Markus Pomrehn

13. März 2016

Zusammenfassung

1 Einführung

2 Beweisformat

2.1 Alt

2.1.1 Vorhandene Beweisregeln

expand: SMTInterpol erweitert assoziative Funktionen, je nach dem ob sie links- oder rechts-assoziativ. Aus $f(t_1, t_2..., t_{n-1}, t_n)$ wird bei links-assoziativen Funktionen $f(f(...f(t_1, t_2), ...), t_n)$ und bei rechts-assoziativen Funktionen $f(t_1, f(..., f(t_{n-1}, t_n)))$. Außerdem wird bei verkettbaren Funktionen aus $f(t_1, t_2, t_3..., t_{n-1}, t_n) = f(t_1, t_2) \wedge f(t_2, t_3) \wedge ... \wedge f(t_{n-1}, t_n)$.

expandDef: SMTLib ermöglicht es Benutzern eigene Funktionen zu definieren. Diese Funktion ersetzt n durch ihre Defintition d.

$$\textsc{ExpandDef}\ \frac{}{n(t) = d[t]}$$

trueNotFalse: Gleichheit von Boolschen Termen, mit einem Term der falsch ist und einem der wahr ist, vereinfache zu falsch.

$$\textsc{TrueNotFalse}\ \frac{}{(=_{i \in I}\ t_i) = false}\ \exists j, k \in I.\ t_j = true \wedge t_k = false$$

constDiff: Gleichheit unterschiedlicher Konstanten ist falsch.

$$\textsc{ConstDiff}\ \frac{}{(=_{i \in I}\ t_i) = false}\ \exists j, k \in I.\ t_j = c_j \wedge t_k = c_k \wedge c_j \neq c_k$$

eqTrue: Gleicheit zu wahr, wird durch ein Konjunkt ersetzt.

$$\textsc{EqTrue}\ \frac{}{(=_{i \in I}\ t_i) = (and_{i' \in I'}\ t_{i'})}\ \exists j \in I.\ t_j = true \wedge I' \subset I \wedge j \notin I'$$

eqFalse: Gleichheit zu falsch, wird durch die Negation der Veroderung der zu falsch gleichen Terme ersetzt.

$$\textsc{EqFalse}\ \frac{}{(=_{i \in I}\ t_i) = (not\ (or_{i' \in I'}\ t_{i'}))}\ \exists j \in I.\ t_j = false \wedge I' \subset I \wedge j \notin I'$$

eqSame: Wenn alle Terme gleich sind, vereinfache zu wahr.

$$\textsc{EqSame}\ \frac{}{(=_{i \in I}\ t_i) = true}\ \forall i, j \in I.\ t_i \equiv t_j$$

eqSimp: Ersetze gleiche Terme durch diesen Term.

$$\textsc{EqSimp}\ \frac{}{(=_{i\in I}\ t_i) = (=_{i'\in I'}\ t_{i'})}\ I' \subset I \wedge |I'| = |\{t_i.\ i \in I\}| \wedge \{t_i.\ i \in I\} = \{t_j.\ j \in I'\}$$

eqBinary: Ersetze Gleichheit von binären Variablen durch die Negation der Veroderung der Negation der Gleichheit.

$$\textsc{EqBinary}\ \frac{(=_{i\in I}\ t_i) = (=_{i'\in I'}\ t_{i'})}{(=_{i\in I}\ t_i) = (not\ (or_{i',i''\in I'}\ (not\ (=\ t_{i'}\ t_{i''}))))}$$

distinctBool: Ersetze Ungleichheit von mehr als 2 boolschen Variablen durch falsch.

$$\textsc{DistinctBool}\ \frac{}{(distinct_{i\in I}\ t_i) = false}\ |I| > 2 \wedge sort(t_i) = Bool$$

distinctSame: Ersetze Ungleichheit gleicher Konstanten zu falsch.

$$\textsc{DistinctSame}\ \frac{}{(=\ distinct_{i\in I}\ t_i) = false}\ \exists j,k \in I.\ t_j \equiv t_k \wedge j \neq k$$

distinctNeg: Ersetze die Ungleichheit einer Variablen und ihrer Negation durch wahr.

