Congruence-Anticongruence Closure[*]

Ján Kľuka

kluka@fmph.uniba.sk

Department of Applied Informatics
Faculty of Mathematics, Physics and Informatics
Comenius University Bratislava
Mlynská dolina
842 48 Bratislava, Slovakia

Abstract

We present in this paper a method for deciding sets of closed, quantifier-free formulas with identities. The method is based on reduction of such a set into a set of identities by introduction of new special function symbols. For the special function symbols, identity in this set is assumed to have a property which is dual to congruence. We prove that under this property, which we call *anticongruence*, the set of identities is a conservative extension of the original set of formulas. We introduce a simple two-rule proof system (the **CA**-proof system) for the sets of identities with anticongruence symbols, and show the system to be sound and complete. We then formulate the congruence closure algorithm in an abstract way that we believe facilitates its understanding, and extended it to the **CA**-closure algorithm to implement the **CA**-proof system.

Keywords: Automated and assisted theorem proving, congruence closure, clauses with identities.

1 Introduction

Intelligent proof assistants, automated theorem provers, satisfiability (SAT) solvers, and model checking algorithms often decide quantifier-free formulas containing identities. Many such decision algorithms primarily concentrate on the propositional content and decide identity only as, so to speak, added-on feature. The examples of resolution with paramodulation or the DPLL-style SAT solving modulo the theory of identity (EUF) immediately come to mind. However, the direct derivation of formulas with identities is so important that we propose to absorb the propositional content of formulas into identities.

The state of a proof in many intelligent proof assistants can be presented with a sequent $\mathit{\Gamma} \Rightarrow \mathit{\Delta}$ whose validity we wish to establish. Without loss of generality we may include among the formulas of $\mathit{\Gamma}$ also the previously proved theorems. These have often an *open* form $\forall\vec{x}A[\vec{x}]$ with $A[\vec{x}]$ quantifier-free. Some formulas of $\mathit{\Delta}$ have usually an existential form $\exists\vec{x}B[\vec{y}]$ with $B[\vec{x}]$ quantifier-free. Suppose that we have succeeded, whether automatically or not, to instantiate some of such formulas to get an equivalent sequent $\mathit{\Gamma}, A_1[\vec{s_1}], \ldots, A_n[\vec{s_n}] \Rightarrow B_1[\vec{t_1}], \ldots, B_m[\vec{t_m}], \mathit{\Delta}$ with all A_i and B_j quantifier-free. It is well-known that we have a computable test for the validity of the sequent $A_1[\vec{s_1}], \ldots, A_n[\vec{s_n}] \Rightarrow B_1[\vec{t_1}], \ldots, B_m[\vec{t_m}]$. If the last sequent is valid, so is the original sequent $\mathit{\Gamma} \Rightarrow \mathit{\Delta}$. Otherwise, a counter-example should help us with choosing a more suitable set of terms for the instantiations of quantifiers and retrying the procedure.

We present a procedure for the decision of quantifier-free sequents, which is a generalization of the well-known algorithms for congruence closure. We convert the negation of the sequent into, basically, a conjunctive normal form with all clauses having at least one positive literal. We then eliminate the clauses by replacing them with suitably chosen identities. We will then try to derive $0 = 1$ using the generalization of a congruence closure algorithm with *anticongruences*. Congruences are axioms of identity of the form $s_1 = t_1 \land \cdots \land s_n = t_n \to f(s_1, \ldots, s_n) = f(t_1, \ldots, t_n)$. Anticongruences have a dual form: $f(s_1, \ldots, s_n) = f(t_1, \ldots, t_n) \to s_1 = t_1 \lor \cdots \lor s_n = t_n$.

In Sect. 2 we discuss the semantics of identity. In Sect. 3 we replace the propositional connectives in sets of clauses by identities. In Sect. 4 we present a generalization of a congruence closure algorithm to deal with anticongruences.

[*]This work has been partially supported by VEGA under the project No. 1/0175/03. The author would like to thank his supervisor, prof. Paul J. Voda.

2 Quasi-Tautologies in Functional Languages

Identity is so central to the verification of computer programs and, for that matter, to the proofs of theorems in mathematics, that we must be able to deal efficiently with the axioms of equality. Our problem is to determine whether a given sequent $\Gamma \Rightarrow \Delta$ with quantifier-free formulas is valid, and if not, then to obtain a (finite) model of Γ which falsifies all formulas of Δ.

It is well-known that the problem is decidable, but we put an additional constraint on the solution that the negative identities should play no role in the procedure. This is because the disequalities $s \neq t$ are known to cause complications in the decision procedures used in SAT-solving (cf. [2]).

In the rest of the paper we will deal exclusively with closed quantifier-free formulas (sentences) of first-order languages. In order to simplify the decision algorithm we eliminate all relation symbols except identity from our languages.

2.1 First-order and Functional Languages

2.1.1 First-order Languages

A *first-order language* $\mathcal{L}$ is given by a (denumerable) set of function and relation symbols. We assume that every $\mathcal{L}$ contains at least one constant (a nullary function symbol). The assumption is needed so we can form closed (ground) terms. Without loss of generality we will designate the constant by 0. The set $T_{\mathcal{L}}$ of *closed* terms of $\mathcal{L}$ is the smallest set consisting of expressions $f(s_1, \ldots, s_n)$ where f is a function symbol of $\mathcal{L}$ of arity $n \geq 0$ and $s_1, \ldots, s_n$ are closed terms.

Closed quantifier-free formulas of $\mathcal{L}$ are formed from the atomic formulas—which are either *identities* $s = t$ with $s, t \in T_{\mathcal{L}}$ or applications of relation symbols $R(\vec{s})$ of arity $n \geq 0$ with $\vec{s} \in T_{\mathcal{L}}$—by the connectives $\neg$, $\vee$, $\wedge$, and $\rightarrow$. In order to simplify the discussion below we include among formulas also expressions $\Gamma \Rightarrow \Delta$, called *sequents*, with Δ, Γ finite sets of formulas. Our sequents are also called *clauses*.

We will deal below exclusively with closed formulas (sentences) and we will henceforth call them just formulas. Formulas will be designated by meta-variables A, B, C and terms by s, t, u, v, both kinds of variables possibly subscripted. In contexts where the language $\mathcal{L}$ is known, we will tacitly understand that the meta-variables range over the corresponding objects from $\mathcal{L}$. For sets of formulas T_1, T_2 we will write T_1, T_2 instead of $T_1 \cup T_2$ and T_1, A instead of $T_1 \cup \{A\}$.

2.1.2 Functional Languages

A first-order language $\mathcal{L}$ is *functional* if it does not contain any relation symbols except identity. Some of the function symbols of positive arities in $\mathcal{L}$ can be designated as *anticongruence symbols*. We call a term in $T_{\mathcal{L}}$ *pure* if no anticongruence symbols occur in it. A *pure language* is without any anticongruence symbols. The restriction to functional languages is justified by the following definitions and by Thm. 1.

For a first-order language $\mathcal{L}$ we designate by $\mathcal{L}_*$ the functional language containing all function symbols of $\mathcal{L}$, and for every n-ary relation symbol R of $\mathcal{L}$ other than identity, an n-ary function symbol R_*. The *translation* A_* of a quantifier-free formula A of $\mathcal{L}$ is obtained by replacing every occurrence of an atomic formula $R(\vec{s})$ in A by the identity $R_*(\vec{s}) = 0$. The function symbols R_* play thus the role of the characteristic functions of relations R.

