universität innsbruck

Seminar Report

# Conditional Nonreachability with Tree Automata

Florian Meßner
florian.g.messner@uibk.ac.at

25 February 2018

**Supervisor:** Dr. Christian Sternagel

**Abstract**

This report describes the use of tree automata for reachability analysis of term rewrite systems, including an extension for conditional term rewrite systems. It includes explanations of a completion algorithm for tree automata to create over and underapproximations of regular languages.

# Contents

**1 Introduction** **1**

**2 Tree Automata** **1**

**3 Tree Automata Completion** **3**

**4 Completion Properties** **4**

**5 Reachability Analysis** **6**

**6 Conditional Term Rewriting** **7**

**7 Conclusion** **8**
7.1 Future Work . . . . . 8

**Bibliography** **9**

# 1 Introduction

Reachability analysis over term rewrite systems is an interesting and useful topic for several purposes including termination and confluence analysis. As term rewrite systems are a Turing-complete computation model [1] they can be used to represent any Turing-computable program. In particular there have been successful attempts to apply termination analysis over term rewrite systems to Java Bytecode [5] and Haskell [3] programs.

An approach presented by Feuillade, Genet and Tong [2] employs tree automata and tree automata completion to approximate reachable terms. This allows for reachability analysis, as terms not part of an overapproximation can never be reached and terms part of an underapproximation are reachable.

In this report I give a brief introduction to tree automata for readers unfamiliar with the topic in Section 2. I present the tree automata completion algorithm in Section 3 alongside necessary auxiliary function. Afterwards I talk about conditions for over and underapproximation of the completed automaton in Section 4. I briefly describe reachability in term rewriting and how tree automata completion can be used for nonreachability analysis in Section 5. Finally I elaborate on conditional term rewriting and an extension of tree automata completion for it in Section 6.

# 2 Tree Automata

In this section I attempt to give readers who are not familiar with tree automata a brief introduction to the topic. I assume a basic understanding of term rewriting [1].

Before looking at tree automata I want to clarify some notation used in this report. A finite set of function symbols with an associated nonnegative arity is $\mathcal{F}$. A finite set disjoint from $\mathcal{F}$ of state symbols with arity zero is $\mathcal{Q}$. A countable (but not necessarily finite) set of variables is $\mathcal{X}$. The set of terms over $\mathcal{F}$ and $\mathcal{X}$ is denoted as $\mathcal{T}(\mathcal{F}, \mathcal{X})$, whereas the set of ground terms, that is terms without variables is denoted as $\mathcal{T}(\mathcal{F})$. A configuration is a ground term over function symbols and states, the set of configurations is denoted by $\mathcal{T}(\mathcal{F} \cup \mathcal{Q})$.

The set of $\mathcal{R}$-descendants of a set of ground terms $E$ with respect to a term rewrite system $\mathcal{R}$ is defined as the set of all ground terms $t$ which can be reached from some term $s \in E$.

**Definition 2.1.** *$\mathcal{R}$-descendants* of $E$: $\mathcal{R}^*(E) = \{t \in \mathcal{T}(\mathcal{F}) \mid \exists s \in E \text{ s.t. } s \to_{\mathcal{R}}^* t\}$

Furthermore I define regular language substitutions as functions $\sigma : \mathcal{X} \mapsto \mathcal{Q}$ mapping variables to states. The set of all such substitutions over $\mathcal{Q}$ and $\mathcal{X}$ is denoted by $\Sigma(\mathcal{Q}, \mathcal{X})$.

With this basis I introduce some more requirements for defining tree automata.

**Definition 2.2.** A *transition* is an ordered pair of configurations and states: $c \to q$ for some $c \in \mathcal{T}(\mathcal{F} \cup \mathcal{Q})$ and $q \in \mathcal{Q}$.

**Definition 2.3.** *Normalized transitions* are a specialization of transitions, further restricting their left-hand side: $c \to q$ for some $q \in \mathcal{Q}$ and either

- $c = q' \in \mathcal{Q}$ is a state or
- $c = f(q_1, \ldots, q_n)$ where $q_1, \ldots, q_n \in \mathcal{Q}$ are states and $f$ is a function symbol of arity $n$ ($f \in \mathcal{F}, ar(f) = n$).

