PCC

Martin Wildmoser

7th February 2005

Contents

theory *EX-MainSum* = *VCGExec*:

0.0.1 Definitions

0.0.2 Name Declarations

constdefs
this::*nat*
this ≡ *0*
k::*nat*
k≡*1*
n::*nat*
n ≡ *2*
res::*nat*
res≡*3*

0.0.3 Program Code

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

constdefs
StartC :: *int* ⇒ *jvm-method cdecl*
StartC arg ≡ (*Start*,(*Object*, [],[(*main*,[],*Integer*,(*2*,*4*,[
Push (*Intg arg*),
Store n,
Push (*Intg 0*),
Store k,
Push (*Intg 0*),
Store res,
Load k,
Load n,
IfIntLeq 10 −− ″*if n* <= *k then terminate else k*++; *res*+=*k*″ ,
Load k,
Push (*Intg 1*),
IAdd,
Store k,
Load k,
Load res,
IAdd,
Store res,
Goto (−*11*),
Load res,
Return −− ″*main-ret*″],
[]))]))

0.0.4 Prove safety using hand-made annotations.

0.0.5 Annotations

constdefs
n0::int
n0 ≡ 5

constdefs
sum-inv::expr
sum-inv ≡ And [Ty (Rg k) Integer, Ty (Rg res) Integer,
Ty (Rg n) Integer,
$((Cn\ (Intg\ 2)) \otimes (Rg\ res)) \doteq (Rg\ k \otimes (Rg\ k \oplus (Cn\ (Intg\ 1)))),$
$(Cn\ (Intg\ 0)) \preceq Rg\ k,\ (Rg\ k) \preceq Rg\ n,$
$Rg\ n \doteq Cn\ (Intg\ n0)]$

0.0.6 Packing code and annotations

constdefs
prog::jbc-prog
prog ≡ (SystemClasses @ [StartC n0],
[((Start,main,6),sum-inv)])

0.0.7 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

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

0.0.8 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-MainSum.vc*, *T*), *t*)), [])]));
*}

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

0.0.9 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*: $[\![2 * a = b * (b+1); b < (5::int); 0 \leq b]\!] \Longrightarrow b + a \leq 14$short proof

lemma *vcg-prog-holds*:
prog ⊢ *vc*
apply (*simp only*: *provable-def vc-def*)
apply (*safe intro*!: *And0 AndI'*)
apply (*simp only*: *deriv-def*, *rule ballI*, *clarsimp simp del*: *evalE-evalEs.simps simp add*: *sem-simps* |
  drule special-bounded-mult, *arith*, *assumption*+, *simp del*: *evalE-evalEs.simps add*: *zadd-zmult-distrib zadd-zmult-distrib2*)+
donemore explicit proof

lemma *vcg-prog-holds2*:
prog ⊢ *vc*
apply (*simp only*: *provable-def vc-def*)
apply (*safe intro*!: *And0 AndI'*)
— We get an initial condition and one condition for edge going out of an annotated position.
— Initial Condition: initF implies isafeF prog (ipc prog)
apply (*simp only*: *deriv-def*)
apply (*rule ballI*)
apply (*clarsimp simp del*: *evalE-evalEs.simps simp add*: *sem-simps*)

— (Start,main,6) nach (Start,main,6) oder (Start,main,19)
— main challenge: arithmetical condition
apply (*drule special-bounded-mult*)
apply (*simp del*: *evalE-evalEs.simps add*: *zadd-zmult-distrib zadd-zmult-distrib2*)+
done

—

0.0.10 Verify program using interval annotations.

0.0.11 Annotations

constdefs
n2::*int*
n2 ≡ *4*

constdefs
res2::*int*
res2 ≡ *2147418112*

constdefs
sum-inv2::*expr*
sum-inv2 ≡ *And* [*Ty* (*Rg k*) *Integer*, *Ty* (*Rg res*) *Integer*,
Ty (*Rg n*) *Integer*,
(*Cn* (*Intg 0*)) ⪯ *Rg res*, (*Rg res*) ⪯ *Cn* (*Intg res2*),
(*Cn* (*Intg 0*)) ⪯ *Rg k*, (*Rg k*) ⪯ *Cn* (*Intg n2*),
Rg n ≐ *Cn* (*Intg n2*)]

0.0.12 Packing code and annotations

constdefs
prog2::*jbc-prog*
prog2 ≡ (*SystemClasses* @ [*StartC n2*],
[((*Start*,*main*,*6*),*sum-inv2*)])Note, that suminv2 is the strongest intervall annotation.

