A Rice’s Theorem for Abstract Semantics

Paolo Baldan
Dipartimento di Matematica, University of Padova, Italy

Francesco Ranzato
Dipartimento di Matematica, University of Padova, Italy

Linpeng Zhang
Department of Computer Science, University College London, UK

Abstract

Classical results in computability theory, notably Rice’s theorem, focus on the extensional content of programs, namely, on the partial recursive functions that programs compute. Later and more recent work investigated intensional generalisations of such results that take into account the way in which functions are computed, thus affected by the specific programs computing them. In this paper, we single out a novel class of program semantics based on abstract domains of program properties that are able to capture nonextensional aspects of program computations, such as their asymptotic complexity or logical invariants, and allow us to generalise some foundational computability results such as Rice’s Theorem and Kleene’s Second Recursion Theorem to these semantics. In particular, it turns out that for this class of abstract program semantics, any nontrivial abstract property is undecidable and every decidable overapproximation necessarily includes an infinite set of false positives which covers all values of the semantic abstract domain.

2012 ACM Subject Classification Theory of computation → Computability; Theory of computation → Recursive functions; Theory of computation → Abstraction

Keywords and phrases Computability Theory, Recursive Function, Rice’s Theorem, Kleene’s Second Recursion Theorem, Program Analysis, Affine Program Invariants

Digital Object Identifier 10.4230/LIPIcs.ICALP.2021.117

Category Track B: Automata, Logic, Semantics, and Theory of Programming


Funding Paolo Baldan: Partially funded by University of Padova, under the SID2018 project “Analysis of STratic Analyses (ASTA)”; Italian Ministry of University and Research, under the PRIN2017 project no. 201784YSZ5 “AnalysisS of Program Analyses (ASPRA)”. Francesco Ranzato: Partially funded by University of Padova, under the SID2018 project “Analysis of STratic Analyses (ASTA)”; Italian Ministry of University and Research, under the PRIN2017 project no. 201784YSZ5 “AnalysisS of Program Analyses (ASPRA)”; Facebook Research, under a “Probability and Programming Research Award”.

Acknowledgements We are grateful to Roberto Giacobazzi for thorough discussions and comments.

1 Introduction

Most classical results in computability theory focus on the so-called extensional properties of programs, i.e., on the properties of the partial functions they compute. Notably, the renowned Rice’s Theorem [25] states that any nontrivial extensional property of programs is undecidable. Despite being very general, Rice’s Theorem and similar results in computability theory, due to the requirement of extensionality, leave out several intensional properties which are of utmost importance in the practice of programming. Essential intensional properties of programs include their asymptotic complexity of computation, their logical invariants (e.g., relations between variables at program points), or any event that might happen during program computation while not affecting the program output.

© Paolo Baldan, Francesco Ranzato, and Linpeng Zhang; licensed under Creative Commons License CC-BY 4.0

48th International Colloquium on Automata, Languages, and Programming (ICALP 2021).
Editors: Nikhil Bansal, Emanuela Merelli, and James Worrell; Article No. 117; pp. 117.1–117.19
Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany
A generalisation of well-established results of computability theory to the realm of program complexity has been put forward by Asperti [1]. A first observation is that Blum’s complexity classes [2], i.e., sets of recursive functions (rather than sets of programs) with some given (lower or upper) bound on their (space and/or time) complexity, are not adequate for investigating the decidability aspects of program complexity: in fact, viewed as program properties they are trivially extensional. Thus, a key idea in [1] is to focus on the so-called complexity cliques, namely, sets of programs (i.e., program indices) closed with respect to their extensional input/output behaviour and their asymptotic complexity. Asperti [1] showed how this approach enables intensional versions of Rice’s theorem, Rice-Shapiro theorem, and Kleene’s second recursion theorem ([8, 28] are standard references for these foundational results) for complexity cliques.

More recently, a different approach has been considered by Moyen and Simonsen in [19], where the classical definition of extensionality has been weakened to a notion of partial extensionality. Roughly, a given set of programs is partially extensional if it includes the set of all programs computing a given partial recursive function. It is shown in [19] that if a set of programs and its complement are partially extensional, then they cannot be both recursive. Interestingly, this result can be further generalised by replacing the extensionality with an equivalence relation on programs satisfying some suitable structural conditions, notably, the existence of a so-called intricated switching family. Moyen and Simonsen [19] show how to derive within their framework intensional versions of Rice’s Theorem – generalising Asperti’s result [1] – and Rice-Shapiro Theorem.

Main Contributions

Along the lines traced by Asperti [1], we investigate whether and how some fundamental extensional results of computability theory can be systematically generalised to intensional aspects of computation, but rather than focussing on specific intensional properties we deal with generic abstract program semantics. More in detail, we distill two fundamental properties of abstract program semantics in our approach: the strong smn property and the existence of a universal fair program, roughly, an interpreter that preserves the abstract semantics. We show that for abstract semantics satisfying the strong smn property and admitting a universal fair program, a generalisation of Kleene’s second recursion theorem can be proved. This, in turn, leads to a generalisation of Rice’s theorem. Besides relying on a general abstract program semantics, inspired by Moyen and Simonsen’s approach [19], we also relax the extensionality condition to partial extensionality. This weakening provides stronger impossibility results as it allows us to show that it is undecidable whether a given program can have a particular semantics, i.e., even nontrivial overapproximations of such properties are undecidable. On a different route, we establish a precise connection with Moyen and Simonsen’s work [19] by showing that for any abstract program semantics satisfying the strong smn property and a structural branching condition (roughly, expressing some form of conditional choice), we can prove the existence of an intricated switching family, which turns out to be the crucial hypothesis in [19] for deriving an intensional version of Rice’s theorem.

Therefore, on the one hand, we generalise the results in [1], going beyond complexity cliques, and, on the other hand, we provide an explicit characterisation of a class of program semantics that admit intricated switching families so that the results in [19] can be applied. Finally, we show some applications of our intensional Rice’s theorem that generalise some undecidability results for intensional properties used in static program analysis. In particular, we focus on program analysis in Karr’s abstract domain of affine relations between program
variables [13]. By exploiting an acute reduction to the undecidable Post correspondence problem, Müller-Olm and Seidl [20] prove that for affine programs with positive affine guards it is undecidable whether a given nontrivial affine relation holds at a given program point or not. Here, we first show that this class of affine programs with positive affine guards, modeled as control flow graphs, turns out to be Turing complete since, by selecting a suitable program semantics, these programs can simulate a URM. Then, this allows us to derive the undecidability result in [20] as a consequence of our results.

The rest of the paper is structured as follows. In Section 2, we provide some background and our basic notions. In Section 3, we introduce the strong smn property, fair universal programs, and the branching condition that will play a fundamental role in our results. In Section 4, we provide our generalisation of Kleene’s second recursion theorem and use it to derive our intensional Rice’s theorem. We also establish an explicit connection with the notion of intricated switching family given in [19]. Section 5 provides some applications of our results to the analysis of affine programs. Section 6 discusses in detail the relation with some of Asperti’s results [1] and with Rogers’ systems of indices [27, 28]. Finally, Section 7 concludes and outlines some directions of future work.

2 Basic Notions

Given an \( n \)-ary partial function \( f : \mathbb{N}^n \to \mathbb{N} \), we denote by \( \text{dom}(f) \) the domain of \( f \) and by \( \text{rng}(f) \triangleq \{ f(\vec{x}) : \vec{x} \in \text{dom}(f) \} \) its range. We write \( f(\vec{x}) \downarrow \) if \( \vec{x} \in \text{dom}(f) \) and \( f(\vec{x}) \uparrow \) if \( \vec{x} \not\in \text{dom}(f) \). Moreover, \( \lambda \vec{x}. \uparrow \) denotes the always undefined function. We denote by \( \mathcal{F}_n \triangleq \mathbb{N}^n \to \mathbb{N} \) the class of all \( n \)-ary (possibly partial) functions and by \( \mathcal{F} \triangleq \bigcup_n \mathcal{F}_n \) the class of all such functions. Additionally, \( \mathcal{C}_n \subseteq \mathcal{F}_n \) denotes the subset of \( n \)-ary partial recursive functions (\( \mathcal{C} \) stands for computable) and \( \mathcal{C} \triangleq \bigcup_n \mathcal{C}_n \) the set of all partial recursive functions.