While tree automata require transitions to be normalized, other transitions will appear in intermediate steps of adding transitions to an automaton where they have to be normalized.

To make things more clear we look at a few different transitions in an example.

**Example 2.4.** Some example transitions for states $q_0, q_1, q_2, q_3$ and function symbols $f, g, h$ are $q_1 \to q_0$ and $h(q_2, q_3) \to q_1$ which are normalized or $h(q_1, f(q_2)) \to q_0$ and $f(g(q_1)) \to q_0$ which are not.

Having introduced all requirements I can now define tree automata.

**Definition 2.5.** A *bottom-up nondeterministic finite tree automaton* is a quadruple $\mathcal{A} = \langle \mathcal{F}, \mathcal{Q}, \mathcal{Q}_f, \Delta \rangle$ of function symbols, states, final states which are a subset of states ($\mathcal{Q}_f \subseteq \mathcal{Q}$) and a set of normalized transitions.

In the remainder I will refer to bottom-up nondeterministic finite tree automata simply as tree automata as they are the only type of tree automata I use. From the set of normalized transitions $\Delta$ I derive a rewriting relation on configurations $\to_{\mathcal{A}}$.

Finally, we can consider the language of an automaton.

**Definition 2.6.** The *language* recognized by a state $q$ in an automaton $\mathcal{A}$ is the set of all ground terms that can be rewritten to $q$: $\mathcal{L}(\mathcal{A}, q) = \{t \in \mathcal{T}(\mathcal{F}) \mid t \to_{\mathcal{A}}^* q\}$

**Definition 2.7.** The *language* recognized by an automaton $\mathcal{A}$ is the union of the languages recognized by the final states of the automaton: $\mathcal{L}(\mathcal{A}) = \bigcup_{q_f \in \mathcal{Q}_f} \mathcal{L}(\mathcal{A}, q_f)$

In other words the language recognized by an automaton is the set of all ground terms $t$ that can be rewritten to some final state $q_f$ of the automaton.

**Example 2.8.** Consider a tree automaton $\mathcal{A} = \langle \mathcal{F}, \mathcal{Q}, \mathcal{Q}_f, \Delta \rangle$ with $\mathcal{F} = \{f, g, a\}$, $\mathcal{Q} = \{q_0, q_1, q_2\}$, $\mathcal{Q}_f = \{q_0\}$ and $\Delta = \{f(q_0) \to q_0, g(q_1) \to q_0, g(q_2) \to q_2, a \to q_1\}$ We look at some rewrite sequences and properties.

- $g(a) \to_{\mathcal{A}} g(q_1) \to_{\mathcal{A}} q_0$
- $f(g(a)) \to_{\mathcal{A}}^* f(q_0) \to_{\mathcal{A}} q_0$
- $\mathcal{L}(\mathcal{A}, q_1) = \{a\}$
- $\mathcal{L}(\mathcal{A}, q_0) = \{f(g(a)), f(f(g(a))), \ldots\} = \{f^*(g(a))\}$
- $\mathcal{L}(\mathcal{A}, q_2) = \emptyset$, $q_2$ can never be reached and is called dead state.

## 3 Tree Automata Completion

The goal of tree automata completion is, given an automaton $\mathcal{A}_0$, find an automaton $\mathcal{A}'$ that overapproximates the $\mathcal{A}$-descendants of the language of $\mathcal{A}_0$, $\mathcal{L}(\mathcal{A}') \supseteq \mathcal{R}^*(\mathcal{L}(\mathcal{A}_0))$. An abstract algorithm for this purpose, successively creates intermediate automata that overapproximate the previous step, $\mathcal{L}(\mathcal{A}_i) \subseteq \mathcal{L}(\mathcal{A}_{i+1})$ until a fixpoint is reached $\mathcal{L}(\mathcal{A}_k) = \mathcal{L}(\mathcal{A}_{k+1})$.

**Definition 3.1.** A *critical pair* between the rewrite relation of a term rewrite system $\mathcal{R}$ and the rewrite relation of an (intermediate) automaton $\mathcal{A}_i$ is some $l\sigma$ for a substitution $\sigma : \mathcal{X} \mapsto \mathcal{Q}$ and a rule $l \to r \in \mathcal{R}$ such that $l\sigma \to^*_{\mathcal{A}_i} q$ and $r\sigma \twoheadrightarrow^*_{\mathcal{A}_i} q$.

