Weakest Safe Context Synthesis by Symbolic Game Semantics and Logical Abduction[*]

Aleksandar S. Dimovski
Faculty of Informatics, Mother Teresa University
Skopje, North Macedonia
aleksandar.dimovski@unt.edu.mk

ABSTRACT

Game semantics provides fully abstract (sound and complete) models for open program fragments with undefined, non-local, identifiers (e.g. library functions). This is achieved by using the "most general" models for undefined identifiers, i.e. the most generic context in which the program fragment will be inserted. Given a safety property as an assertion, we want to find the most permissive models of undefined identifiers, i.e. the weakest safe context, that are sufficient to ensure safety of the given program fragment.

To solve this problem, we present a novel approach that uses symbolic game semantic models of open program fragments and logical abduction. By using symbolic values instead of concrete ones for integers, we represent algorithmic game semantics of program fragments with unbounded integers as finite-state symbolic automata that are amenable for automatic reasoning. Then, we reduce the problem of inferring minimal assumptions on the behaviors of undefined identifiers in order to prove safety of the given program fragment to a logical abduction task. We evaluate our approach and demonstrate its practicality via several examples.

CCS CONCEPTS

• **Theory of computation** → **Logic**; Models of computation; • **Software and its engineering** → *Software notations and tools*;

KEYWORDS

Algorithmic Game Semantics, Symbolic Automata, Logical Abduction, Weakest Safe Contexts, Synthesis

ACM Reference Format:
Aleksandar S. Dimovski. 2025. Weakest Safe Context Synthesis by Symbolic Game Semantics and Logical Abduction. In *The 40th ACM/SIGAPP Symposium on Applied Computing (SAC '25), March 31-April 4, 2025, Catania, Italy.* ACM, New York, NY, USA, 8 pages. https://doi.org/10.1145/3672608.3707849

1 INTRODUCTION

Many programs today contain library methods whose source code is unavailable or too complex to be verified. To verify such open program fragments, we need to know the models (specifications) of these library methods. In this work, we want to find a weakest model (minimal assumptions) for each undefined identifier that ensures correctness (safety) of a program fragment calling these identifiers. We refer to this as a *weakest safe context synthesis problem.*

We study this problem in the game semantics setting. *Game semantics* [2, 22] is a technique for building models of program fragments that are *fully abstract*, i.e. sound and complete with respect to observational equivalence. Its algorithmic subarea [1, 8, 9, 19, 21, 23] aims to apply game semantics models to software verification by providing concrete automata-based representations for them. Here, we use a symbolic representation of algorithmic game semantics [10–12] where symbolic data values are used instead of concrete ones for representing integers. This allows us to obtain compact models of program fragments with infinite data types, such as integers, by using finite-state symbolic automata. Each *complete symbolic play* (accepting word) in the model represents one possible execution path of the program fragment, which is guarded by a conjunction of constraints on the symbols, known as *play condition*, which indicate under what conditions this symbolic play (execution path) is feasible. We say that a program fragment is *safe* if no program execution violates an assertion. Hence, we can use symbolic game semantics models to efficiently verify safety of program fragments. To ensure full abstractness, game semantics uses by default the "most general" (the "weakest possible") models for all free identifiers, thus providing a generic closure of a program fragment.

To address the weakest safe context synthesis problem, we use a counterexample guided synthesis loop with verification and logical inference technique called *abduction* [3, 4, 14] at its core. Abduction is a technique of backward logical reasoning for inferring missing hypothesis in a logical inference task. More specifically, suppose we have a premise P and a desired conclusion C for an inference (P and C are generated as constraints on symbols from the program's model). Abduction infers the simplest and most general explanation E such that (1) $P \wedge E \models C$ and (2) $P \wedge E \not\models$ false. The first condition states that the abduction solution E together with premise P should imply conclusion C, while the second condition states that the abduction solution E should not contradict premise P. The iterative procedure starts by setting the "most general" models for all free identifiers as initial candidate models. Given candidate models for free identifiers (context), we invoke a verifier to check if they ensure safety of all execution paths of the given program fragment. If no counterexample is found, the procedure terminates by reporting the current candidate models. Otherwise, the counterexamples found by the verifier are used by our abduction-based synthesis approach to infer new stronger candidate models for free identifiers. The

[*]Produces the permission block, and copyright information

procedure is then repeated on the updated candidate models. Since our procedure only analyzes implementations of terms (clients) but not of undefined identifiers (libraries), models inferred by our procedure represent specifications that must be checked against implementations of undefined identifiers.

We implemented our approach in the `Symbolic GameChecker` tool [10, 13], which is used for generating and verifying symbolic game models of program fragments. It also calls the `EXPLAIN` tool [4] for solving logical abduction tasks in the combination SMT theory of linear arithmetic and propositional logic. We show how our approach can efficiently synthesize weakest safe models of unknown identifiers in a variety of examples.

The main contributions of this paper are: (1) We define the weakest safe context synthesis problem in the game semantics setting; (2) We present a counterexample guided synthesis algorithm that uses verification and logical abduction for solving this problem; (3) We present an implementation of our approach and demonstrate its practicality via several examples.

1.1 Related Work

The closest to our approach for inferring weakest safe specifications with respect to assertion properties is the work by Albarghouthi et al. [3]. They propose a fully logic-based solution that leverages abduction. They assume existence of a verification tool that is capable of generating one or more counterexamples. For the reported counterexamples, they generate abduction problems and use their solutions to infer the required specifications. In this work, we define the weakest safe specifications synthesis in the settings of game semantics. This brings several distinctive features to our approach, such as: very precise models of second-order functional-imperative programs that take into account all possible contexts. In contrast, the work [3] considers state-based C imperative programs that have some undefined procedures, so the generated specifications represent relations over their arguments and returns.

Logical abduction has found a number of applications in program analysis and synthesis. An abduction algorithm for first-order SMT theories is described in [4] for computing a maximally simple and general solution. This form of SMT-based abduction has also been applied for loop invariant generation [6], and for error explanation and diagnosis of error reports generated by verification tools [5]. It has also been shown how abduction can be used in program synthesis either to infer concrete predicates to plug in for the missing '`if`' guards in partial programs [7] or to fill the missing holes in program sketches [15, 16] with arbitrary expressions [14].

Game semantics for full Idealized Algol has been defined by Abramsky and McCusker [2]. The first applications to software model checking were proposed by Ghica and McCusker [21] by introducing the algorithmic game semantics of IA_2 with finite data types. Several model checking tools based on this representation have been developed [1, 9, 23]. Symbolic game models for IA_2 with infinite data types have been introduced [10], and several their applications for verifying software product line [17, 18] have been proposed in [12, 13]. Moreover, Lin and Tzevelekos [24, 25] have defined symbolic execution game semantics in order to verify open higher-order programs [25] and their contextual equivalence [24].

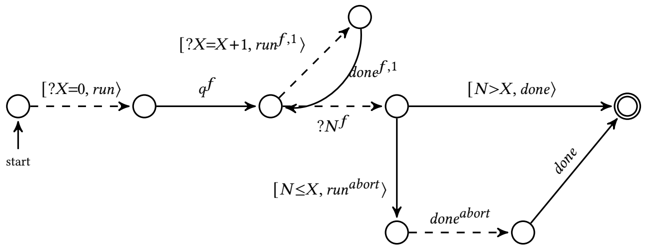

Figure 1: Symbolic game model of M in first iteration.

2 MOTIVATING EXAMPLE

Consider the following program term M:

$$\begin{array}{l} f : \mathsf{com}^{f,1} \to \mathsf{expint}^{f}, \mathit{abort} : \mathsf{com}^{abort} \\ \qquad\qquad\qquad\qquad \mathsf{new}_{\mathsf{int}}\ x := 0 \ \mathsf{in} \\ \qquad\qquad\qquad\qquad \mathsf{assert}\ (f(x := !x + 1) > !x) : \mathsf{com} \end{array}$$

where undefined identifiers are a call-by-name function f and command *abort*, which is executed when the assertion fails, and $!x$ is a de-referencing operation for reading the value in variable x.

First iteration. Our approach starts by setting the weakest possible context as the initial candidate models for free identifiers, i.e. the given term M can be placed in all possible contexts. This means that the undefined call-by-name function f, which may call its argument zero or more times, is represented by the most general model [10, 21]: $q^f \cdot (run^{f,1} \cdot done^{f,1})^* \cdot ?N^f$. The moves tagged with f represent effects of calling q^f and returning an integer via symbol N from function f, while moves tagged with $f, 1$ are effects of function f calling ($run^{f,1}$) and returning ($done^{f,1}$) its argument. Since the argument may be called an arbitrary number of times, we use the Kleene closure $*$. In effect, in the first iteration we obtain the game semantics model of M as shown in Fig. 1. The dashed edges indicate moves of the environment (**O**) and solid edges moves of the term (**P**). Accepting states are designated by an interior circle. Observe that the term communicates with its environment using non-local identifiers f and *abort*. So in the model will only be represented actions associated with f and *abort* as well as with the top-level type com of M. The symbol X is used to keep track of the current value of the local variable x. Note that symbol X occurs only in conditional part of moves. E.g., guarded move $[X=0, run\rangle$ initiates execution of M with *run* under condition that $X=0$.

