On Dart-Zobel Algorithm for Testing Regular Type Inclusion

Lunjin Lu
Oakland University
Rochester, MI 48309
EMail: L2LU@oakland.edu

ABSTRACT
This paper answers open questions about the correctness and the completeness of Dart-Zobel algorithm for testing the inclusion relation between two regular types. We show that the algorithm is complete but incorrect for regular types.

keyword: regular types; type inclusion; algorithm

1. INTRODUCTION
Types are ubiquitous in programming languages [1]. They make programs easier to understand and help detect errors since a large number of errors are type errors.

A *type* is a possibly infinite set of ground terms with a finite representation. An integral part of any type system is its type language that specifies which sets of ground terms are types. To be useful, types should be closed under intersection, union and complement operations. The decision problems such as the emptiness of a type, inclusion of a type in another, equivalence of two types should be decidable. Regular term languages [9, 3], called regular types, satisfy these constraints and have been widely used as types [17, 15, 20, 4, 10, 12, 16, 18, 7, 19, 11, 8, 13, 2, 14].

Most type systems use *tuple distributive regular types* which are strictly less powerful than regular types [17, 15, 20, 10, 12, 16, 18, 7, 19, 11, 8, 13, 2, 14]. Moreover, the class of tuple distributive types are not closed under union. Tuple distributive regular types are regular types closed under tuple distributive closure. Intuitively, the tuple distributive closure of a set of terms is the set of all terms constructed recursively by permuting each argument position among all terms that have the same function symbol [19].

Dart and Zobel [5] present, among others, an inclusion algorithm for regular types with respect to a given set of type definitions without the tuple distributive restriction. However, the completeness and the correctness of the algorithm are left open. This paper provides answers to these open questions. We show that the algorithm is complete but incorrect for regular types.

There were algorithms for deciding the emptiness of regular term languages [9, 3] before Dart and Zobel's work [5]. Thus, the algorithm was intended to solve an already solved problem because the inclusion problem of regular types is equivalent to the emptiness problem of regular term languages. In spite of this, there is a strong impetus to address the correctness and the completeness of Dart-Zobel algorithm for the following reasons. Firstly Dart and Zobel's work [5] was one of early works on regular types and has been widely cited. The algorithm has been experimentally tested [5], and it has also been used in type or type related analyses [4, 6]. Secondly, the algorithm uses tabulation to ensure termination. Tabulation has the effect of pruning subcomputation and has great potential to reduce computational time. Therefore, the algorithm is potentially more efficient than those for deciding the emptiness of regular term languages. Thirdly, when applied to tuple distributive types, the algorithm doesn't need a deterministic regular tree grammar. Instead, it works on a non-deterministic regular tree grammar and compares the tuple-distributive closures of the regular tree languages. In other words, there is no need to transform the regular tree grammar before two types can be compared. Finally, the algorithm deals with built-in types such as integers that can not be defined by a finite number of type rules and that are not dealt with by algorithms for deciding the emptiness of regular term languages.

The remainder of this paper is organised as follows. Section 2 defines regular types by regular term grammars. Section 3 recalls Dart-Zobel algorithm for testing if a regular type is a subset of another regular type. Section 4 addresses the completeness and the correctness of their algorithm that have been left open.

2. REGULAR TYPES
Several equivalent formalisms such as tree automata [9, 3], regular term grammars [9, 3], regular unary logic programs [19] have been used to describe regular types. In [5], Dart and Zobel use regular term grammars to describe regular types that are sets of ground terms over a ranked alphabet Σ.

A regular term grammar is a tuple $\mathsf{G} = \langle \Pi, \Sigma, \Delta \rangle$ where[1]

- Σ is a fixed ranked alphabet. Each symbol in Σ is called a function symbol and has a fixed arity. It is assumed that Σ contains at least one constant that is a function symbol of arity 0.
- Π is a set of symbols called nonterminals. These terminals will be called type symbols as they represent

[1] A start symbol is not needed in our setting.

types. Type symbols are of arity 0. It is assumed that $\Pi \cap \Sigma = \emptyset$.

- Δ is a set of production rules of the form $\alpha \to \tau$ with $\alpha \in \Pi$ and $\tau \in \mathbf{T}(\Sigma \cup \Pi)$ where $\mathbf{T}(\Sigma \cup \Pi)$ is the set of all terms over $\Sigma \cup \Pi$. Terms in $\mathbf{T}(\Sigma \cup \Pi)$ will be called pure type terms.