We use below the notions of structure and of logical consequence ($T \models A$) in the standard way (see e.g. [4]).

Theorem 1. (Elimination of relation symbols) *For a first-order language $\mathcal{L}$, its functional language $\mathcal{L}_*$, and a quantifier-free formula A of $\mathcal{L}$ we have $\models A$ iff $\models A_*$.*

Proof. In the direction ($\leftarrow$) we assume $\models A_*$ and take any structure $\mathcal{M}$ for $\mathcal{L}$. If the domain of $\mathcal{M}$ has at least two elements, we designate by c an element different from $0^{\mathcal{M}}$. We define the structure $\mathcal{M}_*$ for $\mathcal{L}_*$ with the same domain and the same interpretations of function symbols. For a relation symbol R we interpret $R_*^{\mathcal{M}_*}(\vec{x}) = \begin{cases} 0^{\mathcal{M}} & \text{if } R^{\mathcal{M}}(\vec{x}), \\ c & \text{otherwise.} \end{cases}$ By a straightforward induction on the structure of formulas B in $\mathcal{L}$ we prove $\mathcal{M}_* \models B_*$ iff $\mathcal{M} \models B$. From $\mathcal{M}_* \models A_*$ we then get $\mathcal{M} \models A$.

If the domain of $\mathcal{M}$ consists of a single element $0^{\mathcal{M}}$, we construct $\mathcal{M}_*$ for $\mathcal{L}_*$ with a two element domain $\{c, 0^{\mathcal{M}}\}$. We interpret the function symbols f (including the symbols R_*) to yield c on arguments with at least one c, and $0^{\mathcal{M}}$ otherwise, unless f is some R_*, and then we interpret $R_*^{\mathcal{M}_*}(0^{\mathcal{M}}, \dots, 0^{\mathcal{M}})$ as above with $x_1 = 0^{\mathcal{M}}$, ..., $x_n = 0^{\mathcal{M}}$. The rest of the proof is similar as above.

For the proof in the direction $(\rightarrow)$ we assume $\vDash A$ and take any structure $\mathcal{M}_*$ for $\mathcal{L}_*$. We define the structure $\mathcal{M}$ for $\mathcal{L}$ with the same domain and with the same interpretations of function symbols of $\mathcal{L}$. For a relation symbol R of $\mathcal{L}$ we define $R^{\mathcal{M}}(\vec{x})$ to hold iff $\mathcal{M}_* \vDash R_*(\vec{x}) = 0$. We then continue similarly as above. $\square$

2.2 Satisfaction Relation for Binary Relations and Functional Languages

2.2.1 Binary Relations over Terms

We will be investigating propositional consequence for functional languages $\mathcal{L}$ under various properties of the identity symbol. Since the structures used in the first-order logic presuppose the standard properties of identity, we will need to define the semantics of identity by weaker means.

The role of structures will be played by binary relations $\simeq$ over $T_{\mathcal{L}}$. The relation $\simeq$ is an *equivalence* if it is reflexive, symmetric, and transitive. The relation is a *congruence* if it is an equivalence respecting the function symbols of $\mathcal{L}$, i.e., if for every $f \in \mathcal{L}$ of arity $n > 0$ and all terms such that $s_1 \simeq t_1$, ..., $s_n \simeq t_n$, we have $f(s_1, \dots, s_n) \simeq f(t_1, \dots, t_n)$.

A congruence relation $\simeq$ is an *anticongruence* if for all anticongruence symbols $h \in \mathcal{L}$ of arity $m \geq 1$ and terms $\vec{s}$, $\vec{t}$ such that $h(s_1, \dots, s_m) \simeq h(t_1, \dots, t_m)$ we have $s_i \simeq t_i$ for some $i = 1, \dots, m$. Note that for pure languages every congruence relation is trivially an anticongruence.

2.2.2 Satisfaction Relation for Functional Languages

For a given functional language $\mathcal{L}$ we define the relation $\simeq$ *satisfies* A, in writing $\simeq \vDash A$, where $\simeq$ is a relation over $T_{\mathcal{L}}$ and A a formula, to satisfy the following recurrences:

- $\simeq \vDash s = t$ iff $s \simeq t$,
- $\simeq \vDash \neg A$ iff $\simeq \nvDash A$,
- $\simeq \vDash A \vee B$ iff $\simeq \vDash A$ or $\simeq \vDash B$,
- $\simeq \vDash A \wedge B$ iff $\simeq \vDash A$ and $\simeq \vDash B$,
- $\simeq \vDash A \rightarrow B$ iff $\simeq \vDash B$ whenever $\simeq \vDash A$,
- $\simeq \vDash \mathit{\Gamma} \Rightarrow \mathit{\Delta}$ iff whenever $\simeq \vDash A$ for all $A \in \mathit{\Gamma}$, then there is a $B \in \mathit{\Delta}$ such that $\simeq \vDash B$.

We extend the satisfaction relation to (possibly infinite) sets T, called *theories*, and say that $\simeq$ *satisfies* T, in writing $\simeq \vDash T$, if every $A \in T$ is satisfied in $\simeq$. For $q = p, e, c, a$ and $\simeq$ satisfying T we say that $\simeq$ is a *q-model* of T if $\simeq$ is in the order p, e, c, a any relation, equivalence, congruence, or anticongruence. We say that A is a *q-consequence of* T, in writing $T \vDash_q A$, iff every q-model $\simeq$ of T satisfies A. For a set S of formulas we write $T \vDash_q S$ when $T \vDash_q A$ for all $A \in S$.

2.2.3 Conservative Extensions

For $q = p, e, c, a$ we call a theory T_1 in $\mathcal{L}_1$ a *q-extension* of the theory T_2 in $\mathcal{L}_2$ if $\mathcal{L}_1$ is an extension of $\mathcal{L}_2$ (includes all symbols of $\mathcal{L}_2$) and $T_1 \vDash_q T_2$ holds. T_1 *is q-conservative over* T_2 *for* S if T_1 is a q-extension of T_2, and whenever $T_1 \vDash_q A$ for $A \in S$, then already $T_2 \vDash_q A$.

2.2.4 Restrictions and Extensions of Relations over Terms

For a functional language $\mathcal{L}$ let D be a possibly infinite subset of $T_{\mathcal{L}}$. D is *downward closed* if whenever $f(s_1, \dots, s_n) \in D$, then also $s_1, \dots, s_n \in D$. We call such a D a *domain*. We call a relation $\simeq$ over D *e-closed* if it is an equivalence. An equivalence over D is *c-closed* if for every $f(\vec{s}), f(\vec{t}) \in D$ such that $\vec{s} \simeq \vec{t}$ we have $f(\vec{s}) \simeq f(\vec{t})$. A c-closed equivalence is *a-closed* if for every $h(\vec{u}), h(\vec{v}) \in D$ with h an anticongruence symbol such that $h(u_1, \dots, u_m) \simeq h(v_1, \dots, v_m)$, we have $u_i \simeq v_i$ for some $i = 1, \dots, m$.