The completion algorithm joins all critical pairs in each completion step, by adding a transition $r\sigma \to q$ to the new automaton $\mathcal{A}_{i+1}$. Due to the requirements of tree automata, we need to normalize the transition first, requiring some more definitions.

**Definition 3.2.** An *abstraction function* $\alpha$ maps normalized configurations in $\mathcal{T}(\mathcal{F} \cup \mathcal{Q})$ to states: $\alpha : \{f(q_1, \ldots, q_n) | f \in \mathcal{F}, ar(f) = n \text{ and } q_1, \ldots, q_n \in \mathcal{Q}\} \mapsto \mathcal{Q}$

**Definition 3.3.** The *abstraction state* $top_\alpha$ maps (not necessarily normalized) configurations in $\mathcal{T}(\mathcal{F} \cup \mathcal{Q})$ to states, using a given abstraction function.

$$top_\alpha(t) = \begin{cases} t & \text{if } t \in \mathcal{Q} \\ top_\alpha(t) = \alpha(f(top_\alpha(t_1), \ldots, top_\alpha(t_n))) & \text{if } t = f(t_1, \ldots, t_n) \end{cases}$$

Given these functions we have a way to map any configuration to a state, which is required for the normalization of an arbitrary transition as can be seen in Example 3.5.

**Definition 3.4.** The *normalization function* $Norm_\alpha$ maps a transition to a set of normalized transitions, using an abstraction function $\alpha$.

$$Norm_\alpha(s \to q) = \begin{cases} \emptyset & \text{if } s = q \\ \{s \to q\} & \text{if } s \in \mathcal{Q} \text{ and } s \neq q \\ \{f(top_\alpha(t_1), \ldots, top_\alpha(t_n)) \to q\} & \\ \quad \cup \bigcup_{i=1}^{n} Norm_\alpha(t_i \to top_\alpha(t_i)) & \text{if } s = f(t_1, \ldots, t_n) \end{cases}$$

**Example 3.5.** We consider a tree automaton $\mathcal{A}$ with $\mathcal{F} = \{f, g, a\}$, $\mathcal{Q} = \{q_0, q_1, q_2, q_3, q_4\}$, $\mathcal{Q}_f = \{q_0\}$ and $\Delta = \{f(q_1) \to q_0, g(q_1, q_1) \to q_1, a \to q_1\}$.

The language recognized by $q_0$ is $\mathcal{L}(\mathcal{A}, q_0) = \{f(x) | x \in \mathcal{L}(\mathcal{A}, q_1)\}$, where the language recognized by $q_1$ is $\mathcal{L}(\mathcal{A}, q_1) = \mathcal{T}(\{g, a\})$, that is all ground terms over $g$ and $a$.

Now consider a term $s = f(g(q_1, f(a)))$ and an abstraction function $\alpha = \{a \mapsto q_4, f(q_4) \mapsto q_3, g(q_1, q_3) \mapsto q_2\}$. Applying the normalization function to the transition $s \to q_0$ yields the following: $Norm_\alpha(f(g(q_1, f(a))) \to q_0) = \{f(q_2) \to q_0, g(q_1, q_3) \to q_2, f(q_4) \to q_3, a \to q_4\}$.

In particular this requires the abstraction state of $f(a)$, which considers the abstraction function for $\alpha(a) = q_4$ and $\alpha(f(q_4)) = q_3$ for the resulting set of normalized transitions.

Now that we have a method to join critical pairs, we look into the details of the completion algorithm. First I define one step completion of an automaton.

**Definition 3.6.** Given a tree automaton $\mathcal{A} = \langle \mathcal{F}, \mathcal{Q}, \mathcal{Q}_f, \Delta \rangle$, a term rewrite system $\mathcal{R}$ and an abstraction function $\alpha$, the *one-step completed automaton* $\mathcal{C}_{\alpha,\mathcal{R}} = \langle \mathcal{F}, \mathcal{Q}', \mathcal{Q}_f, \Delta' \rangle$ uses a new set of normalized transitions $\Delta' = \Delta \cup \bigcup_{l \to r \in \mathcal{R}, q \in \mathcal{Q}, \sigma \in \Sigma(\mathcal{Q},\mathcal{X}), l\sigma \to_\Delta^* q} Norm_\alpha(r\sigma \to q)$ and a new set of states $\mathcal{Q}' = \{q | c \to q \in \Delta'\}$ with the remaining parameters left unchanged.

