{\mathsf{com}}$. For simplicity we often put predicates such as $!X = 0$ in place of $cond$ on the understanding that they may easily be coded up as expressions (returning 0 iff the predicate does not hold). Let us consider the term $f : \mathsf{com} \to \mathsf{com} \vdash \mathbf{new}\, X\, \mathbf{in}\, (f(X{:=}1); [!X = 0])\, \mathbf{or}\, (f(X{:=}1); [!X = 1]) : \mathsf{com}$. It turns out the term is may-equivalent to $f(\mathbf{skip})$, must-equivalent to $\Omega_{\mathsf{com}}$ and may&must-equivalent to $f(\mathbf{skip})\,\mathbf{or}\,\Omega_{\mathsf{com}}$. The truthfulness of these claims will become apparent at the end of Section 3, in which we give explicit characterizations of both may- and must-approximation.

Game semantics views computation as an exchange of moves (a play) between two players, called O and P. It interprets terms as strategies for P in an abstract game derived from the underlying types. A game model of $\mathsf{EIA}_f + \mu$, based on nondeterministic strategies, has been presented in [17].

$$\frac{}{\Gamma \vdash \mathbf{skip} : \mathsf{com}} \qquad \frac{i \in \{0, \cdots, max\}}{\Gamma \vdash i : \mathsf{exp}} \qquad \frac{}{\Gamma \vdash \Omega_\beta : \beta} \qquad \frac{\Gamma \vdash M : \mathsf{exp}}{\Gamma \vdash \mathbf{succ}(M) : \mathsf{exp}} \qquad \frac{\Gamma \vdash M : \mathsf{exp}}{\Gamma \vdash \mathbf{pred}(M) : \mathsf{exp}}$$

$$\frac{\Gamma \vdash M : \mathsf{com} \quad \Gamma \vdash N : \beta}{\Gamma \vdash M; N : \beta} \qquad \frac{\Gamma \vdash M : \mathsf{exp} \quad \Gamma \vdash N_0 : \beta \quad \Gamma \vdash N_1 : \beta}{\Gamma \vdash \mathbf{if}\, M\, \mathbf{then}\, N_1\, \mathbf{else}\, N_0 : \beta} \qquad \frac{\Gamma \vdash M_1 : \mathsf{com} \quad \Gamma \vdash M_2 : \mathsf{com}}{\Gamma \vdash M_1\, \mathbf{or}\, M_2 : \mathsf{com}}$$

$$\frac{\Gamma \vdash M : \mathsf{var}}{\Gamma \vdash !M : \mathsf{exp}} \qquad \frac{\Gamma \vdash M : \mathsf{var} \quad \Gamma \vdash N : \mathsf{exp}}{\Gamma \vdash M{:=}N : \mathsf{com}} \qquad \frac{\Gamma, X : \mathsf{var} \vdash M : \mathsf{com}}{\Gamma \vdash \mathbf{new}\, X\, \mathbf{in}\, M : \mathsf{com}} \qquad \frac{\Gamma, X : \mathsf{var} \vdash M : \mathsf{exp}}{\Gamma \vdash \mathbf{new}\, X\, \mathbf{in}\, M : \mathsf{exp}}$$

$$\frac{}{\Gamma, x : \theta \vdash x : \theta} \qquad \frac{\Gamma, x : \theta \vdash M : \theta'}{\Gamma \vdash \lambda x^\theta.M : \theta \to \theta'} \qquad \frac{\Gamma \vdash M : \theta \to \theta' \quad \Gamma \vdash N : \theta}{\Gamma \vdash MN : \theta'} \qquad \frac{\Gamma \vdash M : \mathsf{exp} \to \mathsf{com} \quad \Gamma \vdash N : \mathsf{exp}}{\Gamma \vdash \mathbf{mkvar}(M, N) : \mathsf{var}}$$

**Figure 1. Syntax of $\mathsf{EIA}_f$.**

**Definition 2.4.** A (nondeterministic) *strategy* $\sigma$ on an arena $A$, written $\sigma : A$, is a pair $(\mathcal{T}_\sigma, \mathcal{D}_\sigma)$ satisfying the following two conditions. (i) $\mathcal{T}_\sigma$ is a non-empty set of even-length plays of $A$ such that $sab \in \mathcal{T}_\sigma$ entails $s \in \mathcal{T}_\sigma$ (we use $a$ and $b$ to range over moves in $A$). Elements of $\mathcal{T}_\sigma$ are called the *traces* of $\sigma$. (ii) $\mathcal{D}_\sigma$ is a set of odd-length plays of $A$ such that $sa \in \mathcal{D}_\sigma$ implies $s \in \mathcal{T}_\sigma$. Moreover, if $s \in \mathcal{T}_\sigma$, $sa$ is a play and there is no $b$ such that $sab \in \mathcal{T}_\sigma$, then there must exist $d \in \mathcal{D}_\sigma$ such that $d$ is a prefix of $sa$. Elements of $\mathcal{D}_\sigma$ are referred to as the *divergences* of $\sigma$.

For the purpose of modelling Erratic Idealized Algol one considers *single-threaded* strategies whose behaviour is determined by the current thread only. Hence, in the following we focus exclusively on analyzing and representing plays and divergences with single initial moves. In particular, $\mathcal{T}_\sigma$ and $\mathcal{D}_\sigma$ will stand for single-threaded plays and divergences generated by the strategy $\sigma$, $P_A$ will be the set of single-threaded plays in the arena $A$. $M_A$ will stand for the set of moves of $A$. Note that in [17] the full abstraction results have been shown by quotienting with respect to the so-called instrinsic preorder. In the next section we give more explicit characterizations of may- and must-approximation that will ultimately allow us to derive decision procedures for the two problems.

## 3 Program approximation concretely

A play $s \in P_A$ is called *complete* iff all of the questions occurring in it have been answered, i.e. any question-move acts as a justifier to an answer-move. Given a set of plays $\sigma$, let $\mathsf{comp}\,(\sigma)$ be the set of non-empty (single-threaded) *complete* plays in $\sigma$. May-approximation can be characterized by inclusion between the sets of induced complete plays. The proof is essentially the same as in the deterministic case [4].

**Proposition 3.1.** $\Gamma \vdash M_1 \sqsubseteq_{\mathrm{may}} M_2$ if and only if $\mathsf{comp}\,([\![\Gamma \vdash M_1]\!]) \subseteq \mathsf{comp}\,([\![\Gamma \vdash M_2]\!])$.

In order to characterize must-approximation explicitly, it turns out fruitful to view strategies as two-player games between O and P. Specifically, we shall take advantage of reachability games over countable graphs.

**Definition 3.2.** A *reachability game* $\mathcal{G}$ is a tuple $\langle V, V^O, V^P, E, W^O \rangle$ such that $V$ is a countable set, $V = V^O + V^P$, $(V, E)$ is a directed graph and $W^O \subseteq V$. Elements of $V^O$ and $V^P$ are called O- and P-vertices respectively and $W^O$ is the set of winning vertices for O.

Given an initial vertex $v \in V$, the two players make "moves" in $\mathcal{G}$ by picking edges from $E$ repeatedly: if the current vertex is in $V^O$ then O gets to choose an outgoing edge, otherwise P does it; "action" then moves on to the target of the selected edge. The players continue making their choices and moving from one vertex to another as long as they can, i.e. ad infinitum or until a vertex without any outgoing edges is reached.

Formally, a (finite) play of $\mathcal{G}$ is a sequence $v_1 \cdots v_m \in V^*$ $(m \geq 0)$ such that $(v_i, v_{i+1}) \in E$ for any $1 \leq i < m$. An infinite play is a sequence from $V^\infty$ each of whose prefixes is a finite play. Thus, a round of $\mathcal{G}$, started at some initial vertex, will generate either a finite maximal or an infinite play. If a vertex from $W^O$ has been visited during the play, O is declared the winner. By convention, even if no vertex from $W^O$ has been visited, O also wins once a vertex from $V^P$ without outgoing transitions is ultimately reached. P wins in all other cases, i.e. when the resultant play (finite or infinite) does not contain any vertices from $W^O$ and ends in a vertex from $V^O$, if finite.