For any set T of formulas of $\mathcal{L}$, the set of all subterms occurring in the formulas of T is clearly a domain and we call it the *domain of* T. For $q = e, c, a$ the restriction of a q-relation $\simeq$ over $T_{\mathcal{L}}$ to a domain D is clearly a q-closed relation over D. Moreover, it can be easily seen that if two relations $\simeq$ and $\simeq'$ *coincide* on D, i.e., if their restrictions to D are identical, then for every $s, t \in D$ we have $\simeq \vDash s = t$ iff $\simeq' \vDash s = t$. This extends to $\simeq \vDash A$ iff $\simeq' \vDash A$ for any formula A with all subterms occurring in T. The following important lemma asserts a property in the opposite direction:

Lemma 2. (Expansion) *Every a-closed relation $\simeq$ over a domain D of a functional language $\mathcal{L}$ can be* expanded *to an anticongruence $\simeq'$ over $T_{\mathcal{L}}$ such that $\simeq$ and $\simeq'$ coincide on D, and for all function symbols g which do not occur in the terms of D, if $s \in D$ and g occurs in a term t, then $\simeq' \nvDash s = t$.*

Proof. Let $ord\colon D/{\simeq} \to \mathbb{N}$ be an injection into odd natural numbers. We construct the structure $\mathcal{M}$ for $\mathcal{L}$ with the domain a subset of natural numbers consisting of all even numbers and of those odd numbers which are in the range of ord. We interpret an n-ary ($n \geq 0$) function symbol f of $\mathcal{L}$ as follows:

$$f^{\mathcal{M}}(\vec{x}) = \begin{cases} ord([f(\vec{s})]) & \text{if } f(\vec{s}) \in D \text{ and } x_1 = ord([s_1]), \ldots, x_n = ord([s_n]), \\ 2{\cdot}x_1 & \text{otherwise, where if } n = 0, \text{ then } x_1 = 0\,. \end{cases}$$

This is a legal definition because ord is an injection and $\simeq$ is c-closed. Thus if $f(\vec{s}), f(\vec{t}) \in D$ with $ord([s_i]) = ord([t_i])$ for all $i = 1, \ldots, n$, then also $[s_i] = [t_i]$, and hence $[f(\vec{s})] = [f(\vec{t})]$.

By a simple induction on terms s we get if $s \in D$, then $s^{\mathcal{M}} = ord([s])$. Consequently for $s, t \in D$, $s \simeq t$ iff $\mathcal{M} \vDash s = t$. For any terms s and t, we define $s \simeq' t$ to hold iff $\mathcal{M} \vDash s = t$. Clearly, $\simeq'$ is a congruence and its restriction to D is $\simeq$. Moreover, for a function symbol g which does not occur in D, by induction on terms t we have if g occurs in t, then $t^{\mathcal{M}}$ is even, hence $s \not\simeq' t$ for all $s \in D$. Indeed, assume $t = f(\vec{t})$ for some function symbol f and some subterms $\vec{t}$. If $f = g$, then since g does not occur in D, there can be no term $g(\vec{s}) \in D$, so $t^{\mathcal{M}}$ is even: $t^{\mathcal{M}} = 2{\cdot}t_1^{\mathcal{M}}$ or $t^{\mathcal{M}} = 2{\cdot}0$ depending on the arity of g. If $f \neq g$, the arity of f is positive and g occurs in t_i for some i. By the inductive hypothesis $t_i^{\mathcal{M}}$ is even, but then there is no $f(\vec{s}) \in D$ such that $t_i^{\mathcal{M}} = ord([s_i])$, hence $t^{\mathcal{M}} = 2{\cdot}t_1$ is even.

It remains to show that $\simeq'$ is an anticongruence. So assume $\mathcal{M} \vDash h(\vec{u}) = h(\vec{v})$ for an anticongruence symbol h of arity m of $\mathcal{L}$ and consider two cases. If $(h(\vec{u}))^{\mathcal{M}}$ is odd, then there are $\vec{s}$ and $\vec{t}$ such that $h(\vec{s}), h(\vec{t}) \in D$, $u_i^{\mathcal{M}} = ord([s_i])$ and $v_i^{\mathcal{M}} = ord([t_i])$ for all $i = 1, \ldots, m$, and $ord([h(\vec{s})]) = (h(\vec{u}))^{\mathcal{M}} = (h(\vec{v}))^{\mathcal{M}} = ord([h(\vec{t})])$. Since ord is an injection, we have $[h(\vec{s})] = [h(\vec{t})]$, thus $h(\vec{s}) \simeq h(\vec{t})$ from which we get $s_i \simeq t_i$ for some $i = 1, \ldots, m$ as $\simeq$ is a-closed. Hence $u_i^{\mathcal{M}} = s_i^{\mathcal{M}} = t_i^{\mathcal{M}} = v_i^{\mathcal{M}}$ and $u_i \simeq' v_i$. If $(h(\vec{u}))^{\mathcal{M}}$ is even, then $2{\cdot}u_1^{\mathcal{M}} = (h(\vec{u}))^{\mathcal{M}} = (h(\vec{v}))^{\mathcal{M}} = 2{\cdot}v_1^{\mathcal{M}}$ and hence $u_1 \simeq' v_1$. □

2.2.5 Axioms of Equality

Fix a functional language $\mathcal{L}$. For sequences of terms $\vec{s} = s_1, \ldots, s_n$ and $\vec{t} = t_1, \ldots, t_n$ we abbreviate by $\vec{s} = \vec{t}$ the (possibly empty) set $s_1 = t_1, \cdots, s_n = t_n$.

It should be clear that any equivalence over $T_{\mathcal{L}}$ is an e-model of the set $\boldsymbol{E}^{\mathcal{L}}$ of all formulas:

$$s = s \qquad s = t \to t = s \qquad s = t \land t = u \to s = u \tag{1}$$

and vice versa, any relation $\simeq$ satisfying $\boldsymbol{E}^{\mathcal{L}}$ is an equivalence.

If $\simeq$ is a congruence, then it is also a c-model of the set $\boldsymbol{C}^{\mathcal{L}}$ of all sequents:

$$\vec{s} = \vec{t} \Rightarrow f(\vec{s}) = f(\vec{t}) \tag{2}$$

where $f \in \mathcal{L}$ is of arity $n > 0$. Vice versa, any equivalence satisfying $\boldsymbol{C}^{\mathcal{L}}$ is a congruence.

Finally, if $\simeq$ is an anticongruence, then it is also an a-model of the set $\boldsymbol{A}^{\mathcal{L}}$ of all sequents:

$$h(\vec{s}) = h(\vec{t}) \Rightarrow \vec{s} = \vec{t} \tag{3}$$

where h is an anticongruence symbol of $\mathcal{L}$. Vice versa, any congruence satisfying $\boldsymbol{A}^{\mathcal{L}}$ is an anticongruence. Note that the set $\boldsymbol{A}^{\mathcal{L}}$ is empty for pure languages and any congruence for such a language is trivially a model of $\boldsymbol{A}^{\mathcal{L}}$.

Whenever the language $\mathcal{L}$ is known from the context we will drop the superscripts from the names of the sets of identity axioms. When $T \vDash_c A$ holds we say that A *is a quasi-tautological consequence of* T (A *is a quasi-tautology* if T is empty). The above discussion of axioms of identity is formally expressed in Thm. 3. Thm. 4.(1) connects the standard tautological consequence of logic with our relation $\vDash_p$. It follows that the standard logical definition of quasi-tautological consequence as a *tautological consequence from the axioms of identity* (cf. [4]) agrees with our definition above. For the sake of completeness we prove in Thm. 4.(2) the standard connection between quasi-tautological and logical consequence for quantifier-free formulas.