$$\textsc{DistinctNeg}\ \frac{}{(distinct\ t_0\ t_1) = true}\ t_0 \text{ is negation of } t_1$$

distinctTrue: Ersetze Ungleichheit zweier Variablen, bei der die eine wahr ist, durch die Negation der anderen.

$$\textsc{DistinctTrue}\ \frac{}{(distinct\ t_0\ t_1) = (not\ t_{1-i})}\ t_i \equiv true \text{ for } i = 0, 1$$

distinctFalse: Ersetze Ungleichheit zweier Variablen, bei der die eine falsch ist, durch die andere.

$$\textsc{DistinctFalse}\ \frac{}{(distinct\ t_0\ t_1) = t_{1-i}}\ t_i \equiv false \text{ for } i = 0, 1$$

distinctBoolEq: Ersetze Ungleichheit zweier boolschen Variablen, durch Gleichheit der beiden Variablen, wobei eine negiert wird.

$$\textsc{DistinctBoolEq}\ \frac{}{(distinct\ t_0\ t_1) = (=\ t'_0\ t'_1)}\ sort(t_0) = sort(t_1) = Bool \wedge ((t'_0, t'_1) = (\neg t_0, t_1) \vee (t'_0, t'_1) = (t_0, \neg t_1))$$

distinctBinary: Ersetze Ungleichheit mehrerer Variablen, durch die Negation der Veroderung paarweiser Gleichheit.

$$\textsc{DistinctBinary}\ \frac{}{(distinct_{i\in I}\ t_i) = (not\ (or_{i',i''\in I}\ (=\ t_{i'}\ t_{i''})))}$$

notSimp: Negation Vereinfachung.

$$\textsc{NotSimp}\ \frac{}{(not\ F) = F'}\ F' \equiv \begin{cases} false & \text{if } F \equiv true \\ true & \text{if } F \equiv false \\ G & \text{if } F \equiv (not\ G) \end{cases}$$

orSimp: Lasse mehrfaches Vorkommen eines Terms oder falsch weg.

$$\textsc{OrSimp}\ \frac{}{(or_{i\in I}\ t_i) = F}\ F \equiv (or_{i\in I'}\ t_i)\ \text{where}\ I' \subset I\ \text{or}\ F \equiv t_i$$

$$\textsc{OrSimp}\ \frac{}{F \vee \bot = F}$$

orTaut: Vereinfache disjunktive Formel, die wahr oder ein Literal in positiver und negierter Form enthalten, zu wahr.

$$\textsc{OrTaut}\ \frac{}{(or_{i\in I}\ t_i) = true}\ \exists j \in I.\ t_i \equiv\ true \vee \exists j, k \in I.\ t_j \equiv\ (not\ t_k)$$

iteTrue: Ersetze if-then-else (ite), bei denen die Bedingung wahr ist, durch den then Teil.

$$\textsc{IteTrue}\ \frac{}{(ite\ true\ t_1\ t_2) = t_1}$$

iteFalse: Ersetze ite, bei denen die Bedingun falsch ist, durch den else Teil.

$$\textsc{IteFalse}\ \frac{}{(ite\ false\ t_1\ t_2) = t_2}$$

iteSame: Ersetze ite, bei denen der then Teil und der else Teil identisch sind, durch eben diesen Term.

$$\textsc{IteSame}\ \frac{}{(ite\ t_0\ t_1\ t_1) = t_1}$$

iteBool1: Ersetze ite, bei denen der then Teil wahr ist und der else Teil falsch ist, durch die Bedingung.

$$\textsc{IteBool1}\ \frac{}{(ite\ t_0\ true\ false) = t_0}$$

iteBool2: Ersetze ite, bei denen der then Teil falsch ist und der else Teil wahr ist, durch die Negation der Bedingung.

$$\textsc{IteBool2}\ \frac{}{(ite\ t_0\ false\ true) = (not\ t_0)}$$

iteBool3: Ersetze ite, bei denen der then Teil wahr ist, durch die Veroderung der Bedingung und des else Teils.