$v$ is called a *winning vertex* for O if, starting from $v$, O can play in such a way that he always wins. The set containing all such vertices is called O's *winning region* and denoted by $Win^O_{\mathcal{G}}$. P's winning region $Win^P_{\mathcal{G}}$ is defined analogously. Obviously $Win^O_{\mathcal{G}} \cap Win^P_{\mathcal{G}} = \emptyset$. Reachability games belong to the class of *closed* games (whenever O wins, the victory is secured in a finite number of steps), which are well-known to be determined [12]: at each vertex one of the players can secure a win for himself. Consequently, $V = Win^O_{\mathcal{G}} + Win^P_{\mathcal{G}}$.

**Definition 3.3.** Suppose $\mathcal{G}_i = \langle V_i, V_i^O, V_i^P, E_i, W_i^O \rangle$ is a

reachability game for $i = 1, 2$. We say that $\mathcal{G}_1$ and $\mathcal{G}_2$ are *isomorphic* iff there exists a graph isomorphism $f$ between the graphs $(V_1, E_1)$ and $(V_2, E_2)$ such that $f(V_1^O) = V_2^O$, $f(V_1^P) = V_2^P$ and $f(W_1^O) = W_2^O$. Note that, if $\mathcal{G}_1$ and $\mathcal{G}_2$ are isomorphic, for any $v \in V_1$ we have: $v \in Win^O_{\mathcal{G}_1}$ if and only if $f(v) \in Win^O_{\mathcal{G}_2}$.

Given a game $\mathcal{G}$ and a vertex $v \in V$ we define a closely related game $\mathsf{Unf}_v(\mathcal{G})$, which is essentially the "game tree" of $\mathcal{G}$ started in $v$.

**Definition 3.4.** Suppose $\mathcal{G} = \langle V, V^O, V^P, E, W^O \rangle$ is a reachability game and $v \in V$. Let $Play_v^O, Play_v^P$ be the sets of finite plays in $\mathcal{G}$ starting in $v$ and ending in vertices from $V^O$ and $V^P$ respectively. Further, let $Play_v = Play_v^O + Play_v^P$. The *unfolding of $\mathcal{G}$ from $v$* is the game $\mathsf{Unf}_v(\mathcal{G}) = \langle Play_v, Play_v^O, Play_v^P, E_v, W_v^O \rangle$, where $E_v = \{(pu, puw) \mid pu \in Play_v, (u, w) \in E\}$ and $W_v^O = \{pw \in Play_v \mid w \in W^O\}$.

**Remark 3.5.** It is easy to see that the underlying graph of $\mathsf{Unf}_v(\mathcal{G})$ is a tree with root $v$. Winning regions in $\mathsf{Unf}_v(\mathcal{G})$ can be related to those in $\mathcal{G}$ in the following way: for any play $p \in Play_v$ ending in $w \in V$, $w \in Win^O_{\mathcal{G}}$ if and only if $p \in Win^O_{\mathsf{Unf}_v(\mathcal{G})}$.

Next we construct a reachability game for an arbitrary strategy, in which O will win if and only if a non-empty complete position is eventually reached and none of its prefixes is a divergence.

**Definition 3.6.** Suppose $\sigma : A$. The associated reachability game $\mathcal{G}_\sigma$ is defined by $\langle V_\sigma^O + V_\sigma^P, V_\sigma^O, V_\sigma^P, E_\sigma, W_\sigma^O \rangle$ where $V_\sigma^O = \mathcal{T}_\sigma + \{\circ\}$, $V_\sigma^P = \{sa \in P_A \mid s \in \mathcal{T}_\sigma, a \in M_A\}$,

$$\begin{array}{rcl} E_\sigma & = & \{(s_1, s_2) \in (V_\sigma^O \times V_\sigma^P) \cup (V_\sigma^P \times V_\sigma^O) \mid \\ & & \qquad \exists a \in M_A.\, s_2 = s_1 a\} \cup (D_\sigma \times \{\circ\}), \\ W_\sigma^O & = & \{s \in \mathsf{comp}(\sigma) \quad \mid \quad \forall s' \sqsubseteq s. \quad s' \notin \mathcal{D}_\sigma\}. \end{array}$$

Observe that $E_\sigma$ is essentially the tree of plays generated by $\sigma$ (edges correspond to allowable moves in $A$) except for edges of the form $(d, \circ)$, where $d \in \mathcal{D}_\sigma$, which are added for a technical reason. Recall that O is also deemed to win if a vertex from $v \in V_\sigma^P$ with no outgoing edges is reached. By Definition 2.4 (ii), if $sa \in V_\sigma^P$ cannot be extended to a play in $\mathcal{T}_\sigma$, then $sa$ must be a divergence. Because we do not want vertices $d \in D_\sigma$ to become winning for O, we have added an edge from $d$ to the dummy vertex $\circ$ (which has no outgoing transitions and allows P to win instead). Consequently, P will never "get stuck" in $\mathcal{G}_\sigma$, so O can win if and only if the players arrive at a non-empty complete position none of whose prefixes is a divergence. To make notation less verbose, we shall write $Win_\sigma^O$ and $Win_\sigma^P$ for the respective winning regions in $\mathcal{G}_\sigma$ (instead of $Win^O_{\mathcal{G}_\sigma}$ and $Win^P_{\mathcal{G}_\sigma}$ respectively).

Now, using winning regions for $\mathcal{G}_\sigma$, we shall define a new relation $\leq_{\mathrm{must}}$ on strategies over any arena $A$, which will turn out to characterize must-approximation. Intuitively, we want $\sigma_1 \leq_{\mathrm{must}} \sigma_2$ to hold iff whenever the use of $\sigma_2$ causes divergent behaviour, $\sigma_1$ can replicate it. In short, we shall balance all behaviours of $\sigma_2$ unavailable to $\sigma_1$ with winning regions for P. Note that the definition of $\leq_{\mathrm{must}}$ is phrased in terms of both traces and divergences, because a trace may also contribute to a divergence indirectly, when the environment diverges after exploring the trace.

**Definition 3.7.** Suppose $\sigma_1, \sigma_2 : A$. We define $\sigma_1 \leq_{\mathrm{must}} \sigma_2$ to hold iff for any $s \in (\mathcal{T}_{\sigma_2} \cup \mathcal{D}_{\sigma_2}) \setminus (\mathcal{T}_{\sigma_1} \cup \mathcal{D}_{\sigma_1})$ there exists a prefix $s'$ of $s$ such that $s' \in V_{\sigma_1}^P \cap Win_{\sigma_1}^P$.

$\leq_{\mathrm{must}}$ turns out to capture must-approximation in $\mathsf{EIA}_f$.

**Proposition 3.8.** $\Gamma \vdash M_1 \sqsubseteq_{\approx \mathrm{must}} M_2$ if and only if $[\![\Gamma \vdash M_1]\!] \leq_{\mathrm{must}} [\![\Gamma \vdash M_2]\!]$.

**Example 3.9.** We can now reexamine the term $f : \mathsf{com}_{f,1} \to \mathsf{com}_f \vdash \mathbf{new}\, X\, \mathbf{in}\, (f(X{:=}1); [!X = 0])\, \mathbf{or}\, (f(X{:=}1); [!X = 1]) : \mathsf{com}$ from Example 2.3. We have labelled the various occurrences of com to make it easier to trace the origin of moves contributed by them. The corresponding strategy is $\sigma = (\mathcal{T}_\sigma, \mathcal{D}_\sigma)$, where $\mathcal{T}_\sigma$ consists of even-length prefixes of plays matching the regular expression $run\, run_f\, (run_{f,1}\, done_{f,1})^*\, done_f\, done$ and $\mathcal{D}_\sigma$ contains words matching $run\, run_f\, (run_{f,1}\, done_{f,1})^*\, done_f$. Observe that $run \in Win_\sigma^P$. Thanks to Propositions 3.1 and 3.8 it is now easy to ascertain that the equivalences given in Example 2.3 are indeed valid.