EXAMPLE 1. *Let* $\Sigma = \{0, s(), nil, cons(,)\}$ *and* $\Pi = \{Nat, NatList\}$. $\mathsf{G} = \langle \Pi, \Sigma, \Delta\rangle$ *defines natural numbers and lists of natural numbers where*

$$\Delta = \left\{ \begin{array}{l} Nat \to 0, \\ Nat \to s(Nat), \\ Natlist \to nil, \\ Natlist \to cons(Nat, Natlist) \end{array} \right\}$$

∎

The above presentation is slightly different from [5] where production rules with the same type symbol on their lefthand sides are grouped together and called a type rule. For instance, production rules in example 1 are grouped into two type rules $Nat \to \{Nat, Natlist\}$ and $Natlist \to \{nil, cons(Nat, Natlist)\}$.

Types denoted by a pure type term is given by a rewrite rule $\Rightarrow_{\mathsf{G}}$ associated with G. $t \Rightarrow_{\mathsf{G}} s$ if Δ contains a rule $\alpha \to \tau$, α occurs in t and s results from replacing an occurrence of α in t by τ. Let $\Rightarrow_{\mathsf{G}}^*$ be the reflexive and transitive closure of $\Rightarrow_{\mathsf{G}}$. The type denoted by a pure type term τ is defined as follows.

$$[\![\tau]\!]_{\mathsf{G}} \stackrel{def}{=} \{t \in \mathbf{T}(\Sigma) \mid \tau \Rightarrow_{\mathsf{G}}^* t\}$$

$[\![\tau]\!]_{\mathsf{G}}$ is the set of terms over Σ that can be derived from τ by repeatedly replacing the lefthand side of a rule in Δ with its righthand side.

EXAMPLE 2. *Let* G *be the regular term grammar in example 1. We have*

$$\begin{array}{lll} Natlist & \Rightarrow_{\mathsf{G}} & cons(Nat, Natlist) \\ & \Rightarrow_{\mathsf{G}} & cons(s(Nat), Natlist) \\ & \Rightarrow_{\mathsf{G}} & cons(s(0), Natlist) \\ & \Rightarrow_{\mathsf{G}} & cons(s(0), nil) \end{array}$$

Thus, $[\![Natlist]\!]_{\mathsf{G}}$ *contains* $cons(s(0), nil)$ *among others.* ∎

The type represented by a sequence ψ of pure type terms and the type represented by a set Ψ of sequences of pure type terms are defined as follows.

$$\begin{array}{rcl} [\![\epsilon]\!]_{\mathsf{G}} & \stackrel{def}{=} & \{\epsilon\} \\ [\![\langle\tau\rangle + \psi']\!]_{\mathsf{G}} & \stackrel{def}{=} & [\![\tau]\!]_{\mathsf{G}} \times [\![\psi']\!]_{\mathsf{G}} \\ [\![\Psi]\!]_{\mathsf{G}} & \stackrel{def}{=} & \bigcup_{\psi \in \Psi} [\![\psi]\!]_{\mathsf{G}} \end{array}$$

where ϵ is the empty sequence, $+$ is the infix sequence concatenation operator, $\langle\tau\rangle$ is the sequence consisting of the pure type term τ and $\times$ is the Cartesian product operator.

The set Π of nonterminals in Dart and Zobel's type language also contains constant type symbols. Constant type symbols are not defined by production rules and they denote constant types. In particular, Π contains μ denoting the set of all terms over Σ and ϕ denoting the empty set of terms. We will leave out constant type symbols in this paper in order to simplify presentation. Re-introducing constant type symbols will not affect the results of the paper.

Dart-Zobel algorithm works with simplified regular term grammars. A regular term grammar $\mathsf{G} = \langle \Pi, \Sigma, \Delta\rangle$ is simplified if $[\![\alpha]\!]_{\mathsf{G}} \neq \emptyset$ for each $\alpha \in \Pi$ and $\tau \notin \Pi$ for each $(\alpha \to \tau) \in \Delta$. For every regular grammar $\mathsf{G} = \langle \Pi, \Sigma, \Delta\rangle$, there is a simplified $\mathsf{G}' = \langle \Pi', \Sigma, \Delta'\rangle$ such that $\Pi' \subseteq \Pi$ and G and G' are equivalent in the sense $\forall \tau \in \mathbf{T}(\Pi \cup \Sigma).[\![\tau]\!]_{\mathsf{G}} = [\![\tau]\!]_{\mathsf{G}'}$.