Theorem 3. (Reductions of q-consequence) *For a quantifier-free theory T in a functional language $\mathcal{L}$ and a quantifier-free formula A we have:*

1. *$T \vDash_e A$ iff $T, \boldsymbol{E} \vDash_p A$,*
2. *$T \vDash_c A$ iff $T, \boldsymbol{C} \vDash_e A$,*
3. *$T \vDash_a A$ iff $T, \boldsymbol{A} \vDash_c A$.*

Theorem 4. (Tautological and quasi-tautological consequence) *For a quantifier-free theory T in a functional language $\mathcal{L}$ and a quantifier-free formula A we have:*

1. *A is a tautological consequence of T iff $T \vDash_p A$,*
2. *$T \vDash A$ iff $T \vDash_c A$.*

Proof. (1): In the direction ($\rightarrow$) assume that A tautologically follows from T and take any p-model $\simeq$ of T. Construct an assignment I of truth values to identities $s = t$ of $\mathcal{L}$ as $I(s = t) = \mathfrak{t}$ if $s \simeq t$, and $I(s = t) = \mathfrak{f}$ otherwise. We then extend this to any B of $\mathcal{L}$ by showing that B is satisfied in I iff $\simeq \vDash B$. The assignment I satisfies T and so A is true in I and hence $\simeq \vDash A$. In the direction ($\leftarrow$) we assume $T \vDash_p A$ and take any assignment I satisfying T. For $s = t$ in $\mathcal{L}$ we define $s \simeq t$ to hold iff $I(s = t) = \mathfrak{t}$ and continue similarly as above.

(2): The proof is structurally similar to that of (1). In the direction ($\rightarrow$) we take any c-model $\simeq$ of T and construct a structure $\mathcal{M}$ for $\mathcal{L}$ with the domain $T_{\mathcal{L}}/\simeq$ where for a function symbol f of $\mathcal{L}$ we define $f^{\mathcal{M}}([s_1], \ldots, [s_n]) = [f(s_1, \ldots, s_n)]$. We can see, similarly as in the proof of Lemma 2., that this is a legal definition. It is easy to see that $s^{\mathcal{M}} = [s]$ and so $\mathcal{M} \vDash s = t$ iff $s \simeq t$. In the direction ($\leftarrow$) we take any model $\mathcal{M}$ of T and define $s \simeq t$ to hold iff $\mathcal{M} \vDash s = t$. The relation $\simeq$ is a c-model of T. ☐

3 Conversion of Arbitrary Sequents to Sets of Identities

We will transform in Subsect. 3.1 arbitrary sequents to sets of sequents without any additional connectives. In Subsect. 3.2 we will show how to eliminate the sequents in favor of sets of identities with anticongruence symbols. The Conservativity Theorem 6. asserts that the detour through anticongruences is strictly speaking not necessary, but—as the simplicity of the **CA**-proof system presented in Subsect. 3.3 and proved complete in Thm. 7. shows—it is very convenient. The soundness and completeness of our **CA**-closure algorithm presented in Sect. 4 is a direct consequence of Thm. 7.

3.1 Transformation of Sequents into Normal Form

Fix a pure functional language $\mathcal{L}$ for which we can assume w.l.o.g. that it does not contain the constant 1. We will designate by the meta-variable Π finite sets of sequents. We say that Π is *normal* if in every sequent $\Gamma \Rightarrow \Delta$ of Π, the set Δ is non-empty and both sets consist at most of identities.

For a given sequent $\Gamma \Rightarrow \Delta$ in $\mathcal{L}$ we form a set of sequents $\{\Rightarrow A \mid A \in \Gamma\}, \{B \Rightarrow \mid B \in \Delta\}$ and repeatedly apply the the following transformation rules to it:

$$\begin{array}{rcl}
(\Gamma \Rightarrow A \rightarrow B, \Delta), \Pi & \Longrightarrow & (A, \Gamma \Rightarrow B, \Delta), \Pi \\
(A \rightarrow B, \Gamma \Rightarrow \Delta), \Pi & \Longrightarrow & (\Gamma \Rightarrow A, \Delta), (B, \Gamma \Rightarrow \Delta), \Pi \\
(\Gamma \Rightarrow (\Gamma_1 \Rightarrow \Delta_1), \Delta), \Pi & \Longrightarrow & (\Gamma_1, \Gamma \Rightarrow \Delta_1, \Delta), \Pi \\
((\Gamma_1 \Rightarrow \Delta_1), \Gamma \Rightarrow \Delta), \Pi & \Longrightarrow & \{\Gamma \Rightarrow A, \Delta \mid A \in \Gamma_1\}, \{B, \Gamma \Rightarrow \Delta \mid B \in \Delta_1\}, \Pi \\
(\Gamma \Rightarrow A \vee B, \Delta), \Pi & \Longrightarrow & (\Gamma \Rightarrow A, B, \Delta), \Pi \\
(A \vee B, \Gamma \Rightarrow \Delta), \Pi & \Longrightarrow & (A, \Gamma \Rightarrow \Delta), (B, \Gamma \Rightarrow \Delta), \Pi \\
(A \wedge B, \Gamma \Rightarrow \Delta), \Pi & \Longrightarrow & (A, B, \Gamma \Rightarrow \Delta), \Pi \\
(\Gamma \Rightarrow A \wedge B, \Delta), \Pi & \Longrightarrow & (\Gamma \Rightarrow A, \Delta), (\Gamma \Rightarrow B, \Delta), \Pi
\end{array}$$

$$\begin{array}{lcl}(\neg A, \Gamma \Rightarrow \Delta), \Pi & \Longrightarrow & (\Gamma \Rightarrow A, \Delta), \Pi \\ (\Gamma \Rightarrow \neg A, \Delta), \Pi & \Longrightarrow & (A, \Gamma \Rightarrow \Delta), \Pi\,.\end{array}$$

It is easy to prove that this system of rules is terminating and confluent. Moreover, the final set of sequents Π' consists of sequents $\Gamma_1 \Rightarrow \Delta_1$ with Γ_1, Δ_1 containing at most identities.

We form the language $\mathcal{L}^1$ by extending $\mathcal{L}$ with the constant 1 and replace in Π' every sequent with the empty consequent: $\Gamma \Rightarrow$ by the sequent $\Gamma \Rightarrow 0 = 1$ whereby we obtain a normal set of sequents Π in $\mathcal{L}^1$, which we call the *normal form* of the original sequent $\Gamma \Rightarrow \Delta$.

Theorem 5. (Normal form of sequents) *If the sequent $\Gamma \Rightarrow \Delta$ of a pure functional language $\mathcal{L}$ without the constant* 1 *has the normal form Π, then we have $\models_c \Gamma \Rightarrow \Delta$ iff $\Pi \models_c 0 = 1$.*

Proof. We observe that the rules of transformation from Subsect. 3.1 are based on the Gentzen's sequent calculus and can be easily seen to preserve validity, which is by Thm. 4.(2) equivalent to c-validity. Using the notation of Subsect. 3.1, we thus have $\models_c \Gamma \Rightarrow \Delta$ iff $\{\Rightarrow A \mid A \in \Gamma\}, \{B \Rightarrow \mid B \in \Delta\} \models_c \Rightarrow$ iff $\Pi' \models_c \Rightarrow$ iff Π' has no c-model. It remains to show

$$\Pi' \text{ is } c\text{-unsatisfiable iff } \Pi \models_c 0 = 1.$$

In the direction $(\rightarrow)$ we assume $\Pi \not\models_c 0 = 1$, i.e., that there is a c-model $\simeq$ of Π such that $\simeq \not\models 0 = 1$. The congruence $\simeq$ is over $T_{\mathcal{L}^1}$ and its restriction $\simeq'$ to the domain $T_{\mathcal{L}}$ is a congruence over $T_{\mathcal{L}}$. We show $\simeq' \models \Pi'$ by taking any sequent $\Gamma' \Rightarrow \Delta'$ from Π' and by considering two cases. If Δ' is empty, then $\Gamma' \Rightarrow 0 = 1$ is in Π and thus $\simeq \models \Gamma' \Rightarrow 0 = 1$. Since $\simeq \not\models 0 = 1$, we get $\simeq \not\models \Gamma'$, then $\simeq' \not\models \Gamma'$, and hence $\simeq' \models \Gamma' \Rightarrow$. If Δ is not empty, then the sequent is in Π and from $\simeq \models \Gamma' \Rightarrow \Delta'$ we directly get $\simeq' \models \Gamma' \Rightarrow \Delta'$.