## 4 Strategies as automata

In this section we show how to define automata representing strategies that correspond to third-order $\mathsf{EIA}_f$ terms with iteration. In fact, $\mathsf{EIA}_3 + \mathbf{while}$ is the largest fragment of $\mathsf{EIA}_f + \mu$ for which may- and must-approximation can be proved decidable. The translation is defined for terms in $\beta$-normal form and, in this and subsequent sections, whenever we mention $\mathsf{EIA}_f + \mu$-terms, we mean terms in $\beta$-normal form only. The automata we are aiming to construct will have runs corresponding to plays from $\overline{\mathcal{T}_\sigma} = \mathcal{T}_\sigma \cup V_\sigma^P$. Depending on $\theta$ we can decompose $\overline{\mathcal{T}_\sigma}$ as shown in Figure 2 by highlighting initial moves (in any non-empty position) and final ones (only in complete positions). Given $\theta = \theta_1 \to \cdots \theta_k \to \beta$ we define $tail(\theta)$ to be $\beta$. In the Figure, $+$ is meant to represent the disjoint sum so that $[\![\cdots]\!], [\![\cdots]\!]_j, [\![\cdots]\!]_j^r, [\![\cdots]\!]_j^w$ correspond to decomposing the respective sets of complete plays, while $[\![\cdots]\!], [\![\cdots]\!]_j, [\![\cdots]\!]_j^r, [\![\cdots]\!]_j^w$ capture incomplete plays. Note that we have $[\![\cdots]\!] \subseteq [\![\cdots]\!]$, if $tail(\theta) =$

$$\overline{\mathcal{T}_{[\![\Gamma \vdash M:\theta]\!]}} = \begin{cases} run\,[\!|\Gamma \vdash M|\!] + run\,[\![\Gamma \vdash M]\!] \; done & tail(\theta) = \mathsf{com} \\ q\,[\!|\Gamma \vdash M|\!] + q\,\sum_{j=0}^{max}([\![\Gamma \vdash M]\!]_j\, j) & tail(\theta) = \mathsf{exp} \\ read\,[\!|\Gamma \vdash M|\!]^r + read\,\sum_{j=0}^{max}([\![\Gamma \vdash M]\!]^r_j\, j) + \\ \sum_{j=0}^{max}(write(j)\,[\!|\Gamma \vdash M|\!]^w + write(j)\,[\![\Gamma \vdash M]\!]^w_j\; ok) & tail(\theta) = \mathsf{var} \end{cases}$$

$$\mathcal{D}_{[\![\Gamma \vdash M:\theta]\!]} = \begin{cases} run\,(\!|\Gamma \vdash M|\!) & tail(\theta) = \mathsf{com} \\ q\,(\!|\Gamma \vdash M|\!) & tail(\theta) = \mathsf{exp} \\ read\,(\!|\Gamma \vdash M|\!)^r + \sum_{j=0}^{max}(write(j)\,(\!|\Gamma \vdash M|\!)^w_j) & tail(\theta) = \mathsf{var} \end{cases}$$

**Figure 2. Decompositions of traces and divergences**

com, and analogous inclusions also hold in other cases. Divergences can be decomposed in a similar way to incomplete plays (see Figure 2).

For a given $\mathsf{EIA}_3$ + **while**-term we shall construct automata that represent all of the associated distinguished sets (e.g. $[\!|\cdots|\!]$, $[\![\cdots]\!]$ and $(\!|\cdots|\!)$ for $tail(\theta) = \mathsf{com}$) in a suitable sense. In the first two cases ($tail(\theta) = \mathsf{com}, \mathsf{exp}$) the outcome will be a single automaton whose runs capture $[\!|\cdots|\!]$, while the accepting ones represent both $[\![\cdots]\!]$ (resp. $[\![\cdots]\!]_0, \cdots, [\![\cdots]\!]_{max}$) and $(\!|\cdots|\!)$. In order to differentiate complete plays from divergences we use two kinds of final states. In the second case ($tail(\theta) = \mathsf{exp}$), in order to distinguish between $[\![\cdots]\!]_j$ for different $j$, we add extra information to final states in the form of subsets of $\{0, \cdots, max\}$, which can be thought of as returned values and will correspond to the indices involved. For the remaining types $\theta$ such $tail(\theta) = \mathsf{var}$ our construction will yield as many as $max + 2$ automata, corresponding to each of the $max + 2$ possible initial moves in the semantics.

**Definition 4.1.** Let $A$ be an arena corresponding to an $\mathsf{EIA}_f$-type. We define the order of any move in $A$ as follows. If the move is initial then its order is 0. The order of any other question is one greater than the order of any of its enablers (this definition is never ambiguous for the arenas in question). Answer-moves have the same order as the questions that enable them.

Suppose $\Gamma \vdash M : \theta$ is an $\mathsf{EIA}_3$ + **while**-term and $A = [\![\Gamma \vdash \theta]\!]$. Then moves of $A$ can have order at most 3. Besides, any question of order 3 is a P-move and, consequently, any answer of order 3 is an O-move. Let us write $M^{\leq 2}_{AO}$ (resp. $M^{\leq 2}_{AP}$) for the set of O-moves (resp. P-moves) in $A$ whose order does not exceed 2. $M^3_{AO}$ (resp. $M^3_{AP}$) will stand for the set of O-moves (resp. P-moves) of $A$ of order 3. As already mentioned, $M^3_{AO}$ consists of answer-moves and elements of $M^3_{AP}$ are question-moves. Next we define the particular kind of automata that will be used to capture third-order terms with **while**. The automata are instances of visibly pushdown automata [6].

**Definition 4.2.** Suppose $\Gamma \vdash M : \theta$ is a $\mathsf{EIA}_3$ + **while**-term and $A = [\![\Gamma \vdash \theta]\!]$. An $A$-automaton $\mathcal{A}$ is a tuple $\langle\, Q, i, C, D, \Delta, \delta, \mu \,\rangle$ such that: $Q$ is the finite set of states, partitioned into $Q^O$ (O-states) and $Q^P$ (P-states); $i \in Q$ is the initial state; $C \subseteq Q$ and $D \subseteq Q^P$ are two sets of final states; $\Delta$ is the stack alphabet; $\delta \subseteq (Q^O \times M^{\leq 2}_{AO} \times Q^P) \times (Q^P \times M^{\leq 2}_{AP} \times Q^O) \times ((Q^O \times \Delta) \times M^3_{AO} \times Q^P) \times (Q^P \times M^3_{AP} \times (Q^O \times \Delta))$ is the transition function; $\mu : C \to 2^{\{0,\cdots,max\}} \setminus \{\emptyset\}$ is a selector function (its role will be explained below).

Each run (a valid sequence of transitions from the initial state) of $\mathcal{A}$ determines a word from $M^*_A$. We shall write $R(\mathcal{A})$ for the set of words generated by runs of $\mathcal{A}$. The language accepted by $\mathcal{A}$ using the final set $D$ will be denoted by $D(\mathcal{A})$. For any $j \in \{0, \cdots, max\}$ we write $C_j(\mathcal{A})$ for the language accepted by $\mathcal{A}$ using the final set $\{s \in C \mid j \in \mu(s)\}$.

The transition function is divided into four kinds of transitions. The first two are stack-independent transitions, the third kind are pop-transitions and the last group are push-transitions.

**Definition 4.3.** Suppose $\Gamma \vdash M : \theta$ is an $\mathsf{EIA}_3$ + **while**-term and $A = [\![\Gamma \vdash \theta]\!]$. A tuple $\langle\, \mathcal{A}_0, \cdots, \mathcal{A}_k \,\rangle$ of $A$-automata *represents* $\Gamma \vdash M : \theta$ if and only if the following conditions hold.

- In any $\mathcal{A}_i$ ($i = 0, \cdots, k$) the initial state and the final states in $C$ are P-states.
- If $tail(\theta) = \mathsf{com}$ then $k = 0$, $R(\mathcal{A}_0) = [\![\Gamma \vdash M]\!]$, $C_0(\mathcal{A}_0) = [\![\Gamma \vdash M]\!]$ and $D(\mathcal{A}_0) = (\!|\Gamma \vdash M|\!)$. Moreover, we require that $\mu_{\mathcal{A}_0}(s) = \{0\}$ for any $C$-state $s$ of $\mathcal{A}_0$.
- If $tail(\theta) = \mathsf{exp}$, then $k = 0$, $R(\mathcal{A}_0) = [\![\Gamma \vdash M]\!]$, $C_j(\mathcal{A}_0) = [\![\Gamma \vdash M]\!]_j$ ($j = 0, \cdots, max$) and $D(\mathcal{A}_0) = (\!|\Gamma \vdash M|\!)$.
- If $tail(\theta) = \mathsf{var}$, then $k = max + 2$ and $R(\mathcal{A}_j) = [\![\Gamma \vdash M]\!]^w_j$, $R(\mathcal{A}_{max+1}) = [\![\Gamma \vdash M]\!]^r$, $C_0(\mathcal{A}_j) = [\![\Gamma \vdash M]\!]^w_j$, $C_j(\mathcal{A}_{max+1}) = [\![\Gamma \vdash M]\!]^r_j$, $D(\mathcal{A}_j) = (\!|\Gamma \vdash M|\!)^w_j$, $D(\mathcal{A}_{max+1}) = (\!|\Gamma \vdash M|\!)^r$ for any $j = 0, \cdots, max$. Furthermore, we require that $\mu_{\mathcal{A}_j}(s) = \{0\}$ for any $C$-state $s$ of $\mathcal{A}_j$ ($j = 0, \cdots, max$).