3. DART-ZOBEL ALGORITHM

This section recalls Dart and Zobel's inclusion algorithm for regular types. As indicated in section 2, we shall disregard constant type symbols and simplify their algorithm accordingly. We note that without constant type symbols, many functions in their algorithm can be greatly simplified. In place of a type rule, we use the corresponding set of production rules. These superficial changes don't change the essence of the algorithm but facilitate the presentation. We shall assume that G is a simplified regular term grammar and omit references to G where there is no confusion.

We first describe the ancillary functions used in their algorithm. Let $\psi = \tau_1\tau_2\cdots\tau_n$ be a non-empty sequence of pure type terms and Ψ be a set of non-empty sequences of pure type terms. $head(\psi) \stackrel{def}{=} \tau_1$ and $tail(\psi) \stackrel{def}{=} \tau_2\cdots\tau_n$. Define $heads(\Psi) \stackrel{def}{=} \{head(\psi) \mid \psi \in \Psi\}$ and $tails(\Psi) \stackrel{def}{=} \{tail(\psi) \mid \psi \in \Psi\}$. The function *expand* rewrites a non-empty sequence into a set of sequences when necessary.

$$\begin{array}{l} expand(\psi) \stackrel{def}{=} \\ \left\{ \begin{array}{ll} \{\psi\} & \text{if } head(\psi) \notin \Pi \\ \{\langle\tau\rangle + tail(\psi) \mid (head(\psi) \to \tau) \in \Delta\} & \\ & \text{if } head(\psi) \in \Pi \end{array} \right. \\ expands(\Psi) \stackrel{def}{=} \bigcup_{\psi \in \Psi} expand(\psi) \end{array}$$

The function $selects(\tau, \Psi)$ defined below applies when τ is pure type term and $\tau \notin \Pi$ and Ψ is a set of non-empty sequences with $heads(\Psi) \cap \Pi = \emptyset$. The output of $selects(\tau, \Psi)$ is the set of the sequences in Ψ that have the same principal function symbol as τ.

$$\begin{array}{l} selects(f(\tau_1, \cdots, \tau_n), \Psi) \\ \quad \stackrel{def}{=} \{\psi \in \Psi \mid head(\psi) = f(\omega_1, \cdots, \omega_n)\} \end{array}$$

Note that $f(\tau_1, \cdots, \tau_n)$ is a constant when $n = 0$.

The function $open(\psi')$ defined below applies when ψ' is a non-empty sequence with $head(\psi') \notin \Pi$. $open(\psi')$ replaces the head of ψ' with its arguments.

$$open(f(\tau_1, \cdots, \tau_n) + \psi) \stackrel{def}{=} \tau_1\tau_2\cdots\tau_n + \psi$$

When $n = 0$, $open(f(\tau_1, \cdots, \tau_n) + \psi) = \psi$. Without constant type symbols, *open* doesn't need an extra argument as in [5] that is used to test membership of a term in a constant type and to indicate the required number of arguments when the constant type symbol is μ. $opens(\Psi) \stackrel{def}{=} \{open(\psi) \mid \psi \in \Psi\}$.

The inclusion algorithm $subset(\tau_1, \tau_2)$ takes two pure type terms τ_1 and τ_2 and is intended to decide if $[\![\tau_1]\!]_{\mathsf{G}} \subseteq [\![\tau_2]\!]_{\mathsf{G}}$ is true or false. The core part $subsetv$ of the inclusion algorithm takes a sequence ψ of pure type terms and a set Ψ of sequences of pure type terms that are of the same length as ψ and is intended to decide if $[\![\psi]\!]_{\mathsf{G}} \subseteq [\![\Psi]\!]_{\mathsf{G}}$. $subsetv$ takes a third argument C to ensure termination. C is a set of pairs $\langle\beta, \Upsilon\rangle$ where $\beta \in \Pi$ is a type symbol and $\Upsilon \subseteq \mathbf{T}(\Sigma \cup \Pi)$ is a set of pure type terms. A pair $\langle\beta, \Upsilon\rangle$ in C can be read as $[\![\beta]\!]_{\mathsf{G}} \subseteq [\![\Upsilon]\!]_{\mathsf{G}}$.