In the direction $(\leftarrow)$ we assume that Π' has a c-model $\simeq'$. The relation $\simeq'$ is over $T_{\mathcal{L}}$ which forms a domain in $\mathcal{L}^1$. Since $\mathcal{L}$ is pure, $\simeq'$ is trivially a-closed over the domain. By Lemma 2., $\simeq'$ can be expanded to a congruence $\simeq$ over $T_{\mathcal{L}^1}$ such that $\simeq \not\models 0 = 1$. For the proof of the theorem it thus suffices to show $\simeq \models \Pi$ and so we take any sequent $\Gamma' \Rightarrow \Delta'$ in Π. If the sequent is also in Π', then $\simeq' \models \Gamma' \Rightarrow \Delta'$ and hence $\simeq \models \Gamma' \Rightarrow \Delta'$ by coincidence. Otherwise $\Delta' = \{0 = 1\}$ and $\Gamma' \Rightarrow$ is in Π. Since $\simeq' \models \Gamma' \Rightarrow$, we have $\simeq' \not\models \Gamma'$, by coincidence we get $\simeq \not\models \Gamma'$, and hence $\simeq \models \Gamma' \Rightarrow 0 = 1$. □

3.2 Conversion of Normal Sets of Sequents to Systems of Identities

For a given normal set of sequents Π in a pure functional language $\mathcal{L}$ we will now construct an *associated* set of identities Γ in a language $\mathcal{L}^a$. We construct $\mathcal{L}^a$ as an extension of $\mathcal{L}$ by including into it all function symbols of $\mathcal{L}$ and all new function symbols specified by the following procedure which also constructs the initially empty set Γ. The procedure takes each sequent

$$s_1 = t_1, \ldots, s_n = t_n \Rightarrow u_1 = v_1, \ldots, u_m = v_m \tag{4}$$

in Π (note that $m \geq 1$) and

1. adds to $\mathcal{L}^a$ a new n-ary function symbol g and a new m-ary anticongruence symbol h,
2. adds to Γ the identities $g(\vec{s}) = h(\vec{u})$, $g(\vec{t}) = h(\vec{v})$.

Theorem 6. (Conservativity of associated sets of identities) *For every set of identities Γ in $\mathcal{L}^a$ associated with a normal set of sequents Π in a pure $\mathcal{L}$ and every $s = t$ in $\mathcal{L}$ we have $\Gamma \models_a s = t$ iff $\Pi \models_c s = t$. This is an immediate consequence of $\Gamma, \boldsymbol{A}^{\mathcal{L}^a}$ being c-conservative over Π for identities.*

Proof. We first prove that $\Gamma, \boldsymbol{A}^{\mathcal{L}^a}$ is a c-extension of Π by showing $\Gamma, \boldsymbol{A}^{\mathcal{L}^a} \models_c \Pi$, i.e., $\Gamma \models_a \Pi$. So take any sequent (4) of Π and any a-model $\simeq$ of $\Gamma, \vec{s} = \vec{t}$. Thus $\simeq \models g(\vec{s}) = g(\vec{t})$ and hence $\simeq \models h(\vec{u}) = h(\vec{v})$. But then $\simeq \models u_i = v_i$ for some $i = 1, \ldots, m$.

We now prove that $\Gamma, \boldsymbol{A}^{\mathcal{L}^a}$ is c-conservative over Π for identities. So assume $\Gamma \models_a s = t$ for an $s = t$ in $\mathcal{L}$ and take any c-model $\simeq$ of Π. We wish to prove $\Pi \models_c s = t$. Let $\simeq_D$ be the restriction of $\simeq$ to the domain D of $\Pi, s = t$. Clearly, $\simeq_D$ is c-closed. Note that the domain D^a of $\Gamma, s = t$ contains, in addition to the terms of D, also the four terms $g(\vec{s})$, $h(\vec{u})$, $g(\vec{t})$, $h(\vec{v})$ for each sequent (4) of Π. Take the set $\simeq_D \cup \{\langle g(\vec{s}), h(\vec{u})\rangle \mid (g(\vec{s}) = h(\vec{u})) \in \Gamma\}$ and extend it to the least c-closed equivalence $\simeq_{D^a}$ over D^a.

For all sequents (4) in Π this happens by having $[g(\vec{s})]_{\simeq_{D^a}} = \{g(\vec{s}), h(\vec{u}), g(\vec{t}), h(\vec{v})\}$ when $\simeq \models \vec{s} = \vec{t}$ or $\simeq \models \vec{u} = \vec{v}$ holds. Otherwise we have $[g(\vec{s})]_{\simeq_{D^a}} = \{g(\vec{s}), h(\vec{u})\}$ and $[g(\vec{t})]_{\simeq_{D^a}} = \{g(\vec{t}), h(\vec{v})\}$. Note that $\simeq_D = \simeq_{D^a} \cap (D \times D)$.

We now show that $\simeq_{D^a}$ is a-closed. If $h(\vec{u}) \simeq_{D^a} h(\vec{v})$ for the anticongruence symbol h belonging to the sequent (4), then there are two cases. Either $\simeq \models \vec{s} = \vec{t}$ and then, since $\simeq$ is a c-model of Π, $\simeq \models u_i = v_i$ for some $i = 1, \dots, m$. From this we get $u_i \simeq_{D^a} v_i$. Or else $\simeq \models \vec{u} = \vec{v}$, i.e., $u_1 \simeq_D v_1$, and hence $u_1 \simeq_{D^a} v_1$.

By Lemma 2. there is an anticongruence $\simeq'$ over $T_{\mathcal{L}^a}$ which is an expansion of $\simeq_{D^a}$. By the construction of $\simeq_{D^a}$ the relation $\simeq'$ is an a-model of Γ. From the assumption $\Gamma \models_a s = t$ we get $\simeq' \models s = t$ and hence $\simeq \models s = t$ because $\simeq$ and $\simeq'$ coincide on D. □

3.3 The CA-Proof System

We will now present a proof system for proving a-consequences of sets of identities Γ. The reader will note that the system restricts the use of identity axioms from the sets $\boldsymbol{C}$, $\boldsymbol{A}$.