**Theorem 4.4.** For any $\mathsf{EIA}_3$ + **while**-term $\Gamma \vdash M : \theta$ (in $\beta$-normal form) there exists a tuple of $[\![\Gamma \vdash \theta]\!]$-automata

that represents this term. The construction is effective and can be carried out in exponential time.

*Proof.* One advantage of considering $\llbracket \cdots \rrbracket$, $\llparenthesis \cdots \rrparenthesis$ and $\langle\!| \cdots |\!\rangle$ instead of the full semantics $[\![ \cdots ]\!]$ is the fact that they can be defined equationally (by recursion on the syntax) using simpler extended regular expressions and context-free grammars. Our constructions then closely follow the pattern of these definitions.

To account for pointers (it suffices to capture pointers from third-order moves) we maintain labels that record the depth of the occurrence of the corresponding free variable.

The exponential time in the worst case comes from the constructions for $\mathbf{new}\, X\, \mathbf{in}\, M$ and $fM$ (where $f$'s type is order two). See [26] for more details. □

Using the decompositions defined at the beginning of this section we can derive the following result.

**Theorem 4.5.** Let $\Gamma \vdash M : \theta$ be an $\mathsf{EIA}_3 + \mathbf{while}$-term (in $\beta$-normal form) and $\sigma = [\![\Gamma \vdash M : \theta]\!]$. There exists a $[\![\Gamma \vdash \theta]\!]$-automaton $\mathcal{A}$ such that $i \in Q^O, C = \{f\} \subseteq Q^O$, $\mu(f) = \{0\}$, $R(\mathcal{A}) = \overline{\mathcal{T}_\sigma}$, $D(\mathcal{A}) = D_\sigma$ and $C_0(\mathcal{A}) = \mathsf{comp}\,(\sigma)$. $\mathcal{A}$ can be constructed in exponential time.

*Proof.* It suffices to appeal to the previous Theorem, add $i, f$ as new states and define new outgoing transitions as prescribed by $\mu$. □

**Remark 4.6.** (i) Without any loss of generality we can assume that the unique final state from Theorem 4.5 has no outgoing transitions, because a complete position is maximal among single-threaded plays.

(ii) In the proof of Theorem 4.4 the stack is only ever used for interpreting second-order free identifiers, e.g. $f : (\mathsf{com} \to \mathsf{com}) \to \mathsf{com}$, which become available in $\mathsf{EIA}_3$ and its extensions. Consequently, Theorem 4.5 yields finite automata for terms belonging to $\mathsf{EIA}_2 + \mathbf{while}$.

(iii) Further simplifications apply to terms of $\mathsf{EIA}_1$. Since loops are formed only when interpreting $\mathbf{while}$, first- and second-order identifiers, the finite automata corresponding to $\mathsf{EIA}_1$-terms will be loop-free. Hence, for any $\mathsf{EIA}_1$-term, there exists a bound on the length of words that the associated automaton accepts, which we shall call its *depth*. While the size of the automaton produced by Theorem 4.5 for an $\mathsf{EIA}_1$-term can still be exponential with respect to the term size (due to $\mathbf{new}$), its depth turns out to be linear, because the construction for $\mathbf{new}$ does not increase it.

(iv) If $\Gamma \vdash C[\mathbf{new}\, X\, \mathbf{in}\, M]$ is an $\mathsf{EIA}_1$-term (in $\beta$-normal form), then it is equivalent to $\Gamma \vdash \mathbf{new}\, X\, \mathbf{in}\, C[M]$. Consequently, any $\mathsf{EIA}_1$-term $\Gamma \vdash M : \theta$ is equivalent to a term of the form $\lambda x_1^{\beta_1} \cdots x_m^{\beta_m}.\mathbf{new}\, X_1, \cdots, X_n\, \mathbf{in}\, M'$, where $M'$ is $\mathbf{new}$-free and not larger than $M$. Intuitively, globalization of local variables in $\mathsf{EIA}_1$-terms preserves equivalence, because $\mathsf{EIA}_1$ has no mechanisms for repetition.

(v) A practical consequence of the previous point is that, for $\mathsf{EIA}_1$-terms, the $\mathbf{new}$ constructions can be postponed until the very end without violating Theorem 4.5. Hence, it is easy to see that, for $\mathsf{EIA}_1$-terms, the corresponding automaton can be produced in polynomial space in such a way that the size of (the representation of) each state is polynomial (essentially, the state will consist of a state of the automaton corresponding to $M'$ and a tuple of variables corresponding to the values of $X_1, \cdots, X_n$). Consequently, all runs of the automaton can be explored in polynomial space by a depth-first recursive procedure, because the depth of the automaton is linear in the size of $M$ (hence the stack size will be linear) and any configuration is of polynomial size (consequently, each stack frame will be of polynomial size).

## 5 Complexity of may-approximation

By Proposition 3.1, in order to decide may-approximation, it suffices to check containment of the induced sets of complete plays. Thanks to Theorem 4.5 and the subsequent remarks the following upper bounds are obtained.

**Theorem 5.1.** May-approximation is in 2-EXPTIME for $\mathsf{EIA}_3 + \mathbf{while}$, in EXPSPACE for $\mathsf{EIA}_2 + \mathbf{while}$ and in PSPACE for $\mathsf{EIA}_1$.

*Proof.* By Proposition 3.1 along with Theorem 4.5 may-approximation in $\mathsf{EIA}_3 + \mathbf{while}$ can be reduced to the containtment problem for visibly pushdown automata, which is known to be EXPTIME-complete [6]. Since the automata involved can be (at most) exponentially larger than terms, this yields a 2-EXPTIME algorithm for deciding may-approximation between $\mathsf{EIA}_3 + \mathbf{while}$-terms.

Recall from Remark 4.6 (ii) that $\mathsf{EIA}_2 + \mathbf{while}$-terms generate nondeterministic finite automata. The associated containment problem is well-known to be PSPACE-complete [19], implying an EXPSPACE decision procedure for may-aproximation in $\mathsf{EIA}_2 + \mathbf{while}$.

Finally, $\mathsf{EIA}_1$ allows for even faster may-approximation checks. Because the depth of the associated automata is linear (Remark 4.6 (iii)), may-approximation can be contradicted by a word whose size is linear in the size of the terms in question. By Remark 4.6 (v) membership and co-membership can be decided in PSPACE, so the correctness of such a certificate can be verified in polynomial space. Because NPSPACE=PSPACE [35], may-approximation in $\mathsf{EIA}_1$ is in PSPACE. □

**Theorem 5.2.** May-approximation is PSPACE-hard in $\mathsf{EIA}_1$, EXPSPACE-hard for $\mathsf{EIA}_2$ and $\mathsf{EIA}_1 + \mathbf{while}$, and 2-EXPTIME-hard for $\mathsf{EIA}_3$.

In order to justify the remaining results from the table given in the introduction, we observe that the undecidability of may-approximation in $\mathsf{EIA}_4$ follows immediately from the undecidability of contextual equivalence in

the deterministic case [27]. The undecidability of may-approximation in $\mathsf{EIA}_1 + \mu_0$ in turn follows from the undecidability of the containment problem for context-free languages [19] and the encoding of pushdown automata in $\mathsf{EIA}_1 + \mu_0$ from [29], which can easily be generalized to nondeterministic pushdown automata.

The same bounds as for may-approximation apply to the induced notion of may-equivalence. To see that upper bounds are preserved, note that equivalence between two terms can be decided by running two approximation checks. For lower bounds, we observe that in all of our hardness arguments for $M \sqsubseteq_{\approx\,\mathrm{may}} N$, it was already the case that $N \sqsubseteq_{\approx\,\mathrm{may}} M$. Hence, $M \sqsubseteq_{\approx\,\mathrm{may}} N$ was in fact equivalent to $M \cong_{\mathrm{may}} N$. Similarly to may-approximation, the undecidability of may-equivalence in $\mathsf{EIA}_4$ is an immediate consequence of the analogous result in the deterministic case. For $\mathsf{EIA}_1 + \mu_0$ we simply appeal to the undecidability of language equivalence for (nondeterministic) pushdown automata.