Each time the term (**P**) calls f to be executed with the move q^f (or signals termination of the execution of its argument with the move $done^{f,1}$), the environment (**O**) either signals successful completion of f by returning a symbol $?N^f$ of type int or calls its first argument with $run^{f,1}$. When the function f has completed by returning an answer N, the term (**P**) can either terminate successfully with *done* if $(N > X)$; or it can execute *abort* and terminate if $(N \leq X)$. The shortest unsafe play in the game model of M is:

$$[?X = 0, run\rangle \cdot q^f \cdot ?N^f \cdot [N \leq X, run^{abort}\rangle \cdot done^{abort} \cdot done$$

After instantiating input symbols $?N$ with fresh symbolic names:

$$[X = 0, run\rangle \cdot q^f \cdot N^f \cdot [N \leq X, run^{abort}\rangle \cdot done^{abort} \cdot done$$

This instantiated symbolic play is feasible since its condition $(X = 0 \land N \leq X)$ is satisfiable for $(X = N = 0)$. Our goal is to synthesize

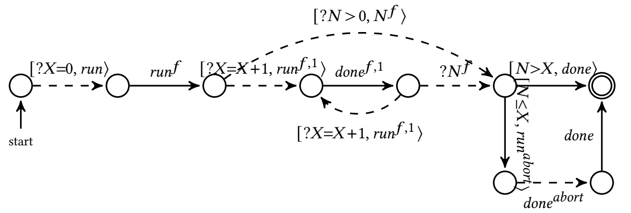

Figure 2: Symbolic game model of M in second iteration.

a model of f that ensures that the above unsafe play is infeasible. To do so, we use this play to construct the following abduction query: $X = 0 \wedge R_f(N) \implies (N > X)$. The left-hand side of the implication encodes the constraints of the play up to the *abort* moves, where the unknown predicate $R_f(N)$ encodes the constraint on the result N returned by f. The right-hand side of this implication encodes the postcondition ensuring that *abort* is not called (assertion is valid). The weakest solution is: $R_f(N) \equiv (N > 0)$. By projecting the above unsafe play on behaviours of f, we obtain the restricted safe play of f: $\underline{q^f \cdot [?N > 0, N^f\rangle}$. This play specifies that if f is a strict function that does not call its argument, then f must return a value greater than 0 in order M to be safe.

Second iteration. If we use the above restricted safe play for the case when f is a strict function, the candidate model of f becomes: $q^f \cdot \big([?N > 0, N^f\rangle + (run^{f,1} \cdot done^{f,1})^{+} \cdot ?N^f\big)$. We invoke the verifier again to check whether this candidate model for f is sufficient to prove safety of the inferred game semantics model of M, which is given in Fig. 2. The verifier discovers that this model is still too weak by reporting the shortest feasible (instantiated) unsafe play:

$$[X_0 = 0, run\rangle \cdot q^f \cdot [X_1 = X_0 + 1, run^{f,1}\rangle \cdot done^{f,1} \cdot N^f$$
$$\cdot [N \leq X_1, run^{abort}\rangle \cdot done^{abort} \cdot done$$

This play is feasible for $(X_0 = 0 \wedge X_1 = N = 1)$. Following the same procedure as above, we construct the abduction query: $X_0 = 0 \wedge X_1 = X_0 + 1 \wedge R_f(N) \implies (N > X_1)$. By solving it and projecting the above unsafe play on behaviours of f, we obtain the restricted safe play of f: $\underline{q^f \cdot run^{f,1} \cdot done^{f,1} \cdot [?N > 1, N^f\rangle}$. It specifies that if f calls its argument once, f must return a value greater than 1 in order M to be safe.

Next iterations. By searching the model of M up to a (user-defined) bound $d \in \mathbb{N}$, [1] we find that the weakest safe environment contains behaviours of f in which if f calls its argument k times ($k \leq d$) the returned value of f is greater than k. That is, the weakest safe model specification for undefined f is:

$$q^f \cdot \big(\Sigma_{k=0}^{d} (run^{f,1} \cdot done^{f,1})^k \cdot [?N > k, N^f\rangle\big)$$

3 SYMBOLIC GAME SEMANTICS

In this work, we consider Idealized Algol (IA), a well studied meta-language in semantics community [26] that enables functional (typed call-by-name λ-calculus) and imperative programming. For the aim of constructing automata-based representation of game semantics, we consider its second-order recursion-free fragment (IA$_2$ for short), where only first-order functions are allowed. Its types are: $D ::= \mathsf{int} \mid \mathsf{bool}$, $B ::= \mathsf{exp}D \mid \mathsf{com} \mid \mathsf{var}D$, $T ::= B \mid B \rightarrow T$, where D, B, and T stand for data types, base types, and first-order function types, respectively. The *syntax* is:

$$M ::= x \mid v \mid \mathsf{skip} \mid \mathsf{diverge} \mid M \,\mathsf{op}\, M \mid \mathsf{if}\, M \,\mathsf{then}\, M \,\mathsf{else}\, M \mid \mathsf{while}\, M \,\mathsf{do}\, M$$
$$\mid M; M \mid M := M \mid !M \mid \mathsf{new}_D\, x := v \,\mathsf{in}\, M \mid \mathsf{mkvar}_D MM \mid \lambda x.M \mid MM$$

where x ranges over a countable set of identifiers, and v ranges over constants of type D, which includes integers ($n \in \mathbb{Z}$) and booleans (tt, ff). *Well-typed terms* are given by typing judgements of the form $\Gamma \vdash M : T$, where $\Gamma = x_1 : T_1, \ldots, x_k : T_k$ is a type *context* consisting of a finite number of typed free identifiers. Typing rules are standard [2, 26], but the general application rule is broken up into linear application and contraction rules:

$$\text{LAPP}\,\frac{\Gamma \vdash M : B \rightarrow T \quad \Delta \vdash N : B}{\Gamma, \Delta \vdash MN : T} \qquad \text{CON}\,\frac{\Gamma, x_1 : T, x_2 : T \vdash M : T'}{\Gamma, x : T \vdash M[x/x_1, x/x_2] : T'}$$

where $M[N/x]$ denotes the capture-avoiding substitution of N for x in M. We use these two rules to have control over multiple occurrences of free identifiers in terms.

The *operational semantics* of IA can be found in [2, 26]. Given a closed term $\vdash M : \mathsf{com}$, we say that M *terminates* if M reduces to skip. We define a *program context* $C[-] : \mathsf{com}$ to be a term with zero or more holes $[-]$ in it, such that if $\Gamma \vdash M : T$ is a term of the same type as the hole then $C[M]$ is a well-typed closed term of type com. $\Gamma \vdash M : T$ is an *approximate* of $\Gamma \vdash N : T$, written $\Gamma \vdash M \sqsubseteq N$, iff for all contexts $C[-] : \mathsf{com}$, s.t. $\vdash C[M] : \mathsf{com}$ and $\vdash C[N] : \mathsf{com}$, if $C[M]$ terminates then $C[N]$ terminates.

We now give a brief overview of symbolic representation of the algorithmic game semantics for IA$_2$ [10]. Let *Sym* be a countable set of symbolic names, ranged over by upper case letters X, Y, Z. For any finite $W \subseteq Sym$, the function $new(W)$ returns a minimal symbolic name which does not occur in W, and sets $W := W \cup \{new(W)\}$. A minimal symbolic name not in W is the one which occurs earliest in a fixed enumeration of all possible symbolic names. Let $SExp^D$ be a set of symbolic expressions of type D generated by data values ($v \in D$), symbols ($X^D \in Sym$), *input symbols* ($?X^D$), and arithmetic-logic operations (op) of type D. We omit to write D when clear from the context. The input symbols $?X^D$ represent a mechanism for dynamically generating new symbolic names. More specifically, $?X$ creates a stream of fresh symbolic names, binding X to the next symbol from its stream whenever $?X$ is evaluated.

