On Satisfiability of Nominal Subtyping with Variance

Aleksandr Misonizhnik
JetBrains Research, Saint Petersburg State University, Russia
misonijnik@gmail.com

Dmitry Mordvinov
JetBrains Research, Saint Petersburg State University, Russia
dmitry.mordvinov@jetbrains.com

Abstract

Nominal type systems with variance, the core of the subtyping relation in object-oriented programming languages like Java, C# and Scala, have been extensively studied by Kennedy and Pierce: they have shown the undecidability of the subtyping between ground types and proposed the decidable fragments of such type systems. However, modular verification of object-oriented code may require reasoning about the relations of open types. In this paper, we formalize and investigate the satisfiability problem for nominal subtyping with variance. We define the problem in the context of first-order logic. We show that although the non-expansive ground nominal subtyping with variance is decidable, its satisfiability problem is undecidable. Our proof uses a remarkably small fragment of the type system. In fact, we demonstrate that even for the non-expansive class tables with only nullary and unary covariant and invariant type constructors, the satisfiability of quantifier-free conjunctions of positive subtyping atoms is undecidable. We discuss this result in detail, as well as show one decidable fragment and a scheme for obtaining other decidable fragments.

2012 ACM Subject Classification Theory of computation → Type theory; Theory of computation → Logic and verification; Software and its engineering → Object oriented languages; Software and its engineering → Automated static analysis; Software and its engineering → Polymorphism; Software and its engineering → Inheritance

Keywords and phrases nominal type systems, structural subtyping, first-order logic, decidability, software verification

Digital Object Identifier 10.4230/LIPIcs.ECOOP.2019.7

Acknowledgements We thank Sophia Drossopoulou, Dmitry Boulytchev and anonymous reviewers for their insight and comments that significantly improved the manuscript.

1 Introduction

Although object-oriented languages like Java, C# or Scala are ubiquitous in modern programming, the investigation of their type systems is still in progress. For example, Java type checking has only recently been shown to be undecidable [7]. One important feature of such languages is that types can appear at runtime and influence program execution (in contrast to the ML programming language family, Haskell, etc., in which type information is erased during compilation). For example, consider the following snippet:

```
1  IDictionary<TKey, TValue> MakeCache<TKey, TValue>()
2  {
3      if (typeof(TKey) == typeof(int))
4          return new SortedDictionary<TKey, TValue>();
5      if (typeof(TKey) == typeof(string))
6          return new Dictionary<TKey, TValue>();
7      throw new InvalidOperationException();
8  }
```

© Aleksandr Misonizhnik and Dmitry Mordvinov;
licensed under Creative Commons License CC-BY

The runtime behaviour of `MakeCache` depends on the value of the formal type parameter `TKey`: the method returns a fresh instance of `SortedDictionary` indexed by integer keys, Dictionary for string keys, or throws an exception otherwise. But what if we would like to *statically* check that `MakeCache` is used correctly, i.e. it does not throw an exception? Unfortunately, the type parameter constraint system of .NET does not allow to specify this restriction, as it does not implement disjunctive constraints.

Our study is motivated by the problem of verification of .NET programs. Modern deductive software verifiers are capable of checking non-trivial properties of programs by proving that under certain *preconditions*, executing a function guarantees certain *postconditions*. For instance, for the example above, the correctness property could be specified as a logical precondition for the method. In Spec#-style [1], this could be expressed as

```
1  IDictionary<TKey, TValue> MakeCache<TKey, TValue>()
2      requires typeof(TKey) == typeof(int) || typeof(TKey) == typeof(string)
3  {
4      ...
5  }
```

In more complex cases, arbitrary boolean combinations (including negation) of subtyping constraints or even quantified specification can be useful. Unfortunately, neither compilers nor modern verifiers (including Spec#) are capable of statically checking such properties because their *assertion language* cannot express properties of types.

Consider another snippet:

```
1  interface ICloneable<out T> { T Clone(); }
2  class Base { }
3  sealed class Derived : Base, ICloneable<Derived>
4  {
5      public Derived Clone() { return new Derived(); }
6  }
7  void F<T>(Base arg1, Derived arg2)
8  {
9      if (arg2 is ICloneable<T> && arg1 is T)
10     {
11         var clone = ((Derived) arg1).Clone();
12         ...
13     }
14 }
```

Using our fictitious extension of Spec# specification language, the specification of the function F would include the following clause:

$$\textit{requires}\ \texttt{Derived} <: \texttt{ICloneable<T>} \wedge \textit{typeof}(arg1) <: \texttt{Base} \wedge \\ \textit{typeof}(arg1) <: \texttt{T} \wedge \textit{typeof}(arg1) <: \texttt{Derived}$$

One might think that the last conjunct, ie $\textit{typeof}(arg1) <: \textit{Derived}$, can be omitted, and that the type cast expression (line 11) never fails: as `Derived` is sealed, the actual type of `arg2` can only be `Derived`. In contrast, the type of `arg1` can be `Base` or a subtype of `Base` type; as `Derived` implements only `ICloneable<Derived>`, `T` may only be `Derived`, therefore, if line 11 is reachable, then `arg1` is `Derived`. However, this reasoning is wrong: as `ICloneable` is a covariant constructor and `Base` is a supertype of `Derived`, `ICloneable<Base>` is a supertype of `Derived`, and line 11 can be reached with `T` = `Base`. Is it possible to determine the satisfiability of such violations automatically?

It would be legitimate to omit the last conjunct, ie $typeof(arg1) <:$ `Derived`, and also omit the cast on line 11, if we knew that

$$\forall \texttt{T}, \texttt{T'}.\texttt{Derived} <: \texttt{ICloneable<T>} \wedge \texttt{T'} <: \texttt{Base} \wedge \texttt{T'} <: \texttt{T} \Rightarrow \texttt{T'} <: \texttt{Derived}$$

Or, equivalently, if we could prove the unsatisfiability of the following assertion:

$$\phi \stackrel{\text{def}}{=} \exists \texttt{T}, \texttt{T'}.\texttt{Derived} <: \texttt{ICloneable<T>} \wedge \texttt{T'} <: \texttt{Base} \wedge \texttt{T'} <: \texttt{T} \wedge \texttt{T'} \not<: \texttt{Derived}$$

In the particular case, ϕ is satisfiable. Namely, take $\texttt{T} = \texttt{Base} = \texttt{T}'$, and the optimizations proposed above would be unsound.

Our examples have demonstrated the relevance of subtype satisfiability in program specification as well as for code optimization. The next question is the design of decision procedures for such questions. However, it turns out that this question in undecidable!

In section 2, we formalize nominal subtyping with variance following definitions and propositions from [8]. We focus our attention on *non-expansive inheritance* fragment [19], which was shown to be decidable [8] and has been adopted in the .NET Framework [4].

In section 3, we formalize and investigate the *first-order satisfiability problem* for nominal subtyping with variance; to the best of our knowledge, this is the first attempt at its detailed examination. Unlike the subtyping problem, which answers the question "Is one type a subtype of another type?", the satisfiability problem answers the question "Are there types that meet the required constraints?". In fact, satisfiability involves the subtyping problem, in the case where the constraints do not contain type variables. This means that the decidability of the subtyping problem does not imply the decidability of the satisfiability problem. We present a number of quantifier-free first-order formulas, demonstrating that the satisfiability problem is tricky even for non-expansive inheritance.

In section 4, we reinforce this by proving the undecidability of this problem, which is the primary contribution of the paper. Our proof uses a remarkably small fragment of the type system; in fact, we show that the subtyping satisfiability problem is undecidable for *non-expansive* class tables (1) without contravariant constructors, (2) with only nullary and unary constructors, (3) for quantifier-free conjunctions of subtyping atoms without negation.

Afterwards, in section 5, we demonstrate one practical decidable fragment which we call *semiground*, and prove its decidability. Using the intuitions from the proof, we provide a scheme to obtain other decidable fragments.

Our results may give rise to the construction of an effective decision procedure for the quantifier-free case. We formulate the problem in terms of satisfiability modulo theory, aiming at the implementation of its decision procedure in SMT-solvers [2, 3]. The support of SMT-reasoning for nominal type theory is useful for software verification tools, static analysers and generation techniques of automated tests, and exploits and patches for object-oriented languages. The first-order theory of nominal types may be used for type specifications in the assertion languages of deductive verifiers, while its decision procedure may be used for solving path conditions of different program branches. It may also be useful in compilers: for example, in improving dead code elimination techniques by proving the unsatisfiability of the path condition for a certain code fragment.