## 6 Complexity of must-approximation

Here we show how one can take advantage of Proposition 3.8 and Theorem 4.5 in order to decide must-approximation in $\mathsf{EIA}_3 + \mathbf{while}$, which turns out to be the largest decidable fragment. By Proposition 3.8 we have $\Gamma \vdash M_1 \sqsubseteq_{\approx\,\mathrm{must}} M_2$ if and only if

$$(\mathcal{T}_{\sigma_2} \cup \mathcal{D}_{\sigma_2}) \;\cap\; (\mathcal{T}_{\sigma_1} \cup \mathcal{D}_{\sigma_1})^c \cap \{s \in P_{[\![\Gamma \vdash \theta]\!]} \mid \forall s' \sqsubseteq s.\ \text{if } s' \in V^P_{\sigma_1} \text{ then } s' \in \mathit{Win}^O_{\sigma_1}\} = \emptyset,$$

where $\sigma_i = [\![\Gamma \vdash M_i]\!]$ $(i = 1, 2)$,

We shall construct visibly pushdown automata accepting each of the three conjuncts distinguished above. The largest of them will be triply exponentially larger than the terms involved, yielding a 3-EXPTIME bound for must-approximation in $\mathsf{EIA}_3 + \mathbf{while}$, because intersection emptiness of a fixed number of VPAs can be decided in polynomial time simply by building up the product automaton [6] and checking it for emptiness.

The VPAs accepting the first two conjuncts can be constructed quite easily via Theorem 4.5. For the second one, the complementation procedure [6] needs to be applied, leading to two exponential blow-ups in total. The last conjunct presents the biggest challenge, because of the reference to the winning region $\mathit{Win}^O_{\sigma_1}$. To handle it, we observe that $\mathcal{G}_{\sigma_1}$ can be viewed as a pushdown reachability game and take advantage of the associated methods for computing winning regions [8, 9].

**Definition 6.1.** A *pushdown game system* is a triple $\mathcal{P} = \langle\, Q, Q^O, Q^P, \Delta, \delta \,\rangle$, where $Q$ is a finite set of states, $Q = Q^O + Q^P$, $\Delta$ is a finite stack alphabet and $\delta \subseteq (Q \times \Delta) \times (Q \times \Delta^*)$ is a finite transition function. We assume the existence of the bottom-of-stack symbol $\perp \in \Delta$, which cannot be taken off the stack.

**Definition 6.2.** Given a pushdown game system $\mathcal{P} = \langle\, Q, Q^O, Q^P, \Delta, \delta \,\rangle$ and a *target set* $W \subseteq Q\Delta^*$, one can define the corresponding reachability game $\mathcal{G}_{\mathcal{P},W} = \langle\, Q\Delta^*, Q^O\Delta^*, Q^P\Delta^*, E, W \,\rangle$, where $E = \{(qxw', q'ww') \mid ((q, x), (q', w)) \in \delta,\ w' \in \Delta^*\}$. Games derived in this way are called *pushdown reachability games*.

Observe that, as VPAs can be determinized, we can determize Theorem 4.5 as well (at an exponential cost). Let $\mathcal{A}_{det}$ be the deterministic VPA for $\Gamma \vdash M_1$ obtained in this way. $\mathcal{A}_{det}$ is crucial to relating $\mathcal{G}_{\sigma_1}$ with pushdown games.

**Lemma 6.3.** There exists a pushdown game system $\mathcal{P} = \langle\, Q, Q^O, Q^P, \Delta, \delta \,\rangle$, a target set $W \subseteq Q$ and $q \in Q$ such that $\mathcal{G}_{\sigma_1}$ is isomorphic to $\mathsf{Unf}_{(q,\perp)}(\mathcal{G}_{\mathcal{P},W})$.

*Proof.* We show how to obtain $\mathcal{P}$ by erasing transition labels in a VPA. Note that Theorem 4.5 (for $\Gamma \vdash M_1$) is not sufficient for this purpose, because the automaton it produces is in general nondeterministic and, consequently, several runs of the automaton might correspond a single play in $\mathcal{G}_{\sigma_1}$. This failure is eliminated if we consider $\mathcal{A}_{det}$ instead. To finish the proof, we need to account for those edges of $\mathcal{G}_{\sigma_1}$ that lead from divergences to $\circ$ in $\mathcal{G}_{\sigma_1}$, and define the target set of the game. The former problem is solved by adding new transitions to $\mathcal{A}_{det}$. Then, to address the latter, we can use additional finite memory to keep track of whether the resultant automaton has encountered a divergence during its run (i.e. whether a $D$-state of $\mathcal{A}_{det}$ was ever entered) and thus identify a set of final states $F \subseteq Q$ such that the goal set can be taken to be $\{f\perp \mid f \in F\}$. We take the initial set of $\mathcal{A}_{det}$ as $q$. □

The next lemma is a handle on the winning regions.

**Lemma 6.4.** Let $\mathcal{P} = \langle\, Q, Q^O, Q^P, \Delta, \delta \,\rangle$ be a pushdown game system and $F \subseteq Q$. Consider the goal set $W = \{f\perp \mid f \in F\}$. There exists a deterministic labelled transition system, $\mathcal{TS}_{\mathcal{P},W} = \langle\, 2^{Q+\{\star\}}, \Delta, \delta' \,\rangle$, constructible from $\mathcal{P}$ in exponential time, such that $qw \in \mathit{Win}^O_{\mathcal{G}_{\mathcal{P},W}}$ if and only if $q \in \delta'(\{\star\}, w^R)$, where $w^R$ is the reversal of $w$.

*Proof.* The given winning set can be captured (in the sense of [8]) with an alternating finite automaton over the state-set $Q + \{\star\}$ such that the size of the transition function is linear in $Q$ and $\star$ is the only accepting state. Bouajanii et al [8] (see also [9]) show how, given an alternating automaton representing the target set, one can construct (in exponential time) another alternating automaton representing the corresponding winning region. Its transition function $\delta''$ can be exponential in the size of $\mathcal{P}$, but the set of states remains $Q + \{\star\}$. To define $\mathcal{TS}_{\mathcal{P},W}$ it now suffices to "reverse" (cf. Theorem 2.13 in [38]) the latter alternating automaton by $\delta'(X, x) = \{q \in Q + \{\star\} \mid \exists Y \subseteq X.(q, x, Y) \in \delta''\}$. Note that the last step remains exponential in the size of $\mathcal{P}$ and thus the lemma follows. □

When the stack content was represented with a word $w \in \Delta^*$ in the definition of pushdown reachability games, the leftmost symbol corresponded to the top of the stack. Consequently, the transition system from the above Lemma processes the stack in the *bottom-up* fashion. This turns out very convenient for us, because the key to capturing the third conjunct is making $\mathcal{A}_{det}$ "aware" of whether it is currently inside O's winning region. For that purpose, $\mathcal{A}_{det}$ can remember and maintain the state that the associated transition system $\mathcal{TS}_{\mathcal{P},W}$ (i.e. the one that Lemma 6.4 yields for the pushdown game system obtained from Lemma 6.3) reaches on processing the current stack contents of $\mathcal{A}_{det}$. This is not difficult to implement for push moves and internal moves. For pop-moves though, the information stored in the finite memory is not sufficient for performing a suitable update, because we then need to "backtrack" in the corresponding run of $\mathcal{TS}_{\mathcal{P},W}$. However, if we arrange for the states of $\mathcal{TS}_{\mathcal{P},W}$ to be pushed on the stack during push-moves, the relevant information will become readily available during pop-moves. To be more precise, during push-moves the modified $\mathcal{A}_{det}$ should, in addition to the usual symbol, push the currently stored state of $\mathcal{TS}_{\mathcal{P},W}$ before updating it to the post-push state.

Note that on the way from $\Gamma \vdash M_1$ to the VPA capturing the third conjunct we have combined three exponential-time procedures: Theorem 4.5, determinization and Lemma 6.4. Hence, the VPA can be produced in triply exponential time.

**Theorem 6.5.** Must-approximation is in 3-EXPTIME for $\mathsf{EIA}_3 + \mathbf{while}$, in 2-EXPTIME for $\mathsf{EIA}_2 + \mathbf{while}$ and in PSPACE for $\mathsf{EIA}_1$.