For any functional language $\mathcal{L}$ and any set of identities Γ the rules of derivation of the **CA**-*proof* system are the following ones:

$$\frac{f(\vec{s}) = f(\vec{t}), \Gamma}{\Gamma} \quad \textbf{C}\text{-rule}$$

$$\frac{u_1 = v_1, \Gamma \mid \cdots \mid u_m = v_m, \Gamma}{\Gamma} \quad \textbf{A}\text{-rule.}$$

The **C**-rules are applicable under conditions that the terms $f(\vec{s})$, $f(\vec{t})$ are pure, occur in Γ, and that $\Gamma \models_e \vec{s} = \vec{t}$ holds. The **A**-rules are applicable under conditions that there is an anticongruence symbol h such that the terms $h(u_1, \dots, u_m)$, $h(v_1, \dots, v_m)$ occur in Γ, and $\Gamma \models_e h(u_1, \dots, u_m) = h(v_1, \dots, v_m)$ holds.

We write $\Gamma \vdash s = t$ if there is a tree built by the rules of derivation for the root Γ such that in every leaf Δ we have $\Delta \models_e s = t$.

Theorem 7. (Soundness and Completeness of the CA-proof system for associated sets) *For every normal set of sequents Π in a pure functional language $\mathcal{L}$, its associated set of identities Γ in $\mathcal{L}^a$, and any identity $s = t$ in the domain of Π we have $\Gamma \vdash s = t$ iff $\Gamma \models_a s = t$.*

Proof. Actually, we prove by induction on the proofs of $\Gamma \vdash s = t$ that the **CA**-proof system is sound for arbitrary set of identities Γ in some $\mathcal{L}$. If the proof is a leaf, then we get $\Gamma \models_a s = t$ because $\Gamma \models_e s = t$. If the last rule used in the proof is a **C**-rule, then we have $\Gamma \models_e \vec{s} = \vec{t}$ and we take any a-model $\simeq$ of Γ. Since $\simeq \models \vec{s} = \vec{t}$, we get $\simeq \models f(\vec{s}) = f(\vec{t})$, and then $\simeq \models s = t$ from the inductive hypothesis. If the last rule used in the proof is an **A**-rule, then we have $\Gamma \models_e h(\vec{u}) = h(\vec{v})$ and we take any a-model $\simeq$ of Γ. Thus $\simeq \models h(\vec{u}) = h(\vec{v})$ and hence $\simeq \models u_i = v_i$ for some $i = 1, \dots, m$. But then $\simeq \models s = t$ by the i-th inductive assumption.

For the proof of completeness restricted to the associated sets we define for an arbitrary set of identities Γ the relation $\simeq_\Gamma$ such that $u \simeq_\Gamma v$ holds iff u, v are in the domain of Γ and $\Gamma \models_e u = v$. The relation is clearly an equivalence over the domain of Γ. We call $\simeq_\Gamma$ *almost closed* if it satisfies the conditions of a-closure except perhaps the conditions for c-closure for the anticongruence symbols h.

Take now the Γ associated to Π from the statement of the theorem and assume that $\Gamma \nvdash s = t$. We construct a proof-like tree with the root Γ such that for every inner node with the label Γ_1 the rule for the node corresponds to an identity axiom for which $\simeq_{\Gamma_1}$ is not almost closed. As there are only finitely many rules applicable to Γ, we can construct such a tree and its leaves must be almost closed. One of the leaves Γ_1 is thus such that $\Gamma_1 \not\models_e s = t$. Let $\simeq$ be the restriction of the equivalence $\simeq_{\Gamma_1}$ to the domain D of Π. The relation is clearly c-closed and trivially a-closed because there are no anticongruence symbols in D. We expand $\simeq$ by Lemma 2. to a (trivial) anticongruence $\simeq'$ over $T_{\mathcal{L}}$.

We prove $\simeq' \models \Pi$ by taking any sequent (4) in Π and assuming $\simeq' \models \vec{s} = \vec{t}$. Then $\vec{s} \simeq \vec{t}$, and hence $\vec{s} \simeq_{\Gamma_1} \vec{t}$. Since Γ_1 is almost closed, we obtain $g(\vec{s}) \simeq_{\Gamma_1} g(\vec{t})$. Since $(g(\vec{s}) = h(\vec{u})), (g(\vec{t}) = h(\vec{v})) \in \Gamma \subseteq \Gamma_1$, we get $h(\vec{u}) \simeq_{\Gamma_1} h(\vec{v})$. Thus $u_i \simeq_{\Gamma_1} v_i$, and hence $\simeq' \models u_i = v_i$, for some $i = 1, \dots, m$.

From $\Gamma_1 \not\models_e s = t$ we get $s \not\simeq_{\Gamma_1} t$, then $s \not\simeq t$, and hence $\simeq' \not\models s = t$, all by coincidence. This shows $\Pi \not\models_c s = t$ and hence $\Gamma \not\models_a s = t$ by the conservativity Thm. 6. □

3.4 Discussion

Note that the size of the set Π' produced by the transformation given in Subsect. 3.1 can be exponential in the size of the original sequent $\Gamma \Rightarrow \Delta$. Using this transformation has greatly simplified the proof of Thm. 5. But a practical application should use a different transformation which yields only polynomial increase in size of the original sequent. This is possible, at the expense of introducing new constants, using an adapted version of the transformation of SAT to 3-SAT. The following rules may be used in place of their respective counterparts from Subsect. 3.1:

$$\begin{array}{rcl}
(A \to B, \Gamma \Rightarrow \Delta), \Pi & \Longrightarrow & (c=0, \Gamma \Rightarrow \Delta), (\Rightarrow A, c=0), (B \Rightarrow c=0), \Pi \\
((\Gamma_1 \Rightarrow \Delta_1), \Gamma \Rightarrow \Delta), \Pi & \Longrightarrow & (c=0, \Gamma \Rightarrow \Delta), \{\Rightarrow A, c=0 \mid A \in \Gamma_1\}, \{B \Rightarrow c=0 \mid B \in \Delta_1\}, \Pi \\
(A \vee B, \Gamma \Rightarrow \Delta), \Pi & \Longrightarrow & (c=0, \Gamma \Rightarrow \Delta), (A \Rightarrow c=0), (B \Rightarrow c=0), \Pi \\
(\Gamma \Rightarrow A \wedge B, \Delta), \Pi & \Longrightarrow & (\Gamma \Rightarrow c=0, \Delta), (c=0 \Rightarrow A), (c=0 \Rightarrow B), \Pi \,.
\end{array}$$

Each application of any of the above rules requires introduction of a new constant c.

Furthermore, any practical application of the **CA**-closure algorithm will use the obvious simplifications of the conversion process described in Subsect. 3.2. We did not use them in order to keep the exposition simple. Whenever in (4) $m = 1$, i.e., the sequent is a *Horn clause*, there is no need to introduce a new anticongruence symbol h into $\mathcal{L}^a$. The addition of the identities $g(\vec{s}) = u_1$ and $g(\vec{t}) = v_1$ into Γ suffices. If we have $n = 0$, there is no need to introduce the constant g into $\mathcal{L}^a$. The addition of $h(\vec{u}) = h(\vec{v})$ into Γ suffices for $m > 1$. If $m = 1$, we just add $u_1 = v_1$.

Thus if Π consists of Horn clauses, then the **CA**-proof system is without ***A***-rules and a congruence closure algorithm suffices to decide whether $s = t$ is a c-consequence of Π.