2 The type system[1]

In this section, we formalize nominal subtyping with variance. Types can either be type variables, denoted by lowercase letters, or constructed types C<$\overline{T}$>, where C is an n-ary type constructor and $\overline{T}$ is a vector of arguments of length n. We omit angle brackets if a type constructor is unary: for example, we write $ABCx$ instead of A<B<C<x>>>. C^rT denotes the type $C \dots CT$, where C occurs r times. We refer to such types as *chains* of r type constructors C ending with T. *Ground* types are types that contain no variables. *Open* types are types that are not ground.

The subtyping relation of nominal type systems with variance is defined via an explicit specification of the names of supertypes and *variances* of type parameters. Such specifications are usually expressed as class tables.

▶ **Definition 2.1.** *A* class table *is a finite set of entries of the form*

$$C\texttt{<}\overline{vx}\texttt{>} \texttt{<::}\ T_1, \dots, T_n$$

Each entry contains a unique declaration of a type constructor and a finite list of constructed types, which are nominal supertypes for all types constructed by this type constructor. The left-hand side of an entry contains the name for constructor C and its formal type parameters x_i with variances *v_i. v_i may be either $\circ$ (invariant) or $+$ (covariant) or $-$ (contravariant). The right-hand side contains a finite list of types T_i obtained from constructors declared in other entries, constructor C, and parameters x_i. To simplify the notation, we omit $\circ$ in class table declarations.*

```
System.Object        <::
System.ValueType     <::  System.Object
IEnumerable          <::  System.Object
IEnumerable<+x>      <::  IEnumerable
ICollection          <::  IEnumerable
ICollection<x>       <::  IEnumerable<x>
Pair<x, y>           <::  System.ValueType
IDictionary          <::  ICollection
IDictionary<x, y>    <::  ICollection<Pair<x, y>>
Dictionary<x, y>     <::  IDictionary<x, y>
                          IDictionary
```

```
interface IEnumerable {}
interface IEnumerable<out x>:
    IEnumerable{}
interface ICollection:
    IEnumerable {}
interface ICollection<x>:
    IEnumerable<x> {}
struct Pair<x, y> {}
interface IDictionary:
    ICollection {}
interface IDictionary<x, y>:
    ICollection<Pair<x, y>> {}
class Dictionary<x, y>:
    IDictionary<x, y>,
    IDictionary {}
```

Listing 1 The declaration of the class `Dictionary` and its class table.

▶ **Example 2.2.** Listing 1 demonstrates a simplified fragment of the class table for the `Dictionary<x, y>` standard container in .NET.

[1] In this section, we follow definitions and propositions from [8].

The i-th formal type parameter of a type constructor C and its variance are denoted by $C\#i$ and $var(C\#i)$ correspondingly: $C\#i \stackrel{\text{def}}{=} x_i$ and $var(C\#i) \stackrel{\text{def}}{=} v_i$. For example, `IEnumerable`$\#1 = x$ and $var($`IEnumerable`$\#1) = +$.

▶ **Definition 2.3.** *A* substitution *is a total mapping from type variables to types, which is an identity everywhere except for a finite set of variables which are mapped into constructed types. The* domain *of a substitution subst is a set of variables mapped to types, and the* range *is the image of the domain. We write substitutions as*

$$[x_1 \mapsto T_1; \ldots, x_n \mapsto T_n] \text{ and } [\overline{x} \mapsto \overline{T}],$$

where $x_1, \ldots, x_n$ are type variables from the domain of substitution and $T_1, \ldots, T_n$ are their images. We write the application of substitution $[\overline{x} \mapsto \overline{T}]$ to type T as $[\overline{x} \mapsto \overline{U}]T$. If the domain of a substitution consists of one type variable x, we omit the brackets:

$$x \mapsto T$$

We use $<::$ not only as a class table separator, but also to denote the binary relation of nominal subtyping. If a class table has an entry $C\texttt{<}\overline{x}\texttt{>} <:: T_i$, then $C\texttt{<}\overline{U}\texttt{>} <:: [\overline{x} \mapsto \overline{U}]T_i$. We write the transitive closure of $<::$ as $<::^+$.

We require class tables to define only acyclic $<::^+$ relations and to be well-formed with respect to the variance of formal type parameters, i.e. variant type parameters should appear only in positions of the same polarity. Furthermore, we require that the supertypes do not overlap: if $C\texttt{<}\overline{x}\texttt{>} <:: T$ and $C\texttt{<}\overline{x}\texttt{>} <:: U$, then for all $\overline{V}$ if $[\overline{x} \mapsto \overline{V}]T = [\overline{x} \mapsto \overline{V}]U$, then $T = U$.

Finally, we can define the subtyping relation.

▶ **Definition 2.4.** *The ground subtyping relation $<:$ is defined by the set of the following rules:*

$$\frac{T <: U}{T <:_+ U} \qquad \frac{}{T <:_\circ T} \qquad \frac{U <: T}{T <:_- U}$$

$$(Var)\ \frac{\text{for each } i \quad T_i <:_{var(C\#i)} U_i}{C\texttt{<}\overline{T}\texttt{>} <: C\texttt{<}\overline{U}\texttt{>}}$$

$$(Super)\ \frac{C\texttt{<}\overline{x}\texttt{>} <:: V \quad [\overline{x} \mapsto \overline{T}]V <: D\texttt{<}\overline{U}\texttt{>}}{C\texttt{<}\overline{T}\texttt{>} <: D\texttt{<}\overline{U}\texttt{>}}\ C \neq D$$

Due to the multiple instantiation inheritance, the Super rule can be applied non-deterministically.

▶ **Example 2.5.** The following sequence of rules should be applied to deduce that `Dictionary<`T, U`>` is a subtype of `IEnumerable<System.Object>`:

	`Dictionary<`T, U`>` $<:$ `IEnumerable<System.Object>`	
$\longrightarrow$	`IDictionary<`T, U`>` $<:$ `IEnumerable<System.Object>`	*by* Super
$\longrightarrow$	`ICollection<Pair<`T, U`>>` $<:$ `IEnumerable<System.Object>`	*by* Super
$\longrightarrow$	`IEnumerable<Pair<`T, U`>>` $<:$ `IEnumerable<System.Object>`	*by* Super
$\longrightarrow$	`Pair<`T, U`>` $<:$ `System.Object`	*by* Var
$\longrightarrow$	`System.ValueType` $<:$ `System.Object`	*by* Super
$\longrightarrow$	`System.Object` $<:$ `System.Object`	*by* Super
$\longrightarrow$		*by* Var

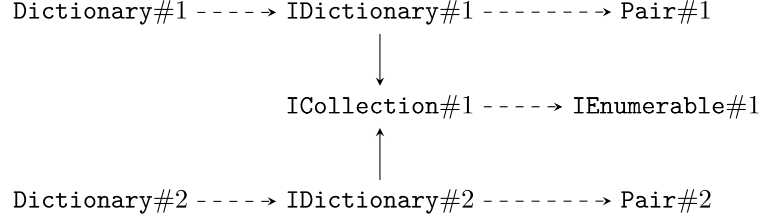

Figure 1 Type parameter dependency graph for Listing 1.

Ground subtyping is a partial order on a set of ground types. The ground subtyping relation has been shown to be undecidable in [8]. In the following, we introduce a notion of *non-expansive inheritance.*

▶ **Definition 2.6.** *A* type parameter dependency graph *is a directed graph with vertices that correspond to formal type parameters and two kinds of edges: for each class table entry* $C\texttt{<}\overline{x}\texttt{>} <:: T$ *and for each subterm* $D\texttt{<}\overline{U}\texttt{>}$ *of* T*,*

- *if* $U_j = x_i$*, then there is a* non-expansive *edge from* $C\#i$ *to* $D\#j$ *(depicted via a dotted arrow);*
- *if* x_i *is a proper subterm of* U_j*, then there is an* expansive *edge from* $C\#i$ *to* $D\#j$ *(depicted via a solid arrow).*

For instance, in Example 2.2, `IDictionary<x,y>` $<::$ `ICollection<Pair<x,y>>` introduces a non-expansive edge from `IDictonary#1` to `Pair#1` and an expansive edge to `ICollection#1`. The complete type parameter dependency graph is shown in Figure 1.