*Proof.* The bound for $\mathsf{EIA}_3 + \mathbf{while}$ follows from the preceding discussion. For $\mathsf{EIA}_2 + \mathbf{while}$, recall from Remark 4.6 that in this case Theorem 4.5 yields finite automata. Thus, the pushdown game system obtained from Lemma 6.3 is simply a bipartite finite graph. Winning regions (vertices) can then be determined without the additional blow-up entailed by appealing to Lemma 6.4. In fact, the calculation of winning regions in this case corresponds to the well-known PTIME-complete problem of alternating graph reachability [21] (see also Theorem 2.3 in [9]). Hence, must-approximation in $\mathsf{EIA}_2 + \mathbf{while}$ is in 2-EXPTIME. A tighter bound can be proved for $\mathsf{EIA}_1$. Recall from Remark 4.6 (iii) that in this case there exists a linear bound (with respect to the term size) on the length of accepted words. Hence, any counterexample to the emptiness of the intersection above can be stored in linear space wrt to the sizes of $M_1$ and $M_2$. By Remark 4.6 (v), the correctness of such a certificate can be verified in polynomial space. To confirm membership in the winning region we rely on the brute-force nondeterministic decision procedure that guesses the winning strategy. □

The above bounds are tight. In fact hardness can be shown for the following problem OMEGA: does a given term $\Gamma \vdash M : \mathsf{com}$ must-approximate $\vdash \Omega_{\mathsf{com}} : \mathsf{com}$? Note that we have $\Gamma \vdash M \sqsubseteq_{\approx \mathrm{must}} \Omega_{\mathsf{com}}$ iff $run \in Win^{P}_{\mathcal{G}_{[\![\Gamma \vdash M]\!]}}$.

**Theorem 6.6.** OMEGA is PSPACE-hard for $\mathsf{EIA}_1$-terms, 2-EXPTIME-hard for $\mathsf{EIA}_2$- and $\mathsf{EIA}_1 + \mathbf{while}$-terms, and 3-EXPTIME-hard for $\mathsf{EIA}_3$.

For undecidability of must-approximation in $\mathsf{EIA}_4$ we observe that, for any Turing machine $T$ and input $w$, the construction in [27] for (deterministic) Idealized Algol yields an $\mathsf{EIA}_4$-term $\vdash M : \theta$ (without $\mathbf{or}$) such that $\epsilon \in Win^{O}_{\mathcal{G}_{[\![\vdash M]\!]}}$ iff $T$ halts on $w$. Hence, OMEGA is undecidable for $\mathsf{EIA}_4$ and so is must-approximation.

For $\mathsf{EIA}_1 + \mu_0$ in turn, we can reduce the undecidable problem of checking intersection emptiness for deterministic context-free languages [19] to OMEGA. Indeed, given DCFLs $L_1, L_2$, using the construction from [29] we can construct $\mathsf{EIA}_1 + \mu_0$-terms $x : \mathsf{exp} \vdash M_1, M_2 : \mathsf{com}$ whose complete plays represent precisely the words in $L_1$ and $L_2$ respectively ($run q_x a_x^1 \cdots q_x a_x^n done$ is used to represent $a^1 \cdots a^n$). Then $L_1 \cap L_2$ is not empty iff $run \in Win^{O}_{\mathcal{G}_{[\![x:\mathsf{exp} \vdash M_1 \mathbf{or} M_2]\!]}}$. Consequently, must-approximation in $\mathsf{EIA}_1 + \mu_0$ must be undecidable.

Identical bounds apply to must-equivalence. The upper bounds are still valid, because checking must-equivalence can be done by checking must-approximation twice. The lower bounds carry over, because they all concerned the problem $\Gamma \vdash M \sqsubseteq_{\approx \mathrm{must}} \Omega$, which is equivalent to $\Gamma \vdash M \cong_{\mathrm{must}} \Omega$. What's more, the bounds are also accurate for may&must-approximation and may&must-equivalence, because in all cases may-approximation is easier to verify than must-approximation and $\Gamma \vdash M \sqsubseteq_{\approx \mathrm{must}} \Omega$ is equivalent to $\Gamma \vdash M \sqsubseteq_{\approx \mathrm{may\&must}} M \mathbf{or} \Omega$ as well as $\Gamma \vdash M \cong_{\mathrm{may\&must}} M \mathbf{or} \Omega$.

## 7 Related and future work

We presented a complete classification of complexity bounds for may-, must- and may&must-approximation as well as the induced notions of equivalence in Erratic Idealized Algol. The results rely on a combination of game semantics with automata-theoretic techniques, notably, methods for solving reachability games.

The usefulness of winning regions in our work raises the question whether game semantics itself could be based on the concept of winning regions (along with a suitable notion of composition) instead of the usual traces and divergences.

The transfer of control in Erratic Idealized Algol adheres to the usual discipline of block entry and exit. This feature turns out to complicate reasoning about must-approximation. Indeed we claim that with non-local flow

control must-approximation corresponds to the preorder $\leq^\sharp$ defined in [17], whose definition is similar to $\leq_{\mathrm{must}}$, but is phrased in terms of divergences rather than winning regions. Consequently, it seems that must-approximation in Erratic Idealized Algol with control could be decided without recourse to reachability games, leading to better complexity bounds. Note that $\leq^\sharp$ is clearly too strong to capture must-equivalence in Erratic Idealized Algol (for instance, it distinguishes $c; \Omega_{\mathsf{com}}$ from $\Omega_{\mathsf{com}}$ and invalidates the must-equivalences from Example 2.3). In fact, we have $\leq^\sharp \subsetneq \leq_{\mathrm{must}}$.

All the terms studied in the paper were assumed to be in $\beta$-normal form. As any $\mathsf{EIA}_f$ term can be $\beta$-normalized the decidability/undecidability results are also relevant to $\mathsf{EIA}_f + \mu$-terms with $\beta$-redexes. One can try to derive complexity bounds in this case by $\beta$-normalization and then applying the decision procedures of this paper. However, the bounds thus obtained are unlikely to be tight. In future work we would like to relate terms with $\beta$-redexes with hierarchical machine models such as those from [5].

## References


[1] S. Abramsky, D. R. Ghica, A. S. Murawski, and C.-H. L. Ong. Applying game semantics to compositional software modelling and verification. *LNCS* 2988: 421–435 (2004).

[2] S. Abramsky and R. Jagadeesan. Games and full completeness for multiplicative linear logic. *J. Symb. Log.* 59(2): 543-574 (1994).

[3] S. Abramsky, R. Jagadeesan, and P. Malacaria. Full abstraction for PCF. *Inf. Comput.* 163: 409–470 (2000).

[4] S. Abramsky and G. McCusker. Linearity, sharing and state: a fully abstract game semantics for Idealized Algol with active expressions. *Algol-like languages*: 297–329 (1997).

[5] R. Alur, S. Kannan, and M. Yannakakis. Communicating hierarchical state machines. *LNCS* 1644: 169–178 (1999).

[6] R. Alur and P. Madhusudan. Visibly pushdown languages. *Proc. of STOC'04*: 202–211 (2004).

[7] A. Bakewell and D. R. Ghica. On-the-fly techniques for games-based software model checking. *Proc. of TACAS*, to appear.

[8] A. Bouajjani, J. Esparza, and O. Maler. Reachability analysis of pushdown automata: Application to model-checking. *LNCS* 1243: 135–150 (1997).

[9] T. Cachat. Symbolic strategy synthesis for games on pushdown graphs. *LNCS* 2380: 704–715 (2002).

[10] R. De Nicola and M. Hennessy. Testing equivalence for processes. *Theor. Comput. Sci.*: 34 (1984).

[11] A. Dimovski, D. R. Ghica, and R. Lazic. A counterexample-guided refinement tool for open procedural programs. *LNCS* 3925: 288–292 (2006).

[12] D. Gale and F. M. Steward. Infinite games with perfect information. *Annals of Mathematics Studies* 28: 245–266 (1953).

[13] D. R. Ghica and G. McCusker. Reasoning about Idealized Algol using regular expressions. *LNCS* 1853: 103–115 (2000).

[14] D. R. Ghica and A. S. Murawski. Compositional model extraction for higher-order concurrent programs. *LNCS* 3920: 303-317 (2006).

[15] E. Grädel, W. Thomas, and T. Wilke, editors. *Automata, Logics, and Infinite Games: A Guide to Current Research*, *LNCS* 2500 (2002).

[16] R. Harmer. *Games and full abstraction for non-deterministic languages*. PhD thesis, University of London, 2000.

[17] R. Harmer and G. McCusker. A fully abstract game semantics for finite nondeterminism. *Proc. of LICS*: 422-430 (1999).

[18] C. A. R. Hoare. *Communicating Sequential Processes* (1985).