This defines a one-step completed automaton as an automaton with all critical pairs of a previous automaton joined using normalized transitions and possibly new states due to the normalization process.

Finally, I can define automata completion using these intermediate definitions.

**Definition 3.7.** The *n-step completed automaton* is defined as follows.

- $\mathcal{A}^0_{\alpha,\mathcal{R}} = \mathcal{A}$
- $\mathcal{A}^{n+1}_{\alpha,\mathcal{R}} = \mathcal{C}_{\alpha,\mathcal{R}}(\mathcal{A}^n_{\alpha,\mathcal{R}})$

This defines the 0-step completion as the unchanged input automaton and the $n+1$-step completed automaton is the one-step completion applied to the $n$-step completed automaton. I define the completed automaton as the fixpoint of this completion $\mathcal{A}^k_{\alpha,\mathcal{R}} = \mathcal{A}^{k+1}_{\alpha,\mathcal{R}} = \mathcal{A}^*_{\alpha,\mathcal{R}}$. This fixpoint, and thus the completed automaton, does however not exist in general, as completion often diverges. The abstraction function $\alpha$ can sometimes be adjusted to ensure termination to generate an overapproximation. I will discuss further conditions for over- and underapproximation in Section 4.

# 4 Completion Properties

In this section I aim to explain the conditions necessary for a completed automaton, if it exists, to be an over- or underapproximation.

In their paper Feuillade, Genet and Tong propose several so called coherence conditions sufficient for the desired properties [2].

**Definition 4.1.** A tree automaton $\mathcal{A}$ and a term rewrite system $\mathcal{R}$ satisfy the *left-coherence* condition if
$\forall \tau : \mathcal{X} \mapsto \mathcal{T}(\mathcal{F}), \forall l \to r \in \mathcal{R}, \forall q \in \mathcal{Q}:$
$l\tau \to_\Delta^* q \Rightarrow \exists \sigma \in \Sigma(\mathcal{Q}, \mathcal{X})$ s.t. $l\tau \to_\Delta^* l\sigma \to_\Delta^* q$

Left-linear term rewrite systems satisfy the left-coherence condition. However, non-left-linear term rewrite systems might still be useful for an overapproximation as long as they satisfy the left-coherence condition. We look at the problem with the completion algorithm and non-left-linear term rewrite systems through an example. Suppose $f(x,x) \to g(x)$ is a rule of a term rewrite system $\mathcal{R}$ and the tree automaton $\mathcal{A}$ contains the transitions

$f(q_1, q_1) \to q_0$ and $f(q_2, q_3) \to q_0$. We can find a valid substitution $\sigma = \{x \mapsto q_1\}$ for a critical pair $f(q_1, q_1) \to q_0$ and $g(q_1) \nrightarrow q_0$. It is however not possible to find such a substitution for the second transition. For completion between the given rule and the second transition we would need to consider the common language of $q_2$ and $q_3$. Instead of computing that, we could also determinize the automaton, which can lead to exponential blowup of states. To avoid both calculations for the completion algorithm it is enough to demand left-coherence.

I also define a stricter version of left-coherence, called *simple left-coherence*. For some tree automaton $\mathcal{A}$, a rule $l \to r$ over terms in $\mathcal{T}(\mathcal{F}, \mathcal{X})$, $\{x_1, \ldots, x_k\}$ a set of nonlinear variables in $l$ and $\mathcal{Y}$ a set of variables distinct from $\mathcal{X}$, we define $Ren(l) = (l', E)$ where $l'$ is the left-hand side of the rule with nonlinear variables replaced with variables from $\mathcal{Y}$ and $E$ is a set of equality constraints.

$$Ren(l) = \begin{cases} (l, \emptyset) & \text{if } l \text{ is either a constant or a variable not in } \{x_1, \ldots, x_k\} \\ (y, \{x = y\}) & \text{if } l \text{ is a variable } x \in \{x_1, \ldots, x_k\} \text{ and } y \text{ is a fresh variable of } \mathcal{Y} \\ (f(t'_1, \ldots, t'_n), \bigcup_{i=1}^{n} E_i) & \text{if } l = f(t_1, \ldots, t_n) \text{ and } Ren(t_i) = (t'_i, E_i) \text{ for all } i = 1 \ldots n \ . \end{cases}$$