▶ **Definition 2.7.** *A class table is* expansive *if its type parameter dependency graph has a cycle with at least one expansive edge.*

▶ **Example 2.8.** The class table Listing 1 is non-expansive, as its type parameter dependency graph Figure 1 does not contain cycles.

▶ **Proposition 2.9.** *The non-expansive ground subtyping relation is decidable.*

▶ **Proposition 2.10.** *Ground subtyping is decidable if a class table has no contravariant constructors.*

Both results have been shown in [8].

3 The `SUBTYPE-SAT` problem

In this section, we formalize the subtype satisfiability problem and show some interesting examples.

In what follows, fix a class table CT. Let $\mathcal{C}$ be a set of constructors in CT. Let $\Sigma = (\mathcal{C}, \{<:\})$ be a first-order signature with equality. Function symbols are identified with constructors in CT. For convenience, the applications of a function symbol C to arguments $\overline{U}$ are still written as $C\texttt{<}\overline{U}\texttt{>}$, or just CU in the unary case. $<:$ is a binary predicate symbol written in infix style. For convenience, $\neg(T <: U)$ and $\neg(T = U)$ are written as $T \not<: U$ and $T \neq U$.

Let $I_{<:}$ be a Σ-structure with the domain $|I_{<:}|$ of all ground types defined by CT, interpreting $<:$ as the subtyping relation from Definition 2.4. Let $\mathcal{T}^{CT}_{<:}$ be a complete first-order Σ-theory of structure $I_{<:}$, i.e. the set of all first-order Σ-sentences which are satisfied by $I_{<:}$ (we assume a usual definition of satisfaction of ϕ by I, denoted $I \vDash \phi$). Given a Σ-sentence ϕ, we say that ϕ is *satisfiable modulo* $\mathcal{T}^{CT}_{<:}$, iff $I \vDash \phi$.

Let $\mathcal{V}$ be a countable set of variables. An *assignment of free variables* is any mapping $v : \mathcal{V} \to |I_{<:}|$ of variables to ground types. Note that free variable assignments are substitutions with a ground range. A formula with free variables ϕ is called satisfiable (valid, unsatisfiable) if $I, v \vDash \phi$ for some (any, no) free variable assignment v. We abbreviate the satisfiability in (I, v) and validity of ϕ with $v \vDash^{CT}_{<:} \phi$ and $\vDash^{CT}_{<:} \phi$ correspondingly.

▶ **Problem 3.1** (`SUBTYPE-SAT` problem)**.** *Given a class table CT and a formula ϕ over Σ, find such a free variable assignment v that $v \vDash^{CT}_{<:} \phi$ or prove its absence.*

We aim to show that although the ground subtyping relation is decidable for both non-expansive class tables and class tables without contravariant constructors, the `SUBTYPE-SAT` problem is undecidable even with both restrictions. We begin with a number of examples demonstrating the complexity of this problem.

▶ **Example 3.2.** Consider a class table

$$\begin{array}{lll} J\texttt{<}+x\texttt{>} & <:: & \\ C & <:: & JC \end{array}$$

and a formula

$$\phi \stackrel{\text{def}}{=} C <: x \wedge C <: y \wedge x \not<: y \wedge y \not<: x$$

Is ϕ satisfiable? Let us consider various possible candidates for the assignment v. Let $v(x) = C$ and $v(y) = C$. In this case, the atom $x \not<: y$ is falsified:

$$v(x) \not<: v(y) = C \not<: C \Leftrightarrow \bot$$

Let $v(x) = C$ and $v(y) = Jy'$ for some y'. Then

$$I(\phi) = C <: C \wedge C <: Jy' \wedge C \not<: Jy' \wedge Jy' \not<: C \Leftrightarrow \bot$$

The case $v(x) = Jx'$ and $v(y) = C$ is symmetrical. The last case is $I(x) = Jx'$ and $I(y) = Jy'$:

$$\begin{array}{l} v(\phi) = C <: Jx' \wedge C <: Jy' \wedge Jx' \not<: Jy' \wedge Jy' \not<: Jx' \Leftrightarrow \\ JC <: Jx' \wedge JC <: Jy' \wedge Jx' \not<: Jy' \wedge Jy' \not<: Jx' \Leftrightarrow \\ C <: x' \wedge C <: y' \wedge x' \not<: y' \wedge y' \not<: x' \end{array}$$

Note that $v(\phi)$ is exactly ϕ up to a renaming of variables. It means that every candidate variable substitution either falsifies the formula, or results in a formula to which the same reasoning applies. As an infinite chain of J is not a valid type, ϕ is unsatisfiable.

The unsatisfiability of ϕ can be intuitively explained in the following way. The satisfiability of ϕ would mean that C has two incomparable supertypes. A set of supertypes of C is exactly $\{J^n C \mid n \geq 0\}$. But for all n, m, $J^n C$ and $J^m C$ are comparable: $n \leq m$ iff $J^n C <: J^m C$.

▶ **Example 3.3.** Consider another class table

$$\begin{array}{lll} E & <:: & \\ J\texttt{<}+x\texttt{>} & <:: & \\ A_1 & <:: & J^{n_1}A_1,\ J^{n_1}E \\ & \vdots & \\ A_m & <:: & J^{n_m}A_m,\ J^{n_m}E, \end{array}$$

where m, $n_i \geq 1$, and the formula

$$\phi \overset{\text{def}}{=} \bigwedge_{1 \leq i \leq m} A_i <: x$$

This formula is satisfied only by v such that

$$v(x) = J^{k \cdot lcm(n_1,\ldots,n_m)}E,$$

where $k \geq 1$ and $lcm(n_1, \ldots, n_m)$ is a least common multiple of $n_1, \ldots, n_m$.

▶ **Example 3.4.** If we replace the class table entry for E in Example 3.3 with

$$E \quad <:: \quad J^{n_1 \cdot \ldots \cdot n_m}E$$

then the formula

$$\phi' \overset{\text{def}}{=} \phi \wedge E \not<: x$$

has a model if and only if the numbers n_1, $\ldots$, n_m are not coprime.

▶ **Example 3.5.** Fix a class table CT and a finite partially ordered set $(P, \leq_P)$. Consider the formula

$$\phi \overset{\text{def}}{=} \bigwedge_{\substack{x,y \in P, \\ x \leq_P y}} x <: y \wedge \bigwedge_{\substack{x,y \in P, \\ x \not\leq_P y}} x \not<: y$$

ϕ has a model if and only if there exists an order-embedding map from P to the set of ground types defined by CT (partially ordered by ground subtyping relation).

▶ **Proposition 3.6.** `SUBTYPE-SAT` *is semidecidable.*

Proof. There is an algorithm that, given ϕ, enumerates all possible ground substitutions v of variables of ϕ and checks $v \vDash^{CT}_{<:} \phi$. Proposition 2.9 guarantees that if ϕ is satisfiable modulo $\mathcal{T}^{CT}_{<:}$, then this algorithm eventually terminates. ◀

In the following section, we show that the set of ground substitution v such that $v \nvDash^{CT}_{<:} \phi$ is not recursively enumerable.