For each type T, we define alphabet $\mathcal{A}_{[\![T]\!]}$ as follows:

$$\mathcal{A}_{[\![\mathsf{int}]\!]} = \mathbb{Z}, \quad \mathcal{A}_{[\![\mathsf{bool}]\!]} = \{tt, ff\}, \quad \mathcal{A}_{[\![\mathsf{exp}D]\!]} = \{q\} \cup \mathcal{A}_{[\![D]\!]},$$
$$\mathcal{A}_{[\![\mathsf{com}]\!]} = \{run, done\}, \mathcal{A}_{[\![\mathsf{var}D]\!]} = \{write(a), read, ok, a \mid a \in \mathcal{A}_{[\![D]\!]}\}$$
$$\mathcal{A}_{[\![B_1^{\langle 1\rangle} \rightarrow \ldots \rightarrow B_k^{\langle k\rangle} \rightarrow B]\!]} = \sum_{1 \leq i \leq k} \mathcal{A}^{\langle i\rangle}_{[\![B_i]\!]} + \mathcal{A}_{[\![B]\!]}$$

Here, + denotes a disjoint union of alphabets. Function types are tagged by a superscript $\langle i\rangle$ to keep record from which type, i.e. which component of the disjoint union, each move comes from. The letters in the alphabet $\mathcal{A}_{[\![T]\!]}$ represent the *moves*, i.e. observable actions that a term of type T can perform. Each move is either a *question* (a demand for information) or an *answer* (a supply of information). For expressions in $\mathcal{A}_{[\![\mathsf{exp}D]\!]}$, there is a *question* move q to ask for the value of the expression, and values from $\mathcal{A}_{[\![D]\!]}$ to *answer* the question. For commands, there is a *question* move *run* to initiate a command, and an *answer* move *done* to signal successful

[1]Note that the model of M has a cycle, so we need to have a bounded exploration of it to guarantee that the search for unsafe plays will terminate in finite time.

termination of a command. For variables, there are *question* moves for writing to the variable, *write*(a), which are acknowledged by the *answer* move *ok*; and a *question* move *read* for reading from the variable, which is *answered* by a value from $\mathcal{A}_{[[D]]}$.

The guarded alphabet $\mathcal{A}^{gu}_{[[T]]}$, induced by $\mathcal{A}_{[[T]]}$, is a set of guarded letters $[b, \alpha\rangle$ where $b \in SExp^{\mathsf{bool}}$ is a boolean condition and α is a symbolic letter that represents a move from $\mathcal{A}_{[[T]]}$ in which all data values $v \in D$ are replaced with symbolic expressions from $SExp^D$. For example, $\mathcal{A}^{gu}_{[[\mathsf{exp}D]]} = \{[b, q\rangle, [b, \alpha\rangle \mid b \in SExp^{\mathsf{bool}}, \alpha \in SExp^D\}$ and $\mathcal{A}^{gu}_{[[\mathsf{com}]]} = \{[b, run\rangle, [b, done\rangle \mid b \in SExp^{\mathsf{bool}}\}$. A guarded letter $[b, \alpha\rangle$ is a symbolic letter α only if b evaluates to true otherwise it is the constant $\emptyset$. We will often write only α for the guarded letter $[tt, \alpha\rangle$. A word $[b_1, \alpha_1\rangle \cdot [b_2, \alpha_2\rangle \ldots [b_n, \alpha_n\rangle$ can be represented as a pair $[b, w\rangle$, where $b = b_1 \wedge b_2 \wedge \ldots \wedge b_n$ is a boolean condition and $w = \alpha_1 \cdot \alpha_2 \ldots \alpha_n$ is a word of symbolic letters.

For any (β-normal) IA$_2$ term, we define a (symbolic) regular-language which represents its game semantics, i.e. its set of complete symbolic plays. A *play* is a sequence of moves played by two players in turns: **P** (Player) which represents the term being modeled, and **O** (Opponent) which represents its context. Every complete symbolic play represents the observable effects of a completed execution path of the given term. It is given as a guarded word $[b, w\rangle$, where b is also called the *play condition.* Assumptions about a symbolic play to be feasible are recorded in its play condition. For infeasible plays, the play condition is unsatisfiable. For any term $\Gamma \vdash M : T$, we define semantics valuation function $[[\Gamma \vdash M : T]]u^\Gamma$, where u^Γ is an *environment* mapping free identifiers $x : T'$ in Γ to their semantics definitions (i.e. regular expressions over $\mathcal{A}^{gu\,\langle x\rangle}_{[[T']]}$). The regular expression $[[\Gamma \vdash M : T]]u^\Gamma$ is defined by recursion on the syntax over the guarded alphabet: $\mathcal{A}^{gu}_{[[\Gamma \vdash T]]} = \big(\sum_{x:T' \in \Gamma} \mathcal{A}^{gu\,\langle x\rangle}_{[[T']]}\big) + \mathcal{A}^{gu}_{[[T]]}$, where moves corresponding to types of free identifiers are tagged with their names to indicate the origin of moves. Hence, $[[\Gamma \vdash M : T]]u^\Gamma$ contains only moves associated with types of free identifiers from Γ and moves of the top-level type T of M.

Free identifiers $x : B_1^{\langle 1\rangle} \to \ldots B_k^{\langle k\rangle} \to B$ are represented by the so-called copy-cat regular expressions, denoted by $u^\Gamma(x)$ which contain all possible behaviours of terms of that type, thus providing the most general context for an open term (see [10] for definitions of all $u^\Gamma(x)$). For example, we have:

$$\begin{aligned} &u^\Gamma(x : \mathsf{exp}D) = q \cdot q^{\langle x\rangle} \cdot ?N^{\langle x\rangle} \cdot N \\ &u^\Gamma(x : \mathsf{exp}D_1^{\langle 1\rangle} \to \ldots \mathsf{exp}D_k^{\langle k\rangle} \to \mathsf{exp}D) = \\ &\quad q \cdot q^{\langle x\rangle} \cdot \big(\textstyle\sum_{1\leq i\leq k} q^{\langle x,i\rangle} \cdot q^{\langle i\rangle} \cdot ?Z_i^{\langle i\rangle} \cdot Z_i^{\langle x,i\rangle}\big)^* \cdot ?X^{\langle x\rangle} \cdot X \end{aligned}$$

When a call-by-name non-local function x with k arguments is called, it may evaluate any of its arguments, zero or more times, in an arbitrary order (hence, the Kleene closure *) and then it returns any allowable answer from its result type. Thus, $u^\Gamma(x)$ describes the generic behavior of a sequential function x. Game semantics of a free identifier $x : B_1^{\langle 1\rangle} \to \ldots B_k^{\langle k\rangle} \to B$ is:

$$[[\Gamma, x : B_1^{\langle x,1\rangle} \to \ldots B_k^{\langle x,k\rangle} \to B^{\langle x\rangle} \vdash x : B_1^{\langle 1\rangle} \to \ldots B_k^{\langle k\rangle} \to B]]u^\Gamma = u^\Gamma(x)$$

The representation of constants is standard: $[[\Gamma \vdash v : \mathsf{exp}D]]u^\Gamma = q \cdot v$, $[[\Gamma \vdash \mathsf{skip} : \mathsf{com}]]u^\Gamma = run \cdot done$, $[[\Gamma \vdash \mathsf{diverge} : \mathsf{com}]]u^\Gamma = \emptyset$. E.g., a constant v is modeled by a play where the initial question q that asks for the value of this expression is answered by constant v.

The representations of some language constructs "c" are given in Table 1. Observe that letter conditions different than tt occur only in plays corresponding to "if" and "while" constructs. In the case of "if" construct, when the value of the first argument given by the symbol Z^{bool} is true then its second argument is run, otherwise if $\neg Z$ is true then its third argument is run. In the case of "while" construct, the guard of while produces the symbol Z^{bool} as answer, so the model is an iteration of plays consisting of the guard producing Z as true concatenated with complete plays of the body, followed by one single play of the guard producing $\neg Z$ as true. A composite term $\mathsf{c}(M_1, \ldots, M_k)$ built out of a language construct "c" and subterms $M_1, \ldots, M_k$ is interpreted by composing the regular expressions for $M_1, \ldots, M_k$ and the regular expression for "c". E.g., $[[\Gamma \vdash M := N : \mathsf{com}]] = [[\Gamma \vdash M : \mathsf{var}D^{\langle 1\rangle}]] \circ [[\Gamma \vdash N : \mathsf{exp}D^{\langle 2\rangle}]] \circ [[:= : \mathsf{var}D^{\langle 1\rangle} \times \mathsf{exp}D^{\langle 2\rangle} \to \mathsf{com}]]$. Composition of regular expressions ($\circ$) is defined as "parallel composition followed by hiding" in CSP style [2]. The parallel composition is matching (synchronizing) of the moves in the shared types, whereas hiding is deleting of all moves from the shared types [10]. The $\mathsf{cell}_v^{\langle x\rangle}$ regular expression in Table 1 is used to impose the good variable behaviour on a local variable x introduced using $\mathsf{new}_D\ x := v\ \mathsf{in}\ M$. Note that X is a symbol used to track the current value of x. The $\mathsf{cell}_v^{\langle x\rangle}$ behaves as a storage cell and plays the most recently written value in x in response to *read*, or if no value has been written yet then answers *read* with the initial value v. The model $[[\Gamma \vdash \mathsf{new}_D\ x := v\ \mathsf{in}\ M]]$ is obtained by constraining $[[\Gamma, x : \mathsf{varD} \vdash M]]$ to only those plays where x exhibits good variable behaviour described by $\mathsf{cell}_v^{\langle x\rangle}$, and then by hiding all moves of x since local variables are not visible outside of their binding scope [10]. The linear application is defined as $[[\Gamma, \Delta \vdash M N : T]] = [[\Delta \vdash N : B^{\langle 1\rangle}]] \circ [[\Gamma \vdash M : B^{\langle 1\rangle} \to T]]$. The contraction $[[\Gamma, x : T^{\langle x\rangle} \vdash M[x/x_1, x/x_2] : T']]$ is obtained from $[[\Gamma, x_1 : T^{\langle x_1\rangle}, x_2 : T^{\langle x_2\rangle} \vdash M : T']]$, such that the moves of x_1 and x_2 are de-tagged so that they represent actions of x.