**Assumption 2.1 (Turing completeness).** Throughout the paper, we assume a fixed Turing complete model and we denote by \( \mathcal{P} \) the corresponding set of programs. Moreover, we consider a fixed Gödel numbering for the programs in \( \mathcal{P} \) and, given an index \( a \in \mathbb{N} \), we write \( P_a \) for the \( a \)-th program in \( \mathcal{P} \). A program can take a varying number \( n \) of inputs and we denote by \( \phi_a^{(n)} \in \mathcal{C}_n \) the \( n \)-ary partial function computed by \( P_a \). Therefore, by Turing completeness, \( \{ \phi_a^{(n)} | a, n \in \mathbb{N} \} = \mathcal{C} \) must hold.

The binary relation between programs that compute the same \( n \)-ary function is called Rice’s equivalence and denoted by \( \sim_R^n \), i.e.,

\[
a \sim_R^n b \iff \phi_a^{(n)} = \phi_b^{(n)}. \]

Classical Rice’s theorem [25] compares the extension of programs, i.e., the functions they compute, and shows that unions of equivalence classes of programs computing the same function are undecidable. In Asperti’s work [1], by relying on the notion of complexity clique, the asymptotic program complexity can be taken into account. The idea here is to further generalise the approach in [1] by considering generic program semantics rather than asymptotic program complexity. Additionally, an equivalence relation on program semantics allows us to further abstract and identify programs with different extensional semantics. More precisely, such an equivalence relation allows us to reason on semantic program properties that may not hold with functional equivalence.
Definition 2.2 (Abstract semantics). An abstract semantics is a pair \( \langle \pi, \equiv_\pi \rangle \) where:

1. \( \pi : \mathbb{N}^2 \to \mathcal{F} \) associates a program index \( a \) and arity \( n \) with an \( n \)-ary function \( \pi_a^{(n)} \in \mathcal{F}_n \), called semantics of \( a \);
2. \( \equiv_\pi \subseteq \mathcal{F} \times \mathcal{F} \) is an equivalence relation between functions.

Given \( n \in \mathbb{N} \), the \( n \)-ary program equivalence induced by an abstract semantics \( \langle \pi, \equiv_\pi \rangle \) is the equivalence \( \sim_\pi^n \subseteq \mathbb{N} \times \mathbb{N} \) defined as follows: for all \( a, b \in \mathbb{N} \),

\[
 a \sim_\pi^n b \iff \pi_a^{(n)} \equiv_\pi \pi_b^{(n)}.
\]

The notation for the case of arity \( n = 1 \) will be simplified by omitting the arity, e.g., \( \phi_a \) instead of \( \phi_a^{(1)} \) and \( \sim_\pi \) in place of \( \sim_\pi^1 \). Abstract semantics can be viewed as a generalisation of the notion of system of indices (or numbering), as found in standard reference textbooks [22, 28] and discussed in detail later in Section 6.2. Let us now show how the standard extensional interpretation of programs, complexity and complexity cliques can be cast into our setting.

Example 2.3 (Concrete semantics). The concrete input/output semantics can be trivially seen as an abstract semantics \( \langle \phi, = \rangle \) where \( \phi_a^{(n)} \) is the \( n \)-ary function computed by \( P_a \) and \( = \) is the equality between functions. Observe that this concrete semantics induces an \( n \)-ary program equivalence which is Rice’s equivalence \( \sim_R^n \).

Example 2.4 (Domain semantics). For a given set of inputs \( S \subseteq \mathbb{N} \), consider \( \langle \phi, \equiv_S \rangle \) where \( \phi_a^{(n)} \) is the \( n \)-ary function computed by \( P_a \) and for \( f, g : \mathbb{N}^n \to \mathbb{N} \), their equivalence is defined by \( f \equiv_S g \iff \text{dom}(f) \cap S = \text{dom}(g) \cap S \).

Example 2.5 (Blum complexity). Let \( \Phi : \mathbb{N}^2 \to \mathcal{C} \) be a Blum complexity [2], i.e., for all \( a \in \mathbb{N} \) and \( \vec{x} \in \mathbb{N}^n \), (1) \( \Phi_a^{(n)}(\vec{x}) = m \) holds, and (2) for all \( m \in \mathbb{N} \), the predicate \( \Phi_a^{(n)}(\vec{x}) = m \) is decidable. Letting \( \Theta(f) \) to denote the standard Big Theta complexity class of a function \( f \), the pair \( \langle \Phi, \equiv_\Phi \rangle \) defined by

\[
 \Phi_a^{(n)} \equiv_\Phi \Phi_b^{(n)} \iff \Phi_a^{(n)}(\vec{y}) \in \Theta(\Phi_b^{(n)}(\vec{y}))
\]

is an abstract semantics.

Example 2.6 (Complexity clique). Complexity cliques as defined by Asperti in [1] can be viewed as an abstract semantics \( \langle \pi, \equiv_\pi \rangle \), that we will refer to as the complexity clique semantics. For each arity \( n \) and program index \( a \) let us define:

\[
 \pi_a^{(n)} \triangleq \lambda \vec{y} \langle \phi_a^{(n)}(\vec{y}), \Phi_a^{(n)}(\vec{y}) \rangle
\]

where \( \langle \ldots, \ldots \rangle : \mathbb{N}^2 \to \mathbb{N} \) is an effective bijective encoding for pairs and \( \Phi : \mathbb{N}^2 \to \mathcal{C} \) is a Blum complexity. The equivalence \( \equiv_\pi \) is defined as follows: for all \( a, b, n \in \mathbb{N} \),

\[
 \pi_a^{(n)} \equiv_\pi \pi_b^{(n)} \iff \phi_a^{(n)}(\vec{y}) \equiv_\Phi \pi_b^{(n)}(\vec{y})
\]

Classical Rice’s theorem states the undecidability of extensional program properties. Following [19], we parameterise extensional sets by means of a generic equivalence relation.

Definition 2.7 (~-extensional set). Let \( \sim \subseteq \mathbb{N} \times \mathbb{N} \) be an equivalence relation between programs whose equivalence classes are denoted by \( [a]_\sim \). A set of indices \( A \subseteq \mathbb{N} \) is called:

\( \sim \)-extensional when for all \( a, b \in A \), if \( a \in A \) and \( a \sim b \) then \( b \in A \);

partially \( \sim \)-extensional when there exists \( a \in \mathbb{N} \) such that \( [a]_\sim \subseteq A \);

universally \( \sim \)-extensional when for all \( a \in \mathbb{N} \), \( [a]_\sim \cap A \neq \emptyset \).
In words, a set $A$ is \sim-extensional if $A$ is a union of \sim-equivalence classes, partially \sim-extensional if $A$ contains at least a whole \sim-equivalence class, and universally \sim-extensional if $A$ contains at least an element from each \sim-equivalence class, i.e., its complement $\mathbb{N} \setminus A$ is not partially \sim-extensional. Notice that if $A$ is not trivial (i.e., $A \neq \emptyset$ and $A \neq \mathbb{N}$) and \sim-extensional then $A$ is partially \sim-extensional and not universally \sim-extensional. Let us observe that \sim-$\mu$-extensionality is the standard notion of extensionality so that classical Rice’s theorem [25] states that if $A$ is \sim-$\mu$-extensional and not trivial then $A$ is not recursive.\footnote{In [19], the term “extensional” is replaced by “compatible” when one refers to generic equivalence relations $\sim$.}