4 SUBTYPE-SAT is undecidable

$$
\begin{array}{lll}
A_1\texttt{<}+x\texttt{>} & <:: & \\
 & \vdots & \\
A_m\texttt{<}+x\texttt{>} & <:: & \\
R\texttt{<}+x\texttt{>} & <:: & \\
S\texttt{<}+x\texttt{>} & <:: & \\
R_0 & <:: & RR_0,\ E \\
S_0 & <:: & SS_0,\ E \\
E & <:: & \\
U_1\texttt{<}+x\texttt{>} & <:: & \overline{A_{U_1}}x,\ W_1x,\ Sx,\ Rx \\
 & \vdots & \\
U_n\texttt{<}+x\texttt{>} & <:: & \overline{A_{U_n}}x,\ W_nx,\ Sx,\ Rx \\
V_1\texttt{<}+y\texttt{>} & <:: & \overline{A_{V_1}}y,\ W_1y,\ Sy,\ Ry \\
 & \vdots & \\
V_n\texttt{<}+y\texttt{>} & <:: & \overline{A_{V_n}}y,\ W_ny,\ Sy,\ Ry
\end{array}
$$

$$
\begin{array}{lll}
W_1\texttt{<}+x\texttt{>} & <:: & \\
 & \vdots & \\
W_n\texttt{<}+x\texttt{>} & <:: & \\
G & <:: & U_1G,\dots,U_nG,\ E \\
H & <:: & V_1H,\dots,V_nH,\ E \\
P & <:: & U_1P,\dots,U_nP,\ E \\
Q & <:: & V_1Q,\dots,V_nQ,\ E \\
W & <:: & W_1W,\dots,W_nW,\ E \\
D & <:: & \overline{A_{U_1}}D,\dots,\overline{A_{U_n}}D,\ E
\end{array}
$$

Listing 2 Class table PCP-CT.

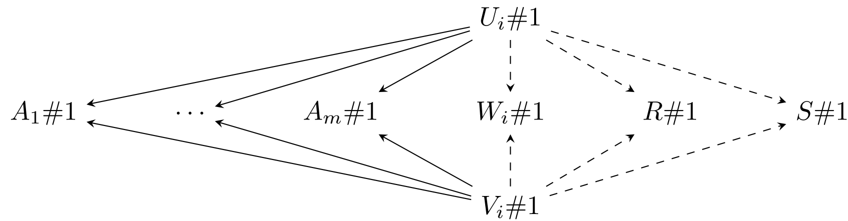

Figure 2 Type parameter dependency graph for PCP-CT.

We prove the undecidability of SUBTYPE-SAT via a reduction from the Post Correspondence Problem.

The Post Correspondence Problem

Let $\{(\overline{A_{U_1}}, \overline{A_{V_1}}), \dots, (\overline{A_{U_n}}, \overline{A_{V_n}})\}$ be a set of pairs of non-empty words over a finite alphabet $\{A_1, \dots, A_m\}$. The Post Correspondence Problem (PCP) is to determine whether or not there exists a sequence of indices $i_1, \dots, i_r$ such that $\overline{A_{U_{i_1}}} \dots \overline{A_{U_{i_r}}} = \overline{A_{V_{i_1}}} \dots \overline{A_{V_{i_r}}}$.

It is a well-known fact that PCP is undecidable [13].

We use a class table from Listing 2 in our reduction. Note that it is non-expansive as its type parameter dependency graph (see Figure 2) has no cycles, and it has no contravariant constructors.

Consider the SUBTYPE-SAT problem for a formula ψ with type parameters x, y, z, q, p, t and class table PCP-CT, where ψ is defined as follows:

$$\phi_0 \stackrel{\text{def}}{=} R_0 <: p \wedge S_0 <: q \wedge W <: z$$
$$\phi_1 \stackrel{\text{def}}{=} G <: x \wedge P <: x \wedge x <: p \wedge x <: q \wedge x <: z \wedge \phi_0$$
$$\phi_2 \stackrel{\text{def}}{=} H <: y \wedge Q <: y \wedge y <: p \wedge y <: q \wedge y <: z \wedge \phi_0$$
$$\psi \stackrel{\text{def}}{=} D <: t \wedge x <: t \wedge y <: t \wedge t \not<: E \wedge \phi_1 \wedge \phi_2$$

The main idea of this reduction is to represent the words in $\{ A_1, \ldots, A_m \}^+$ as chains of covariant constructors terminating with E. For example, words $\overline{A_{U_i}}$ and $\overline{A_{V_j}}$ are encoded as $\overline{A_{U_i}}E$ and $\overline{A_{V_j}}E$. The enumeration of PCP solutions is encoded into the PCP-CT and ψ. We demonstrate a non-deterministic process, consistently refining the type variables of ψ by replacing them with a type constructor applied to fresh variables, and then simplifying the new formula.

▶ **Definition 4.1.** *A* ground *substitution is a substitution with only ground types in its range. An* elementary *substitution is a substitution whose range contains only constructed types with type variables as their arguments. A substitution is* complete *for a formula f, if its domain is exactly all type variables of f.*

Substitutions may be *composed*. The composition of $u = [\, x_1 \mapsto a_1; \ldots; x_n \mapsto a_n \,]$ and $v = [\, y_1 \mapsto b_1; \ldots; y_m \mapsto b_m \,]$ is obtained by removing from the substitution $[\, x_1 \mapsto va_1; \ldots; x_n \mapsto va_n; y_1 \mapsto b_1; \ldots; y_m \mapsto b_m \,]$ those pairs $y_i \mapsto b_i$ for which $y_i \in \{x_1, \ldots, x_k\}$. For instance,

$$x \mapsto Cx' \odot x' \mapsto Dx'' = x \mapsto CDx''$$

We also define a composition of *substitution sets*:

$$\{ \, subst_1^{left}; \, \ldots; \, subst_n^{left} \, \} \odot \{ \, subst_1^{right}; \, \ldots; \, subst_k^{right} \, \} \stackrel{\text{def}}{=} \bigcup_{\substack{1 \le i \le n \\ 1 \le j \le k}} \{ \, subst_i^{left} \odot subst_j^{right} \, \}$$

Note that as ψ is a conjunction of atoms, its satisfiability implies the satisfiability of its arbitrary subformula. Now let us consider complete substitutions that do not falsify the subformula ϕ_0.

▶ **Lemma 4.2.** *For the formula*

$$\phi_0 \stackrel{\text{def}}{=} R_0 <: p \wedge S_0 <: q \wedge W <: z,$$

a substitution does not falsify ϕ_0 iff it can be represented as a composition of substitutions

$$p \mapsto Rp, \; p \mapsto E, \; p \mapsto R_0, \; q \mapsto Sq, \; q \mapsto E, \; q \mapsto S_0, \; z \mapsto W_i z, \; z \mapsto E, \; z \mapsto W$$

with $1 \le i \le n$.

Proof. The type constructor R_0 only has nominal supertypes with head constructors R_0, R and E. Hence, after an application of a substitution different from $\{ \, p \mapsto Rp; \; p \mapsto E; \; p \mapsto R_0 \, \}$ to ϕ_0, the VAR and SUPER rules cannot be applied to simplify the formula, therefore ϕ_0 becomes false. The application of both $p \mapsto E$ and $p \mapsto R_0$ satisfy an atom $R_0 <: p$; an application of $p \mapsto Rp$ turns this atom into itself. Therefore, only the compositions of these substitutions do not falsify the formula.

A similar argument works for the q and z type variables. ◀

▶ **Lemma 4.3.** *For the formula*

$$\phi_1 \stackrel{\text{def}}{=} G <: x \wedge P <: x \wedge x <: p \wedge x <: q \wedge x <: z \wedge \phi_0,$$

only elementary complete substitutions

$$[\,p \mapsto Rp;\ q \mapsto Sq;\ z \mapsto W_i z;\ x \mapsto U_i x\,]$$

$$[\,p \mapsto E;\ q \mapsto E;\ z \mapsto E;\ x \mapsto E\,]$$

with $1 \leq i \leq n$ *do not falsify it.*

Proof. The proof is by a case splitting into possible substitutions.

As ϕ_0 is a subformula of ϕ_1, Lemma 4.2 implies that the candidate substitutions to z that do not falsify ϕ_1 immediately are the ones from the set $\{\, z \mapsto W_i z;\ z \mapsto E;\ z \mapsto W \,\}$.

- $[\,z \mapsto W\,]\phi_1 = G <: x \wedge x <: W \wedge \ldots$

  By transitivity of subtyping, this entails the $G <: W$, which is false. Therefore, the substitutions with $z \mapsto W$ falsify ϕ_1.

- $[\,z \mapsto W_i z\,]\phi_1 = G <: x \wedge P <: x \wedge x <: W_i z \wedge \ldots$

  In order for this formula to be satisfiable, the substitution should map x to a common supertype for G and P, and it should have a nominal supertype constructed with W_i. The only such substitutions are $\{\, x \mapsto U_i x;\ x \mapsto W_i x \,\}$.