The functions $subset$ and $subsetv$ are defined in the following. Where several alternative definitions of $subsetv$ apply, the first is used.

$$subset(\tau_1, \tau_2) \stackrel{def}{=} subsetv(\langle\tau_1\rangle, \{\langle\tau_2\rangle\}, \emptyset)$$

$$subsetv(\psi, \Psi, C) \stackrel{def}{=}$$
$$\begin{cases} false & \text{if } \Psi = \emptyset \\ true & \text{if } \psi = \epsilon \\ subsetv(tail(\psi), tails(\Psi), C) & \\ \quad \text{if } \langle head(\psi), \Upsilon\rangle \in C \text{ and } heads(\Psi) \supseteq \Upsilon & \\ \forall \psi' \in expand(\psi).subsetv(\psi', \Psi, C \cup \{\langle head(\psi), heads(\Psi)\rangle\}) & \\ \quad \text{if } head(\psi) \in \Pi & \\ subsetv(open(\psi), opens(selects(head(\psi), expands(\Psi))), C) & \\ \quad \text{if } head(\psi) = f(\tau_1, \cdots, \tau_n) & \end{cases}$$

The second condition $heads(\Psi) \supseteq \Upsilon$ for the third alternative is obviously mistaken to be $heads(\Psi) \subseteq \Upsilon$ in [5]. The first two alternatives deal with two trivial cases. The third alternative uses pairs in C to force termination. As we shall see later, this is fine for tuple distributive regular types but is problematic for regular types in general. The fourth alternative expands ψ into a set of sequences ψ' and compares each of them with Ψ. The fifth alternative applies when $\psi = f(\tau_1, \cdots, \tau_n) + \psi'$. Sequences in Ψ are expanded and the expanded sequences of the form $f(\sigma_1, \cdots, \sigma_n) + \omega'$ are selected. ψ and the set of the selected sequences are then compared after replacing $f(\tau_1, \cdots, \tau_n)$ with $\tau_1 \cdots \tau_n$ in ψ and replacing $f(\sigma_1, \cdots, \sigma_n)$ with $\sigma_1 \cdots \sigma_n$ in each $f(\sigma_1, \cdots, \sigma_n) + \omega'$.

4. CORRECTNESS AND COMPLETENESS

We now address the correctness and the completeness of Dart-Zobel algorithm. We first show that the algorithm is incorrect by means of a counterexample.

4.1 Correctness - a counterexample

EXAMPLE 3. *Let* $\mathsf{G} = \langle\Pi, \Sigma, \Delta\rangle$ *with* $\Pi = \{\alpha, \beta, \theta, \sigma, \omega\}$, $\Sigma = \{a, b, g(), h(,)\}$ *and*

$$\Delta = \left\{ \begin{array}{ll} \alpha \to g(\omega), & \beta \to g(\theta) \mid g(\sigma), \\ \theta \to a \mid h(\theta, a), & \sigma \to b \mid h(\sigma, b), \\ \omega \to a \mid b \mid h(\omega, a) \mid h(\omega, b) & \end{array} \right\}$$

where, for instance, $\theta \to a \mid h(\theta, a)$ *is an abbreviation of two rules* $\theta \to a$ *and* $\theta \to h(\theta, a)$. *Let* $\Sigma_h = \Sigma \setminus \{h\}$. *We have*

$$\begin{array}{lcl} [\![\theta]\!]_{\mathsf{G}} & = & \{t \in \mathbf{T}(\Sigma_h) | t \textit{ is left-skewed and leaves of } t \textit{ are } a\text{'s}\} \\ [\![\sigma]\!]_{\mathsf{G}} & = & \{t \in \mathbf{T}(\Sigma_h) | t \textit{ is left-skewed and leaves of } t \textit{ are } b\text{'s}\} \\ [\![\omega]\!]_{\mathsf{G}} & = & \{t \in \mathbf{T}(\Sigma_h) | t \textit{ is left-skewed}\} \\ [\![\alpha]\!]_{\mathsf{G}} & = & \{g(t) \mid t \in [\![\omega]\!]_{\mathsf{G}}\} \\ [\![\beta]\!]_{\mathsf{G}} & = & \{g(t) \mid t \in [\![\theta]\!]_{\mathsf{G}} \cup [\![\sigma]\!]_{\mathsf{G}}\} \end{array}$$

Let $t = g(h(h(a, b), a))$. $t \in [\![\alpha]\!]_{\mathsf{G}}$ *and* $t \notin [\![\beta]\!]_{\mathsf{G}}$. *Therefore,* $[\![\alpha]\!]_{\mathsf{G}} \not\subseteq [\![\beta]\!]_{\mathsf{G}}$. *The incorrectness of Dart-Zobel algorithm is illustrated by showing* $subset(\alpha, \beta) = true$ *as follows. Let* $C_0 = \{\langle\alpha, \{\beta\}\rangle\}$. *We have*

$$\begin{array}{rcl} subset(\alpha, \beta) & = & subsetv(\langle\alpha\rangle, \{\langle\beta\rangle\}, \emptyset) \\ & & \textit{by def. of subset} \\ & = & subsetv(\langle g(\omega)\rangle, \{\langle\beta\rangle\}, C_0) \\ & & \textit{by 4th def. of subsetv} \\ & = & subsetv(\langle\omega\rangle, \{\langle\theta\rangle, \langle\sigma\rangle\}, C_0) \\ & & \textit{by 5th def. of subsetv} \end{array}$$