$$\textsc{IteBool3}\ \frac{}{(ite\ t_0\ true\ t_2) = (or\ t_0\ t_2)}$$

iteBool4: Ersetze ite, bei denen der then Teil falsch ist, durch die Negation der Veroderung von Bedingung und der Negation des else Teils.

$$\textsc{IteBool4}\ \frac{}{(ite\ t_0\ false\ t_2) = (not\ (or\ t_0\ (not\ t_2)))}$$

iteBool5: Ersetze ite, bei denen der else Teil wahr ist, durch die Veroderung der Negation der Bedingung und des then Teils.

$$\textsc{IteBool5}\ \frac{}{(ite\ t_0\ t_1\ true) = (or\ (not\ t_0)\ t_1)}$$

iteBool6: Ersetze ite, bei denen der else Teil falsch ist, durch die Negation der Veroderung der negierten Bedingung und then Teils.

$$\textsc{IteBool6}\ \frac{}{(ite\ t_0\ t_1\ false) = (not\ (or\ (not\ t_0)\ (not\ t_1)))}$$

andToOr: Ersetze Verundungen durch die Negation der Veroderung der Negationen.

$$\textsc{AndToOr}\ \frac{}{(and\ t_i) = (not\ (or\ (not\ t_i)))}$$

xorToDistinct: Ersetze XOR durch Ungleich.

$$\textsc{XorToDistinct}\ \frac{}{(xor\ t_1\ t_2) = (distinct\ t_1\ t_2)}$$

impToOr: Ersetze Implikation durch Oder.

$$\textsc{ImpToOr}\ \frac{}{(implies_{i\in I}\ t_i\ t) = (or_{i\in I}\ t\ (not\ t_i))}$$

canonicalSum: Summen werden so dargestellt, das nur die äusserste Funktion Plus ist. Mehrfaches auftreten wird zu einem Produkt zusammengefasst.

$$\textsc{CanonicalSum}\ \frac{}{t = t'}\ t' \cong t$$

leqToLeq0: Ersetze kleiner gleich durch kleiner gleich Null.

$$\textsc{LeqToLeq0}\ \frac{}{(<=\ t_1\ t_2) = (<=\ t'\ 0)}\ t' \cong t_1 - t_2$$

ltToLeq0: Ersetze kleiner durch kleiner gleich Null.

$$\textsc{LtToLeq0}\ \frac{}{(<\ t_1\ t_2) = (not\ (<=\ t'\ 0))}\ t' \cong t_2 - t_1$$

geqToLeq0: Ersetze größer gleich durch kleiner gleich.

$$\textsc{GeqToLeq0}\ \frac{}{(>=\ t_1\ t_2) = (<=\ t'\ 0)}\ t' \cong t_2 - t_1$$

gtToLeq0: Ersetze größer durch kleiner gleich.

$$\textsc{GtToLeq0}\ \frac{}{(>\ t_1\ t_2) = (not\ (<=\ t'\ 0))}\ t' \cong t_1 - t_2$$

leqTrue: Ersetze (<= c 0) durch wahr, falls c ≤ 0 ist.

$$\textsc{LeqTrue}\ \frac{}{(<=\ c\ 0) = true}\ c \leq 0$$

leqFalse: Ersetze (<= c 0) durch falsch, falls c > 0 ist.

$$\textsc{LeqFalse}\ \frac{}{(<=\ c\ 0) = false}\ c > 0$$

desugar: Diese Regeln werden angewandt, wenn Integers und Reals gemischt in der Formel auftauchen.

$$\textsc{desugar}\ \frac{}{F[t] = F[t.0]}\ t \text{ is an integer} \qquad \textsc{desugar}\ \frac{}{F[t] = F[to\_real(t)]}\ t \text{ is not an integer}$$

divisible: Entfernen des Teilbar-Operators.