  - $[\,x \mapsto W_i x\,]\phi_1 = R_0 <: p \wedge W_i x <: p \wedge \ldots$

    This formula has the atom $W_i x <: p$, which is not falsified only if the substitution $p \mapsto W_i p$ is applied to the type variable p. But by Lemma 4.2, the candidate substitutions into p are $\{\, p \mapsto Rp;\ p \mapsto E;\ p \mapsto R_0 \,\}$. Therefore, each substitution containing $[\,x \mapsto W_i x;\ z \mapsto W_i z\,]$ falsifies ϕ_1.

  - $[\,x \mapsto U_i x\,]\phi_1 = R_0 <: p \wedge U_i x <: p \wedge S_0 <: q \wedge U_i x <: q \wedge U_i x <: W_i z \wedge \ldots$

    Common supertypes of R_0 and U_i could have only one head constructor, namely R; symmetrically, the common supertypes S_0 and U_i could be constructed only by S. Therefore, the substitutions $p \mapsto Rp$ and $q \mapsto Sq$ do not falsify the ϕ_1, while substitutions from the set

    $$[\,z \mapsto W_i z;\ x \mapsto U_i x\,] \odot \{\, p \mapsto E;\ p \mapsto R_0 \,\} \odot \{\, q \mapsto E;\ q \mapsto S_0 \,\}$$

    falsify ϕ_1. Hence, in this case, only the substitution

    $$[\,z \mapsto W_i z;\ x \mapsto U_i x;\ p \mapsto Rp;\ q \mapsto Sq\,]$$

    does not falsify ϕ_1.

- $[\,z \mapsto E\,]\phi_1 = G <: x \wedge P <: x \wedge x <: E \wedge \ldots$

  In order for this formula to be satisfiable, the substitution should map x to a common supertype for G and P, which is a subtype of E. The only appropriate substitution is $x \mapsto E$. The application of the substitution $[\,z \mapsto E;\ x \mapsto E\,]$ to ϕ_1 gives

  $$G <: E \wedge P <: E \wedge E <: p \wedge E <: q \wedge E <: E \wedge R_0 <: p \wedge S_0 <: q \wedge W <: E,$$

which simplifies into

$$E <: p \wedge E <: q \wedge R_0 <: p \wedge S_0 <: q.$$

In order for this formula to be satisfiable, the substitution should map p to a common supertype of R_0 and E, and q should be mapped into a common supertype of S_0 and E. The only appropriate substitution is $[\,p \mapsto E;\, q \mapsto E\,]$. Therefore the substitution

$$[\,z \mapsto E;\, x \mapsto E;\, p \mapsto E;\, q \mapsto E\,]$$

does not falsify ϕ_1.

We have considered all possible cases, among which only the substitutions

$$[\,z \mapsto W_i z;\, x \mapsto U_i x;\, p \mapsto Rp;\, q \mapsto Sq\,]$$

$$[\,z \mapsto E;\, x \mapsto E;\, p \mapsto E;\, q \mapsto E\,]$$

with $1 \leq i \leq n$ do not falsify ϕ_1. ◀

▶ **Lemma 4.4.** *For the formula*

$$\phi_2 \stackrel{\text{def}}{=} H <: y \wedge Q <: y \wedge y <: p \wedge y <: q \wedge y <: z \wedge \phi_0,$$

only elementary complete substitutions

$$[\,p \mapsto Rp;\, q \mapsto Sq;\, z \mapsto W_i z;\, y \mapsto V_i y\,]$$

$$[\,p \mapsto E;\, q \mapsto E;\, z \mapsto E;\, y \mapsto E\,]$$

with $1 \leq i \leq n$ *do not falsify it.*

Proof. Similar to the proof of Lemma 4.3. ◀

Lemma 4.3 and Lemma 4.4 imply that only elementary complete substitutions

$$[\,p \mapsto Rp;\, q \mapsto Sq;\, z \mapsto W_i z;\, x \mapsto U_i x;\, y \mapsto V_i y\,]$$

$$[\,p \mapsto E;\, q \mapsto E;\, z \mapsto E;\, x \mapsto E;\, y \mapsto E\,]$$

do not falsify $\phi_1 \wedge \phi_2$.

$\phi_1 \wedge \phi_2$ has a very important property: the application of the substitution

$$[\,p \mapsto Rp;\, q \mapsto Sq;\, z \mapsto W_i z;\, x \mapsto U_i x;\, y \mapsto V_i y\,]$$

to it and the simplification of the resulting formula turn $\phi_1 \wedge \phi_2$ into itself:

$$\begin{array}{l}
G <: U_i x \wedge P <: U_i x \wedge U_i x <: Rp \wedge U_i x <: Sq \wedge U_i x <: W_i z \wedge \\
H <: V_i y \wedge Q <: V_i y \wedge V_i y <: Rp \wedge V_i y <: Sq \wedge V_i y <: W_i z \wedge \\
R_0 <: Rp \wedge S_0 <: Sq \wedge W <: W_i z \\
\Leftrightarrow \\
U_i G <: U_i x \wedge U_i P <: U_i x \wedge Rx <: Rp \wedge Sx <: Sq \wedge W_i x <: W_i z \wedge \\
V_i H <: V_i y \wedge V_i Q <: V_i y \wedge Ry <: Rp \wedge Sy <: Sq \wedge W_i y <: W_i z \wedge \\
RR_0 <: Rp \wedge SS_0 <: Sq \wedge W_i W <: W_i z \\
\Leftrightarrow \\
G <: x \wedge P <: x \wedge x <: p \wedge x <: q \wedge x <: z \wedge \\
H <: y \wedge Q <: y \wedge y <: p \wedge y <: q \wedge y <: z \wedge \\
R_0 <: p \wedge S_0 <: q \wedge W <: z \\
= \phi_1 \wedge \phi_2
\end{array}$$

Note also that application of the substitution

$$[\,p \mapsto E;\ q \mapsto E;\ z \mapsto E;\ x \mapsto E;\ y \mapsto E\,]$$

and simplification turn $\phi_1 \wedge \phi_2$ into a true.

Notation

Let $N = \{\,1, \ldots, n\,\}$. We denote the set of finite sequences in N by $N^{<\omega}$. For $J \in N^{<\omega}$, $J = j_1 \ldots j_r$, we denote by $U_J T$ a chain of type constructors $U_{j_1} \ldots U_{j_r} T$; we define $V_J T$ and $W_J T$ similarly. Sometimes we write J^r to emphasize that the length of J is r.

▶ **Theorem 4.5.** *The formula $\phi_1 \wedge \phi_2$ has a set of models with the interpretations I_J such that*

$$v_J(x) = U_J E, \quad v_J(y) = V_J E, \quad v_J(z) = W_J E,$$
$$v_J(p) = R^r E, \quad v_J(q) = S^r E$$

where $J \in N^{<\omega}$, and r is the length of J.

Proof. As we have shown, only the compositions of the following substitutions do not falsify the formula $\phi_1 \wedge \phi_2$ immediately:

$$subst_i \stackrel{\text{def}}{=} [\,p \mapsto Rp;\ q \mapsto Sq;\ z \mapsto W_i z;\ x \mapsto U_i x;\ y \mapsto V_i y\,]$$
$$subst_{end} \stackrel{\text{def}}{=} [\,p \mapsto E;\ q \mapsto E;\ z \mapsto E;\ x \mapsto E;\ y \mapsto E\,]$$

Note that as free variable substitutions are ground substitutions, we may compose them. As $subst_i$ does not change $\phi_1 \wedge \phi_2$ after simplification, and $subst_{end}$ satisfies it, a satisfying substitution for $\phi_1 \wedge \phi_2$ may only be a composition of the finite number of $subst_i$, ending with the ground substitution $subst_{end}$, i.e. $subst_{j_1} \odot \ldots \odot subst_{j_r} \odot subst_{end}$.

Thus the only satisfying ground substitutions of $\phi_1 \wedge \phi_2$ are:

$$v_J = subst_{j_1} \odot \ldots \odot subst_{j_r} \odot subst_{end} =$$
$$= [\,p \mapsto R^r E;\ q \mapsto S^r E;\ z \mapsto W_J E;\ x \mapsto U_J E;\ y \mapsto V_J E\,]$$ ◀

▶ **Lemma 4.6.** *Let $\overline{L}$ be a chain of "letter" constructors, i.e. constructors from $\{A_1, \ldots, A_m\}$, $J^r \in N^{<\omega}$ with $r > 0$. Then*

$$U_J E <: \overline{L}E \vee U_J E <: \overline{L}D$$

is satisfiable if and only if

$$\overline{A_{U_{j_1}}} \ldots \overline{A_{U_{j_r}}} E = \overline{L}E$$

Proof. We prove the claim by induction on r.