3.1 Weakest Safe Environments

The following formal results are proved before [10].

Proposition 3.1 ([10]). *For any IA$_2$ term, the set $[[\Gamma \vdash M : T]]u^\Gamma$ is a (symbolic) regular-language without infinite summations defined over its effective finite alphabet. Moreover, a finite-state symbolic automata $\mathcal{A}[[\Gamma \vdash M : T]]u^\Gamma$ recognizing it is effectively constructible.*

Given a complete *symbolic* play $[b, w\rangle \in \mathcal{L}([[\Gamma \vdash M : T]]u^\Gamma)$, we now show how to generate the corresponding complete *concrete* plays, in which concrete values are used instead of symbols. Recall that an input symbol $?Z$ creates a stream of fresh symbolic names for each instantiation of $?Z$. Thus, whenever $?Z$ is met in a symbolic play, the mechanism for fresh symbol generation is used to dynamically instantiate it with a new fresh symbolic name from its stream. This symbolic name binds all occurrences of Z that follow in the play until a new $?Z$ is met which overrides the previous symbolic name with the next symbolic name taken from its stream. For example, the unsafe play corresponding to program path of the

$$
\begin{array}{l}
[\![\mathsf{op} : \mathsf{exp}D_1^{\langle 1\rangle} \times \mathsf{exp}D_2^{\langle 2\rangle} \to \mathsf{exp}D]\!] = q \cdot q^{\langle 1\rangle}\cdot ?Z^{\langle 1\rangle} \cdot q^{\langle 2\rangle}\cdot ?Z'^{\langle 2\rangle} \cdot (Z\,\mathsf{op}\,Z') \\
[\![; : \mathsf{com}^{\langle 1\rangle} \times \mathsf{com}^{\langle 2\rangle} \to \mathsf{com}]\!] = run \cdot run^{\langle 1\rangle} \cdot done^{\langle 1\rangle} \cdot run^{\langle 2\rangle} \cdot done^{\langle 2\rangle} \cdot done \\
[\![\mathsf{if} : \mathsf{expbool}^{\langle 1\rangle} \times \mathsf{com}^{\langle 2\rangle} \times \mathsf{com}^{\langle 3\rangle} \to \mathsf{com}]\!] = [tt, run\rangle \cdot [tt, q^{\langle 1\rangle}\rangle \cdot [tt, ?Z^{\langle 1\rangle}\rangle \cdot \\
\qquad\qquad \big([Z, run^{\langle 2\rangle}\rangle \cdot [tt, done^{\langle 2\rangle}\rangle + [\neg Z, run^{\langle 3\rangle}\rangle \cdot [tt, done^{\langle 3\rangle}\rangle\big) \cdot [tt, done\rangle \\
[\![\mathsf{while} : \mathsf{expbool}^{\langle 1\rangle} \times \mathsf{com}^{\langle 2\rangle} \to \mathsf{com}]\!] = [tt, run\rangle \cdot [tt, q^{\langle 1\rangle}\rangle \cdot [tt, ?Z^{\langle 1\rangle}\rangle \cdot \big([Z, run^{\langle 2\rangle}\rangle \cdot [tt, done^{\langle 2\rangle}\rangle \cdot [tt, q^{\langle 1\rangle}\rangle \cdot [tt, ?Z^{\langle 1\rangle}\rangle\big)^* \cdot [\neg Z, done\rangle \\
[\![:= : \mathsf{var}D^{\langle 1\rangle} \times \mathsf{exp}D^{\langle 2\rangle} \to \mathsf{com}]\!] = run \cdot q^{\langle 2\rangle}\cdot ?Z^{\langle 2\rangle} \cdot write(Z)^{\langle 1\rangle} \cdot ok^{\langle 1\rangle} \cdot done \\
[\![! : \mathsf{var}D^{\langle 1\rangle} \to \mathsf{exp}D]\!] = q \cdot read^{\langle 1\rangle}\cdot ?Z^{\langle 1\rangle} \cdot Z \\
\mathsf{cell}_v^{\langle x\rangle} = ([?X{=}v, read^{\langle x\rangle}\rangle \cdot X^{\langle x\rangle})^* \cdot \big(write(?X)^{\langle x\rangle} \cdot ok^{\langle x\rangle} \cdot (read^{\langle x\rangle} \cdot X^{\langle x\rangle})^*\big)^*
\end{array}
$$

Table 1: Symbolic representations of some language constructs

motivating term M from Section 2 where f calls its argument once is: $[?X = 0, run\rangle \cdot q^f \cdot [?X = X+1, run^{f,1}\rangle \cdot done^{f,1} ?N^f \cdot [N \leq X, run^{abort}\rangle \cdot done^{abort} \cdot done$. After instantiating its input symbols $?X$ and $?N$, we obtain the play: $[X_0 = 0, run\rangle \cdot q^f \cdot [X_1 = X_0+1, run^{f,1}\rangle \cdot done^{f,1} \cdot N^f \cdot [N \leq X_1, run^{abort}\rangle \cdot done^{abort} \cdot done$, where X_0, X_1, and N are different symbolic names generated by streaming symbols $?X$ and $?N$. We write $[\![b, w\rangle\!\rangle$ for the symbolic play obtained after instantiating all input symbols in the play $[b, w\rangle$.

Let *Eval* be the set of evaluations, i.e. the set of total functions from W to $\mathcal{A}_{[\![\mathsf{int}]\!]} \cup \mathcal{A}_{[\![\mathsf{bool}]\!]}$. We use ρ to range over *Eval*. Given a word of symbolic letters w, let $\rho(w)$ be a word where every symbolic name X is replaced by the concrete value $\rho(X)$. Given a guarded (instantiated) word $[\![b, w\rangle\!\rangle$, define $\rho([\![b, w\rangle\!\rangle) = \rho(w)$ if $\rho(b) =$ true; otherwise $\rho([\![b, w\rangle\!\rangle) = \emptyset$ if $\rho(b) =$ false. The concretization of a symbolic regular-language is defined as: $\gamma\,(\mathcal{L}(R)) = \{\rho[\![b, w\rangle\!\rangle \mid [b, w\rangle \in \mathcal{L}(R), \rho \in Eval\}$. Let $([\![\Gamma \vdash M : T]\!]u^\Gamma)^{CR}$ denotes the set of all complete concrete plays obtained as in [21], where concrete values and infinite summations in regular expressions are used.

Proposition 3.2 ([10]). *For any IA_2 term, $\gamma\,([\![\Gamma \vdash M : T]\!]u^\Gamma) = ([\![\Gamma \vdash M : T]\!]u^\Gamma)^{CR}$.*

Suppose that there is a special free identifier *abort* of type com. We define an assertion: $\mathsf{assert} {=} \lambda b{:}\mathsf{expbool}.\mathsf{if}\ b\ \mathsf{then\ skip\ else}\ abort$. We assume that there is only one assertion in term $\Gamma \vdash M$ and $Ass(M)$ denotes its boolean expression. $\Gamma \vdash M$ is *safe* iff $\Gamma \vdash M[\mathsf{skip}/abort] \sqsubseteq M[\mathsf{diverge}/abort]$; otherwise it is *unsafe*. A play is *safe* if it contains no moves from $\mathcal{A}^{\langle abort\rangle}_{[\![\mathsf{com}]\!]}$; otherwise it is *unsafe*.