$$\textsc{divisible}\ \frac{}{((\_\ divisible\ 1)\ t) = true}$$

$$\textsc{divisible}\ \frac{}{((\_\ divisible\ n)\ t) = false}\ t \text{ is integer constant not divisible by } n$$

$$\textsc{divisible}\ \frac{}{((\_\ divisible\ n)\ t) = true}\ t \text{ is integer constant divisible by } n$$

$$\textsc{divisible}\ \frac{}{((\_\ divisible\ n)\ t) = (=\ \ t\ (*\ n\ (div\ t\ n)))}$$

modulo: Ersetze Modulo durch Geteilt.

$$\textsc{modulo}\ \frac{}{(mod\ x\ y) = (+\ x\ (*\ (-y)\ (div\ x\ y)))}$$

modulo1: Modulo 1 ist gleich 0.

$$\textsc{mod1}\ \frac{}{(mod\ c\ 1) = 0}$$

modulo-1: Modulo -1 ist gleich 0.

$$\textsc{mod-1}\ \frac{}{(mod\ c\ (-1)) = 0}$$

moduloConst: Berechne den Rest, falls es Konstanten sind.

$$\textsc{modConst}\ \frac{}{(mod\ c_1\ c_2) = d}\ d \equiv \begin{cases} c_1 - c_2\lfloor\frac{c_1}{c_2}\rfloor \text{ if } c_2 > 0 \\ c_1 - c_2\lceil\frac{c_1}{c_2}\rceil \text{ if } c_2 < 0 \end{cases}$$

div1: Geteilt durch 1 ist gleich dem Dividenden.

$$\textsc{div1}\ \frac{}{(div\ c\ 1) = c}$$

div-1: Geteilt durch -1 ist gleich dem negativem Dividenden.

$$\textsc{div-1}\ \frac{}{(div\ c\ (-1)) = d}\ d \equiv -c$$

divConst: Berechne die Division, wenn es um Konstanten geht.

$$\textsc{divConst}\ \frac{}{(div\ c_1\ c_2) = d}\ d \equiv \begin{cases} \lfloor\frac{c_1}{c_2}\rfloor \text{ if } c_2 > 0 \\ \lceil\frac{c_1}{c_2}\rceil \text{ if } c_2 < 0 \end{cases}$$

toInt: Wandle reelle Zahlen in Integer um.

$$\textsc{toInt}\ \frac{}{(to\_int\ r) = v}\ v \equiv \lfloor r \rfloor$$

toReal: Wandle Integer Zahlen in reelle um.

$$\textsc{toReal}\ \frac{}{(to\_realc) = c.0}$$

flatten: Verschachtelte Oder werden zu einem grossen umgewandelt.

strip: Entfernen der Benennung.

$$\textsc{strip}\ \frac{}{(!\ F\ :annotations) = F}$$

storeOverStore: Bei einem verschachteltem Store wird das Innere entfernt.

$$\textsc{storeOverStore}\ \frac{}{(store\ (store\ a\ i\ v_1)\ i\ v_2) = (store\ a\ i\ v_2)}$$

selectOverStore: Wenn man auf die Position zugreift, die gerade beschrieben wurde, erhält man das was gerade geschrieben wurde und wenn man eine andere Position liest, bekommt das was da vorher schon drin stand.

$$\textsc{selectOverStore}\ \frac{}{(select\ (store\ a\ i\ v)\ i) = v}$$

$$\textsc{selectOverStore}\ \frac{}{(select\ (store\ a\ c_1\ v)\ c_2) = (select\ a\ c_2)}\ c_1 \neq c_2$$

storeRewrite: Es gibt 2 Möglichkeiten einen store in einen select umzuwandeln.

$$\textsc{storeRewrite}\ \frac{}{(=\ a\ (store\ a\ i\ v)) = (=\ (select\ a\ i)\ v)}$$

$$\textsc{storeRewrite}\ \frac{}{(=\ (store\ a\ i\ v)\ a) = (=\ (select\ a\ i)\ v)}$$