**Definition 4.2.** An automaton $\mathcal{A}$ and a term rewrite system $\mathcal{R}$ satisfy the simple left-coherence condition if for all rules $l \to_{\mathcal{R}} r$ such that $Ren(l) = (l', E)$:
$\forall (x = y) \in E, \forall \sigma \in \Sigma(\mathcal{Q}, \mathcal{X}), \forall q, q_x, q_y \in \mathcal{Q}$ :
$l'\sigma \to_{\Delta}^{*} q \wedge \sigma(x) = q_x \neq q_y = \sigma(y) \Longrightarrow \mathcal{L}(\mathcal{A}, q_x) \cap \mathcal{L}(\mathcal{A}, q_y) = \emptyset$.

Simple left-coherence implies left coherence, thus it is sufficient to show this property. Symmetrical to left-coherence there is also the right-coherence condition.

**Definition 4.3.** A tree automaton $\mathcal{A}$ and a term rewrite system $\mathcal{R}$ are said to be *right-coherent* if either $\mathcal{R}$ is right-linear, or
$\forall q \in \mathcal{Q} : \exists t \in \mathcal{T}(\mathcal{F}) : \mathcal{L}(\mathcal{A}, q) \subseteq \mathcal{R}^*(t)$.

In particular in a term rewrite system that is not right linear, for all states $q$ there must be some ground term $t$ such that the $\mathcal{R}$-descendants of $t$ overapproximate the language recognized by the state $q$. Note that this condition only considers the initial automaton in contrast to the left-coherence condition.

Lastly, there is also a coherence condition for abstraction functions.

**Definition 4.4.** An abstraction function $\alpha$ is called *coherent* with tree automaton $\mathcal{A}$ and term rewrite system $\mathcal{R}$ if

- for all configurations in its domain $t \in Dom(\alpha)$ and for all states in its range $q \in \mathcal{Q} \cap Ran(\alpha)$
- if $\alpha(t) = q$ then $t \to q \in \mathcal{A}$ and

- there is a ground term $t' \in \mathcal{T}(\mathcal{F})$ such that $t' \to_{\mathcal{A}}^* t$ and $\mathcal{L}(\mathcal{A}, q) \subseteq \mathcal{R}^*(\{t'\})$.

Informally, this condition demands that whenever $\alpha$ maps some configuration $t$ to a state $q$ either q is a fresh state not in $\mathcal{A}$ or terms recognized by $q$ are some term $t'$ which is recognized by $t$, or $\mathcal{R}$-descendants of it.

Using these conditions I can define over- and underapproximation of tree automata completion.

**Lemma 4.5.** *If completion terminates and the term rewrite system $\mathcal{R}$ and the completed tree automaton $\mathcal{A}^*_{\alpha,\mathcal{R}}$ satisfy the left-coherence condition then the language of the completed automaton overapproximates the $\mathcal{R}$-descendants of the language recognized by the original automaton: $\mathcal{L}(\mathcal{A}^*_{\alpha,\mathcal{R}}) \supseteq \mathcal{R}^*(\mathcal{L}(\mathcal{A}))$*

**Lemma 4.6.** *If the term rewrite system $\mathcal{R}$ and the tree automaton $\mathcal{A}$ satisfy the right-coherence condition and $\alpha$ is an injective abstraction function coherent with $\mathcal{R}$ and $\mathcal{A}$ then every step of the completion procedure, including the fixpoint, underapproximates the $\mathcal{R}$-descendants of the language recognized by the original automaton: $\forall n \in \mathbb{N} : \mathcal{L}(\mathcal{A}^n_{\alpha,\mathcal{R}}) \subseteq \mathcal{R}^*(\mathcal{L}(\mathcal{A}))$*

These two results can be combined to define the exact case of tree automata completion.

