Available Projects

René Thiemann

April 25, 2024

Contents

1 Congruence Closure (2-3 persons)

We consider a set ground equations GE such as

- f(g(a)) = h(b)
- f(b) = b
- g(a) = b

  and are interested in the question whether a particular equation is implied GE. For instance the sequence of equality-steps

- f(h(b)) = f(f(g(a))) = f(f(b)) = f(b)

  proves that f(h(b)) = f(b) follows from E.

  Whereas it is easy to validate a given sequence of equality-steps, the problem is to detect whether such a sequence exists for a given equation. To this end, the congruence closure algorithm has been developed which should be partially verified in this project.

> Basic knowledge of term rewriting is helpful for this project. The describtion of the algorithm is based on *Franz Baader and Tobias Nipkow*, *Term Rewriting and All That*, *Chapter 4.3*.

theory *Project-Congruence-Closure*
  imports
    Main
begin

1.1 Definition of Algorithm

We start by definining ground terms where the type of symbols are just strings.

type-synonym *symbol* = *string*

datatype *trm* = *Fun symbol trm list*

type-synonym *eqs* = (*trm* × *trm*)*set*

Define the set of subterms of a term, e.g., the subterms of f(g(a),b) would be $\{f(g(a),b),\ g(a),\ a,\ b\}$.

fun *subt* :: *trm* ⇒ *trm set* **where**
  subt (*Fun f ts*) = *undefined*

Prove two useful lemmas about subterms.

lemma *self-subt*: *u* ∈ *subt u* **sorry**

lemma *subt-trans*: *s* ∈ *subt t* ⟹ *t* ∈ *subt u* ⟹ *s* ∈ *subt u* **sorry**

For a set of ground-equalities, the congruence closure algorithm is in particular interested in all subterms that occur in the equalities.

definition *subt-eqs* **where** *subt-eqs GE* = ⋃ ((λ (*l*,*r*). *subt l* ∪ *subt r*) ‘ *GE*)

From now on fix a specific set of ground-equalities GE.

context
  fixes *GE* :: *eqs*
begin

Define an equality step where one can either replace one side of an equation in GE by the other side (a root-step), or where one can apply a step in a context.

inductive-set *estep* :: *trm rel* **where**
  root: *undefined* ⟹ *undefined* ∈ *estep*
| *ctxt*: (*s*,*t*) ∈ *estep* ⟹ (*Fun f* (*before* @ *s* # *after*), *Fun f* (*before* @ *t* # *after*)) ∈ *estep*

The other important definition is the Cong-operation which given a set of equalities derives new equalities of these by reflexivity, symmetry, transitivity or context.

inductive-set *Cong* :: *eqs* ⇒ *eqs* **for** *E* **where**
  C-keep: *eq* ∈ *E* ⟹ *eq* ∈ *Cong E*
| *C-refl*: (*t*,*t*) ∈ *Cong E*
| *C-sym*: (*s*,*t*) ∈ *E* ⟹ (*t*,*s*) ∈ *Cong E*
| *C-trans*: (*s*,*t*) ∈ *E* ⟹ (*t*,*u*) ∈ *E* ⟹ (*s*,*u*) ∈ *Cong E*
| *C-cong*: *length ss* = *length ts* ⟹ (∀ *i* < *length ts*. (*ss* ! *i*, *ts* ! *i*) ∈ *E*) ⟹ (*Fun f ss*, *Fun f ts*) ∈ *Cong E*

Let us now fix to terms s and t where we are interested in whether GE implies s = t.

context
  fixes *s t* :: *trm*
begin

In the congruence closure algorithm one only is interested in equalities of terms in S.

definition *S* **where** *S* = *subt s* ∪ *subt t* ∪ *subt-eqs GE*

definition *CongS* **where** *CongS E* = *Cong E* ∩ (*S* × *S*)

CCA defines the equalities that are obtained in the i-th iteration of the congruence closure algorithm, which iteratively applies the *local.CongS* operation starting from *GE*.

definition *CCA* **where** *CCA i* = (*CongS* ^^ *i*) *GE*

Prove the following simple inclusions.

lemma *GE-S*: *GE* ⊆ *S* × *S* **sorry**

lemma *GE-CCA*: *GE* ⊆ *CCA i* **sorry**

1.2 Completeness of CCA

The crucial result of the congruence closure algorithm is given in the following lemma on the completeness of the algorithm: if the algorithm has stabilized in the i-th iteration, then all equations in *local.S* × *local.S* that can be derived with arbitrary many steps are also contained in the equalities of CCA.

lemma *esteps-imp-CCA*: **assumes** *CongS* (*CCA i*) = *CCA i*
  shows (*u*,*v*) ∈ *estep*^* ∩ (*S* × *S*) ⟶ (*u*,*v*) ∈ *CCA i*
proof