Proposition 3.3 ([10]). *$\Gamma \vdash M : T$ is safe iff all plays in $[\![\Gamma \vdash M : T]\!]u^\Gamma$ are safe.*

We use u to denote an *environment* mapping free identifiers $x : T \in \Gamma$ to symbolic regular-expressions defined over the alphabet $\mathcal{A}^{gu\,\langle x\rangle}_{[\![T]\!]}$. We write $[\![\Gamma \vdash M{:}T]\!]u$ for the game semantics of $\Gamma \vdash M{:}T$, where every $x \in \Gamma$ behaves according to its specification $u(x)$.

Definition 3.4 (Safe Environment). We say that an environment u is *safe* for a term $\Gamma \vdash M{:}T$ if $[\![\Gamma \vdash M{:}T]\!]u$ contains only safe plays.

While safe environments are sufficient for ensuring safety, they may be stronger than necessary. For example, $u(x) = \emptyset$ for all $x \in \Gamma$ is always safe environment. Since we want to synthesize *weakest safe environment*, we define partial order on environments. We write $u > u'$, and say that u is weaker than u', if we have: (i) $\forall x \in \Gamma. \gamma(u(x)) \supseteq \gamma(u'(x))$; (ii) $\exists x \in \Gamma. \gamma(u(x)) \neq \gamma(u'(x))$.

Definition 3.5 (Weakest Safe Environment). We define the *weakest safe environment* u^* for $\Gamma \vdash M{:}T$, if (*i*) $[\![\Gamma \vdash M{:}T]\!]u^*$ is safe; (*ii*) for all u, such that $u > u^*$, $[\![\Gamma \vdash M{:}T]\!]u$ is not safe.

That is, u^* is the weakest safe environment for $\Gamma \vdash M{:}T$ if we cannot weaken any $u^*(x)$ while preserving safety of $[\![\Gamma \vdash M{:}T]\!]u^*$.

4 ABDUCTION

Standard abduction facilitates the inference of a single unknown predicate $R(\mathbf{x})$ defined over a vector of variables $\mathbf{x}$, known as *abducible*, from a formula $R(\mathbf{x}) \wedge \chi \implies C$. That is, the standard abduction finds a formula ϕ over variables $\mathbf{x}$, such that (1) $\phi \wedge \chi \not\models$ false; and (2) $\phi \wedge \chi \models C$. A solution ϕ to the standard abduction problem is an interpretation of $R(\mathbf{x})$ that consistently strengthens the left-hand side of the implication in order to make the implication logically valid. Every solvable abduction problem has a unique logically weakest solution (up to logical equivalence). The `Abduce`$(\chi, C, \mathbf{x})$, which computes the logically weakest solution of "$R(\mathbf{x}) \wedge \chi \implies C$", is implemented in the `EXPLAIN` tool [4].

We now define (linear) multi-abduction that allows the inference of multiple unknown predicates defined over different variables.

Definition 4.1 (Linear Multi-Abduction). Given $\bigwedge_{i=1}^n R_i(\mathbf{x}_i) \wedge \chi \implies C$, the (linear) multi-abduction finds a mapping Δ from each R_i to a formula ϕ_i over variables $\mathbf{x}_i$, such that (1) $\bigwedge_{i=1}^n \phi_i \wedge \chi \not\models$ false; and (2) $\bigwedge_{i=1}^n \phi_i \wedge \chi \models C$.

A solution Δ is a logically weakest solution if no ϕ_i can be weakened. That is, for any ϕ' that is logically weaker than $\Delta(R_i) = \phi_i$, the mapping $\Delta[R_i \mapsto \phi']$ is not a solution. In contrast to standard abduction problems, (linear) multi-abduction problems do not have unique weakest solutions (see [3] for details). The procedure `MultiAbduce`$(\chi, C, \mathcal{A})$ that takes as input formulas χ and C as well as a set $\mathcal{A}$ of abducibles of the form $R_i(\mathbf{x}_i)$, and computes the logically weakest solution Δ of the multi-abduction problem is described in [3]. In short, it first calls `Abduce`$(\chi, C, vars(\mathcal{A}))$, where $vars(\mathcal{A})$ is the set of all variables in $\mathcal{A}$, in order to solve the following standard abduction problem: $R^*(vars(\mathcal{A})) \wedge \chi \implies C$. That is, all abducibles $R_1, \ldots R_n$ in $\mathcal{A}$ are combined into one abducible R^*. Thus, a solution ψ to R^* contains all solutions to $R_1, \ldots R_n$. Therefore, the solution ψ to the above problem is decomposed into a set of formulas $\phi_1, \ldots, \phi_n$ defined over $\mathbf{x}_1, \ldots, \mathbf{x}_n$ respectively, such that $\wedge_i \phi_i \implies \psi$ and $\phi_1, \ldots, \phi_n$ are as weak as possible. The decomposition is performed by first finding an initial solution Δ that for each R_i has an interpretation of the form $\mathbf{x}_i = \mathbf{m}_i$ ($\mathbf{m}_i$ is a vector of constants), which is obtained by finding a model M of $(\chi \wedge \psi)$ and setting $\mathbf{x}_i = M(\mathbf{x}_i)$. To weaken the solution for any R_i, we fix the solutions for all other R_j's and solve $R_i(\mathbf{x}_i) \wedge \wedge_{i \neq j} \Delta(R_j) \implies \psi$ by calling Abduce$(\wedge_{i \neq j} \Delta(R_j), \psi, \mathbf{x}_i)$. That is, we use the current solution for all other R_j's and infer the weakest R_i that still implies ψ. We refer to [3] for more details on this procedure.

Proposition 4.2 ([4]). *If the abduction problem is solvable, then* MultiAbduce$(\chi, C, \mathcal{A})$ *terminates with a weakest solution containing a fewest number of variables.*

5 SYNTHESIZING WEAKEST SAFE ENVIRONMENTS

A term $\Gamma \vdash M : T$ is called *simple* (linear) if it can be typed without applying the contraction rule to any free identifier from Γ. We can only use the linear application rule, where every linear free identifier can be used exactly once in the derivation of M. In this section, we first present our synthesis algorithm MinSafeEnv for simple terms, and then we generalize it for arbitrary terms.

Algorithm. Algorithm 1 describes our procedure, MinSafeEnv, for synthesizing a weakest safe environment. It takes as inputs a simple term $\Gamma \vdash M : T$ and a bound $d \in \mathbb{N}$, and returns as output the weakest safe environment u^* for $\Gamma \vdash M : T$. Given a candidate environment u, we construct a symbolic automaton $[[\Gamma \vdash M : T]]u$ and invoke a verifier (model checker) at line 3 to check it and thus establish if the current candidate environment ensures safety of all execution paths of M. Initially, u is set to the most general environment u^Γ (line 1). If the verifier finds no unsafe feasible play, then MinSafeEnv terminates reporting the current candidate environment u as the weakest safe (line 4). Otherwise, the algorithm goes through a refinement loop where the found unsafe feasible plays are used to generate stronger candidate environments u' until $[[\Gamma \vdash M : T]]u'$ is proven safe. In particular, the counterexample c reported by the verifier is used to encode a multi-abduction query at line 5. By solving the query at line 6, we generate a new updated environment u at line 7 and continue our refinement loop.

Verify. The model checker is aiming to find a feasible unsafe play in the symbolic model $[[\Gamma \vdash M : T]]u$ with respect to the given assertion in M. However, the presence of while commands and free identifiers of function types introduces infinite plays (traces), i.e. a cycle, in $[[\Gamma \vdash M : T]]u$. Hence, the verification may not terminate for safe terms when there are longer and longer infeasible unsafe plays, or the model may contain an infinite number of feasible unsafe plays thus preventing MinSafeEnv to terminate. To address this termination problem, we use a bound $d \in \mathbb{N}$ to define a bounded exploration of the symbolic model. In particular, we place a numeric bound d (instead of Kleene closure $*$) on the number of iterations of the while-loop and on the number of times an undefined function can call its arguments.

$$[[\mathsf{while} : \mathsf{exp}^{\langle 1 \rangle} \times \mathsf{com}^{\langle 2 \rangle} \to \mathsf{com}]] = \\ \quad run \cdot q^{\langle 1 \rangle} \cdot ?Z^{\langle 1 \rangle} \cdot \Sigma_{l=0}^{d} \big([Z, run^{\langle 2 \rangle} \rangle \cdot done^{\langle 2 \rangle} \cdot q^{\langle 1 \rangle} \cdot ?Z^{\langle 1 \rangle} \big)^l \cdot [\neg Z, done \rangle$$

$$u(x : B_1^{\langle 1 \rangle} \to \dots B_k^{\langle k \rangle} \to \mathsf{exp} D) = q \cdot q^{\langle x \rangle} \cdot \Sigma_{l=0}^{d} \big(\textstyle\sum_{1 \leq i \leq k} R_{B_i}^{\langle x,i \rangle} \big)^l \cdot ?N^{\langle x \rangle} \cdot N$$

As a result of the above definitions, we obtain a bounded model of a term $\Gamma \vdash M : T$, denoted $[[\Gamma \vdash M : T]]^d u$, which is verified by a call Verify$([[\Gamma \vdash M : T]]^d u)$.

