PCC

Martin Wildmoser

7th February 2005

Contents

theory *EX-Sum* = *VCGExec*:

0.0.1 Example - Sum

0.0.2 Name Declarations

constdefs
Adder :: *cname*
Adder ≡ ''*Adder*''
sum :: *mname*
sum ≡ ''*sum*''
n::*vname*
n ≡ ''*n*''
this::*nat*
this ≡ *0*
k::*nat*
k≡*1*
res::*nat*
res≡*2*

—

0.0.3 Program Code

```
consts comment ::instr ⇒ string ⇒ instr ((- −− -) [61,60] 60)
defs comment-def [simp]:
comment i s ≡ i

constdefs arg::int
arg ≡ 65535

constdefs
 StartC :: jvm-method cdecl
 StartC ≡ (Start,(Object, [],[(main,[],Integer,(2,2,[
 New Adder,
 Store 1,
 Load 1,
 Push (Intg arg),
 Putfield n Adder,
 Load 1,
 Invoke sum 0 −− ''main-call'',
 Return −− ''main-ret'',
 Push (Intg −1), Return −− ''handler for NullPointer Ex.'' ,
 Push (Intg −1), Return −− ''handler for ClassCast Ex.'',
 Push (Intg −1), Return −− ''handler for OutOfMemory Ex.''],
 [(0,7,NullPointer,8,0),
  (0,7,ClassCast,10,0),
  (0,7,OutOfMemory,12,0)]))]))

constdefs
  AdderC :: jvm-method cdecl
  AdderC ≡ (Adder, (Object,[(n,Integer)],[
(sum,[],Integer,(2,2,[
  Push (Intg 0) −− ''sum-pre'' ,
  Store k,
  Push (Intg 0),
  Store res,
  Load k −− ''sum-inv'',
  Load this,
  Getfield n Adder,
  IfIntLeq 10,
  Load k,
  Push (Intg 1),
  IAdd,
  Store k,
  Load res,
  Load k,
  IAdd,
```

$Store\ res$,
$Goto\ {-12}\ {--}\ ''jumps\ to\ sum\text{-}inv''$,
$Load\ res$,
$Return\ {--}\ ''sum\text{-}post'']$,[]))]))

—

0.0.4 Annotations

constdefs

$main\text{-}call::expr$
$main\text{-}call \equiv And\ [Gf\ n\ Adder\ (St\ 0) \doteq Cn\ (Intg\ arg)]$
$main\text{-}ret::expr$
$main\text{-}ret \equiv St\ 0 \doteq Cn\ (Intg\ ((arg * (arg+1))\ div\ 2))$
$sum\text{-}pre::expr$
$sum\text{-}pre \equiv And\ [\ Ty\ (Gf\ n\ Adder\ (Rg\ 0))\ Integer,\ Rg\ 0 \doteq Call\ (St\ 0),$
$Gf\ n\ Adder\ (Rg\ 0) \doteq Call\ (Gf\ n\ Adder\ (St\ 0)),$
$(Gf\ n\ Adder\ (Rg\ 0)) \preceq (Cn\ (Intg\ 65535)),$
$(Cn\ (Intg\ 0)) \preceq (Gf\ n\ Adder\ (Rg\ 0))\ ]$
$sum\text{-}inv::expr$
$sum\text{-}inv \equiv And\ [\ Ty\ (Gf\ n\ Adder\ (Rg\ 0))\ Integer,$
$((Cn\ (Intg\ 2)) \otimes (Rg\ res)) \doteq (Rg\ k \otimes (Rg\ k \oplus (Cn\ (Intg\ 1)))),$
$Gf\ n\ Adder\ (Rg\ 0) \doteq Call\ (Gf\ n\ Adder\ (St\ 0)),$
$(Gf\ n\ Adder\ (Rg\ 0)) \preceq (Cn\ (Intg\ 65535)),$
$(Cn\ (Intg\ 0)) \preceq (Gf\ n\ Adder\ (Rg\ 0)),\ Rg\ k \preceq (Gf\ n\ Adder\ (Rg\ 0)),$
$(Cn\ (Intg\ 0)) \preceq Rg\ k]$

$sum\text{-}post::expr$
$sum\text{-}post \equiv And\ [Ty\ (Gf\ n\ Adder\ (Rg\ 0))\ Integer,\ St\ 0 \doteq Rg\ res,$
$Gf\ n\ Adder\ (Rg\ 0) \doteq Call\ (Gf\ n\ Adder\ (St\ 0)),$
$((Cn\ (Intg\ 2)) \otimes (Rg\ res)) \doteq$
$((Call\ (Gf\ n\ Adder\ (St\ 0))) \otimes ((Call\ (Gf\ n\ Adder\ (St\ 0))) \oplus (Cn\ (Intg\ 1))))]$