[19] J. E. Hopcroft and J. D. Ullman. *Introduction to Automata Theory, Languages and Computation* (1979).

[20] J. M. E. Hyland and C.-H. L. Ong. On Full Abstraction for PCF. *Inf. Comput.* 163(2):285–408 (2000).

[21] N. Immerman. Number of quantifiers is better than number of tape cells. *J. Comput. Syst. Sci.* 22(3): 384–406 (1981).

[22] A. Legay, A. S. Murawski, J. Ouaknine, and J. Worrell. On automated verification of probabilistic programs. *Proc. of TACAS*, to appear.

[23] P. Malacaria and C. Hankin. Generalized flowcharts and games (extended abstract). *LNCS* 1443: 363–374 (1998).

[24] P. Malacaria and C. Hankin. A new approach to control flow analysis. *LNCS* 1383: 95–108 (1998).

[25] P. Malacaria and C. Hankin. Non-deterministic games and program analysis: an application to security. *Proc. of LICS*: 443–452 (1999).

[26] A. S. Murawski. `http://users.comlab.ox.ac.uk/andrzej.murawski/long.pdf`

[27] A. S. Murawski. On program equivalence in languages with ground-type references. *Proc. of LICS*: 108–117 (2003).

[28] A. S. Murawski. Games for complexity of second-order call-by-name programs. *Theor. Comput. Sci.* 343(1/2): 207–236 (2005).

[29] A. S. Murawski, C.-H. L. Ong, and I. Walukiewicz. Idealized Algol with ground recursion and DPDA equivalence. *LNCS* 3580: 917–929 (2005).

[30] A. S. Murawski and J. Ouaknine. On probabilistic program equivalence and refinement. *LNCS* 3653: 156–170 (2005).

[31] A. S. Murawski and I. Walukiewicz. Third-order Idealized Algol with iteration is decidable. *LNCS* 3441: 202–218 (2005).

[32] C.-H. L. Ong. Observational equivalence of 3rd-order Idealized Algol is decidable. *Proc. of LICS*: 245–256 (2002).

[33] C.-H. L. Ong. On model-checking trees generated by higher-order recursion schemes. *Proc. of LICS*: 81–90 (2006).

[34] J. C. Reynolds. The essence of Algol. *Algorithmic Languages*: 345–372. North Holland, 1978.

[35] W. J. Savitch. Relationships between nondeterministic and deterministic tape complexities. *J. Comput. Syst. Sci.* 4(2):177–192 (1970).

[36] G. Sénizergues. L(A)=L(B)? decidability results from complete formal systems. *Theor. Comput. Sci.* 251(1-2):1–166 (2001).

[37] R. J. van Glabbeek. The linear time - branching time spectrum I. In *Handbook of Process Algebra*, Chapter 1: 3–99. Elsevier, 2001.

[38] S. Yu. Regular languages. In *Handbook of Formal Languages*, Chapter 1: 41–110. Springer Verlag, 1997.

## A Proof of Proposition 3.8

*Proof.* W.l.o.g. we assume that $\Gamma = \emptyset$, i.e. $\vdash M_1, M_2 : \theta$. If $M_1, M_2$ are not closed, one can simply take $\lambda\Gamma.M_1, \lambda\Gamma.M_2$ instead, where $\lambda\Gamma$ stands for a sequence of $\lambda$-abstractions with respect to each variable from $\Gamma$. Let $\sigma_i = [\![\vdash M_i]\!]$ $(i = 1, 2)$.

($\neg R \Rightarrow \neg L$) Suppose there exists $s \in (\mathcal{T}_{\sigma_2} \cup \mathcal{D}_{\sigma_2}) \setminus (\mathcal{T}_{\sigma_1} \cup \mathcal{D}_{\sigma_1})$ such that all prefixes of $s$ which are in $V^P_{\sigma_1}$ are not in $Win^P_{\sigma_1}$ (they must be in $Win^O_{\sigma_1}$ then). Observe that $s \neq \epsilon$, because $\epsilon \in \mathcal{T}_{\sigma_1} \cap \mathcal{T}_{\sigma_2}$.

Let $t$ be the longest odd-length prefix of $s$ such that $t \in V^P_{\sigma_1}$. Note that $t$ is well-defined, because $s \neq \epsilon$ and the first move of $s$, viewed as a single-move sequence, is in $V^P_{\sigma_1}$. Let $t'$ be an odd-length prefix of $t$. Observe that $t' \in V^P_{\sigma_1}$ and thus $t' \in Win^O_{\sigma_1}$. In particular, no prefix of $t'$ is in $\mathcal{D}_{\sigma_1}$. Recall that $t' \in Win^O_{\sigma_1}$ implies that O has a winning strategy in the game $G_{\sigma_1}$ started from $t'$, i.e., regardless of P's responses, O will eventually be able to extend $t'$ to a complete position without encountering any divergences. Let $\mathcal{T}^{\mathrm{e}}_{t'}$ (resp. $\mathcal{T}^{\mathrm{o}}_{t'}$) be the set of even-length (resp. odd-length) suffixes of $t'$ generated by O's winning strategy when deployed against all possible behaviours of P in the game $G_{\sigma_1}$ with starting vertex $t'$. Recall that $t'$ ranges over odd-length prefixes of $t$. Let $\mathcal{T}^{\mathrm{o}} = \bigcup_{t'} \mathcal{T}^{\mathrm{o}}_{t'}$, $\mathcal{T}^{\mathrm{e}} = \bigcup_{t'} \mathcal{T}^{\mathrm{e}}_{t'}$. Define the set $\mathcal{U} \subseteq P_{[\![\theta \to \mathsf{com}]\!]}$ by

$$\{ \quad \epsilon \quad \} \quad \cup \quad \{ \quad run\, u \quad | \quad u \in \mathcal{T}^{\mathrm{o}} \quad \} \quad \cup \quad \{ \quad run\, u\, done \quad | \quad u \in \mathsf{comp}\,(\mathcal{T}^{\mathrm{e}}) \quad \} \ .$$

- Suppose $t = s$. Since $t$ has odd length, we have $s \in \mathcal{D}_{\sigma_2}$. Consider the strategy $\tau = (\mathcal{U}, \emptyset) : [\![\theta \to \mathsf{com}]\!]$. Then $\mathcal{D}_{\sigma_1;\tau} = \emptyset$ by choice of $\tau$. However, $\mathcal{D}_{\sigma_2;\tau} \neq \emptyset$, because $run\, s = run\, t \in \mathcal{U} = \mathcal{T}_\tau$ and $s \in \mathcal{D}_{\sigma_2}$. By $\omega$-algebraicity and definability results from [17], there exists $\tau' = [\![x : \theta \vdash C[x] : \mathsf{com}]\!]$ such that $\mathcal{D}_{\sigma_1;\tau'} = \emptyset$ and $\mathcal{D}_{\sigma_2;\tau'} \neq \emptyset$. Consequently, by Adequacy [17], $C[M_1] \Downarrow_{\mathrm{must}}$ holds, but $C[M_2] \Downarrow_{\mathrm{must}}$ does not.
- Suppose $t$ is a proper prefix of $s$. Let $tp$ be a (not necessarily proper) prefix of $s$. Then we claim that $tp \notin \mathcal{T}_{\sigma_1}$.
  - If $tp = s$ then indeed $tp \notin \mathcal{T}_{\sigma_1}$, because $s \notin \mathcal{T}_{\sigma_1}$.
  - If $tp$ is a proper prefix of $s$ and $tp \in \mathcal{T}_{\sigma_1}$, then we would have $tpo \in V^P_{\sigma_1}$, where $tpo$ is a prefix of $s$. This would contradict the choice of $t$.

  Consider the strategy $\tau = (\mathcal{U}, \{run\, t\, p\}) : [\![\theta \to \mathsf{com}]\!]$. By choice of $\mathcal{U}$ and thanks to $tp \notin \mathcal{T}_{\sigma_1}$, we have $\mathcal{D}_{\sigma_1;\tau} = \emptyset$. However, $\mathcal{D}_{\sigma_2;\tau} \neq \emptyset$, because $tp \in \mathcal{T}_{\sigma_2}$ (since $tp$ is a prefix of $s \in \mathcal{T}_{\sigma_2} \cup \mathcal{D}_{\sigma_2}$). As in the previous case, we can now conclude that $\vdash M_1 \sqsubseteq_{\mathrm{must}} M_2$ does not hold.