Algorithm 1: **MinSafeEnv**$(\Gamma \vdash M : T, d)$

```
1 $u = u^\Gamma$;
2 while true do
3     $c$ = Verify($[[\Gamma \vdash M:T]]^d u$) ;
4     if $c$ == null then return $u$;
5     $\chi, \mathcal{A}$ = Encode($c$);
6     $\Delta$ = MultiAbduce($\chi, Ass(M), \mathcal{A}$) ;
7     $u$ = Update($u, c, \Delta$);
```

The procedure Verify$([[\Gamma \vdash M : T]]^d u)$ uses model checking algorithms [10, 20] to search for feasible unsafe plays in the symbolic automaton $[[\Gamma \vdash M : T]]^d u$. This boils down to checking if an 'unsafe' state q' with transition $q \xrightarrow{[b, run^{abort} \rangle} q'$ is reachable in the model. We use the breadth-first search (BFS) algorithm to visit all states in the model that are reachable from the initial state and check whether one of them is 'unsafe'. This way, BFS finds the shortest unsafe play (run^{abort} occurs earliest in it). When an unsafe play is found, we call an external SMT solver (e.g. Yices) to check consistency (i.e. satisfiability modulo the background theories) of its play condition. If the condition is consistent, the corresponding counterexample is reported. Otherwise, the BFS continues with the search.

Using Propositions 3.1, 3.2, and 3.3 and the bounded exploration of models, we can prove:

Proposition 5.1. Verify$([[\Gamma \vdash M : T]]^d u)$ *is correct and terminates modulo the exploration bound d and decidable theories in SMT.*

Encode. We now present a technique that encodes a counterexample c into a single multi-abduction query. The procedure Encode takes as input an instantiated play c, and constructs a formula χ and a set of abducibles $\mathcal{A}$ as follows. The play c is of the form: $[b_1, \alpha_1 \rangle \dots [b_k, \alpha_k \rangle \cdot [b_{k+1} \wedge \neg Ass(M), run^{abort} \rangle \cdot done^{abort} \dots [b_n, \alpha_n \rangle$. The formula χ includes all boolean conditions that occur in moves up to run^{abort} (but excluding $\neg Ass(M)$). That is, $\chi = \bigwedge_{i=1}^{k+1} b_i$. For all symbols of the form $N^{\langle tag \rangle}$ that occur in symbolic parts of moves, that is in α_i for $1 \leq i \leq k$, we introduce a predicate $R_{tag}(N)$ that represents the unknown constraint of the component (free identifier from Γ or M) corresponding to $\langle tag \rangle$. We define a set $\mathcal{A}$ of all such obtained predicates $R_{tag}(N)$. The constructed multi-abduction query is: $(\bigwedge_{R_{tag}(N) \in \mathcal{A}} R_{tag}(N)) \wedge (\bigwedge_{i=1}^{k+1} b_i) \implies Ass(M)$.

Example 5.2. Consider the term: $x : \mathsf{expint}^x, y : \mathsf{expint}^y, abort : \mathsf{com}^{abort} \vdash \mathsf{assert}(x + y \geq 1) : \mathsf{com}$. The model of this term for initial environment u^Γ is: $run \cdot q^x \cdot ?X^x \cdot q^y \cdot ?Y^y \cdot ([X + Y \geq 1, done \rangle + [X + Y < 1, run^{abort} \rangle \cdot done^{abort} \cdot done)$. The instantiated shortest unsafe play is: $run \cdot q^x \cdot X^x \cdot q^y \cdot Y^y \cdot [X + Y < 1, run^{abort} \rangle \cdot done^{abort} \cdot done$, and the multi-abduction query is: $R_x(X) \wedge R_y(Y) \implies (X + Y \geq 1)$. This is an example of a multi-abduction query that has more than one weakest solution. Two solutions are $R_x(X) \equiv (X \geq 0)$, $R_y(Y) \equiv (Y \geq 1)$; and $R_x(X) \equiv (X \geq 1)$, $R_y(Y) \equiv (Y \geq 0)$. The procedure MultiAbduce will return one of them.

Update. Given a (uninstantiated) counterexample c and a solution Δ to the multi-abduction problem inferred from it, we now show how the current candidate environment u is updated by calling procedure Update(u, c, Δ). For each free identifier $x : T \in \Gamma$, we generate a restricted play c_x to x by gathering all moves in c that are associated with x. For any move of the form $[b, ?N^{\langle x,j \rangle}\rangle$ in c_x, where we use superscript $\langle x, 0 \rangle$ to denote the tag $\langle x \rangle$, such that the predicate $R_{x,j}(N)$ exists in Δ, we replace that move by $[b \wedge \Delta(R_{x,j}(N)), N^{x,j}\rangle$, where the first occurrence of N in $\Delta(R_{x,j}(N))$ is replaced with $?N$. The obtained restricted safe play is denoted $c_x[\Delta]$. Then we update $u(x)$ with $(u(x) \backslash c_x) \cup c_x[\Delta]$. If the multi-abduction query does not have a solution, we update $u(x)$ with $(u(x) \backslash c_x)$ since there are no feasible safe paths in M when free identifiers behave as in c.

Example 5.3. Reconsider the term from Example 5.2. The counterexample c is: $run \cdot q^x \cdot ?X^x \cdot q^y \cdot ?Y^y \cdot [X + Y < 1, run^{abort}\rangle \cdot done^{abort} \cdot done$, and the solution Δ is $R_x(X) \equiv (X \geq 0)$, $R_y(Y) \equiv (Y \geq 1)$. The restricted plays are $c_x = q^x \cdot ?X^x$ and $c_y = q^y \cdot ?Y^y$. Let u' be $\texttt{Update}(u^\Gamma, c, \Delta)$. Then, $u'(x) = q^x \cdot [?X \geq 0, X^x\rangle$ and $u'(y) = q^y \cdot [?Y \geq 1, Y^y\rangle$.

Example 5.4. Reconsider the term M from Section 2. In the second iteration, the counterexample c is: $[X_0 = 0, run\rangle \cdot q^f \cdot [X_1 = X_0 + 1, run^{f,1}\rangle \cdot done^{f,1} \cdot N^f \cdot [N \leq X_1, run^{abort}\rangle \cdot done^{abort} \cdot done$, the solution Δ is $R_f(N) \equiv (N > 1)$, and the current environment is $u(f) = q^f \cdot \big([?N > 0, N^f\rangle + (run^{f,1} \cdot done^{f,1})^+ \cdot ?N^f\big)$. The restricted play is $c_f = q^f \cdot run^{f,1} \cdot done^{f,1} \cdot ?N^f$, and the updated environment is $u'(f) = q^f \cdot \big([?N > 0, N^f\rangle + run^{f,1} \cdot done^{f,1} \cdot [?N > 1, N^f\rangle + \Sigma_{k \geq 2}(run^{f,1} \cdot done^{f,1})^k \cdot ?N^f\big)$.

Generalization. Let $\Gamma \vdash M : T$ be an arbitrary term in which identifiers from Γ may occur multiple times in M. Let $\Gamma' \vdash M' : T$ be derived without using the contraction rule for Γ', such that $\Gamma \vdash M : T$ is obtained from it by applying the contraction rule for identifiers from Γ. First, we compute $u' = \texttt{MinSafeEnv}(\Gamma' \vdash M' : T, d)$. Then, we define a new environment u whose domain is the set of identifiers from Γ. For each identifier $x \in \Gamma$ that has multiple occurrences $x_1, \ldots, x_k \in \Gamma'$ in one execution path of M, we define one behaviour (play) of $u(x)$ to be concatenation of behaviours (plays) of $u'(x_1), \ldots, u'(x_k)$, where the first play $u'(x_1)$ describes the behaviour of first call to x in M, the second play $u'(x_2)$ describes second call to x in M, etc.