0.0.13 Generate ML code for the VCG

0.0.14 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 = *prog2*

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*

0.0.15 Verification Condition

ML {∗ *use EX-Sum.ML* ∗}
ML {∗ *wf prog*; ∗}
ML {∗ *val vc2* = *opt* (*vcg prog*); ∗}
ML {∗ *val pvc* = (*Pretty.str-of* (*Sign.pretty-term* (*sign-of* (*the-context* ())) (*term-of-expr vc2*))); ∗}

— now we trasfer the ML result back to Isabelle
ML-setup {∗
val t = *term-of-expr vc2*;
val T = *fastype-of t*;
Context.>> (*fn thy* => *thy* |>
Theory.add-consts-i [(*vc2*, *T*, *NoSyn*)] |>
(*fst o PureThy.add-defs-i false* [((*vc2-def*, *Logic.mk-equals* (*Const* (*EX-MainSum.vc2*, *T*), *t*)), [])]));
∗}

— the verification condition is now defined as constant vc2::expr

lemma *vcg-prog2-holds*:
prog2 ⊢ *vc2*
apply (*simp only*: *provable-def vc2-def*)
apply (*safe intro*!: *And0 AndI'*)
apply (*simp only*: *deriv-def*, *rule ballI*, *clarsimp simp del*: *evalE-evalEs.simps simp add*: *sem-simps*)
— problem: annotation is not inductive!

1. ⋀*a aa b ab ac ba bb x xa*.
⟦((*a*, *aa*, *b*), (*ab*, *ac*, *ba*), *bb*) ∈ *safe*□*prog2*; *0* ≤ *xa*; *xa* ≤ *2147418112*; *0* ≤ *x*;
x ≤ *4*;
the-Bool
(*evalE prog2* ((*a*, *aa*, *b*), (*ab*, *ac*, *ba*), *bb*)
(*Pos* (''*Start*'', ''*main*'', *Suc* (*Suc* (*Suc* (*Suc* (*Suc* (*Suc 0*))))))));
int (*length ba*) = *1*;
the-Bool (*evalE prog2* ((*a*, *aa*, *b*), (*ab*, *ac*, *ba*), *bb*) (*Ty* (*Rg 0*) *NT*)) ∨
the-Bool
(*evalE prog2* ((*a*, *aa*, *b*), (*ab*, *ac*, *ba*), *bb*) (*Ty* (*Rg 0*) (*Class* ''*Start*'')));
evalE prog2 ((*a*, *aa*, *b*), (*ab*, *ac*, *ba*), *bb*) (*Rg* (*Suc 0*)) = *Intg x*;
evalE prog2 ((*a*, *aa*, *b*), (*ab*, *ac*, *ba*), *bb*) (*Rg* (*Suc* (*Suc 0*))) = *Intg 4*;
evalE prog2 ((*a*, *aa*, *b*), (*ab*, *ac*, *ba*), *bb*) (*Rg* (*Suc* (*Suc* (*Suc 0*)))) = *Intg xa*;
x ≠ *4*⟧
⟹ *xa* + *x* ≤ *2147418111*

oopsVerification fails because suminv2 is not inductive!**constdefs**
sum-inv2I::*expr*
sum-inv2I ≡ *And* [*Ty* (*Rg k*) *Integer*, *Ty* (*Rg res*) *Integer*,

Ty (*Rg n*) *Integer*,
Cn (*Intg 0*) $\preceq$ *Rg res*, *Rg res* $\preceq$ ((*Cn MX*) $\ominus$ (*Rg k* $\oplus$ *Cn* (*Intg 1*))),
Cn (*Intg 0*) $\preceq$ *Rg k*, *Rg k* $\preceq$ *Cn* (*Intg n2*),
Rg n $\doteq$ *Cn* (*Intg n2*)]

0.0.16 Packing code and annotations

constdefs
prog2'::jbc-prog
prog2' ≡ (*SystemClasses* @ [*StartC n2*],
[((*Start*,*main*,*6*),*sum-inv2I*)])

0.0.17 Generate ML code for the VCG

0.0.18 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 = *prog2'*

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*

0.0.19 Verification Condition

ML {∗ *use EX-Sum.ML* ∗}
ML {∗ *wf prog*; ∗}
ML {∗ *val vc2'* = *opt* (*vcg prog*); ∗}
ML {∗ *val pvc* = (*Pretty.str-of* (*Sign.pretty-term* (*sign-of* (*the-context* ())) (*term-of-expr vc2'*))); ∗}

— now we trasfer the ML result back to Isabelle
ML-setup {∗
val t = *term-of-expr vc2'*;
val T = *fastype-of t*;
Context.>> (*fn thy* => *thy* |>
Theory.add-consts-i [(*vc2'*, *T*, *NoSyn*)] |>