\section{Fair and Strong ssmn Semantics}

In this section, we identify some fundamental properties of abstract semantics that will be later used in our intensional computability results. A first basic property stems from the fundamental smn theorem and intuitively amounts to requiring that the operation of fixing some parameters of a program is effective and preserves its abstract semantics.

\begin{definition}[Strong smn semantics] An abstract semantics $\langle \pi, \equiv_\pi \rangle$ has the \textit{strong smn (ssmn)} property if, given $m, n \geq 1$, there exists a total computable function $s : \mathbb{N}^{m+2} \to \mathbb{N}$ such that for all $a, b \in \mathbb{N}$, $\vec{x} \in \mathbb{N}^m$:

$$
\lambda \vec{y}. \pi_{a}^{(n+1)}(\phi_{b}^{(m)}(\vec{x}), \vec{y}) \equiv_\pi \pi_{s(a,b,\vec{x})}^{(n)}.
$$

(1)

In such a case, the abstract semantics $\langle \pi, \equiv_\pi \rangle$ is called \textit{strong smn}.
\end{definition}

The above definition requires the property (1) which is slightly stronger than one would expect. The natural generalisation of the standard smn property, in the style, e.g., of [1], would amount to asking that, given $m, n \geq 1$, there exists a total computable function $s : \mathbb{N}^{m+1} \to \mathbb{N}$ such that for any program index $a \in \mathbb{N}$ and input $\vec{x} \in \mathbb{N}^m$, it holds $\lambda \vec{y}. \pi_{a}^{(m+n)}(\vec{x}, \vec{y}) \equiv_\pi \pi_{s(a,\vec{x})}^{(n)}$. The concrete semantics $\langle \phi, = \rangle$ of Example 2.3 clearly satisfies this ssmn property. In fact, the function $\lambda a, b, \vec{x}, \vec{y}. \pi_{a}^{(n+1)}(\phi_{b}^{(m)}(\vec{x}), \vec{y})$ is computable (by composition, relying on the existence of universal functions), hence the existence of a total computable $s : \mathbb{N}^{m+2} \to \mathbb{N}$ such that $\lambda \vec{y}. \pi_{a}^{(n+1)}(\phi_{b}^{(m)}(\vec{x}), \vec{y}) \equiv_\pi \pi_{s(a,\vec{x})}^{(n)}$ holds, as prescribed by Definition 3.1, follows by the standard smn theorem. It is easily seen that the same applies to the domain semantics of Example 2.4.

The reason for the stronger requirement (1) in Definition 3.1 is that, to deal with generic abstract semantics, thus going beyond asymptotic complexity, a suitable smn definition needs to embody a condition on program composition (of $a$ and $b$ in Definition 3.1). Indeed, if we consider the semantics based on program complexity (i.e., Examples 2.5 and 2.6), it turns out that whenever they enjoy the smn property in [1, Definition 11] and, additionally, they satisfy the linear time composition hypothesis in [1, Section 4] relating the asymptotic complexities of a program composition to those of its components, then they are ssmn semantics according to Definition 3.1. More details on the relationship with Asperti’s approach [1] will be given later in Section 6.1.

Note that for an ssmn abstract semantics $\langle \pi, \equiv_\pi \rangle$, there always exists a program whose denotation is equivalent to the always undefined function, namely,

$$
\text{for any arity } n \in \mathbb{N} \text{ there exists a program index } e_0 \in \mathbb{N} \text{ such that } \pi_{e_0}^{(n)} \equiv_\pi \lambda \vec{y}. \uparrow. \quad (2)
$$
In fact, if \( b \) is a program index for the always undefined function \( \lambda \vec{y}.\uparrow \), then, by (1), we have that \( \lambda \vec{y}.\pi_{n+1}^{(n)}(\phi_{b}(0), \vec{y}) = \lambda \vec{y}.\pi_{n+1}^{(n)}(0, b, 0) \), so that we can pick \( e_0 \triangleq s(0, b, 0) \).

It is also worth exhibiting an example of abstract semantics which is not ssmn. Let \( \pi_a(\vec{x}) \) be defined as the number of different variables accessed in a computation of the program \( a \) on the input \( \vec{x} \). Then, let us observe that the mere fact that \( \pi_a \) is always a total function trivially makes the abstract semantics \( \langle \pi, = \rangle \) non-ssmn.

To generalise Kleene’s second recursion theorem, besides the ssmn property, we need to postulate the existence of so-called fair universal programs, namely, programs that can simulate every other program w.r.t. a given abstract semantics. This generalises the analogous notion in [1, Definition 26], where this simulation is specific to complexity cliques and must preserve both the computed function and its asymptotic complexity.

**Definition 3.2 (Fair semantics).** An index \( u \in \mathbb{N} \) is a fair universal program for an abstract semantics \( \langle \pi, \equiv \rangle \) and an arity \( n \in \mathbb{N} \) if for all \( a \in \mathbb{N} \):

\[
\pi_{a(n)}^{(n)} \equiv \pi \lambda \vec{y}.\pi_{u(n+1)}^{(n+1)}(a, \vec{y}).
\]

An abstract semantics is fair if it admits a fair universal program for every arity.

Clearly, the concrete (Example 2.3) and domain (Example 2.4) semantics are fair. In general, as noted in [1], the existence of a fair universal program may not only depend on the reference abstract semantics, but also on the underlying computational model. For instance, when considering program complexity, as argued by Asperti [1, Section 6] by relying on some remarks by Blum [3], multi-tape Turing machines seem not to admit fair universal programs. By contrast, single tape Turing machines do have fair universal programs, despite the fact that this is commonly considered a folklore fact and cannot be properly quoted. Hereafter, when referring to the complexity-based semantics of Examples 2.5 and 2.6, we will implicitly use that they are ssmn and fair semantics.

## 4 Kleene’s Second Recursion Theorem and Rice’s Theorem

In this section, we show how some foundational results of computability theory can be extended to a general abstract semantics. The first approach relies on a generalisation of Kleene’s second recursion theorem, which is then used to derive a corresponding Rice’s theorem. A second approach consists in identifying conditions that ensure the existence of an intricated switching family in the sense of [19], from which Rice’s theorem also follows.

### 4.1 Kleene’s Second Recursion Theorem

We show that Kleene’s second recursion theorem holds for any fair ssmn abstract semantics. This generalises the analogous result proved by Asperti [1, Section 5] for complexity cliques.

**Theorem 4.1 (Intensional Second Recursion Theorem).** Let \( \langle \pi, \equiv \rangle \) be a fair ssmn abstract semantics. For any total computable function \( h : \mathbb{N} \to \mathbb{N} \) and arity \( n \in \mathbb{N} \), there exists an index \( a \in \mathbb{N} \) such that \( a \sim^a_{\pi} h(a) \).

As an example, this result, instantiated to the complexity semantics of Example 2.5, entails the impossibility of designing a program transform that modifies the asymptotic complexity of every program, even without preserving its input-output behavior.
Example 4.2 (Fixpoints of Blum complexity semantics). Let \( \langle \Phi, \equiv_{\text{a}} \rangle \) be the Blum complexity semantics of Example 2.5. A program transform \( h : \mathbb{N} \to \mathbb{N} \) is a total computable function which maps indices of programs into indices of transformed programs. By applying Theorem 4.1, for any arity \( n \in \mathbb{N} \), we know that there exists an index of a program \( a \) such that \( a \sim_{n}^{a} h(a) \) holds, so that the program transform \( h \) necessarily does not alter the asymptotic complexity of, at least, the program \( a \).