Base step: $r = 1$.

$$U_{j_1} E <: \overline{L}E \vee U_{j_1} E <: \overline{L}D \Leftrightarrow \overline{A_{U_{j_1}}} E <: \overline{L}E \vee \overline{A_{U_{j_1}}} E <: \overline{L}D$$

As $\overline{A_{U_{j_1}}} E$, $\overline{L}E$ and $\overline{L}D$ are constructed from covariant type constructors $A_1, \ldots, A_m$ without the right hand side of the class table (i.e. without the strict *nominal* supertypes), we may conclude that

$$\overline{A_{U_{j_1}}} E <: \overline{L}E \vee \overline{A_{U_{j_1}}} E <: \overline{L}D \Leftrightarrow \overline{A_{U_{j_1}}} E = \overline{L}E \vee \overline{A_{U_{j_1}}} E = \overline{L}D \Leftrightarrow \overline{A_{U_{j_1}}} E = \overline{L}E$$

Induction step

Let $r = k + 1$, $J = j_1 \cdot J'$, length of J' is k.

$$U_{j_1} U_{J'} E <: \overline{L} E \vee U_{j_1} U_{J'} E <: \overline{L} D \Leftrightarrow$$
$$\Leftrightarrow \overline{A_{U_{j_1}}} U_{J'} E <: \overline{L} E \vee \overline{A_{U_{j_1}}} U_{J'} E <: \overline{L} D \Leftrightarrow$$

As $\overline{A_{U_{j_1}}} E$, $\overline{L} E$ and $\overline{L} D$ are constructed from covariant type constructors $A_1, \ldots, A_m$, which do not have strict nominal supertypes, we must require $\overline{L} = \overline{A_{U_{j_1}} L'}$.

$$\Leftrightarrow \overline{A_{U_{j_1}}} U_{J'} E <: \overline{A_{U_{j_1}} L'} E \vee \overline{A_{U_{j_1}}} U_{J'} E <: \overline{A_{U_{j_1}} L'} D \Leftrightarrow$$
$$\Leftrightarrow U_{J'} E <: \overline{L'} E \vee U_{J'} E <: \overline{L'} D \Leftrightarrow \text{(I.H.)}$$
$$\Leftrightarrow \overline{A_{U_{j_2}}} \ldots \overline{A_{U_{j_r}}} E = \overline{L'} E \Leftrightarrow \overline{A_{U_{j_1}} A_{U_{j_2}}} \ldots \overline{A_{U_{j_r}}} E = \overline{L} E$$ ◀

▶ **Lemma 4.7.** *Let $\overline{L}$ be a chain of constructors from $\{A_1, \ldots, A_m\}$, $J^r \in N^{<\omega}$ with $r > 0$. Then*

$$V_J E <: \overline{L} E \vee V_J E <: \overline{L} D$$

is satisfiable if and only if

$$\overline{A_{V_{j_1}}} \ldots \overline{A_{V_{j_r}}} E = \overline{L} E$$

Proof. Similar to the proof of Lemma 4.6. ◀

▶ **Lemma 4.8.** *The formula*

$$\phi_{J^r} \stackrel{\text{def}}{=} D <: t \wedge U_J E <: t \wedge V_J E <: t \wedge t \not<: E$$

is satisfiable if and only if $r > 0$ and

$$\phi'_J \stackrel{\text{def}}{=} \overline{A_{V_{j_1}}} \ldots \overline{A_{V_{j_r}}} E = \overline{A_{U_{j_1}}} \ldots \overline{A_{U_{j_r}}} E$$

is valid.

Proof. Let J be an empty sequence. Then ϕ_J becomes

$$D <: t \wedge E <: t \wedge t \not<: E.$$

D and E have only one common supertype E. But the substitution of E into t falsifies the formula because of the atom $t \not<: E$. Hence if J is an empty sequence, ϕ_J is unsatisfiable.

Let J be a non-empty sequence.

($\Rightarrow$)

Let ϕ_J be satisfiable. Then D and $U_J E$ should have a common supertype. It cannot be E, as for all i, E is not a supertype for U_i. Consider all other supertypes of D. Those are chains of constructors from $\{ A_1, \ldots, A_m \}$, terminated by either D or E. In other words, the only candidate supertypes are $\overline{L} D$ and $\overline{L} E$, where $\overline{L}$ are non-empty chains of constructors from $\{ A_1, \ldots, A_m \}$.

By Lemma 4.6 and Lemma 4.7, if $v_J \vDash^{CT}_{<:} \phi_J$, then

$$v_J(t) = \overline{A_{U_{j_1}}} \ldots \overline{A_{U_{j_r}}} E = \overline{A_{V_{j_1}}} \ldots \overline{A_{V_{j_r}}} E$$

As ϕ_J is satisfiable, ϕ'_J is true.

($\Leftarrow$)
Let ϕ'_J be valid, then the interpretation

$$v_J(t) = \overline{A_{U_{j_1}}} \dots \overline{A_{U_{j_r}}} E = \overline{A_{V_{j_1}}} \dots \overline{A_{V_{j_r}}} E,$$

satisfies ϕ_J:

$$v_J(\phi_J) = D <: v_J(t) \wedge U_J E <: v_J(t) \wedge V_J E <: v_J(t) \wedge v_J(t) \not<: E =$$

$$D <: \overline{A_{U_{j_1}}} \dots \overline{A_{U_{j_r}}} E \wedge U_J E <: \overline{A_{U_{j_1}}} \dots \overline{A_{U_{j_r}}} E \wedge$$
$$V_J E <: \overline{A_{V_{j_1}}} \dots \overline{A_{V_{j_r}}} E \wedge \overline{A_{U_{j_1}}} \dots \overline{A_{U_{j_r}}} E \not<: E \Leftrightarrow$$

$$\overline{A_{U_{j_1}}} \dots \overline{A_{U_{j_r}}} E <: \overline{A_{U_{j_1}}} \dots \overline{A_{U_{j_r}}} E \wedge$$
$$\overline{A_{V_{j_1}}} \dots \overline{A_{V_{j_r}}} E <: \overline{A_{V_{j_1}}} \dots \overline{A_{V_{j_r}}} E \wedge$$
$$\overline{A_{U_{j_1}}} \dots \overline{A_{U_{j_r}}} E \not<: E \Leftrightarrow \top$$ ◀

▶ **Theorem 4.9.** *The formula*

$$\phi_J \stackrel{\text{def}}{=} D <: t \wedge U_J E <: t \wedge V_J E <: t \wedge t \not<: E$$

is satisfiable if and only if J is a solution to PCP with the pairs of words

$$\{(\overline{A_{U_1}}, \overline{A_{V_1}}), \dots, (\overline{A_{U_n}}, \overline{A_{V_n}})\}$$

over the alphabet $\{ A_1, \dots, A_m \}$, i.e.

$$\overline{A_{U_{j_1}}} \dots \overline{A_{U_{j_r}}} = \overline{A_{V_{j_1}}} \dots \overline{A_{V_{j_r}}}$$

Proof. By Lemma 4.8, ϕ_J is satisfiable if and only if $r > 0$ and

$$\phi'_J \stackrel{\text{def}}{=} \overline{A_{V_{j_1}}} \dots \overline{A_{V_{j_r}}} E = \overline{A_{U_{j_1}}} \dots \overline{A_{U_{j_r}}} E$$

is valid.
($\Rightarrow$)
Let ϕ'_J be valid. Then J is such a non-empty sequence of indices that the concatenation of words $\overline{A_{U_{j_1}}}, \dots, \overline{A_{U_{j_r}}}$ equals the concatenation of words $\overline{A_{V_{j_1}}}, \dots, \overline{A_{V_{j_r}}}$. Therefore J is a solution to PCP. ($\Leftarrow$)
Let J be a solution to PCP. That means that the types $\overline{A_{V_{j_1}}} \dots \overline{A_{V_{j_r}}} E$ and $\overline{A_{U_{j_1}}} \dots \overline{A_{U_{j_r}}} E$ are equal. Besides, as J solves PCP, it is non-empty. This implies the validity of ϕ'_J. ◀

▶ **Theorem 4.10.** *The formula*

$$\psi \stackrel{\text{def}}{=} D <: t \wedge x <: t \wedge y <: t \wedge t \not<: E \wedge \phi_1 \wedge \phi_2$$

is satisfiable if and only if PCP with the pairs of words

$$\{(\overline{A_{U_1}}, \overline{A_{V_1}}), \dots, (\overline{A_{U_n}}, \overline{A_{V_n}})\}$$

over the alphabet $\{ A_1, \dots, A_m \}$ has a solution.