(fst o PureThy.add-defs-i false [((vc2′-def, Logic.mk-equals (Const (EX-MainSum.vc2′, T), t)), [])]));
*}

lemma *vcg-prog2′-holds*:
prog2′ ⊢ *vc2′*
apply (*simp only*: *provable-def vc2′-def*)
apply (*safe intro*!: *And0 AndI′*)
apply (*simp only*: *deriv-def*, *rule ballI*, *clarsimp simp del*: *evalE-evalEs.simps simp add*: *sem-simps*)
— problem: annotation is not inductive!

```
1. ⋀a aa b ab ac ba bb x xa.
      ⟦((a, aa, b), (ab, ac, ba), bb) ∈ safe□prog2′; 0 ≤ xa; xa ≤ 2147483646 − x; 0 ≤ x;
       x ≤ 4;
       the-Bool
        (evalE prog2′ ((a, aa, b), (ab, ac, ba), bb)
          (Pos (''Start'', ''main'', Suc (Suc (Suc (Suc (Suc (Suc 0))))))));
       int (length ba) = 1;
       the-Bool (evalE prog2′ ((a, aa, b), (ab, ac, ba), bb) (Ty (Rg 0) NT)) ∨
       the-Bool
        (evalE prog2′ ((a, aa, b), (ab, ac, ba), bb) (Ty (Rg 0) (Class ''Start'')));
       evalE prog2′ ((a, aa, b), (ab, ac, ba), bb) (Rg (Suc 0)) = Intg x;
       evalE prog2′ ((a, aa, b), (ab, ac, ba), bb) (Rg (Suc (Suc 0))) = Intg 4;
       evalE prog2′ ((a, aa, b), (ab, ac, ba), bb) (Rg (Suc (Suc (Suc 0)))) = Intg xa;
       x ≠ 4⟧
      ⟹ 2 ∗ x + xa ≤ 2147483644
```

oops

— Fails again. No linear invariant seems to be sufficient.

—

0.0.20 Verify robust program using intervals.

constdefs
StartCR :: *int* ⇒ *jvm-method cdecl*
StartCR arg ≡ (*Start*,(*Object*, [],[
(*main*,[],*Integer*,(*2*,*4*,[
Push (*Intg arg*),
Store n,
Push (*Intg 0*),
Store res,
Push (*Intg 0*),
Store k,
Load k −− ″*LOOP*: *suminv3*″,
Load n,
IfIntLeq 13 −− ″*if n* <= *k then terminate else k*++; *res*+=*k*″ ,
Load res,
Push (*Intg 2147418113*) −− ″*2147418112 is the maximum value res can have at this position without causing an overflow*″,
IfIntLeq 10,
Load k −− ″*k*: [*0*,*arg*], *res*:[*0*,*2147418112*]″,
Push (*Intg 1*),
IAdd,
Store k,
Load k,
Load res,
IAdd,
Store res,
Goto (−*14*) −− ″*goto LOOP*″,
Load res,
Return −− ″*main-ret*″],
[]))]))

constdefs
n3::*int*
n3 ≡ *65535*

constdefs
sum-inv3::*expr*
sum-inv3 ≡ *And* [*Ty* (*Rg k*) *Integer*, *Ty* (*Rg res*) *Integer*,
Ty (*Rg n*) *Integer*,
(*Cn* (*Intg 0*)) ⪯ *Rg res*, (*Rg res*) ⪯ *Cn* (*Intg MAX*),
(*Cn* (*Intg 0*)) ⪯ *Rg k*, (*Rg k*) ⪯ (*Cn* (*Intg 65535*)),
Rg n ⪯ *Cn* (*Intg 65535*)]

0.0.21 Packing code and annotations

constdefs
prog3::*jbc-prog*
prog3 ≡ (*SystemClasses* @ [*StartCR n3*],
[((*Start*,*main*,*6*),*sum-inv3*)])

0.0.22 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 = *prog3*

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*

0.0.23 Verification Condition

ML {* *use EX-Sum.ML* *}
ML {* *wf prog*; *}
ML {* *val vc3* = *opt* (*vcg prog*); *}
ML {* *val pvc* = (*Pretty.str-of* (*Sign.pretty-term* (*sign-of* (*the-context* ())) (*term-of-expr vc3*))); *}

— now we trasfer the ML result back to Isabelle
ML-setup {*
val t = *term-of-expr vc3*;
val T = *fastype-of t*;
Context.>> (*fn thy* => *thy* |>
Theory.add-consts-i [(*vc3*, *T*, *NoSyn*)] |>
(*fst o PureThy.add-defs-i false* [((*vc3-def*, *Logic.mk-equals* (*Const* (*EX-MainSum.vc3*, *T*), *t*)), [])]));
*}

lemma *vcg-prog3-holds*:
prog3 ⊢ *vc3*
apply (*simp only*: *provable-def vc3-def*)
apply (*safe intro*!: *And0 AndI'*)

apply (*simp only*: *deriv-def*, *rule ballI*, *clarsimp simp del*: *evalE-evalEs.simps simp add*: *sem-simps*)
done

end