This second recursion theorem allows us to obtain an intensional version of Rice’s theorem for fair and ssrn abstract semantics. Inspired by [19], we generalise the statement to cover partial extensional properties.

Theorem 4.3 (Rice by fair and ssrn semantics). Let \( \langle \pi, \equiv_{\text{a}} \rangle \) be a fair and ssrn semantics. If \( A \subseteq \mathbb{N} \) is partially \( \sim_{n}^{a} \)-extensional and not universally \( \sim_{n}^{a} \)-extensional, for some arity \( n \in \mathbb{N} \), then \( A \) is not recursive.

Fig. 1 provides a graphical representation of this result: if we can find two program indices \( a_{0}, a_{1} \in \mathbb{N} \) such that \( A \) overapproximates the \( \equiv_{\text{a}} \)-equivalence class \( [a_{1}]_{\sim_{a}} \) and \( A \) does not intersect \( [a_{0}]_{\sim_{a}} \), then \( A \) cannot be recursive. For example, as observed in Section 3, the asymptotic complexity on a suitable computational model such as single tape Turing machines is a fair ssrn semantics, so that Theorem 4.3 applies. Let us illustrate some further applications of Theorem 4.3.

Example 4.4 (Halting set). Let \( \langle \phi, \equiv_{\|n\|} \rangle \) be the domain semantics of Example 2.4 with \( S = \mathbb{N} \), hence \( f \equiv_{n} g \) when \( \text{dom}(f) = \text{dom}(g) \). The halting set \( K \equiv \{ \alpha \in \mathbb{N} \mid \phi_{\alpha}(a) \downarrow \} \) can be proved to be non-recursive by resorting to Theorem 4.3 for \( \langle \phi, \equiv_{\|n\|} \rangle \). Let \( e_{0}, e_{1} \in \mathbb{N} \) be such that \( \phi_{e_{0}} = \lambda x.1 \) and \( \phi_{e_{1}} = \lambda x.1 \). Since \( [e_{1}]_{\equiv_{\|n\|}} \) is the set of programs that compute total functions, we have that \( [e_{1}]_{\equiv_{\|n\|}} \subseteq K \). Moreover, \( [e_{0}]_{\equiv_{\|n\|}} \) is the set of nonterminating programs for any input, so that \( [e_{0}]_{\equiv_{\|n\|}} \cap K = \emptyset \). This means that \( \langle \phi, \equiv_{\|n\|} \rangle \) satisfies the hypotheses of Theorem 4.3, thus entailing that \( K \) is not recursive.

Example 4.5 (Complexity sets). Let \( \langle \phi, = \rangle, \langle \Phi, \equiv_{\phi} \rangle \) be, resp., the semantics of Examples 2.3 and 2.5. Let \( \text{sort} : \mathbb{N} \to \mathbb{N} \) be a total function that takes as input an encoded sequence of numbers and outputs the encoding of the corresponding sorted sequence. It turns out that by applying Theorem 4.3, the following sets can be proved to be non-recursive:

1. \( A \equiv \{ \alpha \mid \Phi_{\alpha} \in \Theta(n \log n) \land \phi_{\alpha} = \text{sort} \} \).
2. \( B \equiv \{ \alpha \mid \Phi_{\alpha} \in \mathcal{O}(n \log n) \} \).
3. \( C \equiv \{ \alpha \mid \Phi_{\alpha} \in \Omega(n \log n) \} \).

Let \( is, ms \) be different implementations of \( \text{sort} \), i.e., \( \phi_{is} = \phi_{ms} = \text{sort} \), such that \( \Phi_{is} \in \Theta(n^{2}) \) and \( \Phi_{ms} \in \Theta(n \log n) \) – \( is \) and \( ms \) could be, resp., insertion and merge sort. Recall that \( \sim_{R} \) denotes the Rice equivalence induced by \( \langle \phi, = \rangle \) (i.e., \( a \sim_{R} b \Leftrightarrow \phi_{a} = \phi_{b} \)), and, in turn, let \( \sim_{\Phi_{R}} \equiv \sim_{\Phi} \cap \sim_{R} \) be the equivalence induced by the complexity clique semantics of Example 2.6, which is a fair ssrn semantics. Then, we have that:
(1) since $[is]_{\sim_{\omega}} \cap A = \emptyset$ and $[ms]_{\sim_{\omega}} \subseteq A$, by Theorem 4.3 $A$ is non-recursive;
(2) since $[is]_{\sim_{\omega}} \cap B = \emptyset$ and $[ms]_{\sim_{\omega}} \subseteq B$, by Theorem 4.3 $B$ is non-recursive;
(3) let $e$ be any program index such that $\Phi_e \in \Theta(1)$. Since $[e]_{\sim_{\omega}} \cap C = \emptyset$ and $[is]_{\sim_{\omega}} \subseteq C$,
by Theorem 4.3, the set $C$ is non-recursive.

It is worth remarking that in Example 4.5, $n \log n$ could be replaced by any function, thus showing the undecidability of the asymptotic complexities “big O” (case (2)) and “big Omega” (case (3)). Let us also point out that Example 4.4 shows how easily the halting set $K$ can be proved to be non-recursive by applying Theorem 4.3.

### 4.2 Branching Semantics

Let us investigate the connection between our results and the key notion of intricated switching family used by Moyen and Simonsen [19] for proving their intensional version of Rice’s theorem. Firstly, we argue that every ssnn abstract semantics admits an intricated switching family whenever it is able to express a suitable form of conditional branching. This allows us to derive an intensional Rice’s theorem. Moreover, we show that for fair and ssnn semantics, the identity can always play the role of intricated switching family.

**Definition 4.6** (Branching and discharging semantics). An abstract semantics $\langle \pi, \equiv_{\pi} \rangle$ is branching if, given $n \geq 1$, there exists a total computable function $r : \mathbb{N}^2 \to \mathbb{N}$ such that for all $a, b, c_1, c_2, x \in \mathbb{N}$ such that $c_1 \neq c_2$:

$$\lambda y.\pi^{(n)}_{r(a,b,c_1,c_2)}(x, y) \equiv_{\pi} \begin{cases} 
\lambda y.\pi_a^{(n)}(x, y) & \text{if } x = c_1 \\
\lambda y.\pi_b^{(n)}(x, y) & \text{if } x = c_2 \\
\lambda y.↑ & \text{otherwise}
\end{cases}$$

Moreover, $\langle \pi, \equiv_{\pi} \rangle$ is (variable) discharging if, for all $n \geq 1$, there exists a total computable function $t : \mathbb{N} \to \mathbb{N}$ such that for all $a, x \in \mathbb{N}$:

$$\pi_a^{(n)} \equiv_{\pi} \lambda y.\pi_{t(a)}^{(n+1)}(x, y).$$

Hence, intuitively, an abstract semantics is branching when it is able to model the branching structure of conditional statements with multiple positive guards, while the property of being variable discharging holds when one can freely add fresh and unused variables without altering the abstract semantics. Let us recall the notion of intricated switching family from [19, Definition 5].

**Definition 4.7** (Intricated switching family [19, Definition 5]). Let $\sim \subseteq \mathbb{N} \times \mathbb{N}$ be an equivalence relation on program indices. An intricated switching family (ISF) w.r.t. $\sim$ is an indexed set of total computable functions $\{\sigma_{a,b}\}_{a,b\in\mathbb{N}}$, with $\sigma_{a,b} : \mathbb{N} \to \mathbb{N}$, such that for all $a, b \in \mathbb{N}$, the sets $A_{a,b} = \{x \in \mathbb{N} \mid \sigma_{a,b}(x) \sim a\}$ and $B_{a,b} = \{x \in \mathbb{N} \mid \sigma_{a,b}(x) \sim b\}$ are recursively inseparable (i.e., no recursive $C$ exists such that $A_{a,b} \subseteq C$ and $C \cap B_{a,b} = \emptyset$).