4 An Algorithm for the CA-Closure

We will now present an algorithm which systematically applies the ***C*** and ***A*** rules of the **CA**-proof system (see Subsect. 3.3) with the correctness expressed by Thm. 7. Our algorithm is a generalization (to deal with anticongruences) of the well-known $\mathrm{O}(n\log(n))$ congruence closure algorithms [1, 3]. In contrast to the two algorithms, no transformation of terms into directed graphs (as in [1]) or into a curried form (as in [3]) is needed. We achieve the same efficiency (for the ***C***-closure that is, the ***A***-closure is inherently exponential) with an organization of terms $f(s_1, \ldots, s_n)$ into a tree with branches $\langle [f(s_1, \ldots, s_n)], f, [s_1], \ldots, [s_n], \emptyset \rangle$. We present the algorithm on two levels: an abstract one expressed in the language of set theory, and a more concrete one with the data structures specified in details sufficient for a complexity estimate.

We fix for the rest of this section a normal set of sequents Π in some pure functional language $\mathcal{L}$ and its associated set of identities Γ with the domain D in the language $\mathcal{L}^a$.

4.1 Set-Theoretical Implementation of the CA-Closure Algorithm

4.1.1 Equivalence Trees

Define $\simeq_\Gamma$ to be over D and such that $s \simeq_\Gamma t$ holds iff $\Gamma \vDash_e s = t$. We call an equivalence $\simeq$ over D a *coarsening of* $\simeq_\Gamma$ if $\simeq_\Gamma \subseteq \simeq$ and for every pair of identities $g(\vec{s}) = h(\vec{u})$, $g(\vec{t}) = h(\vec{v})$ in Γ we have $[h(\vec{u})] = \{g(\vec{s}), h(\vec{u})\}$ or $[h(\vec{u})] = \{g(\vec{s}), h(\vec{u}), g(\vec{t}), h(\vec{v})\}$. Note that $\simeq_\Gamma$ is its own coarsening.

Take any coarsening $\simeq$ of $\simeq_\Gamma$ and construct the set

$$T(\simeq) = \{\langle [f(s_1, \ldots, s_n)], f, [s_1], \ldots, [s_n], \emptyset \rangle \mid f(s_1, \ldots, s_n) \text{ is pure in } D\}\,.$$

We designate by $Tr(\simeq)$ the closure of $T(\simeq)$ under the initial segments of its sequences, i.e., the least set T such that $T(\simeq) \subseteq T$ and with every $\langle e, p \rangle \in T$ also $p \in T$. The reader will observe that the set $Tr(\simeq)$ is a *tree* in the set-theoretical sense. The finite sequences $p \in Tr(\simeq)$ are its *nodes*. Every node $\langle e, p \rangle \in Tr(\simeq)$ is a *child* of p, and p is the *parent* of $\langle e, p \rangle$. The node $\langle \emptyset \rangle$ is the *root* and nodes without children are *leaves*. We call e the *label* of the node $\langle e, p \rangle$. The tree $Tr(\simeq)$ is finitely branching (each node has finitely many children) with every *path*, i.e., a leaf in our particular case, finite.

This makes $T(\simeq)$ the set of paths through the tree $Tr(\simeq)$. We have $D/{\simeq} = \{e \mid \exists p\, \langle e, p \rangle \in T(\simeq)\}$.

4.1.2 C-Closure of Equivalence Trees

We call an equivalence tree $T(\simeq)$ with $\simeq$ a coarsening of $\simeq_\Gamma$ **C**-*closed* if $\simeq$ is almost c-closed ($\simeq$ does not have to be a congruence for the anticongruence symbols). We will apply those **C**-rules to $T(\simeq)$ which correspond to pure congruence axioms for which $T(\simeq)$ is not closed until we obtain a **C**-closed tree. We can determine that a **C**-rule is applicable to $T(\simeq)$ by finding a node $\langle f, [s_1], \ldots, [s_n], \emptyset\rangle$ in the tree with at least two children (which must be leaves). In other words, we have $\langle [f(\vec{s})], f, [s_1], \ldots, [s_n], \emptyset\rangle$ and $\langle [f(\vec{t})], f, [t_1], \ldots, [t_n], \emptyset\rangle$ in $T(\simeq)$ such that $[s_1] = [t_1]$, $\ldots$, $[s_n] = [t_n]$, and $[f(\vec{s})] \neq [f(\vec{t})]$.

We coarsen $\simeq$ to $\simeq'$ by joining the classes $[f(\vec{s})]$ and $[f(\vec{t})]$ in such a way that $\simeq'$ is the least extension of $\simeq$ for which $f(\vec{s}) \simeq' f(\vec{t})$ holds. This happens by having

$$T(\simeq') = \mathit{Join\_classes}([f(\vec{s})], [f(\vec{t})], T(\simeq))\,.$$

The function *Join_classes* is defined as

$$\mathit{Join\_classes}(a, b, T) = \{\langle e \star c, f, e_1 \star c, \ldots, e_n \star c, \emptyset\rangle \mid \langle e, f, e_1, \ldots, e_n, \emptyset\rangle \in T\}$$

where $c = a \cup b$ and

$$e \star c = \begin{cases} e & \text{if } e \cap c = \emptyset, \\ c & \text{otherwise.} \end{cases}$$

4.1.3 A-Closure of Equivalence Trees

We call a **C**-closed equivalence tree $T(\simeq)$ **A**-*closed* if $\simeq$ is almost closed (see the proof of Thm. 7.). An **A**-rule is applicable to $T(\simeq)$ if there is in $T(\simeq)$ a leaf $\langle [g(\vec{s})], g, [s_1], \ldots, [s_n], \emptyset\rangle$ such that $[g(\vec{s})] = [h(\vec{u})] = [h(\vec{v})]$, and $[u_1] \neq [v_1]$, $\ldots$, $[u_m] \neq [v_m]$. Note that this is possible only if $[s_1] = [t_1]$, $\ldots$, $[s_n] = [t_n]$ holds.

In order to **A**-close $T(\simeq)$ we recursively **A**-close the equivalence trees obtained by **C**-closing the trees $\mathit{Join\_classes}([u_i], [v_i], T(\simeq))$ for all $i = 1, \ldots, m$.

4.2 Efficient Implementation of the C-Closure Algorithm

The full **CA**-algorithm discussed on an abstract level in the preceding subsection runs in exponential time. The most efficient algorithms known for congruence closure, which corresponds to the **C**-closure of an equivalence tree with n nodes, need time $\mathrm{O}(n \log(n))$ on average. Fig. 1 outlines in abstract terms an $\mathrm{O}(n \log(n))$ implementation of the **C**-closure algorithm.

4.2.1 Data Structures

The imperative algorithm in Fig. 1 operates by side-effects on the global variable *tree* which is implemented as an array of ten-tuples indexed by values of type *nodes*. The values of type *nodes* correspond to the nodes in the tree $Tr(\simeq)$ but they are implemented as numbers so they can be used as indices to the array. The values of type *labels* are (logically speaking) representatives of equivalence classes for $\simeq$ and thus they are terms from D. This type is implemented as a subtype of *nodes* where the term $f(s_1, \ldots, s_n)$ is identified with the node $\langle f, [a_1], \ldots, [a_n], \emptyset\rangle$ where a_1, $\ldots$, a_n are of type *labels* and represent in that order the classes $[s_1]$, $\ldots$, $[s_n]$.

The algorithm accesses the global variable *tree* through a set of accessing/modifying properties (similar to object properties in Delphi or C$^\sharp$). The properties are presented on an abstract level, most of them behaving like functions (e.g. *children*). For instance, the application *children*$[p]$ yields the finite set $\{\langle e, p\rangle \mid \langle e, p\rangle \in Tr(\simeq)\}$ of children of the node p. The set is implemented as a doubly-linked list starting from the node *tree*$[p]$.*first_child* and strung together through the fields *next_sibling*, *prev_sibling*. We need a doubly-linked list of siblings so that the removal of a child from *children*$[p]$ can be implemented in time $\mathrm{O}(1)$.