**Lemma 4.7.** *If the term rewrite system $\mathcal{R}$ and the tree automaton $\mathcal{A}$ satisfy the right-coherence condition, $\alpha$ is an injective abstraction function coherent with $\mathcal{R}$ and $\mathcal{A}$, the completed automaton exists and $\mathcal{R}$ and $\mathcal{A}^*_{\alpha,\mathcal{R}}$ satisfy the left-coherence condition then the completion is exact: $\mathcal{L}(\mathcal{A}^*_{\alpha,\mathcal{R}}) = \mathcal{R}^*(\mathcal{L}(\mathcal{A}))$*

# 5 Reachability Analysis

In this section I describe how tree automata combined with the completion algorithm can be used for reachability analysis in term rewriting. Reachability in general is the question, given terms $s$ and $t$, whether there are some substitutions $\sigma, \tau$ for which $s\sigma \to_{\mathcal{R}}^* t\tau$. In the context of this report we focus on ground terms, so the problem is simply whether $s \to_{\mathcal{R}}^* t$, or more interestingly the nonreachability problem is to find out whether $s \not\to_{\mathcal{R}}^* t$. Determining nonreachability is among other things useful for termination and confluence proofs over term rewrite systems and thus indirectly also for proofs over programs in languages such as Haskell, as mentioned in Section 1.

As described in Section 4, when completion terminates and several conditions are satisfied, the completed automaton recognizes an overapproximation of the $\mathcal{R}$-descendants of the language recognized by an initial automaton. The original language can be replaced by an arbitrary regular language $E$. In particular we can look at the special case $E = \{s\}$, thus $\mathcal{R}^*(E)$ is the set of all ground terms reachable from $s$. Now if $t$ is not a member of an overapproximation of this set, it is also not a member of the $\mathcal{R}$-descendants and thus not reachable from $s$. Hence it is sufficient to check whether the completed automaton accepts $t$.

# 6 Conditional Term Rewriting

This section gives an introduction to conditional term rewriting as defined by Feuillade, Genet and Tong [2], which is just one way to look at conditional term rewrite systems (CTRS). Other types of CTRS will be presented in Section 7. Finally we will present the extension of the completion algorithm to the conditional case.

Essentially, conditional term rewriting is an extension of term rewriting where a subset of the rewrite rules are so called conditional rules. These rules come with a set of, in the context of this paper, joinability conditions. Here the variables of the conditions must be a subset of the variables in the left side of the rule. A conditional rule $l \to r \Leftarrow c_1 \downarrow c_2$ is said to be enabled for a substitution $\sigma$ if $c_1\sigma \downarrow c_2\sigma$, which means that there is some common reduct $u$ such that $c_1\sigma \to^* u$ and $c_2\sigma \to^* u$. Multiple conditions are simply a conjunction of singleton joinability conditions and behave as such.

This yields a rewriting relation $\to_{\mathcal{R}}^{\downarrow n}$ with

- $\to_{\mathcal{R}}^{\downarrow 0} = \to_{\mathcal{R}_{nc}}$ relation over non conditional rules
- $a \to_{\mathcal{R}}^{\downarrow n+1} b \Leftrightarrow a \to_{\mathcal{R}}^{\downarrow 0} b$ or
  $\exists \sigma, p \in Pos(a), (l \to r \text{ if } s \downarrow t) \in \mathcal{R}$ such that
  $a|_p = l\sigma, b = a[r\sigma]_p$ and $\exists u \in \mathcal{T}(\mathcal{F})$ such that
  $s\sigma \to_{\mathcal{R}}^{\downarrow n*} u$ and $t\sigma \to_{\mathcal{R}}^{\downarrow n*} u$

Feuillade, Genet and Tong propose that it is sufficient to extend the tree automata completion algorithm for the conditional case, to support this type of term rewrite system.

This version of the completion algorithm is similar to the previous one, however the set of states in step $i$ is now split $\mathcal{Q}_i = \mathcal{Q}_0 \cup \mathcal{Q}_{i,new} \cup \mathcal{Q}_{i,cond}$ where $\mathcal{Q}_0$ is the initial set of states of $\mathcal{A}_0$, $\mathcal{Q}_{i,new}$ is the set of states added through normalization of transitions and indexed with natural numbers and $\mathcal{Q}_{i,cond}$ is the set of states added for conditional rules and indexed with configurations in $\mathcal{T}(\mathcal{F} \cup \mathcal{Q})$.