Example 5.5. Consider the term

$$\begin{array}{ll} n : \mathsf{expint}^{\langle n \rangle}, abort : \mathsf{com}^{abort} \vdash & \mathsf{new_{int}}\, x := 0 \text{ in} \\ & \text{while } (!x < 3) \text{ do } x := !x + n; \\ & assert(!x \leq 6) : \mathsf{com} \end{array}$$

where n is an undefined expression. Its symbolic model is: $[?X = 0, run\rangle \cdot \big([X < 3, q^{\langle n \rangle}\rangle \cdot [?X = X + ?N, N^{\langle n \rangle}\rangle\big)^* \cdot \big([X \geq 3 \wedge X \leq 6, done\rangle + [X \geq 3 \wedge X > 6, run^{abort}\rangle \cdot done^{abort} \cdot done\big)$. Assume the exploration bound is $d = 3$. The shortest feasible unsafe play is: $[X_0 = 0, run\rangle \cdot [X_0 < 3, q^{\langle n \rangle}\rangle \cdot [X_1 = X_0 + N_0, N_0^{\langle n \rangle}\rangle \cdot [X_1 \geq 3 \wedge X_1 > 6, run^{abort}\rangle \cdot done^{abort} \cdot done$. The corresponding abduction query is: $R_{N_0}(N_0) \wedge (X_0 = 0 \wedge X_0 < 3 \wedge X_1 = X_0 + N_0 \wedge X_1 \geq 3) \implies (X_1 \leq 6)$. The solution is: $R_{N_0}(N_0) \equiv (3 \leq N_0 \leq 6)$. The corresponding safe play for n when n is called once (i.e. body of while is executed once) is: $q^{\langle n \rangle} \cdot [3 \leq ?N \leq 6, N^{\langle n \rangle}\rangle$. Similarly, we obtain safe plays for n when n is called twice (i.e. body of while is executed twice) to be: $q^{\langle n \rangle} \cdot [0 \leq ?N \leq 2, N^{\langle n \rangle}\rangle \cdot q^{\langle n \rangle} \cdot [1 \leq ?N \leq 4, N^{\langle n \rangle}\rangle$, as well as when n is called thrice (i.e. body of while is executed thrice) to be:

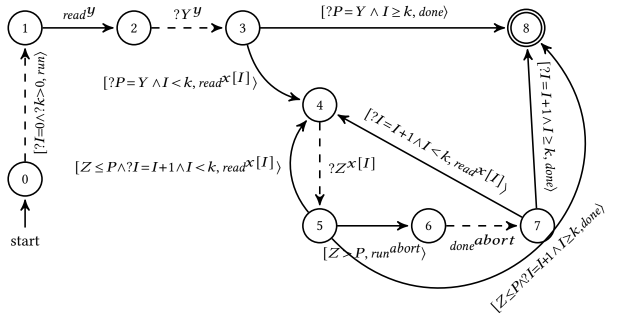
start
$[?I{=}0 \wedge ?k{>}0, run\rangle$
$read^y$
$?Y^y$
$[?P{=}Y \wedge I \geq k, done\rangle$
$[?P{=}Y \wedge I < k, read^{x[I]}\rangle$
$[Z \leq P \wedge ?I{=}I{+}1 \wedge I < k, read^{x[I]}\rangle$
$?Z^{x[I]}$
$[?I{=}I{+}1 \wedge I < k, read^{x[I]}\rangle$
$[?I{=}I{+}1 \wedge I \geq k, done\rangle$
$[Z > P, run^{abort}\rangle$
$done^{abort}$
$[Z \leq P \wedge ?I{=}I{+}1 \wedge I \geq k, done\rangle$

Figure 3: The game model for the linear search term.

$q^{\langle n \rangle} \cdot [0 \leq ?N \leq 1, N^{\langle n \rangle}\rangle \cdot q^{\langle n \rangle} \cdot [0 \leq ?N \leq 1, N^{\langle n \rangle}\rangle \cdot q^{\langle n \rangle} \cdot [1 \leq ?N \leq 4, N^{\langle n \rangle}\rangle$. That is, $u(n) = (q^{\langle n \rangle} \cdot [3 \leq ?N \leq 6, N^{\langle n \rangle}\rangle) + (q^{\langle n \rangle} \cdot [0 \leq ?N \leq 2, N^{\langle n \rangle}\rangle \cdot q^{\langle n \rangle} \cdot [1 \leq ?N \leq 4, N^{\langle n \rangle}\rangle) + (q^{\langle n \rangle} \cdot [0 \leq ?N \leq 1, N^{\langle n \rangle}\rangle \cdot q^{\langle n \rangle} \cdot [0 \leq ?N \leq 1, N^{\langle n \rangle}\rangle \cdot q^{\langle n \rangle} \cdot [1 \leq ?N \leq 4, N^{\langle n \rangle}\rangle)$.

Correctness. By Proposition 5.1 and Proposition 4.2, we can prove:

THEOREM 5.6. *The procedure* $\texttt{MinSafeEnv}(\Gamma \vdash M : T, d)$ *terminates and is correct modulo the exploration bound d and decidable theories in SMT and abducible solvers.*

6 IMPLEMENTATION

We have extended the `Symbolic GameChecker` tool [10] to develop a prototype implementation of our approach for generating weakest safe environments of open terms. `Symbolic GameChecker` converts any IA$_2$ term into a symbolic automaton representing its game semantics, and then performs bounded exploration of the automaton for feasible unsafe traces (plays). It calls an external SMT solver, `Yices` (http://yices.csl.sri.com) to determine satisfiability of play conditions, and the `EXPLAIN` tool [4] to solve the SMT-based multi-abduction query. The tool is implemented in Java and can be found at: `https://aleksdimovski.github.io/symbolicgc.html`. (version for weakest safe contexts). We now illustrate our tool.

Consider the following version of the linear search algorithm:

$$\begin{array}{l} x[k] : \mathsf{varint}^{x[-]},\ y : \mathsf{expint}^y,\ \mathsf{abort} : \mathsf{com}^{abort} \vdash \\ \qquad \mathsf{new_{int}}\ i := 0 \text{ in } \mathsf{new_{int}}\ p := !y \text{ in} \\ \qquad \text{while } (!i < k) \text{ do } \{ \\ \qquad\qquad \text{assert } (!x[i] \leq !p); \\ \qquad\qquad i := i + 1; \} \ : \mathsf{com} \end{array}$$

The meta variable $k > 0$ represents the size of array x, which will be replaced by several different values. In the above term, first the input expression y is copied into the local variable p. Then the non-local array x is examined to check if all its elements are less or equal to the value stored in p. If this is false, the assertion fails.

The symbolic model for this term is given in Fig. 3. The array $x[k]$ is given a symbolic representation [10] where the array size k and the index of the array elements represent symbols. We use symbols I and P to track the current values of local variables i and p. The symbol I is used to represent the index of an array element. If the value Z read from the environment $\mathbf{O}$ for some array element $X[I]$, where $0 \leq I < k$, is less or equal to the value Y read from the environment $\mathbf{O}$ for undefined expression y, then the assertion

Bench.	d	Max-M	Fin-M	Iter	Time	Time-Abd
Motivate M	3	29	13	3	3.07 s	2.70 s
Motivate M	5	41	17	5	5.20 s	4.61 s
Motivate M	7	53	21	7	8.53 s	7.55 s
$\texttt{Linear}_3$	3	19	11	3	6.51 s	6.10 s
$\texttt{Linear}_5$	5	26	13	5	66.4 s	64.5 s
$\texttt{Linear}_6$	6	41	14	6	203.9 s	200.2 s

Table 2: Performance results.

is valid. Otherwise, the assertion fails and an unsafe behaviour is exercised. To successfully analyze an array with size $k > 0$, we need to have a bounded exploration with $d = k$. For example, when $k = 1$, the feasible unsafe play is when $x[0]$ (i.e. symbol Z) is greater than y (i.e. symbols Y and P). We obtain a multi-abduction query, and the solution to the initial abduction query is $(Z \leq Y)$. Then we call `Yices` to find a model of $(Z \leq Y)$, thus obtaining $(Z = 0, Y = 0)$. By fixing $(Y = 0)$, we solve abduction query $R_Z(Z) \wedge (Y = 0) \implies (Z \leq Y)$, thus obtaining $(Z \leq 0)$. By fixing $(Z \leq 0)$, we solve abduction query $R_Y(Y) \wedge (Z \leq 0) \implies (Z \leq Y)$, thus obtaining $(Y \geq 0)$. Finally, we report the following safe models for y: $read^y \cdot [?Y \geq 0, Y^y\rangle$ and for $x[0]$: $read^{x[0]} \cdot [?Z \leq 0, Z^{x[0]}\rangle$.