The property *cwl* (*c*hild *w*ith *l*abel) behaves like an (imperative) relation which can be viewed as a partial map yielding for a label a and a node p the child q of p with the label a if it exists, i.e., if $\langle a, p, q\rangle \in \mathit{cwl}$. The last corresponds to $q = \langle [a], p\rangle \in Tr(\simeq)$.

The partial map *cwl* is rather sparse. For a tree with n nodes the map is defined on $\mathrm{O}(n)$ pairs of arguments out of possible $\mathrm{O}(n^2)$. In order to obtain the average complexity $\mathrm{O}(n \log(n))$ of the **C**-closure algorithm it is crucial that the child q be yielded on average in time $\mathrm{O}(1)$. Note that the naive traversal of the set *children*$[p]$ gives the access-time $O(n)$. Probably the best implementation of such a map is by a hash table of the size $O(n)$ (the variable *cwl_implem* in Fig. 1) with hash conflicts doubly-linked through the nodes in the array *tree*. However, at a negligible loss of efficiency we can also choose to implement *cwl* by a balanced tree or by a trie.

```
type nodes                                        — $Tr(\simeq)$
     labels $\subseteq$ nodes                     — the set of representatives of classes in $D/\simeq$
prop children: nodes $\to \mathcal{P}$(nodes)     — $children[p] = \{\langle e, p\rangle \mid \langle e, p\rangle \in Tr(\simeq)\}$
     uses: labels $\to \mathcal{P}$(nodes)        — $uses[a] = \{\langle [a], p\rangle \mid \langle [a], p\rangle \in Tr(\simeq)\}$
     label: nodes $\to$ labels                    — $label\big[\langle [a], p\rangle\big] = a$
     parent: nodes $\to$ nodes                    — $parent\big[\langle [a], p\rangle\big] = p$
     cwl: $\mathcal{P}$(labels $\times$ nodes $\times$ nodes)   — $cwl = \{\langle a, p, \langle [a], p\rangle\rangle \mid \langle [a], p\rangle \in Tr(\simeq)\}$
var  tree: array [nodes] of record parent, first_child: nodes; num_children: $\mathbb{N}$; label: labels;
       next_sibling, prev_sibling, next_use, prev_use, next_conflict, prev_conflict: nodes end
     cwl_implem: hashed array [labels, nodes] of nodes

procedure C-CLOSURE
  while there are sibling leaves with labels $a$, $b$
    JOIN_CLASSES($a, b$)

procedure JOIN_CLASSES($a, b$)
  assume w.l.o.g. that $|uses[a]| \geq |uses[b]|$
  for each $n$ in $uses[b]$
    if $label[n] = b$
      $p \leftarrow parent[n]$
      REMOVE($n$)
      if $\langle a, p, m\rangle \in cwl$
        REMOVE($m$)
        $n \leftarrow$ MERGE_NODES($m, n$)
      $label[n] \leftarrow a$
      ADD($p, n$)

procedure REMOVE($n$)
  remove $\langle label[n], parent[n], n\rangle$ from $cwl$
  remove $n$ from $uses\big[label[n]\big]$
  remove $n$ from $children\big[parent[n]\big]$

function MERGE_NODES($p, q$)
  assume w.l.o.g. that $|children[p]| \geq |children[q]|$
  for each $n$ in $children[q]$
    $c \leftarrow label[n]$
    REMOVE($n$)
    if $\langle c, p, m\rangle \in cwl$
      REMOVE($m$)
      $n \leftarrow$ MERGE_NODES($m, n$)
    ADD($p, n$)
  return $p$

procedure ADD($p, n$)
  $parent[n] \leftarrow p$
  add $n$ to $children[p]$
  add $n$ to $uses\big[label[n]\big]$
  add $\langle label[n], p, n\rangle$ to $cwl$
```

Figure 1: Efficient implementation of the **C**-closure algorithm.

4.2.2 The Working of the Algorithm

The main procedure **C**-CLOSURE of the algorithm in Fig. 1 looks for two siblings leaves with labels a and b and joins them, in a similar way as the function *Join_classes* from Subsect. 4.1.2, by calling the procedure JOIN_CLASSES(a, b). The only difference is that the former works with equivalence classes, while the latter with their representatives.

JOIN_CLASSES finds all occurrences of the label b in the tree by traversing the set $uses[b]$ (implemented by a doubly-linked list) of all nodes with the label b. For every node $\langle b, p\rangle$ in the set, its descendants $\langle \ldots, b, p\rangle$ are merged by MERGE_NODES$(\langle a, p\rangle, \langle b, p\rangle)$ with the descendants $\langle \ldots, a, p\rangle$ of the sibling node $\langle a, p\rangle$ of $\langle b, p\rangle$ if there is such. The label b of $\langle b, p\rangle$ is renamed to a because a will be the new representative of the joined class $[a] \cup [b]$.

The procedure MERGE_NODES(p, q) traverses all children $\langle c, q\rangle$ of q making $\langle c, q\rangle$ a child of p unless there is a child $\langle c, p\rangle$ of p. In that case, the procedure first recursively merges $\langle c, q\rangle$ with $\langle c, p\rangle$, and then makes the result a child of p.

In both JOIN_CLASSES and MERGE_NODES it is crucial that the **for each** loop traverses the smaller set and accesses the children from the larger set in the constant time. This assures that each out of the n

nodes in the tree is removed from some smaller set and added to a larger set at most $\log(n)$ times (the size of the set obtained by the joined and merged nodes is at least double the size of the smaller set). Thus the time complexity of the algorithm is on $O(n \log(n))$ (on average because of the hashing).

5 Conclusions

We have shown how to absorb the propositional content of formulas into sets of identities where the congruence axioms take care of conjunctions, the anticongruence axioms of disjunctions. Negations $\neg A$ are translated to $A \Rightarrow 0 = 1$, and sequents are taken care of by identities connecting congruence with anticongruence symbols. Of course, it is well-known in logic that formulas can be reduced to identities. For instance, the *Primitive recursive arithmetic* (PRA) can be presented in this way. However, one needs a certain amount of arithmetic for this. We have shown that the reduction is possible without any arithmetic by employing in a pleasingly symmetric way congruences and anticongruences.

While we have implemented the **CA**-algorithm, the implementation is not yet ready to be benchmarked and compared to state-of-the-art automated theorem provers. We are now working on a congruence closure based proof system and procedure for generally quantified sequents (clauses).

References

[1] Peter J. Downey, Ravi Sethi, and Robert E. Tarjan. *Variations on the Common Subexpression Problem.* Journal of the ACM, 27(4):758–771, 1980.

[2] Harald Ganzinger, George Hagen, Robert Nieuwenhuis, Albert Oliveras, and Cesare Tinelli. DPLL(T): Fast Decision Procedures. *16th International Conference on Computer Aided Verification (CAV), Boston, July 2004.* LNCS 3114, pp. 175–188, Springer, 2004.

[3] Robert Nieuwenhuis and Albert Oliveras. Congruence closure with integer offsets. *10th Int. Conf. on Logics for Programming, AI and Reasoning (LPAR), Almaty, Kazakhstan, September 2003.* LNAI 2850, pp. 78–90, Springer, 2003.

[4] Joseph R. Shoenfield. *Mathematical Logic.* Addison-Wesley, Reading, Massachusetts, 1967.