Proof. By Theorem 4.5, the formula $\phi_1 \wedge \phi_2$ has a set of solutions with the interpretations $v_J(I)$ such that:

$$v_J(x) = U_J E, \quad v_J(y) = V_J E, \quad v_J(z) = W_J E,$$
$$v_J(p) = R^r E, \quad v_J(q) = S^r E$$

Hence the satisfiability of ψ is equivalent to the satisfiability of

$$\psi' \stackrel{\text{def}}{=} \bigvee_{J \in N^{<\omega}} v_J(\psi) = \bigvee_{J \in N^{<\omega}} D <: t \wedge U_J E <: t \wedge V_J E <: t \wedge t \not<: E = \bigvee_{J \in N^{<\omega}} \phi_J$$

By Theorem 4.5, ϕ_J is satisfiable if and only if J is a solution to PCP. Therefore ψ' is satisfiable if and only if there exists a sequence of indices J that solves PCP. ◀

▶ **Corollary 4.11.** *SUBTYPE-SAT is undecidable.*

▶ **Corollary 4.12.** *SUBTYPE-SAT is undecidable even for non-expansive class tables without contravariant constructors.*

▶ **Corollary 4.13.** *SUBTYPE-SAT is undecidable even for non-expansive class tables with only constant and unary constructors.*

▶ **Corollary 4.14.** *SUBTYPE-SAT is undecidable even for quantifier-free conjunctions of literals.*

▶ **Corollary 4.15.** *SUBTYPE-SAT is undecidable even for quantifier-free conjunctions of positive literals.*

Proof. To show this, we simply need to exclude the trivial solution with all variables mapped to E without using the atom $t \not<: E$ in ψ.

This can be done, for instance, by adding a new entry

$$D_0 \quad <:: \quad \overline{A_{U_1}} D, \ldots, \overline{A_{U_n}} D$$

into `PCP-CT` and altering ψ to

$$\psi \stackrel{\text{def}}{=} D_0 <: t \wedge x <: t \wedge y <: t \wedge \phi_1 \wedge \phi_2$$ ◀

5 Decidable fragments of `SUBTYPE-SAT`

In this section, we introduce several fragments of the `SUBTYPE-SAT` and prove their decidability. Decidability could be achieved by restricting the class table or the formula. A semiground fragment constrains the formula, allowing the class table to be arbitrary (but non-expansive). Using the intuitions from the proof, we conjecture another decidable fragment that constraints the shape of the class table, leaving the formula to be arbitrary.

▶ **Definition 5.1.** *A* semiground *atom (literal) is an atom (literal) with at least one ground argument. A* normalized semiground *atom (literal) is an atom (literal) with one ground argument and one variable argument (note that the SUBTYPE-SAT language only has binary predicate symbols $<:$ and $=$).*

A (normalized) semiground formula is a quantifier-free formula that contains only (normalized) semiground atoms. A semiground fragment *is a set of semiground formulas.*

▶ **Theorem 5.2.** *Each semiground atom is logically equivalent to some normalized semiground formula.*

Proof. The theorem obviously holds in case an atom is already normalized. If both arguments of an atom are ground, it is equivalent to either $\top$ or $\bot$. It remains to consider only the atom $T <: U$ with both T and U constructed. In this case we apply the subtyping rules:

- Let $T = C\texttt{<}\overline{T}\texttt{>}$ and $U = C\texttt{<}\overline{U}\texttt{>}$. Then we simplify $T <: U$ using (VAR) rule:

  $$C\texttt{<}\overline{T}\texttt{>} <: C\texttt{<}\overline{U}\texttt{>} \Leftrightarrow \bigwedge_i T_i <:_{var(C\#i)} U_i$$

  As either $C\texttt{<}\overline{T}\texttt{>}$ or $C\texttt{<}\overline{U}\texttt{>}$ is ground, either every T_i, or every U_i is ground. This means that every atom of the resulting formula is still semiground.
- Let $T = C\texttt{<}\overline{T}\texttt{>}$ and $U = D\texttt{<}\overline{U}\texttt{>}$ with $C \neq D$. Then we apply the (SUPER) rule:

  $$C\texttt{<}\overline{T}\texttt{>} <: D\texttt{<}\overline{U}\texttt{>} \Leftrightarrow \bigvee_j D\texttt{<}\overline{W_j}\texttt{>} <: D\texttt{<}\overline{U}\texttt{>},$$

  where $D\texttt{<}\overline{W_j}\texttt{>} = [\,\overline{x} \mapsto \overline{T}\,]V_j$ and $C\texttt{<}\overline{x}\texttt{>} <:: V_j$. Obviously, all atoms in this formula are still semiground.

As the non-deterministic application of subtyping rules with occurence checks is a decision procedure for non-expansive inheritance [8], the process is either terminating, resulting in a normalized formula, or eventually the chain of simplifications loops on some atom. In this case, the atom is tautologically false. ◀

▶ **Corollary 5.3.** *Each semiground formula is logically equivalent to some normalized semiground formula.*

▶ **Definition 5.4.** *Let* $Q \stackrel{\text{def}}{=} A_1 \wedge \ldots \wedge A_n$ *be a conjunct.* Maximal connected subconjuncts *is a set of conjuncts* $\{Q_1, \ldots, Q_m\}$ *such that*
1. $Q = Q_1 \wedge \ldots \wedge Q_m$
2. Q_i *and* Q_j *have distinct literals if* $i \neq j$
3. *If* A_i *and* A_j *have common variables then they occur in the same* Q_k

▶ **Lemma 5.5.** *If SUBTYPE-SAT is decidable for the conjunction of normalized semiground literals (with the only free variable* x*) of the form*

$$\bigwedge_i T_i <: x \wedge \bigwedge_j x <: U_j \wedge \bigwedge_k V_k \not<: x \wedge \bigwedge_l x \not<: W_l,$$

then SUBTYPE-SAT is decidable for the whole semiground fragment.

Proof. Let ψ be a semiground formula. By Corollary 5.3

$$\psi \sim^{CT}_{<:} \bigvee_i \bigwedge_j \psi_{i,j},$$

where $\psi_{i,j}$ is a normalized semiground literal. Each conjunct can be divided into maximal connected subconjuncts in such a way that each one of them contains only one variable. The algorithm then can check the satisfiability of each group separately. ◀

We define a set of substitutions

$$CtorSubsts = \{\, x \mapsto C\texttt{<}\overline{x}\texttt{>} \,\},$$

where C is a constructor from CT, and $\overline{x}$ is a vector of distinct variables.

▶ **Lemma 5.6.** *Let* ψ *be a conjunction of normalized semiground literals with only one free variable. Then* $\psi(x)$ *is equisatisfiable with*

$$\psi^{subst} \stackrel{\text{def}}{=} \bigvee_{subst \in CtorSubsts} subst\, \psi(x)$$

▶ **Lemma 5.7.** *The `SUBTYPE-SAT` problem is decidable for conjunctions of normalized semiground literals with one free variable.*

Proof. Let ψ_0 be a conjunction of normalized semiground literals with one free variable. Apply Lemma 5.6. ψ_0^{subst} is a semiground formula. If it is ground, we may check its satisfiablity and terminate. Otherwise, we may act as in Lemma 5.5: apply Corollary 5.3 and convert the formula to DNF. ψ_0^{subst} is satisfiable iff one of the conjuncts is satisfiable.