0.0.5 Packing code and annotations

constdefs

$prog::jbc\text{-}prog$
$prog \equiv (SystemClasses\ @\ [AdderC, StartC],$
$[((Start, main, 6), main\text{-}call),$
$((Start, main, 7), main\text{-}ret),$
$((Start, main, 8), TT),$
$((Start, main, 10), TT),$
$((Start, main, 12), TT),$
$((Adder, sum, 0), sum\text{-}pre),$
$((Adder, sum, 4), sum\text{-}inv),$
$((Adder, sum, 18), sum\text{-}post)])$

—

0.0.6 Generate ML code for the VCG

generate-code (*EX-Sum.ML*) [*term-of*]
  wf-jvm-prog-phi = λ Φ (*P*::*jvm-prog*). *wf-jvm-prog-phi* Φ *P*
  wf = *wf*
  opt = *opt*
  vcg = *vcgTy*
  prog = *prog*

— If a program is not wellformed, the following functions help to find out why.

  phi-prog = *map-of2* (*convert-pt* (*prog-kil* (*fst prog*)))
  wfS = *wfS*
  wf-jvm-prog-phiS = λ Φ (*P*::*jvm-prog*). *wf-jvm-prog-phiS* Φ *P*
  initjob = *initjob* (*fst prog*) *Start* ''*main*''
  nxtjob = *nxtjob* (*fst prog*) *Start* ''*main*''
  printjob = λ *job*. *printjob* (*fst prog*) *Start* ''*main*'' *job*
  parsejob = *parsejob*

0.0.7 Verification Condition

ML {∗ *use EX-Sum.ML* ∗}
ML {∗ *wf prog*; ∗}
ML {∗ *val vc* = *opt* (*vcg prog*); ∗}
ML {∗ *val pvc* = (*Pretty.str-of* (*Sign.pretty-term* (*sign-of* (*the-context* ())) (*term-of-expr vc*))); ∗}

— now we trasfer the ML result back to Isabelle
ML-setup {∗
val t = *term-of-expr vc*;
val T = *fastype-of t*;
Context.>> (*fn thy* => *thy* |>
  Theory.add-consts-i [(*vc*, *T*, *NoSyn*)] |>
  (*fst o PureThy.add-defs-i false* [((*vc-def*, *Logic.mk-equals* (*Const* (*EX-Sum.vc*, *T*), *t*)), [])]));
∗}

— the verification condition is now defined as constant vc::expr

0.0.8 Proving the Verification Condition

Here we prove the vc (via the semantics of formulae). The evaluation simpset from VCGexec.thy is tuned for this purpose. Some of the evaluation rules (evalEevalEs.simps) are ommitted, because this keeps the expansion overhead small.

The following lemma is used for the verification of the example. Alternatively one could use a tactic for bounded arithemtics.**lemma** *special-bounded-mult*: $[\![\ 2 * (a::int) = b * (b+1);\ 0 \leq (b::int);\ b \leq c\ ]\!] \Longrightarrow\ b + a \leq c + ((c * (c + 1))\ div\ 2)$

declare *evalE-evalEs.simps* [*simp del*]

declare *sem-simps* [*simp add*]