Let $C_1 = C_0 \cup \{\langle\omega, \{\theta, \sigma\}\rangle\}$. *By the fourth definition of subsetv and the above equation,*

$$subset(\alpha, \beta) = \left(\begin{array}{rr} & subsetv(\langle a\rangle, \{\langle\theta\rangle, \langle\sigma\rangle\}, C_1) \\ \wedge & subsetv(\langle b\rangle, \{\langle\theta\rangle, \langle\sigma\rangle\}, C_1) \\ \wedge & subsetv(\langle h(\omega, a)\rangle, \{\langle\theta\rangle, \langle\sigma\rangle\}, C_1) \\ \wedge & subsetv(\langle h(\omega, b)\rangle, \{\langle\theta\rangle, \langle\sigma\rangle\}, C_1) \end{array} \right) \quad (1)$$

By applying the fifth and then the second definitions of subsetv, $subsetv(\langle a\rangle, \{\langle\theta\rangle, \langle\sigma\rangle\}, C_1) = subsetv(\epsilon, \{\epsilon\}, C_1) = true$. *In the same way, we obtain* $subsetv(\langle a\rangle, \{\langle\theta\rangle, \langle\sigma\rangle\}, C_1) = true$.

$$\begin{array}{ll} \multicolumn{2}{l}{subsetv(\langle h(\omega, a)\rangle, \{\langle\theta\rangle, \langle\sigma\rangle\}, C_1)} \\ = & subsetv(\langle\omega, a\rangle, \{\langle\theta, a\rangle, \langle\sigma, b\rangle\}, C_1) \\ & \textit{by 5th def. of subsetv} \\ = & subsetv(\langle a\rangle, \{\langle a\rangle, \langle b\rangle\}, C_1) \\ & \textit{by 3rd def. of subsetv} \\ = & subsetv(\epsilon, \{\epsilon\}, C_1) \\ & \textit{by 5th def. of subsetv} \\ = & true \\ & \textit{by 2nd def. of subsetv} \end{array}$$

We can show $subsetv(\langle h(\omega, a)\rangle, \{\langle\theta\rangle, \langle\sigma\rangle\}, C_1) = true$ *in the same way as above. Therefore, by equation 1,* $subset(\alpha, \beta) = true$ *and subset is incorrect for regular types.* ∎

The problem with the algorithm stems from the way the set C is used in the third definition of $subsetv$. As the above example indicates, the third definition of $subsetv$ severs the dependency between the terms in a tuple, i.e., subterms of a term.

In [5], Dart and Zobel show by an example that their algorithm works for some regular types which are not tuple distributive. We don't know what is the largest subclass of the class of regular types for which the algorithm is correct.

4.2 Completeness

We now prove that Dart-Zobel algorithm is complete for regular types in the sense that $subset(\tau_1, \tau_2) = true$ whenever $[\![\tau_1]\!]_{\mathsf{G}} \subseteq [\![\tau_2]\!]_{\mathsf{G}}$. Let C be a set of pairs $\langle \beta, \Upsilon \rangle$ with $\beta \in \Pi$ and $\Upsilon \subseteq \mathbf{T}(\Sigma \cup \Pi)$. A pair $\langle \beta, \Upsilon \rangle$ in C states that the denotation of β is included in that of Υ, i.e., $[\![\beta]\!]_{\mathsf{G}} \subseteq [\![\Upsilon]\!]_{\mathsf{G}}$ for regular types. Define

$$\Gamma_{C,\mathsf{G}} \stackrel{def}{=} \wedge_{\langle \beta, \Upsilon \rangle \in C} [\![\beta]\!]_{\mathsf{G}} \subseteq [\![\Upsilon]\!]_{\mathsf{G}}$$

The completeness of *subset* follows from the following theorem which asserts the completeness of *subsetv*.