The proof is by induction on the number of steps and then by the size of the starting term *u*. This is expressed as follows in Isabelle.

assume $(u,v) \in estep\widehat{\ }* \cap (S \times S)$
then obtain n **where** $*$: $u \in S$ $v \in S$ $(u,v) \in estep\widehat{\ }\widehat{\ }n$
by (*auto simp*: *rtrancl-power*)
obtain m **where** $m = (n,size\ u)$ **by** *auto*
with $*$ **show** $(u,v) \in CCA\ i$
proof (*induction m arbitrary*: *u v n rule*: *wf-induct*[*OF wf-measures*[*of* [*fst*,*snd*]]])
case (*1 m u v n*)

For handling the induction, we first convert the derivation into a function which gives us all intermediate terms via function w.

from *1(4)*[*unfolded relpow-fun-conv*] **obtain** w
where w: $w\ 0 = u$ $w\ n = v$ $(\forall i<n.\ (w\ i,\ w\ (Suc\ i)) \in estep)$ **by** *auto*

And the proof now proceeds by case-analysis on whether any of these steps was a root step or whether all steps are non-root.

show *?case* **sorry**
qed
qed

Next, completeness of CCA is easily established

lemma *esteps-imp-CCA-st*: **assumes** $CongS\ (CCA\ i) = CCA\ i$
shows $(s,t) \in estep\widehat{\ }* \longrightarrow (s,t) \in CCA\ i$
sorry

1.3 Soundness of CCA

The crucial step to prove soundness is the following lemma, which might require some further auxiliary lemmas.

lemma *Cong-esteps*: $E \subseteq estep\widehat{\ }* \Longrightarrow Cong\ E \subseteq estep\widehat{\ }*$ **sorry**

But you can easily verify that $?E \subseteq estep^* \Longrightarrow Cong\ ?E \subseteq estep^*$ is the key to prove soundness of CCA.

lemma *CCA-imp-esteps*: $CCA\ i \subseteq estep\widehat{\ }*$ **sorry**

1.4 Correctness of CCA

Having soundness and completeness, correctness is simple.

theorem *congruence-closure-correct*: **assumes** $CongS\ (CCA\ i) = CCA\ i$
shows $(s,t) \in estep\widehat{\ }* \longleftrightarrow (s,\ t) \in CCA\ i$
sorry

1.5 Termination of CCA

The precondition $local.CongS\ (local.CCA\ i) = local.CCA\ i$ can be discharged proving termination of the congruence closure algorithm which just computes the least i such that the precondition is satisfied. The existence of such an i follows from the fact that CCA i is increasing with increasing i and CCA i is bounded by the finite set of terms S x S, assuming finiteness of GE.

Formulating and proving these facts in Isabelle is another task of this project, if it is conducted as a 3-person project.

context
  assumes *finite GE*
begin

lemma *finite-S*: *finite S* **sorry**

lemma *CCA-SS*: $CCA\ n \subseteq S \times S$ **sorry**

lemma *CCA-mono*: $CCA\ n \subseteq CCA\ (Suc\ n)$ **sorry**

lemma *i-exists*: $\exists\ i.\ CongS\ (CCA\ i) = CCA\ i$ **sorry**

definition *fixpointI* $= (LEAST\ i.\ CongS\ (CCA\ i) = CCA\ i)$

lemma *fixpointI*: $CongS\ (CCA\ fixpointI) = CCA\ fixpointI$
  sorry

lemma *congruence-closure*: $(s,t) \in estep\hat{}* \longleftrightarrow (s,\ t) \in CCA\ fixpointI$
  using *congruence-closure-correct*[*OF fixpointI*] **.**

Design an algorithm to compute *local.fixpointI* and prove its termination. The algorithm itself of course must not use *local.fixpointI*, but the measure for proving termination might very well depend on this unknown constant.

end
end
end
end

2 Propositional Logic (2 persons)

Soundness and completeness of a logic establish that the syntactic notion of provability is equivalent to the semantic notation of logical entailment.

In this project you will formally prove soundness and completeness of a specific set of natural deduction rules for propositional logic.

theory *Project-Logic*
  imports *Main*
begin

2.1 Syntax and Semantics

Propositional formulas are defined by the following data type (that comes with some syntactic sugar):

type-synonym *id* = *string*
datatype *form* =
  Atom id
| *Bot* ($\bot_p$)
| *Neg form* ($\neg_p$ - [*68*] *68*)
| *Conj form form* (**infixr** $\wedge_p$ *67*)
| *Disj form form* (**infixr** $\vee_p$ *67*)
| *Impl form form* (**infixr** $\rightarrow_p$ *66*)

Define a function *eval* that evaluates the truth value of a formula with respect to a given truth assignment.