lemma *vcg-prog-holds*:
prog ⊢ *vc*
apply (*simp only*: *provable-def vc-def*)
apply (*safe intro*!: *And0 AndI′*)

```
 1. prog,[] ⊢ And [Ty (Gf ''n'' ''Adder'' (Rg 0)) Integer,
                   (Cn (Intg 2) ⊗ Rg (Suc (Suc 0))) ≐
                   (Rg (Suc 0) ⊗ Num (Rg (Suc 0)) num-op.Plus (Cn (Intg 1))),
                   Gf ''n'' ''Adder'' (Rg 0) ≐ Call (Gf ''n'' ''Adder'' (St 0)),
                   Gf ''n'' ''Adder'' (Rg 0) ≼ Cn (Intg 65535),
                   Cn (Intg 0) ≼ Gf ''n'' ''Adder'' (Rg 0),
                   Rg (Suc 0) ≼ Gf ''n'' ''Adder'' (Rg 0), Cn (Intg 0) ≼ Rg (Suc 0),
                   Pos (''Adder'', ''sum'', Suc (Suc (Suc (Suc 0)))), Cn (Intg 1) ≺ FrNr,
                   Neg (And [Neg (Ty (Rg 0) NT), Neg (Ty (Rg 0) (Class ''Adder''))]),
                   Ty (Rg (Suc 0)) Integer, Ty (Rg (Suc (Suc 0))) Integer] ⊃
              (And [Neg (Rg 0 ≐ Cn Null)] ⊃
               And [And [Gf ''n'' ''Adder'' (Rg 0) ≼ Rg (Suc 0)] ⊃
                    And [(Cn (Intg 2) ⊗ Rg (Suc (Suc 0))) ≐
                         (Call (Gf ''n'' ''Adder'' (St 0)) ⊗
                          Num (Call (Gf ''n'' ''Adder'' (St 0))) num-op.Plus
                           (Cn (Intg 1)))],
                    And [Neg (Gf ''n'' ''Adder'' (Rg 0) ≼ Rg (Suc 0))] ⊃
                    And [Num (Cn (Intg 1)) num-op.Plus (Rg (Suc 0)) ≼
                         Cn (Intg 2147483647),
                         And [Ty (Num (Cn (Intg 1)) num-op.Plus (Rg (Suc 0))) Integer] ⊃
                         And [Num (Num (Cn (Intg 1)) num-op.Plus (Rg (Suc 0))) num-op.Plus
                               (Rg (Suc (Suc 0))) ≼
                              Cn (Intg 2147483647),
                              And [Ty (Num (Num (Cn (Intg 1)) num-op.Plus (Rg (Suc 0)))
                                        num-op.Plus (Rg (Suc (Suc 0))))
                                    Integer] ⊃
                              And [(Cn (Intg 2) ⊗
                                    Num (Num (Cn (Intg 1)) num-op.Plus (Rg (Suc 0)))
                                     num-op.Plus (Rg (Suc (Suc 0)))) ≐
                                   (Num (Cn (Intg 1)) num-op.Plus (Rg (Suc 0)) ⊗
                                    Num (Num (Cn (Intg 1)) num-op.Plus (Rg (Suc 0)))
                                     num-op.Plus (Cn (Intg 1))),
                                   Num (Cn (Intg 1)) num-op.Plus (Rg (Suc 0)) ≼
                                   Gf ''n'' ''Adder'' (Rg 0),
                                   Cn (Intg 0) ≼
                                   Num (Cn (Intg 1)) num-op.Plus (Rg (Suc 0))]]]])
 2. prog,[] ⊢ And [Ty (Gf ''n'' ''Adder'' (Rg 0)) Integer, St 0 ≐ Rg (Suc (Suc 0)),
                   Gf ''n'' ''Adder'' (Rg 0) ≐ Call (Gf ''n'' ''Adder'' (St 0)),
                   (Cn (Intg 2) ⊗ Rg (Suc (Suc 0))) ≐
```

$(Call\ (Gf\ ''n''\ ''Adder''\ (St\ 0)) \otimes$
$Num\ (Call\ (Gf\ ''n''\ ''Adder''\ (St\ 0)))\ num\text{-}op.Plus\ (Cn\ (Intg\ 1))),$
$Pos\ (''Adder'', ''sum'',$
$Suc\ (Suc\ (Suc\ (Suc\ (Suc$
$(Suc\ (Suc\ (Suc\ (Suc\ (Suc\ (Suc\ (Suc\ (Suc\ (Suc\ (Suc\ (Suc\ (Suc\ (Suc\ 0)))))))))))))))))),$
$Call\ (And\ [And\ [TT,\ And\ [Gf\ ''n''\ ''Adder''\ (St\ 0) \doteq Cn\ (Intg\ 65535)]],$
$Pos\ (''Start'', ''main'',$
$Suc\ (Suc\ (Suc\ (Suc\ (Suc\ (Suc\ 0))))))]),$
$Cn\ (Intg\ 1) \prec FrNr,\ Ty\ (St\ 0)\ Integer,$
$Neg\ (And\ [Neg\ (Ty\ (Rg\ 0)\ NT),\ Neg\ (Ty\ (Rg\ 0)\ (Class\ ''Adder''))]),$
$Ty\ (Rg\ (Suc\ 0))\ Integer,\ Ty\ (Rg\ (Suc\ (Suc\ 0)))\ Integer] \supset$
$(St\ 0 \doteq Cn\ (Intg\ 2147450880))$

apply (*simp only*: *deriv-def*, *rule ballI*, *clarsimp* |
drule-tac c=65534 **in** *special-bounded-mult* |
simp add: *zadd-zmult-distrib zadd-zmult-distrib2*)+
— nprfsize = 101768
done

end