Moyen and Simonsen [19, Theorem 3] show that if an equivalence $\sim$ admits an ISF, then every partially $\sim$-extensional and not universally $\sim$-extensional set is not recursive. A simplified version of their intensional result, tailored for our setting, can be stated as follows.

---

2 For the sake of simplicity, [19, Definition 5] is here instantiated to the case of recursive sets.
Theorem 4.8 ([19, Theorem 3]). Let \( \sim \subseteq \mathbb{N} \times \mathbb{N} \) be an equivalence relation. If \( A \subseteq \mathbb{N} \) is partially \( \sim \)-extensional, not universally \( \sim \)-extensional and there exists an ISF w.r.t. \( \sim \) then \( A \) is not recursive.

Branching semantics allow us to derive the following intensional version of Rice’s Theorem.

Theorem 4.9 (Rice by branching, discharging and ssmn semantics). Let \( \langle \pi, \equiv_{\pi} \rangle \) be a branching, discharging and ssmn semantics. If \( A \subseteq \mathbb{N} \) is partially \( \sim_n^{\pi} \)-extensional and not universally \( \sim_n^{\pi} \)-extensional for some arity \( n \in \mathbb{N} \), then \( A \) is not recursive.

Let us discuss more in detail the relationship with the approach in [19]. Firstly, it turns out that a fair ssmn semantics always admits a canonical ISF, namely, the identity \( \text{Id} \equiv \{(\lambda x.x)_{a,b}\}_{a,b \in \mathbb{N}} \).

Proposition 4.10. Let \( \langle \pi, \equiv_{\pi} \rangle \) be a fair and ssmn semantics. Then, the identity \( \text{Id} \) is an ISF w.r.t. \( \sim_n^{\pi} \), for all \( n \geq 1 \).

Let us point out that the identity function has not been exploited in [19], that instead focuses on the standard switching family. It turns out that the identity function plays a key role as ISF.

Proposition 4.11. Let \( \sim \subseteq \mathbb{N} \times \mathbb{N} \) be an equivalence relation. The following statements are equivalent:

1. Every set \( A \subseteq \mathbb{N} \) partially \( \sim \)-extensional and not universally \( \sim \)-extensional is non-recursive.
2. The identity \( \text{Id} \) is an ISF w.r.t. \( \sim \).
3. There exists an ISF w.r.t. \( \sim \).

Therefore, the above result roughly states that the identity function is the “canonical” ISF, meaning that if an ISF exists, then \( \text{Id} \) is an ISF as well. Moreover, the intensional Rice’s Theorem 4.8 of [19] provides a sufficient condition (i.e., the existence of an ISF) for a partially and not universally extensional set to be undecidable. Proposition 4.11 enhances Theorem 4.8 by showing that such a sufficient condition is necessary as well, or, equivalently, that a partially and not universally extensional set is undecidable iff there exists an ISF.

We conclude this section by discussing an alternative notion of branching, which requires the preservation of a full conditional statement with positive and negative guards. This is an adaptation to our framework of a property that would be needed to exploit a so-called standard switching family as defined in [19, Example 1].

Definition 4.12 (Strongly branching semantics). An abstract semantics \( \langle \pi, \equiv_{\pi} \rangle \) is strongly branching if, given \( n \geq 1 \), there exists a total computable function \( r : \mathbb{N}^3 \to \mathbb{N} \) such that for all \( a, b, c, x \in \mathbb{N} \):

\[
\lambda \vec{y}.\pi^{(n)}_{r(a,b,c)}(x, \vec{y}) \equiv_{\pi} \begin{cases} 
\lambda \vec{y}.\pi^{(n)}_a(x, \vec{y}) & \text{if } x = c \\
\lambda \vec{y}.\pi^{(n)}_b(x, \vec{y}) & \text{otherwise}
\end{cases}
\]

Despite appearing to be more natural, the preservation of conditionals with positive and negative conditions is a stronger requirement than the one we considered in Definition 4.6. Indeed, it turns out that every ssmn and strongly branching semantics is a branching semantics.

Proposition 4.13 (Strongly branching implies branching). If \( \langle \pi, \equiv_{\pi} \rangle \) is an ssmn and strongly branching semantics, then \( \langle \pi, \equiv_{\pi} \rangle \) is branching.
4.3 An Application to Static Program Verifiers

We adapt the general definition of static program verifier of Cousot et al. [7, Definition 4.3] to our framework. Given a program property \( P \subseteq \mathbb{N} \) to check, a static program verifier is a total recursive function \( V : \mathbb{N} \rightarrow \{0, 1\} \), which is sound when for all \( p \in \mathbb{N} \), \( V(p) = 1 \Rightarrow p \in P \), while \( V \) is precise if the reverse implication also holds, i.e., when \( V(p) = 1 \iff p \in P \) holds. Informally, soundness guarantees that only false negatives are allowed, i.e., \( \mathbb{N} \setminus P \) is merely a subset of \( \{ p \in \mathbb{N} : V(p) = 0 \} \), while precise verifiers output true positives and true negatives only (i.e., they decide \( P \)).

Classical Rice’s theorem clearly entails the impossibility of designing a precise verifier for a nontrivial extensional property. However, one may wonder whether there exist sound verifiers with “few” false negatives. By applying our intensional Theorem 4.3, we are able to show that sound but imprecise verifiers necessarily have at least one false negative for each equivalence class of programs, even for intensional properties.

Example 4.14 (Constant value verifier). Assume we are interested in checking if a program can output a given constant value, for instance, zero with the aim of statically detecting division-by-zero bugs. Let \( V \) be a sound static verifier for the set \( P_{\neq 0} \triangleq \{ p \in \mathbb{N} | 0 \in \text{rng}(\phi_p) \} \) of programs that output zero for some input. The set \( N \triangleq \{ p \in \mathbb{N} | V(p) = 0 \} \) is recursive since \( V \) is assumed to be a total computable function. By soundness of \( V \), we have that \( \mathbb{N} \setminus P_{\neq 0} \subseteq N \), so that \( N \) includes, for example, the programs computing the constant function \( \lambda x.1 \). Therefore, \( N \) is partially extensional, and, by Theorem 4.3, \( N \) has to be universally extensional. This means that for any computable function \( f \in \mathcal{C} \) there exists a program \( p \in \mathbb{N} \) that computes \( f \) such that \( V(p) = 0 \). Thus, when \( 0 \in \text{rng}(f) \) holds (e.g., for \( f = \lambda x.0 \)), \( V \) necessarily outputs a false negative for \( p \). Hence, \( V \) outputs infinitely many false negatives.

Example 4.15 (Complexity verifier). Consider a speculative sound static verifier \( V \) for recognizing programs that meet some lower bound, for instance, programs having a cubic lower bound \( P_{\Omega(n^3)} \triangleq \{ p \in \mathbb{N} | \Phi_p = \Omega(n^3) \} \). Thus, \( N \triangleq \{ p \in \mathbb{N} | V(p) = 0 \} \) has to be recursive and if \( \sim_\Phi \) is the program equivalence induced by the Blum complexity semantics \( (\Phi, \equiv_\Phi) \) of Example 2.5 then, by soundness of \( V \), we have, for example, \( \{ p \in \mathbb{N} | \Phi_p = \Theta(1) \} \subseteq N \). This means that \( N \) is partially \( \sim_\Phi \)-extensional and, by Theorem 4.3, \( N \) is universally extensional, namely, \( V \) will output 0 for at least a program in each Blum complexity class. For instance, even some programs with an exponential lower bound will be wrongly classified by \( V \) as programs that do not meet a cubic lower bound.