The algorithm searches critical pairs without considering conditions. Each of these pairs either involves a conditional or a non-conditional rule. Non-conditional rules are treated as before, this time adding new states from normalization to $\mathcal{Q}_{i+1,new}$. For conditional rules $l \to r$ if $c_1 \downarrow c_2$, if at least one of the states $q_{c_1\sigma}$ and $q_{c_2\sigma}$ are not in $\mathcal{Q}_{i,cond}$, the states are created and added and so are the required normalized transitions $Norm_\alpha(c_1\sigma \to q_{c_1\sigma}) \cup Norm_\alpha(c_2\sigma \to q_{c_2\sigma})$.

Afterwards, or if the states already existed, the intersection of their recognized languages is checked for emptiness $\mathcal{L}(\mathcal{A}_i, q_{c_1\sigma}) \cap \mathcal{L}(\mathcal{A}_i, q_{c_2\sigma}) \neq \emptyset$.

If this set is not empty we know the condition is true and we treat the rule as if it was a non-conditional rule. Otherwise, if the set is empty, we consider this rule not enabled for this step of completion.

Again, we can consider properties of the completion result. For some tree automaton $\mathcal{A}_0$ that is an overapproximation of a regular language $\mathcal{L}(\mathcal{A}_0) \supseteq E$ and a left linear conditional term rewrite system $\mathcal{R}$, if $\mathcal{A}'$ is the result of the completion of $\mathcal{A}_0$ with respect to $\mathcal{R}$, then the language recognized by $\mathcal{A}'$ is an overapproximation of the $\mathcal{R}$-descendants of $E$, $\mathcal{R}^*(E) \subseteq \mathcal{R}^*(\mathcal{L}(\mathcal{A}_0)) \subseteq \mathcal{L}(\mathcal{A}')$.

# 7 Conclusion

After a brief introduction to tree automata for readers unfamiliar with the topic, I have described the completion algorithm by Feuillade, Genet and Tong [2], which allows us to generate automata accepting an overapproximation of a desired language. I have explained conditions for under and overapproximation of the completed automaton. Finally I have explained conditional term rewriting and an extension of the completion algorithm for conditional term rewrite systems.

## 7.1 Future Work

The presented completion algorithm deals with conditional term rewrite systems with joinability conditions. However, oftentimes conditional term rewrite systems use reachability conditions, that is a rule $l \to r$ may only be used if $c_2$ is reachable from $c_1$. More work is required to determine whether an adjustment of the algorithm for such conditions is feasible.

Furthermore, the conditional term rewrite systems dealt with are type 1 CTRS. The condition $\mathit{Var}(l) \supseteq \mathit{Var}(c_1) \cup \mathit{Var}(c_2)$ means that the conditions may not add variables that are not already part of the left-hand side of the rule. There are however three other, less-restrictive types of CTRS. In type 2 CTRS, new variables may appear in conditions, but not in the right-hand side $\mathit{Var}(l) \supseteq \mathit{Var}(r)$. Another step further are type 3 CTRS, which allow new variables in the right-hand side, that do not appear in the left-hand side, but do appear in the conditions $\mathit{Var}(l) \cup \mathit{Var}(c_i) \supseteq \mathit{Var}(r)$. Finally type 4 knows no restrictions [4]. It is not immediately clear whether the algorithms are useful for higher type conditional term rewrite systems and this might be an interesting topic to examine in future work.

## References

[1] F. Baader and T. Nipkow. *Term rewriting and all that*. Cambridge university press, 1999.

[2] G. Feuillade, T. Genet, and V. V. T. Tong. Reachability analysis over term rewriting systems. *Journal of Automated Reasoning*, 33(3-4):341–383, 2004.

[3] J. Giesl, S. Swiderski, P. Schneider-Kamp, and R. Thiemann. Automated termination analysis for haskell: From term rewriting to programming languages. In *International Conference on Rewriting Techniques and Applications*, pages 297–312. Springer, 2006.

[4] A. Middeldorp and E. Hamoen. Completeness results for basic narrowing. *Applicable Algebra in Engineering, Communication and Computing*, 5(3-4):213–253, 1994.

[5] C. Otto, M. Brockschmidt, C. Von Essen, and J. Giesl. Automated termination analysis of java bytecode by term rewriting. In *LIPIcs-Leibniz International Proceedings in Informatics*, volume 6. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2010.