Choose the conjunct non-deterministically and divide it into maximal connected subconjuncts. The algorithm then checks the satisfiability of each subconjunct ϕ_1, i.e. we have reduced the problem to itself. The described procedure enumerates the (possibly infinite) sequence $\{ \psi_i \}_{i \in \mathbb{N}}$.

Without loss of generality, we may conclude that all literals in ψ_i are distinct and that all conjuncts have an identical free variable. Two conjuncts are equal if they are identical as sets of literals. Note that if $\psi_i = \psi_j$ for some $i \neq j$, then ψ_i is unsatisfiable, and we may terminate.

For the conclusion of the proof, we refer to the notion of inheritance closure from [8]. It is known that the inheritance closure of a finite set of types within a non-expansive class table is finite [8]. Now, note that all ground types of ψ_i are obtained by application of subtyping rules, thus they are elements of the inheritance closure for the set of ground types of ψ_0. That means that only a finite number of literals may occur in $\{ \psi_i \}_{i \in \mathbb{N}}$. As every literal occurs in each conjunct no more than once, eventually $\psi_i = \psi_j$ for $i < j$. Therefore, our procedure terminates. ◀

▶ **Theorem 5.8.** *`SUBTYPE-SAT` is decidable for queries with a semiground fragment.*

Proof. By Lemma 5.5 and Lemma 5.7. ◀

Formulas from Example 3.3 and Example 3.4 refer to the described fragment and their satisfiability can be checked. And the formula from Theorem 4.10 goes beyond the fragment, which is consistent with the undecidability of PCP.

The proof of Theorem 5.2 can be used to produce a generalized scheme for obtaining new decidable fragments. We describe this scheme below.

▶ **Definition 5.9.** *A* normalized *atom (literal) is an atom (literal) that does not have open constructed types. In other words, a normalized atom is either a normalized semiground atom, or $x <: y$, or $x = y$, where x and y are variables. A normalized formula is a quantifier-free formula that only contains normalized atoms.*

Consider a conjunct whose maximal connected subconjuncts consist of a single element. Acting similarly to Lemma 5.6, we simplify the obtained formula, convert it to DNF, and split it into maximal connected subconjuncts. If we can impose some restrictions on the class table or the formula so that the number of different obtained subconjuncts is finite, then we have a criterion for the termination of the procedure.

▶ **Example 5.10.** Let the class table be organized in such a way that for each pair of constructors C and D, the literals $C{<}\overline{x}{>} <: D{<}\overline{y}{>}$ and $C{<}\overline{x}{>} \not<: D{<}\overline{y}{>}$ are transformed via the application of Corollary 5.3 into such a disjunction of normalized conjuncts that each of them may be partitioned into maximal connected subconjuncts, containing no more than one variable from $\overline{x}$ and no more than one variable from $\overline{y}$. Applying the scheme described above to normalized conjuncts with n different free variables, we can only obtain normalized

conjuncts which have no more than n free variables. Moreover, each quantifier-free formula can be simplified into a normalized formula. As the number of distinct types in an inheritance closure is bounded [8], the number of conjuncts is bounded, so the procedure will terminate.

This idea may be used to introduce some syntactic restrictions on the shape of the class table. We leave it for the future work.

6 Related work

There is a long line of research on decidability of ground subtyping of nominal type systems with variance. One of the latest studies has shown the Turing-completeness of Java subtyping [7]. C++ templates are also known to be Turing-complete [18]. Scala [12], OCaml [10, 15] and Haskell type systems with extensions are undecidable as well. However, .NET subtyping is decidable [5, 8]. These papers formalize and investigate *type checking* in certain programming languages. In contrast, we investigate the subtyping in the presence of open types, founded on the results on ground subtyping.

Constraint satisfiability

The most relevant recent work is [16]. Motivated by the same goals, it reduces the satisfiability problem of type-based partially ordered sets to the first-order satisfiability problem and proposes to use SMT-solvers to solve the constraints. Unlike our work, the type system under consideration is a nominal fragment of Java type system without generics.

The satisfiability problem for subtyping constraints and its computational complexity for more general (in comparison to [16]) fragments of type systems is explored in [14, 6, 9, 11]. These works explore constraints on finite and recursive types, structural and non-structural subtyping and type constructors with covariant and contravariant type parameters; our work studies a more general problem.

The paper [17] shows the undecidability of first-order subtyping constraints for non-structural subtyping. This result entails the undecidability of `SUBTYPE-SAT`, but it uses a significantly larger fragment of first-order logic (in particular, universally quantified formulas). Our proof uses only quantifier-free conjunctions of positive atoms and those features of nominal subtyping with variance which are not present in the non-structural case.

7 Conclusion

We have introduced the satisfiability problem of nominal subtyping with variance and studied some of its properties. The undecidability of the problem has been proven using a noticeably small fragment of the type system. We have also discussed a number of non-trivial decidable fragments and a scheme to obtain other decidable fragments. Finding more extensive decidable fragments is an open problem: for example, it could be done by introducing syntactic restrictions on the shape of the class table. In addition, it would be interesting to compare the decidable fragments with the fragment that is most widely used in practice. Another area of future work is the construction of effective procedures for solving the `SUBTYPE-SAT` problem. Subsequently, such decision procedures can be implemented in SMT-solvers, which makes them easy to use in a variety of SMT-based program analysis approaches.

References

1. Mike Barnett, K Rustan M Leino, and Wolfram Schulte. The Spec# programming system: An overview. In *International Workshop on Construction and Analysis of Safe, Secure, and Interoperable Smart Devices*, pages 49–69. Springer, 2004.
2. Clark Barrett and Cesare Tinelli. Satisfiability Modulo Theories. In *Handbook of Model Checking*, pages 305–343. Springer, 2018.
3. Leonardo Mendonça de Moura and Nikolaj Bjørner. Satisfiability modulo theories: introduction and applications. *Commun. ACM*, 54(9):69–77, 2011.
4. ECMA ECMA. 335: Common language infrastructure (CLI), 2005.
5. Burak Emir, Andrew Kennedy, Claudio V. Russo, and Dachuan Yu. Variance and Generalized Constraints for C# Generics. In *ECOOP*, volume 4067 of *Lecture Notes in Computer Science*, pages 279–303. Springer, 2006.
6. Alexandre Frey. Satisfying Subtype Inequalities in Polynomial Space. In *SAS*, volume 1302 of *Lecture Notes in Computer Science*, pages 265–277. Springer, 1997.
7. Radu Grigore. Java generics are Turing complete. In *POPL*, pages 73–85. ACM, 2017.
8. Andrew J Kennedy and Benjamin C Pierce. On decidability of nominal subtyping with variance, 2007.
9. Viktor Kuncak and Martin C. Rinard. Structural Subtyping of Non-Recursive Types is Decidable. In *LICS*, pages 96–107. IEEE Computer Society, 2003.
10. Mark Lillibridge. *Translucent sums: A foundation for higher-order module systems*. PhD thesis, Carnegie Mellon University, 1997.
11. Joachim Niehren, Tim Priesnitz, and Zhendong Su. Complexity of Subtype Satisfiability over Posets. In *ESOP*, volume 3444 of *Lecture Notes in Computer Science*, pages 357–373. Springer, 2005.
12. Martin Odersky. Scaling DOT to Scala–soundness, 2016.
13. Emil L Post. A variant of a recursively unsolvable problem. *Bulletin of the American Mathematical Society*, 52(4):264–268, 1946.
14. Vaughan R. Pratt and Jerzy Tiuryn. Satisfiability of Inequalities in a Poset. *Fundam. Inform.*, 28(1-2):165–182, 1996.
15. Andreas Rossberg. Undecidability of OCaml type checking, 1999.
16. Elena Sherman, Brady J. Garvin, and Matthew B. Dwyer. Deciding Type-Based Partial-Order Constraints for Path-Sensitive Analysis. *ACM Trans. Softw. Eng. Methodol.*, 24(3):15:1–15:33, 2015.
17. Zhendong Su, Alexander Aiken, Joachim Niehren, Tim Priesnitz, and Ralf Treinen. The first-order theory of subtyping constraints. In *POPL*, pages 203–216. ACM, 2002.
18. Todd L. Veldhuizen. C++ Templates are Turing complete, 2003.
19. Mirko Viroli. On the recursive generation of parametric types. Technical report, Technical Report DEIS-LIA-00-002, Universita di Bologna, 2000.