fun *eval* :: (*id* ⇒ *bool*) ⇒ *form* ⇒ *bool*
  where
  eval v φ ⟷ *undefined*

Using *eval*, define semantic entailment of a formula from a list of formulas.

definition *entails* :: *form list* ⇒ *form* ⇒ *bool* (**infix** ⊨ *51*)
  where
  $\Gamma \models \varphi$ ⟷ *undefined*

2.2 Natural Deduction

The natural deduction rules we consider are captured by the following inductive predicate *proves* P φ, with infix syntax $P \vdash \varphi$, that holds whenever a formula φ is provable from a list of premises P.

inductive *proves* (**infix** ⊢ *58*)
  where
  premise: $\varphi \in set\ P \Longrightarrow P \vdash \varphi$
| *conjI*: $P \vdash \varphi \Longrightarrow P \vdash \psi \Longrightarrow P \vdash \varphi \wedge_p \psi$
| *conjE1*: $P \vdash \varphi \wedge_p \psi \Longrightarrow P \vdash \varphi$
| *conjE2*: $P \vdash \varphi \wedge_p \psi \Longrightarrow P \vdash \psi$
| *impI*: $\varphi \# P \vdash \psi \Longrightarrow P \vdash (\varphi \rightarrow_p \psi)$
| *impE*: $P \vdash \varphi \Longrightarrow P \vdash \varphi \rightarrow_p \psi \Longrightarrow P \vdash \psi$
| *disjI1*: $P \vdash \varphi \Longrightarrow P \vdash \varphi \vee_p \psi$
| *disjI2*: $P \vdash \psi \Longrightarrow P \vdash \varphi \vee_p \psi$
| *disjE*: $P \vdash \varphi \vee_p \psi \Longrightarrow \varphi \# P \vdash \chi \Longrightarrow \psi \# P \vdash \chi \Longrightarrow P \vdash \chi$
| *negI*: $\varphi \# P \vdash \bot_p \Longrightarrow P \vdash \neg_p \varphi$
| *negE*: $P \vdash \varphi \Longrightarrow P \vdash \neg_p \varphi \Longrightarrow P \vdash \bot_p$
| *botE*: $P \vdash \bot_p \Longrightarrow P \vdash \varphi$
| *dnegE*: $P \vdash \neg_p \neg_p \varphi \Longrightarrow P \vdash \varphi$

Prove that ⊢ is monotone with respect to premises, that is, we can arbitrarily extend the list of premises in a valid prove.

lemma *proves-mono*:
assumes $P \vdash \varphi$ **and** *set* $P \subseteq$ *set* Q
shows $Q \vdash \varphi$
sorry

Prove the following derived natural deduction rules that might be useful later on:

lemma *dnegI*:
assumes $P \vdash \varphi$
shows $P \vdash \neg_p \neg_p\ \varphi$
sorry

lemma *pbc*:
assumes $\neg_p\ \varphi\ \#\ P \vdash \bot_p$
shows $P \vdash \varphi$
sorry

lemma *lem*:
$P \vdash \varphi \lor_p \neg_p\ \varphi$
sorry

lemma *neg-conj*:
assumes $\chi \in \{\varphi, \psi\}$ **and** $P \vdash \neg_p\ \chi$
shows $P \vdash \neg_p\ (\varphi \land_p \psi)$
sorry

lemma *neg-disj*:
assumes $P \vdash \neg_p\ \varphi$ **and** $P \vdash \neg_p\ \psi$
shows $P \vdash \neg_p\ (\varphi \lor_p \psi)$
sorry

lemma *trivial-imp*:
assumes $P \vdash \psi$
shows $P \vdash \varphi \rightarrow_p \psi$
sorry

lemma *vacuous-imp*:
assumes $P \vdash \neg_p\ \varphi$
shows $P \vdash \varphi \rightarrow_p \psi$
sorry

lemma *neg-imp*:
assumes $P \vdash \varphi$ **and** $P \vdash \neg_p\ \psi$
shows $P \vdash \neg_p\ (\varphi \rightarrow_p \psi)$
sorry

2.3 Soundness

Prove soundness of $\vdash$ with respect to $\models$.

lemma *proves-sound*:
  assumes $P \vdash \varphi$
  shows $P \models \varphi$
  sorry

2.4 Completeness

Prove completeness of $\vdash$ with respect to $\models$ in absence of premises.

lemma *prove-complete-Nil*:
  assumes $[] \models \varphi$
  shows $[] \vdash \varphi$
  sorry

Now extend the above result to also incorporate premises.

lemma *proves-complete*:
  assumes $P \models \varphi$
  shows $P \vdash \varphi$
  sorry

Conclude that semantic entailment is equivalent to provability.

lemma *entails-proves-conv*:
  $P \models \varphi \longleftrightarrow P \vdash \varphi$
  sorry

end