2.1.2 Beispielbeweis

Zeile 1: *(@clause*

Zeile 2: *(@eq*

Zeile 3: *(@eq*

Zeile 4: *@asserted (or c (not (or a b (not a)))))*

Zeile 5: *(@rewrite (! (= (or a b (not a)) true) :orTaut))*

Zeile 6: *(@rewrite (! (= (not true) false) :notSimp)*

Zeile 7: *(@rewrite (! (= (or c false) c) :orSimp)))*

Zeile 8: *(@intern (= c (! c :quoted))))*

Zeile 9: *(! c :quoted))*

Im alten Beweisformat wird nicht darauf eingegangen in welchem Kontext das umzuschreibende Element auftaucht und ob es mehrfach auftritt. Die rewrite-Regeln werden einfach aufgesammelt. Zeile 5 - 7 sind die rewrite-Regeln. Zeile 4 wird asserted. Das Ergebnis des @eq in Zeile 3 ist c und das des @eqs aus Zeile 2 ist das gesamt Ergebnis (! c :quoted).

2.2 Neu

2.2.1 Hinzukommende Beweisregeln

Reflexivität: Gibt für einen Term t ein Beweis für (= t t) zurück
Transitivität: Gibt für einen Beweis für (= a b) und (= b c) ein Beweis für (= a c) zurück, die Regel soll links assoziativ sein
Kongruenz: Gibt für einen Beweis für (= s (f t_1 ... t_n)) und (= t t') einen Beweis für (= s (f t_1' ... t_n')) wobei t_i' = t' falls t_i = t und für den Rest gilt t_i' = t_i.

2.2.2 Beispielbeweis

Zeile 1: *(@clause*
Zeile 2: *(@eq*
Zeile 3: *(@asserted (or c (not (or a b (not a)))))*
Zeile 4: *(@trans*
Zeile 5: *(@cong*
Zeile 6: *(@refl (or c (not (or a b (not a)))))*
Zeile 7: *(@trans*
Zeile 8: *(@cong*
Zeile 9: *(@refl (not (or a b (not a))))*
Zeile 10: *(@rewrite (! (= (or a b (not a)) true) :orTaut)))*
Zeile 11: *(@rewrite (! (= (not true) false) :notSimp))))*
Zeile 12: *(@rewrite (! (= (or c false) c) :orSimp))*
Zeile 13: *(@intern (= c (! c :quoted)))))*
Zeile 14: *(! c :quoted))*

Beim neuen Beweisformat wird mit der Kongruenz-Regel genau beschrieben an welcher Stelle ersetzt wird, was dann mit der Transitivität-Regel weiter umgeschrieben kann. In obigem Beweis wird mit der Kongruenz-Regel bis zur innersten Ersetzung abgestiegen und dort umgeschrieben. Die Eingabe sind die Reflexivität-Regel (@refl (not (or a b (not a)))) und der Rewrite-Schritt (@rewrite (! (= (or a b (not a)) true) :orTaut))). Das Ergebnis ist der Beweis (= (not (or a b (not a) (not true)). Die drüber stehenden Transitivität liefert mit dem Beweis der Kongruenz und dem Rewrite-Schritt (@rewrite (! (= (not true) false) :notSimp)) den Beweis für (= (not (or a b (not a))) false). Mit diesem Beweis und der Reflexivität-Regel, liefert die übergeordnete Kongruenz-Regel (= (or c (not (or a b (not a)))) (or c false)). Darauf folgt eine 3-fache Transitivität-Regel, die hat als Eingabe den Beweis der Kongruenz-Regel (= (or c (not (or a b (not a)))) (or c false)), die Rewrite-Schritt (@rewrite (! (= (or c false) c) :orSimp)) und den Intern-Schritt (@intern (= c (! c :quoted))))). Das Ergebnis ist der Beweis (= (or c (not (or a b (not a)))) (! c :quoted)) so das das @eq das gewünschte (! c :quoted) ausspuckt.

3 Entwurf

Wie wird ein Beweis generiert? Beispiel Formel und der Transformationsbeweis

4 Implementation

Alt: NonRecursive beschreiben. Klassen beschreiben: ProofTracker, TermCompiler, Clausifier

Neu: Änderungen an ProofTracker/TermCompiler, Clausifier beschreiben. Neue Interaktion und Interface zwischen den Klassen beschreiben.

5 Verwandte Arbeiten

6 Fazit und Ausblick