As shown by Cousot et al. [7, Theorem 5.4], precise static verifiers cannot be designed (unless for trivial program properties). The examples above prove that, additionally, we cannot have any certain information on an input program \( p \) whenever the output of a sound (and imprecise) verifier for \( p \) is 0. In fact, when this happens, \( p \) could compute any partial function (cf. Example 4.14) or have any complexity (cf. Example 4.15).

5 On the Decidability of Affine Program Invariants

Karr’s abstract domain [13] consisting of affine equalities between program variables, such as \( 2x - 3y = 1 \), is well known and widely used in static program analysis [18, 26]. Karr [13] put forward an algorithm that infers for each program point \( q \) of a control flow graph modelling an affine program \( P \) (i.e., an unguarded program with non-deterministic branching and affine assignments) a set of affine equalities that hold among the variables of \( P \) when the control reaches \( q \), namely, an affine invariant for \( P \). Müller-Olm and Seidl [20] show that
Karr’s algorithm actually computes the strongest affine invariant for affine programs (this result has been extended to a slightly larger class of affine programs in [23, Theorem 5.1]). Moreover, they design a more efficient algorithm implementing this static analysis and they extend in [21] the algorithm for computing bounded polynomial invariants, i.e., the strongest polynomial equalities of degree at most a given $d \in \mathbb{N}$. Later, Hrushovski et al. [11] put forward a sophisticated algorithm for computing the strongest unbounded polynomial invariants of affine programs, by relying on the Zariski closure of semigroups.

On the impossibility side, Müller-Olm and Seidl [20, Section 7] prove that for affine programs allowing positive affine guards it is undecidable whether a given nontrivial affine equality holds at a given program point or not. In practical applications, static analyses on Karr’s domain of guarded affine programs ignore non-affine Boolean guards, while for an affine guard $b$, the current affine invariant $i$ is propagated through the positive branch of $b$ by the intersection $i \cap b$, that remains an affine subspace. By the aforementioned undecidability result [20, Section 7], this latter analysis algorithm for guarded affine programs turns out to be sound but necessarily imprecise, thus inferring affine invariants which are not the strongest ones.

Müller-Olm and Seidl [20, Section 7] prove their undecidability result by exploiting an acute reduction to the undecidable Post correspondence problem, inspired by early reductions explored in data flow analysis [9, 12]. In this section, we show that our Theorem 4.9 allows us to derive and extend this undecidability result by exploiting an orthogonal intensional approach. More precisely, we prove that any nontrivial (and not necessarily affine) relation on the states of control flow graphs of programs allowing: (1) zero, variable and successor assignments, resp., $x := 0$, $x := y$ and $x := y + 1$, and (2) positive equality guards $x = y$ and $x = v^?$, turns out to be undecidable. Since these control flow graphs form a subclass of affine programs with positive affine guards, the undecidability result of Müller-Olm and Seidl [20, Section 7] is retrieved as a consequence.

We consider control flow graphs that consist of program points connected by edges labeled by assignments and guards. Variables are denoted by $x_i$, with $i \in \mathbb{N}$, and store values ranging in $\mathbb{N}$, while Karr’s abstract domain is designed for variables assuming values in $\mathbb{Q}$. Clearly, from a computability perspective, this is not a restriction simply by considering a computable bijection between $\mathbb{N}$ and $\mathbb{Q}$.

Definition 5.1 (Basic affine control flow graph). A basic affine control flow graph (BACFG) is a tuple $G = (N, E, s, e)$, where $N$ is a finite set of nodes, $s, e \in N$ are the start and end nodes, and $E \subseteq N \times \text{Com} \times N$ is a set of labelled edges, where the set Com of commands consists of assignments of type $x_n := 0$, $x_n := x_m$, $x_n := x_m + 1$, and equality guards of type $x_n = x_m^?$, $x_n = v^?$, with $v \in \mathbb{N}$.

Let us remark that BACFGs only include basic affine assignments and positive affine guards, in particular inequality checks such as $x_n \neq x_m^?$ and $x_n \neq v^?$ are not allowed. Thus, BACFGs are a subclass of affine programs with positive affine guards.

As in dataflow analysis and abstract interpretation [6, 9, 26], BACFGs have a collecting semantics where, given a set of input states $I_n$, each program point is associated with the set of states that occur in some program execution from some state in $I_n$. A finite number of variables may occur in a BACFG, so that a state of a BACFG $G$ is a tuple $(x_1, \ldots, x_k) \in \mathbb{N}^k$, where $k$ is the maximum variable index occurring in $G$ and $k = 0$ is a degenerate case for trivial BACFGs with $\mathbb{N}^0 = \{\bullet\}$. The collecting transfer function $f_{\mathcal{C}}(\cdot) : \text{Com} \rightarrow \wp(\mathbb{N}^k) \rightarrow \wp(\mathbb{N}^k)$ for $k \in \mathbb{N}$ variables and with $n, m \in [1, k]$ is defined as follows:
A Rice’s Theorem for Abstract Semantics

Let us recall that an ssnm abstract semantics needs an underlying Turing complete concrete semantics of programs (cf. Assumption 2.1). A crucial observation is that any URM (Unlimited Register Machine\(^3\)) program, provided with suitable operational semantics, can be simulated by a BACFG, that is, BACFGs turn out to be Turing complete despite not including full (both positive and negative) Boolean tests.

\(^3\) Recall that URMs are a Turing complete computational model [8].
can be simulated by the BACFGs depicted in Figures 2 and 3. While the BACFGs in Figure 2 are trivial, let us describe more in detail how to simulate a jump instruction by the BACFG in Figure 3. Intuitively, a difficulty arises for simulating the negative branch $x_n \neq x_m?$. Here, the BACFG at node $q_i$ initialises a fresh unused variable $z$ with both $x_n + 1$ and $x_m + 1$ and transfers the control to a node $inc_i$ where $z$ is incremented infinitely many times. Thus, in the least fixpoint solution, at node $inc_i$ the variable $z$ stores any value $v > \min(x_m, x_n)$, including $z = \max(x_m, x_n)$. Suppose now that $x_n > x_m$ holds: in this case, the guard $x_n = z?$ between nodes $inc_i$ and $q_{i+1}$ eventually will be made true and at the node $q_{i+1}$ the store will retain the original values of all variables ($x_m$ and $x_n$ included), except for the new variable $z$ which will be ignored by the remaining nodes. The case $x_m > x_n$ is analogous. Therefore, it turns out that the node $q_{i+1}$ will be reached if and only if $x_m \neq x_n$ holds, while $q_p$ will be reached if and only if $x_m = x_n$ holds, thus providing a simulation for the jump instruction $j(m,n,p)$.

### 5.2 Concrete and Abstract Semantics

One key insight is that the concrete semantics is defined on the URM programs that satisfy the Assumption 2.1 of Turing completeness, while the abstract semantics is defined on BACFGs. Let us consider two Gödel numberings for BACFGs and URMs, so that for an index $a \in \mathbb{N}$, $G_a$ and $RM_a$ denote, resp., the $a$-th BACFG and URM programs. The concrete semantics $\langle \phi_a = \rangle$ of URMs, for an index $a \in \mathbb{N}$ and an arity $n \in \mathbb{N}$, is defined as follows: for all $\bar{x} \in \mathbb{N}^n$,

$$\phi_a^{(n)}(\bar{x}) \triangleq \begin{cases} y & \text{if } RM_a \text{ on input } \bar{x} \text{ halts with value } y \text{ on its first register,} \\ \uparrow & \text{otherwise.} \end{cases}$$

On the other hand, the abstract semantics of BACFGs is as follows.
Definition 5.4 (State semantics of BACFGs). Let $Q \subseteq \wp(\mathbb{N}^n)$ be a predicate on sets of states with $t \in \mathbb{N}$ variables. The state semantics $\langle Q, = \rangle$ of BACFGs, for any index $a \in \mathbb{N}$ and arity $n \in \mathbb{N}$, is given by the function $Q_a^{(n)} : \mathbb{N}^n \rightarrow \{0, 1\}$ defined as follows: for all $\vec{x} \in \mathbb{N}^n$,

$$Q_a^{(n)}(\vec{x}) \triangleq \begin{cases} 
1 & \text{if } [G_a]_{\vec{x}}[e_a] \neq \emptyset \land [G_a]_{\vec{x}}[e_a] |_t \in Q \\
0 & \text{if } [G_a]_{\vec{x}}[e_a] \neq \emptyset \land [G_a]_{\vec{x}}[e_a] |_t \not\in Q \\
\uparrow & \text{if } [G_a]_{\vec{x}}[e_a] = \emptyset
\end{cases}$$

where $e_a$ is the end node of the $a$-th BACFG $G_a$.