THEOREM 4. *Let ψ be a sequence of pure type terms and Ψ a set of sequences of pure type terms of the same length as ψ, C a set of pairs $\langle \beta, \Upsilon \rangle$ with $\beta \in \Pi$ and $\Upsilon \subseteq \mathbf{T}(\Sigma \cup \Pi)$. If $\Gamma_{C,\mathsf{G}} \models [\![\psi]\!]_{\mathsf{G}} \subseteq [\![\Psi]\!]_{\mathsf{G}}$ then $subsetv(\psi, \Psi, C) = true$.*

PROOF. *Assume $subsetv(\psi, \Psi, C) = false$. The proof is done by showing $\Gamma_{C,\mathsf{G}} \not\models [\![\psi]\!]_{\mathsf{G}} \subseteq [\![\Psi]\!]_{\mathsf{G}}$. This is accomplished by induction on $\langle dp(\psi, \Psi, C), lg(\psi) \rangle$ where $lg(\psi)$ is the length of ψ and $dp(\psi, \Psi, C)$ is the depth of the computation tree for $subsetv(\psi, \Psi, C)$. Define $\langle k, l \rangle < \langle k', l' \rangle \stackrel{def}{=} (k < k') \vee (k = k') \wedge (l < l')$.*

Basis. $dp(\psi, \Psi, C) = 0$ and $lg(\psi) = 0$. $\psi = \epsilon$ and $\Psi = \emptyset$ since $subsetv(\psi, \Psi, C) = false$. Let $\mathbf{t} = \epsilon$. $\mathbf{t} \in [\![\psi]\!]_{\mathsf{G}}$ and $\mathbf{t} \notin [\![\Psi]\!]_{\mathsf{G}}$. So, $\Gamma_{C,\mathsf{G}} \not\models [\![\psi]\!]_{\mathsf{G}} \subseteq [\![\Psi]\!]_{\mathsf{G}}$.

Induction. $dp(\psi, \Psi, C) \neq 0$ or $lg(\psi) \neq 0$. By the definition of subsetv,

(a) $\Psi = \emptyset$; *or*

(b) $subsetv(tail(\psi), tails(\Psi), C) = false$ *and there is* $\Upsilon \subseteq \mathbf{T}(\Sigma \cup \Pi)$ *such that* $(\langle head(\psi), \Upsilon \rangle \in C) \wedge (heads(\Psi) \supseteq \Upsilon)$; *or*

(c) $head(\psi) \in \Pi$ *and* $\exists . \psi' \in expand(\psi).subsetv(\psi', \Psi, C') = false$ *where* $C' = C \cup \{\langle head(\psi), heads(\Psi) \rangle\}$; *or*

(d) $head(\psi) = f(\tau_1, \cdots, \tau_n)$ *and* $subsetv(\psi', \Psi', C) = false$ *where* $\psi' = open(\psi)$ *and* $\Psi' = opens(selects(head(\psi), expands(\Psi)))$.

It remains to prove that $\Gamma_{C,\mathsf{G}} \not\models [\![\psi]\!]_{\mathsf{G}} \subseteq [\![\Psi]\!]_{\mathsf{G}}$ in each of the cases (a)-(d). The case (a) is trivial as G is simplified and hence $[\![\psi]\!]_{\mathsf{G}} \neq \emptyset$.

In the case (b), we have $dp(tail(\psi), tails(\Psi), C) \leq dp(\psi, \Psi, C)$ and $lg(tail(\psi)) < lg(\psi)$. By the induction hypothesis, $\Gamma_{C,\mathsf{G}} \not\models [\![tail(\psi)]\!]_{\mathsf{G}} \subseteq [\![tails(\Psi)]\!]_{\mathsf{G}}$. Thus, $\Gamma_{C,\mathsf{G}} \models \exists \mathbf{t}'.(\mathbf{t}' \in [\![tail(\psi)]\!]_{\mathsf{G}} \wedge \mathbf{t}' \notin [\![tails(\Psi)]\!]_{\mathsf{G}})$. Let $t \in [\![head(\psi)]\!]_{\mathsf{G}}$ and $\mathbf{t} = \langle t \rangle + \mathbf{t}'$. Note that t exists as G is simplified. We have $\Gamma_{C,\mathsf{G}} \models \mathbf{t} \in [\![\psi]\!]_{\mathsf{G}} \wedge \mathbf{t} \notin [\![\Psi]\!]_{\mathsf{G}}$. So, $\Gamma_{C,\mathsf{G}} \not\models [\![\psi]\!]_{\mathsf{G}} \subseteq [\![\Psi]\!]_{\mathsf{G}}$.