In Table 2, we represent performances for analyzing the motivating example M from Section 2 (denoted `Motivate M`) as well as the linear search term for different values of $k = 3, 5$, and 6 (denoted $\texttt{Linear}_k$). We list the exploration bound d, the size of the biggest game model generated during analysis `Max-M`, the size of the final game model in the last iteration `Fin-M`, the number of iterations performed `Iter`, the total time taken to analyze a term `Time`, and the time taken by `EXPLAIN` to solve abduction queries `Time-Abd` (both in seconds). We ran our tool on a 64-bit Intel®Core^{TM} i7-1165G7 CPU@2.80GHz, VM Ubuntu 18.04 LTS, with 8 GB memory. We can observe that as the exploration bound d increases, the number of iterations and running times increase as well. In the case of $\texttt{Linear}_k$, we observe the exponential growth of time mostly due to the fact that `EXPLAIN` needs more time for solving abduction queries. In fact, note that `EXPLAIN` takes more than 90% of the total analysis time. For bigger values of k, we obtain longer counterexamples with more constraints, which leads to more complex abduction queries.

7 CONCLUSION

In this work, we have proposed a novel approach for inferring weakest safe model specifications of unknown identifiers in the settings of game semantics and we have shown the effectiveness of this approach on some practical examples.

REFERENCES

[1] Samson Abramsky, Dan R. Ghica, Andrzej S. Murawski, and C.-H. Luke Ong. 2004. Applying Game Semantics to Compositional Software Modeling and Verification. In *10th International Conference, TACAS 2004, Proceedings (LNCS)*, Vol. 2988. Springer, Barcelona, 421–435. https://doi.org/10.1007/978-3-540-24730-2_32

[2] Samson Abramsky and Guy McCusker. 1996. Linearity, Sharing and State: a fully abstract game semantics for Idealized Algol with active expressions. *Electr. Notes Theor. Comput. Sci.* 3 (1996), 2–14. https://doi.org/10.1016/S1571-0661(05)80398-6

[3] Aws Albarghouthi, Isil Dillig, and Arie Gurfinkel. 2016. Maximal specification synthesis. In *Proceedings of the 43rd Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL 2016.* ACM, St. Petersburg, 789–801. https://doi.org/10.1145/2837614.2837628

[4] Isil Dillig and Thomas Dillig. 2013. Explain: A Tool for Performing Abductive Inference. In *Computer Aided Verification - 25th International Conference, CAV 2013. Proceedings (LNCS)*, Vol. 8044. Springer, Saint Petersburg, 684–689. https://doi.org/10.1007/978-3-642-39799-8_46

[5] Isil Dillig, Thomas Dillig, and Alex Aiken. 2012. Automated error diagnosis using abductive inference. In *ACM SIGPLAN Conference PLDI '12.* ACM, Edinburgh, 181–192. https://doi.org/10.1145/2254064.2254087

[6] Isil Dillig, Thomas Dillig, Boyang Li, and Kenneth L. McMillan. 2013. Inductive invariant generation via abductive inference. In *Proceedings of the 2013 ACM SIGPLAN International Conference on OOPSLA 2013.* ACM, Indianapolis, 443–456. https://doi.org/10.1145/2509136.2509511

[7] Thomas Dillig, Isil Dillig, and Swarat Chaudhuri. 2014. Optimal Guard Synthesis for Memory Safety. In *Computer Aided Verification - 26th International Conference, CAV 2014. Proceedings (LNCS)*, Vol. 8559. Springer, Vienna, 491–507. https://doi.org/10.1007/978-3-319-08867-9_32

[8] Aleksandar Dimovski, Dan R. Ghica, and Ranko Lazic. 2005. Data-Abstraction Refinement: A Game Semantic Approach. In *12th International Symposium on Static Analysis, SAS '05 (LNCS)*, Vol. 3672. Springer, London, 102–117. https://doi.org/10.1007/11547662_9

[9] Aleksandar Dimovski and Ranko Lazic. 2007. Compositional software verification based on game semantics and process algebra. *STTT* 9, 1 (2007), 37–51. https://doi.org/10.1007/s10009-006-0005-y

[10] Aleksandar S. Dimovski. 2014. Program verification using symbolic game semantics. *Theor. Comput. Sci.* 560 (2014), 364–379. https://doi.org/10.1016/j.tcs.2014.01.016

[11] Aleksandar S. Dimovski. 2016. Symbolic Game Semantics for Model Checking Program Families. In *Model Checking Software - 23nd International Symposium, SPIN 2016, Proceedings (LNCS)*, Vol. 9641. Springer, Eindhoven, 19–37.

[12] Aleksandar S. Dimovski. 2017. Probabilistic Analysis Based On Symbolic Game Semantics and Model Counting. In *Proceedings Eighth International Symposium on Games, Automata, Logics and Formal Verification, GandALF 2017 (EPTCS)*, Vol. 256. OPA, Rome, 1–15. https://doi.org/10.4204/EPTCS.256.1

[13] Aleksandar S. Dimovski. 2018. Verifying annotated program families using symbolic game semantics. *Theor. Comput. Sci.* 706 (2018), 35–53. https://doi.org/10.1016/j.tcs.2017.09.029

[14] Aleksandar S. Dimovski. 2023. Generalized Program Sketching by Abstract Interpretation and Logical Abduction. In *Static Analysis - 30th International Symposium, SAS 2023, Proceedings (LNCS)*, Vol. 14284. Springer, Lisbon, 212–230. https://doi.org/10.1007/978-3-031-44245-2_11

[15] Aleksandar S. Dimovski. 2023. Quantitative program sketching using decision tree-based lifted analysis. *J. Comput. Lang.* 75 (2023), 101206. https://doi.org/10.1016/j.cola.2023.101206

[16] Aleksandar S. Dimovski, Sven Apel, and Axel Legay. 2021. Program Sketching using Lifted Analysis for Numerical Program Families. In *NASA Formal Methods - 13th International Symposium, NFM 2021, Proceedings (LNCS)*, Vol. 12673. Springer, Virtual Event, 95–112. https://doi.org/10.1007/978-3-030-76384-8_7

[17] Aleksandar S. Dimovski, Sven Apel, and Axel Legay. 2022. Several lifted abstract domains for static analysis of numerical program families. *Sci. Comput. Program.* 213 (2022), 102725. https://doi.org/10.1016/j.scico.2021.102725

[18] Aleksandar S. Dimovski, Claus Brabrand, and Andrzej Wasowski. 2019. Finding suitable variability abstractions for lifted analysis. *Formal Aspects Comput.* 31, 2 (2019), 231–259. https://doi.org/10.1007/s00165-019-00479-y

[19] Aleksandar S. Dimovski, Dan R. Ghica, and Ranko Lazic. 2006. A Counterexample-Guided Refinement Tool for Open Procedural Programs. In *Model Checking Software, 13th International SPIN Workshop, 2006, Proceedings (LNCS)*, Vol. 3925. Springer, Vienna, 288–292. https://doi.org/10.1007/11691617_17

[20] Aleksandar S. Dimovski, Axel Legay, and Andrzej Wasowski. 2020. Generalized abstraction-refinement for game-based CTL lifted model checking. *Theor. Comput. Sci.* 837 (2020), 181–206. https://doi.org/10.1016/j.tcs.2020.06.011

[21] Dan R. Ghica and Guy McCusker. 2003. The regular-language semantics of second-order Idealized ALGOL. *Theor. Comput. Sci.* 309, 1-3 (2003), 469–502. https://doi.org/10.1016/S0304-3975(03)00315-3

[22] J. M. E. Hyland and C.-H. Luke Ong. 2000. On Full Abstraction for PCF: I, II, and III. *Inf. Comput.* 163, 2 (2000), 285–408. https://doi.org/10.1006/inco.2000.2917

[23] Stefan Kiefer, Andrzej S. Murawski, Joël Ouaknine, Björn Wachter, and James Worrell. 2013. Algorithmic probabilistic game semantics - Playing games with automata. *Formal Methods in System Design* 43, 2 (2013), 285–312. https://doi.org/10.1007/s10703-012-0173-1

[24] Vasileios Koutavas, Yu-Yang Lin, and Nikos Tzevelekos. 2022. From Bounded Checking to Verification of Equivalence via Symbolic Up-to Techniques. In *28th International Conference, TACAS 2022 (LNCS)*, Vol. 13244. Springer, Munich, 178–195. https://doi.org/10.1007/978-3-030-99527-0_10

[25] Yu-Yang Lin and Nikos Tzevelekos. 2020. Symbolic Execution Game Semantics. In *5th International Conference on FSCD 2020 (LIPIcs)*, Vol. 167. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, Paris, 27:1–27:24. https://doi.org/10.4230/LIPIcs.FSCD.2020.27

[26] John C. Reynolds. 1997. *The essence of Algol.* In: O'Hearn, P.W., Tennent, R.D. (eds), *Algol-like languages*, Birkhaüser, Basel.