Predicates of type $Q \subseteq \wp(\mathbb{N}^n)$ are also known as hyperproperties [5] and the state semantics of Definition 5.4 models the validity of a given predicate $Q$ at the end node of a BACFG. Note that it is not restrictive to consider the end node, since this can be arbitrarily chosen in a BACFG.

Theorem 5.5. The state semantics of Definition 5.4 is ssnm, branching and discharging.

Let us now consider a state semantics $\langle Q, = \rangle$ for some predicate $Q \subseteq \wp(\mathbb{N}^n)$. For all $n \geq 1$, let us define two sets $A^{\mathcal{V}Q}$ and $A^{\mathcal{Z}Q}$, by distinguishing two cases depending on whether $Q$ includes the empty set, that models nontermination, or not:

1. if $\emptyset \not\in Q$ then $A^{\mathcal{V}Q} \triangleq \{ a \in \mathbb{N} | \forall \vec{y}. Q_a^{(n)}(\vec{y}) = 1 \}$ and $A^{\mathcal{Z}Q} \triangleq \{ a \in \mathbb{N} | \exists \vec{y}. Q_a^{(n)}(\vec{y}) = 1 \}$;
2. if $\emptyset \in Q$ then $A^{\mathcal{V}Q} \triangleq \{ a \in \mathbb{N} | \forall \vec{y}. Q_a^{(n)}(\vec{y}) \in \{1, \uparrow\} \}$ and $A^{\mathcal{Z}Q} \triangleq \{ a \in \mathbb{N} | \exists \vec{y}. Q_a^{(n)}(\vec{y}) \in \{1, \uparrow\} \}$.

Hence, $A^{\mathcal{V}Q}$ ($A^{\mathcal{Z}Q}$) is the set of BACFGs such that $Q$ holds at $e_a$ for any (some) input state. It turns out that if $A^{\mathcal{V}Q}$ is nontrivial then it is not recursive. Indeed, observe that $A^{\mathcal{V}Q}$ is $\sim_Q$-extensional, so that Theorem 5.5 enables applying Theorem 4.9 to $\langle Q, = \rangle$. The same argument applies to the existential version $A^{\mathcal{Z}Q}$. We have therefore the following consequence.

Corollary 5.6. If $Q$ is not trivial then $A^{\mathcal{V}Q}$ and $A^{\mathcal{Z}Q}$ are not recursive.

Corollary 5.6 means that we cannot decide if a nontrivial predicate $Q$ holds at a given program point of a BACFG for all input states, neither whether there exists an input state that will make $Q$ true. It is worth remarking that the predicates $Q$ are arbitrary and include, but are not limited to, relational predicates between program variables such as affine equalities of Karr’s abstract domain. Let us define some noteworthy examples of predicates:

1. Given a set of affine equalities $\text{aff} = \{ a_j \cdot \vec{x} = b_j \}_{j=1}^m$, with $a_j \in \mathbb{Z}^t$ and $b_j \in \mathbb{Z}$, $Q_{\text{aff}} \triangleq \{ S \in \wp(\mathbb{N}^t) \mid \forall \vec{v} \in S \forall j \in [1, m]. a_j \cdot \vec{v} = b_j \}$;
2. Given $i \in [1, t]$ and $c \in \mathbb{N}$, $Q_{\text{conc}} \triangleq \{ S \in \wp(\mathbb{N}^t) \mid \exists \vec{v} \in S. v_i = c \}$;
3. Given a size $k \in \mathbb{N}$, $Q_{\text{fin}} \triangleq \{ S \in \wp(\mathbb{N}^t) \mid |S| = k \}$ and $Q_{\text{fin}} \triangleq \cup_{k \in \mathbb{N}} Q_{\text{fin}}$.

Therefore, Corollary 5.6 for $A^{\mathcal{V}Q, a}$ entails the undecidability result of Müller-Olm and Seidl [20, Section 7] discussed above. The predicate $Q_{\text{conc}}$ can be used to derive the undecidability of checking if some variable $x_i$ may store a given constant $c$ for affine programs with positive affine guards, e.g., for $c = 0$ this amounts to the undecidability of detecting division-by-zero bugs. Finally, with $Q_{\text{fin}}$, we obtain the undecidability of dead code elimination. $Q_{\text{fin}}$ entails the well-known undecidability of constant detection [9, 24], while the existential predicate $Q_{\text{fin}}$ encodes whether some program point may only have finitely many different states.
6 Discussion of Related Work

In this section we discuss in detail the relation with some of Asperti’s results [1] and with Rogers’ systems of indices [27, 28].

6.1 Relation with Asperti’s Approach

We show that our ssmn property in Definition 3.1 is a generalisation of the smn property in Asperti’s approach [1], in a way that the Kleene’s second recursion theorem and Rice’s theorem for complexity cliques in [1] arise as instances of the corresponding results in our approach. Let us first recall and elaborate on the axioms for the complexity of function composition studied by Lischke [15, 16, 17] and assumed in [1, Section 4].

Definition 6.1 (Linear time and space complexity composition). Consider a given concrete semantics $\phi$ and a Blum complexity $\Phi$. The pair $\langle \phi, \Phi \rangle$ has the linear time composition property if there exists a total computable function $h : \mathbb{N}^2 \to \mathbb{N}$ such that for all $i, j \in \mathbb{N}$:

1. $\phi_{h(i,j)} = \phi_i \circ \phi_j$,
2. $\Phi_{h(i,j)} \in \Theta(\Phi_i \circ \Phi_j)$.

If (2) is replaced by

2’. $\Phi_{h(i,j)} \in \Theta(\max\{\Phi_i \circ \Phi_j, \Phi_j \})$

then $\langle \phi, \Phi \rangle$ has the linear space composition property.

Roughly speaking, the linear time composition property states that there exists a program $h(i, j)$ which computes the composition $\phi_i(\phi_j(x))$ in an amount of time which is asymptotically equivalent to the sum of the time needed for computing $P_i$ on input $\phi_j(x)$ and the time to compute $P_j$ on input $x$. On the other hand, the linear space composition property aims at modeling the needed space, so that rather than adding the complexities of $P_i$ and $P_j$, their maximum is considered, since this intuitively is the maximum amount of space needed for computing a composition of programs.

By observing that $\Theta(\max\{\Phi_i \circ \Phi_j, \Phi_j \}) = \Theta(\Phi_i \circ \Phi_j + \Phi_j)$ we can merge the linear time and space properties of Definition 6.1 and extend them for $n$-ary compositions as follows.