In the case (c), $dp(\psi', \Psi, C') < dp(\psi, \Psi, C)$. By the induction hypothesis, $\Gamma_{C'\mathsf{G}} \not\models [\![\psi']\!]_{\mathsf{G}} \subseteq [\![\Psi]\!]_{\mathsf{G}}$. Note that $\Gamma_{C',\mathsf{G}} = \Gamma_{C,\mathsf{G}} \wedge ([\![head(\psi)]\!]_{\mathsf{G}} \subseteq [\![heads(\Psi)]\!]_{\mathsf{G}})$. So, we have $\Gamma_{C,\mathsf{G}} \models [\![\psi]\!]_{\mathsf{G}} \not\subseteq [\![\Psi]\!]_{\mathsf{G}} \vee [\![head(\psi)]\!]_{\mathsf{G}} \not\subseteq [\![heads(\Psi)]\!]_{\mathsf{G}}$. Assume $\Gamma_{C,\mathsf{G}} = true$. Either (i) $[\![\psi]\!]_{\mathsf{G}} \not\subseteq [\![\Psi]\!]_{\mathsf{G}}$ or (ii) $[\![head(\psi)]\!]_{\mathsf{G}} \not\subseteq [\![heads(\Psi)]\!]_{\mathsf{G}}$. In the case (i), $\exists \mathbf{t}'.(\mathbf{t}' \in [\![\psi']\!]_{\mathsf{G}}) \wedge (\mathbf{t}' \notin [\![\Psi]\!]_{\mathsf{G}})$. By proposition 5.26 in [5], we have $\Gamma_{C,\mathsf{G}} \not\models [\![\psi]\!]_{\mathsf{G}} \subseteq [\![\Psi]\!]_{\mathsf{G}}$. In the case (ii), $\exists t.(t \in [\![head(\psi)]\!]_{\mathsf{G}}) \wedge (t \notin [\![heads(\Psi)]\!]_{\mathsf{G}})$. Let $\mathbf{t}' \in [\![tail(\psi)]\!]_{\mathsf{G}}$ and $\mathbf{t} = \langle t \rangle + \mathbf{t}'$. Note that $\mathbf{t}'$ exists as G is simplified. We have $\mathbf{t} \in [\![\psi]\!]_{\mathsf{G}} \wedge \mathbf{t} \notin [\![\Psi]\!]_{\mathsf{G}}$. So, $\Gamma_{C,\mathsf{G}} \not\models [\![\psi]\!]_{\mathsf{G}} \subseteq [\![\Psi]\!]_{\mathsf{G}}$ in the case (c).

In the case (d), we have $\psi' = \tau_1 \cdots \tau_n + tail(\psi)$ and $dp(\psi', \Psi', C) < dp(\psi, \Psi, C)$. By the induction hypothesis, $\Gamma_{C,\mathsf{G}} \not\models [\![\psi']\!]_{\mathsf{G}} \subseteq [\![\Psi']\!]_{\mathsf{G}}$. Thus, $\Gamma_{C,\mathsf{G}} \models \exists \mathbf{t}_1.\exists \mathbf{t}_2.(lg(\mathbf{t}_1) = n) \wedge ((\mathbf{t}_1 + \mathbf{t}_2) \in [\![\psi']\!]_{\mathsf{G}}) \wedge ((\mathbf{t}_1 + \mathbf{t}_2) \notin [\![\Psi']\!]_{\mathsf{G}})$, which implies $\Gamma_{C,\mathsf{G}} \models \exists \mathbf{t}_1.\exists \mathbf{t}_2.(\langle f(\mathbf{t}_1) \rangle + \mathbf{t}_2) \in [\![\psi]\!]_{\mathsf{G}}) \wedge ((\langle f(\mathbf{t}_1) \rangle + \mathbf{t}_2) \notin [\![\Psi]\!]_{\mathsf{G}})$. So, $\Gamma_{C,\mathsf{G}} \not\models [\![\psi]\!]_{\mathsf{G}} \subseteq [\![\Psi]\!]_{\mathsf{G}}$. □

The completeness of *subset* is a corollary of the above theorem.