($\neg L \Rightarrow \neg R$) Suppose it is not the case that $\vdash M_1 \sqsubseteq_{\mathrm{must}} M_2$. Then, by soundness results from [17], there exists $\tau : [\![\theta \to \mathsf{com}]\!]$ such that $\mathcal{D}_{\sigma_1;\tau} = \emptyset$ and $\mathcal{D}_{\sigma_2;\tau} \neq \emptyset$. We need to show that there exists $s \in (\mathcal{T}_{\sigma_2} \cup \mathcal{D}_{\sigma_2}) \setminus (\mathcal{T}_{\sigma_1} \cup \mathcal{D}_{\sigma_1})$ such that, for any prefix $t$ of $s$, if $t \in V^P_{\sigma_1}$ then $t \in Win^O_{\sigma_1}$. We examine all possible causes of divergence during the composition of $\sigma_2$ and $\tau$, and in each case identify a suitable $s$.

- Suppose $run\, u \in \mathcal{T}_{\sigma_2} \not\downarrow \mathcal{T}_\tau$. Then, since $\mathcal{D}_{\sigma_1;\tau} = \emptyset$, there must exist a prefix $run\, u'$ of $run\, u$ such that $u' \in \mathcal{T}_{\sigma_2} \setminus \mathcal{T}_{\sigma_1}$. Let $s = u'$.
- Suppose the divergence is caused by some interaction sequence $u$.
  - If $u \upharpoonright [\![\theta]\!] \in \mathcal{D}_{\sigma_2}$ then, since $\mathcal{D}_{\sigma_1;\tau} = \emptyset$, we must have $u \upharpoonright [\![\theta]\!] \in \mathcal{D}_{\sigma_2} \setminus \mathcal{D}_{\sigma_1}$. Let $s = u \upharpoonright [\![\theta]\!]$.
  - Suppose $u \upharpoonright ([\![\theta]\!], [\![\mathsf{com}]\!]) = run\, u' \in \mathcal{D}_\tau$. Then, since $\mathcal{D}_{\sigma_1;\tau} = \emptyset$, we must have $u' \in \mathcal{T}_{\sigma_2} \setminus \mathcal{T}_{\sigma_1}$. Let $s = u'$.

Since we have $\mathcal{D}_{\sigma_1;\tau} = \emptyset$, any prefix of $s$ which is in $V^P_{\sigma_1}$ must also be in $Win^P_{\sigma_1}$.

□

$$\frac{}{s, V \Downarrow s, V}$$

$$\frac{s, M \Downarrow s', i}{s, \mathbf{succ}(M) \Downarrow s', (i+1) \bmod max} \qquad \frac{s, M \Downarrow s', i}{s, \mathbf{pred}(M) \Downarrow s', (i-1) \bmod max}$$

$$\frac{s, M \Downarrow s', \mathbf{skip} \quad s', N \Downarrow s'', V}{s, M; N \Downarrow s'', V} \qquad \frac{s(X \mapsto 0), M \Downarrow s', V}{s \setminus X, \mathbf{new}\, X\, \mathbf{in}\, M \Downarrow s' \setminus X, V}$$

$$\frac{s, M \Downarrow s', 0 \quad s', N_0 \Downarrow s'', V}{s, \mathbf{if}\, M\, \mathbf{then}\, N_1\, \mathbf{else}\, N_0 \Downarrow s'', V} \qquad \frac{i > 0 \quad s, M \Downarrow s', i \quad s', N_1 \Downarrow s'', V}{s, \mathbf{if}\, M\, \mathbf{then}\, N_1\, \mathbf{else}\, N_0 \Downarrow s'', V}$$

$$\frac{s, M_1 \Downarrow s', \mathbf{skip}}{s, M_1\, \mathbf{or}\, M_2 \Downarrow s', \mathbf{skip}} \qquad \frac{s, M_2 \Downarrow s', \mathbf{skip}}{s, M_1\, \mathbf{or}\, M_2 \Downarrow s', \mathbf{skip}}$$

$$\frac{s, M \Downarrow s', x}{s, !M \Downarrow s', s'(x)} \qquad \frac{s, M \Downarrow s', \mathbf{mkvar}(M', N') \quad s', N' \Downarrow s'', i}{s, !M \Downarrow s'', i}$$

$$\frac{s, N \Downarrow s', i \quad s', M \Downarrow s'', x}{s, M{:=}N \Downarrow s''(x \mapsto i), \mathbf{skip}}$$

$$\frac{s, N \Downarrow s', i \quad s', M \Downarrow s'', \mathbf{mkvar}(M', N') \quad s'', M'i \Downarrow s''', \mathbf{skip}}{s, M{:=}N \Downarrow s''', \mathbf{skip}}$$

$$\frac{s, M \Downarrow s', \lambda x^{\theta}.M' \quad s', M'[N/x] \Downarrow s'', V}{s, MN \Downarrow s'', V}$$

$$\frac{s, M(\mu x^{\theta}.M) \Downarrow s', V}{s, \mu x^{\theta}.M \Downarrow s', V}$$

$V$ stands for terms in *canonical* form:

$$V ::= \mathbf{skip} \mid i \mid x \mid \lambda x^{\theta}.M \mid \mathbf{mkvar}(M, N).$$

There is no rule for $\Omega_{\beta}$, as the term is not supposed to converge.

**Figure 3. Operational semantics**

$$\frac{}{s, V \Downarrow_{\mathrm{must}}}$$

$$\frac{s, M \Downarrow_{\mathrm{must}}}{s, \mathbf{succ}(M) \Downarrow_{\mathrm{must}}} \qquad \frac{s, M \Downarrow_{\mathrm{must}}}{s, \mathbf{pred}(M) \Downarrow_{\mathrm{must}}}$$

$$\frac{s, M \Downarrow_{\mathrm{must}} \quad \forall_{s'}(s, M \Downarrow s', \mathbf{skip} \Rightarrow s', N \Downarrow_{\mathrm{must}})}{s, M; N \Downarrow_{\mathrm{must}}}$$

$$\frac{s, M \Downarrow_{\mathrm{must}} \quad \forall_{s'}(s, M \Downarrow s', 0 \Rightarrow s', N_0 \Downarrow_{\mathrm{must}}) \quad \forall_{s'}\forall_{i>0}(s, M \Downarrow s', i \Rightarrow s', N_1 \Downarrow_{\mathrm{must}})}{s, \mathbf{if}\ M\ \mathbf{then}\ N_1\ \mathbf{else}\ N_0 \Downarrow_{\mathrm{must}}}$$

$$\frac{s, M \Downarrow_{\mathrm{must}} \quad \forall_{s'}(s, M \Downarrow s', \mathbf{skip} \Rightarrow s', N \Downarrow_{\mathrm{must}})}{s, M; N \Downarrow_{\mathrm{must}}}$$

$$\frac{s, M_1 \Downarrow_{\mathrm{must}} \quad s, M_2 \Downarrow_{\mathrm{must}}}{s, M_1\ \mathbf{or}\ M_2 \Downarrow_{\mathrm{must}}}$$

$$\frac{s, M \Downarrow_{\mathrm{must}} \quad \forall_{s'}(s, M \Downarrow s', \mathbf{mkvar}(M', N') \Rightarrow s', N' \Downarrow_{\mathrm{must}})}{s, !M \Downarrow_{\mathrm{must}}}$$

$$\frac{s, N \Downarrow_{\mathrm{must}} \quad \forall_{s',i}(s, N \Downarrow s', i \Rightarrow s', M \Downarrow_{\mathrm{must}} \wedge \forall_{s''}(s', M \Downarrow s'', \mathbf{mkvar}(M', N') \Rightarrow s'', M'i \Downarrow_{\mathrm{must}}))}{s, !M \Downarrow_{\mathrm{must}}}$$

$$\frac{s(X \mapsto 0), M \Downarrow_{\mathrm{must}}}{s \setminus X, \mathbf{new}\ X\ \mathbf{in}\ M \Downarrow_{\mathrm{must}}}$$

$$\frac{M \Downarrow_{\mathrm{must}} \quad \forall_{s',M'}(s, M \Downarrow s', \lambda x.M' \Rightarrow s', M'[N/x] \Downarrow_{\mathrm{must}})}{MN \Downarrow_{\mathrm{must}}}$$

$$\frac{s, M(\mu x^{\theta}.M) \Downarrow_{\mathrm{must}}}{s, \mu x^{\theta}.M \Downarrow_{\mathrm{must}}}$$

**Figure 4. The must-converge predicate**