Definition 6.2 (Linear complexity composition). Given a concrete semantics $\phi$ and a Blum complexity $\Phi$, the pair $\langle \phi, \Phi \rangle$ has the linear complexity composition property if, given $n, m \geq 1$, there exists a total computable function $h : \mathbb{N}^2 \to \mathbb{N}$ such that for all $i, j \in \mathbb{N}$:

$$
\phi_{h(i,j)}^{(m+n)}(\vec{x}, \vec{y}) = \lambda \vec{x} \lambda \vec{y}. \phi_{\phi_i^{(n+1)}(\phi_j^{(m)}(\vec{x}), \vec{y})}^{(m+n)}(\vec{x}, \vec{y}),
\Phi_{h(i,j)}^{(m+n)}(\vec{x}, \vec{y}) \in \Theta(\lambda \vec{x} \lambda \vec{y}. (\phi_{\phi_i^{(n+1)}(\phi_j^{(m)}(\vec{x}), \vec{y})}^{(m+n)}(\vec{x}, \vec{y}) + \Phi_{\phi_j^{(m)}(\vec{x}, \vec{y})}^{(m+n)}(\vec{x}, \vec{y}))).$$

We can now recall the smn property as defined in [1, Definition 11].

Definition 6.3 (Asperti’s smn property). Given a concrete semantics $\phi$, a Blum complexity $\Phi$ and $m, n \geq 1$, the pair $\langle \phi, \Phi \rangle$ has the Asperti’s smn property if there exists a total computable function $s : \mathbb{N}^{m+1} \to \mathbb{N}$ such that $\forall e \in \mathbb{N}, \vec{x} \in \mathbb{N}^m$:

$$
\lambda \vec{y}. \phi_{s(e, \vec{x})}^{(n)}(\vec{x}, \vec{y}) = \phi_{\phi_i^{(n)}(e, \vec{x})}^{(m+n)}(\vec{x}, \vec{y}),
\lambda \vec{y}. \Phi_{s(e, \vec{x})}^{(n)}(\vec{x}, \vec{y}) \in \Theta(\lambda \vec{y}. \Phi_{\phi_i^{(n)}(e, \vec{x})}^{(m+n)}(\vec{x}, \vec{y})).$$

Informally, the smn property of Definition 6.3 states that the operation of fixing parameters preserves both the concrete semantics and the asymptotic complexity. Under these assumptions, we can show that Asperti’s complexity clique semantics satisfies our ssmn property.
Lemma 6.4. Let $(\pi, \equiv_\pi)$ be the complexity clique semantics of Example 2.6. If $(\pi, \equiv_\pi)$ satisfies Asperti’s ssrn and linear complexity composition properties then $(\pi, \equiv_\pi)$ is ssrn.

This result, together with the observation that the notion of fairness (Definition 3.2) instantiated to the complexity clique semantics is exactly that of [1, Definition 26], allows us to retrieve Kleene’s second recursion theorem and Rice’s theorem for complexity cliques in [1] as instances of our corresponding results given in Section 4.1.

6.2 Relation with Systems of Indices

As mentioned in Section 2, our definition of abstract semantics resembles the acceptable systems of indices [22, Definition II.5.1] or numberings [28, Exercise 2-10], firstly studied by Rogers [27]. In this section we discuss how such notions compare.

Definition 6.5 (System of indices [22, Definition II.5.1]). A system of indices is a family of functions $\{\psi^n\}_{n \in \mathbb{N}}$ such that each $\psi^n : \mathbb{N} \to C_n$ is a surjective map that associates program indices to $n$-ary partial recursive functions.

- $\{\psi^n\}_{n \in \mathbb{N}}$ has the parametrization (or ssrn) property if for every $m, n \in \mathbb{N}$ there is a total computable function $s : \mathbb{N}^{m+1} \to \mathbb{N}$ such that $\forall e \in \mathbb{N}, x \in \mathbb{N}^m$:

  $$\lambda y.\psi^{m+n}_e(x, y) = \psi^n_{s(e, x)}.$$

- $\{\psi^n\}_{n \in \mathbb{N}}$ has the enumeration property if for every $n \in \mathbb{N}$ there exists $u \in \mathbb{N}$ such that for all and $e \in \mathbb{N}$ and $y \in \mathbb{N}^n$:

  $$\psi^n_e = \lambda y.\psi^{n+1}_u(e, y).$$

Any standard Gödel numbering associating a program with the function it computes is a system of indices with the parametrization and enumeration properties. Moreover, exactly as we did in Example 2.3, any system of indices $\{\psi^n\}_{n \in \mathbb{N}}$ can be viewed as an abstract semantics $(\pi, =)$ with $\pi^n_a = \psi^n_a$. In this context, the enumeration and parametrization properties correspond to our fairness and ssrn conditions: fairness is exactly enumeration while ssrn follows from parametrization and enumeration, as discussed in Section 3 for the concrete semantics (cf. Example 2.3).

A system of indices is defined to be acceptable if it allows to get back and forth with a given system of indices satisfying the parametrization and enumeration properties through a pair of total computable functions.

Definition 6.6 (Acceptable system of indices [27, Definition 4]). Let $\{\varphi^n\}_{n \in \mathbb{N}}$ be a given system of indices with the parametrization and enumeration properties. A system of indices $\{\psi^n\}_{n \in \mathbb{N}}$ is acceptable if there exist two total computable functions $f, g : \mathbb{N} \to \mathbb{N}$ such that for all $a, n \in \mathbb{N}$:

$$\psi^n_a = \varphi^n_{f(a)} \quad \text{and} \quad \varphi^n_a = \psi^n_{g(a)}.$$  

As shown in [22, Proposition II.5.3], it turns out that a system of indices is acceptable if and only if it satisfies both enumeration and parametrization (a proof of this characterization was first given by Rogers [27, Section 2]). Consequently, an acceptable system of indices $\{\psi^n\}_{n \in \mathbb{N}}$ can be viewed as an abstract semantic $(\pi, =)$, where $\pi^n_a = \psi^n_a$, which, by this characterization of acceptability, is ssrn and fair, and therefore, by Theorem 4.1 it enjoys Kleene’s second recursion theorem, as already known from [22, Corollary II.5.4]. Under this
perspective, a generic abstract semantics according to Definition 2.2 can be viewed as a proper generalisation of the notion of acceptable system of indices, which merely encodes a change of program numbering and does not allow to take into account an actual abstraction of the concrete input/output behaviour of programs.

7 Conclusion and Future Work

This work generalises some traditional extensional results of computability theory, notably Kleene’s second recursion theorem and Rice’s theorem, to intensional abstract program semantics that include the complexity cliques investigated by Asperti [1]. Our approach was also inspired by Moyen and Simonsen [19] and relies on weakening the classical definition of extensional program property to a notion of partial extensionality w.r.t. abstract program semantics that satisfy some structural conditions. As an application, we strengthened and generalised a result by Müller-Olm and Seidl [20] proving that for affine programs with positive affine guards it is undecidable whether an affine relation holds at a given program point. Our results also shed further light on the claim that these undecidability results hinge on the Turing completeness of the underlying computational model, as argued in [19].

As future work, a natural question would be to investigate intensional extensions of Rice-Shapiro’s theorem that fit our framework based on abstract semantics. This appears to be a nontrivial challenge. Generalisations of Rice-Shapiro’s theorem have been given in [1, Section 5] and [19, Section 5.1]. A generalisation in the vein of the approach in [1] seems to be viable, but would require structural assumptions on abstract program semantics that, while natural in [1] whose focus is on complexity properties, would be artificial for abstract program semantics and would limit a general applicability. A further stimulating research topic is to apply our approach to abstract semantics as defined by abstract interpretation of programs, in particular for investigating the relationship with the notion of abstract extensionality studied by Bruni et al. [4]. Finally, while our framework relies on the assumption of an underlying Turing complete computational model, in a different direction, one could try to consider intensional properties for classes of programs indexing subrecursive functions (e.g., primitive recursive functions), whose extensional properties have been already studied (see, e.g., [10, 14]). Despite the fact that we suppose that our approach will fall short on these program classes, as one cannot expect to have a universal program inside the class itself or the validity of Kleene’s second recursion theorem, we think that this represents an intriguing research challenge.

References