COROLLARY 5. *Let τ_1 and τ_2 be pure type terms. If $[\![\tau_1]\!]_{\mathsf{G}} \subseteq [\![\tau_2]\!]_{\mathsf{G}}$ then $subset(\tau_1, \tau_2) = true$.*

PROOF. *$subset(\tau_1, \tau_2) = subsetv(\langle \tau_1 \rangle, \{\langle \tau_2 \rangle\}, \emptyset)$ by the definition of subset. We have $\Gamma_{\emptyset,\mathsf{G}} \models [\![\langle \tau_1 \rangle]\!]_{\mathsf{G}} \subseteq [\![\{\langle \tau_2 \rangle\}]\!]_{\mathsf{G}}$ since $[\![\tau_1]\!]_{\mathsf{G}} \subseteq [\![\tau_2]\!]_{\mathsf{G}}$. The corollary now follows from the above theorem as $\Gamma_{\emptyset,\mathsf{G}} = true$.* □

5. CONCLUSION

We have provided answers to open questions about the correctness and the completeness of Dart-Zobel algorithm for testing inclusion of one regular type in another. The algorithm is complete but incorrect for regular types.

6. REFERENCES

[1] L. Cardelli and P. Wegner. On understanding types, data abstraction, and polymorphism. *ACM computing surveys*, 17(4):471–522, 1985.

[2] M. Codish and V. Lagoon. Type dependencies for logic programs using ACI-unification. In *Proceedings of the 1996 Israeli Symposium on Theory of Computing and Systems*, pages 136–145. IEEE Press, June 1996.

[3] H. Comon, M. Dauchet, R. Gilleron, D. Lugiez, S. Tison, and M. Tommasi. *Tree Automata Techniques and Applications*. Draft, 1998.

[4] P.W. Dart and J. Zobel. Efficient run-time type checking of typed logic programs. *Journal of Logic Programming*, 14(1-2):31–69, 1992.

[5] P.W. Dart and J. Zobel. A regular type language for logic programs. In F. Pfenning, editor, *Types in Logic Programming*, pages 157–189. The MIT Press, 1992.

[6] S. K. Debray, P. López-Garcia, and M. Hermenegildo. Non-failure analysis for logic programs. In Lee Naish, editor, *Logic programming: Proceedings of the fourteenth International Conference on Logic Programming*, pages 48–62. The MIT Press, July 1997.

[7] T. Fruhwirth, E. Shapiro, M.Y. Vardi, and E. Yardeni. Logic programs as types for logic programs. In *Proceedings of Sixth Annual IEEE Symposium on Logic in Computer Science*, pages 300–309. The IEEE Computer Society Press, 1991.

[8] J.P. Gallagher and D.A. de Waal. Fast and precise regular approximations of logic programs. In M. Bruynooghe, editor, *Proceedings of the Eleventh International Conference on Logic Programming*, pages 599–613. The MIT Press, 1994.

[9] F. Gécseg and M. Steinby. *Tree Automata.* Akadémiai Kiadó, 1984.

[10] N. Heintze and J. Jaffar. A finite presentation theorem for approximating logic programs. In *Proceedings of the seventh Annual ACM Symposium on Principles of Programming Languages*, pages 197–209. The ACM Press, 1990.

[11] N. Heintze and J. Jaffar. Semantic types for logic programs. In F. Pfenning, editor, *Types in Logic Programming*, pages 141–155. The MIT Press, 1992.

[12] D. Jacobs. Type declarations as subtype constraints in logic programming. *SIGPLAN Notices*, 25(6):165–73, 1990.

[13] L. Lu. Type analysis of logic programs in the presence of type definitions. In *Proceedings of the 1995 ACM SIGPLAN Symposium on Partial Evaluation and Semantics-Based program manipulation*, pages 241–252. The ACM Press, 1995.

[14] L. Lu. A polymorphic type analysis in logic programs by abstract interpretation. *Journal of Logic Programming*, 36(1):1–54, 1998.

[15] P. Mishra. Towards a theory of types in Prolog. In *Proceedings of the IEEE international Symposium on Logic Programming*, pages 289–298. The IEEE Computer Society Press, 1984.

[16] U.S. Reddy. Types for logic programs. In *Proceedings of the 1990 North American Conference on Logic Programming*, pages 836–40. The MIT Press, 1990.

[17] M. Soloman. Type definitions with parameters. In *Conference Record of the Fifth ACM Symposium on Principles of Programming Languages*, pages 31–38, 1978.

[18] E. Yardeni, T. Fruehwirth, and E. Shapiro. Polymorphically typed logic programs. In K. Furukawa, editor, *Logic Programming. Proceedings of the Eighth International Conference*, pages 379–93. The MIT Press, 1991.

[19] E. Yardeni and E. Shapiro. A type system for logic programs. *Journal of Logic Programming*, 10(2):125–153, 1991.

[20] J. Zobel. Derivation of polymorphic types for Prolog programs. In J.-L. Lassez, editor, *Logic Programming: Proceedings of the fourth international conference*, pages 817–838. The MIT Press, 1987.