Property Directed Self Composition

Ron Shemer, Arie Gurfinkel, Sharon Shoham, Yakir Vizel

Akshatha Shenoy
TCS Research
Pune

11th July 2021

Motivating Example

```
int doubleSquare(bool h, int x){
    int z, y=0;
    if(h) { z = 2*x; }
    else { z = x; }

    while (z>0) {
        z--;
        y = y+x;
    }

    if(!h) { y = 2*y; }
    return y;
}
```

Figure: Program that computes $2x^2$

Double Square Program

- Need to talk about any 2 runs of the program.
- Instead 2 copies of the same program, 1^{st} copy on variables $x1$, $h1$, $z1$... and 2^{nd} copy on $x2$, $h2$, $z2$....

```
assume(x1 == x2)

y1 = doubleSquare(x1, h1)
y2 = doubleSquare(x2, h2)

assert(y1 == y2)
```

Figure: Sequential composition of 2 doubleSquare program copies

```
main(bool h1, int x1, bool h2, int x2){
    int z1, z2, y1=0, y2=0;
    assume(x1 == x2);
    if(h1)  {   z1 = 2*x1;  }
    else{   z1 = x1;  }
    while (z1>0) {
        z1--;
        y1 = y1+x1;
    }
    if(!h1)  {   y1 = 2*y1; }

    if(h2)  {   z2 = 2*x2;  }
    else{   z2 = x2;  }
    while (z2>0) {
        z2--;
        y2 = y2+x2;
    }
    if(!h2)  {   y2 = 2*y2; }
    assert(y1 == y2);
}
```

Figure: Sequential composition of 2 doubleSquare program copies

k-safety property

Properties that refer to k executions
Example: Determinism is a 2-safety property

```
main(bool h1, int x1, bool h2, int x2){
    int z1, z2, y1=0, y2=0;
    assume(x1 == x2);
    if(h1)  {   z1 = 2*x1;   }
    else{   z1 = x1;   }
    while (z1>0) {
        z1--;
        y1 = y1+x1;
    }
    if(!h1)  {   y1 = 2*y1; }

    if(h2)  {   z2 = 2*x2;   }
    else{   z2 = x2;   }
    while (z2>0) {
        z2--;
        y2 = y2+x2;
    }
    if(!h2)  {   y2 = 2*y2; }
    assert(y1 == y2);
}
```

Figure: Any composition

```
main(int x1, int x2){
    int z1, z2, y1=0, y2=0;
    assume(x1 == x2);
    z1 = 2*x1;
    while (z1>0) {
        z1--;
        y1 = y1+x1;
    }

    z2 = x2;
    while (z2>0) {
        z2--;
        y2 = y2+x2;
    }
    y2 = 2*y2;
    assert(y1 == y2);
}
```

Figure: Simplified version of sequential composition initialising h1 = *True* and h2 = *False*

Non trivial composition

```
main(int x1, int x2){
    int z1, z2, y1=0, y2=0;

    assume(x1 == x2);

    z1 = 2*x1;

    z2 = x2;

    while (z1>0 && z2>0) {
        z1--; y1 = y1+x1;
        z1--; y1 = y1+x1;
        z2 --;
        y2 = y2+x2;

    }
    y2 = 2*y2;

    assert(y1 == y2);
}
```

Figure: Another composition for the version initialising h1 = *True* and h2 = *False*

Non trivial composition with a "simple" proof

```
main(int x1, int x2){

    int z1, z2, y1=0, y2=0;
    assume(x1 == x2);

    z1 = 2*x1;
    z2 = x2;                                y1 = 2*y2
    while (z1>0 && z2>0) {
        z1--; y1 = y1+x1;
        z1--; y1 = y1+x1;
        z2 --;
        y2 = y2+x2;

    }
    y2 = 2*y2;

    assert(y1 == y2);
}
```

Figure: Composition has simpler inductive invariants

Main ideas of the paper

- Every interleaving gives rise to a composition.

Main ideas of the paper

- Every interleaving gives rise to a composition.
- Search for a composition that admits a simple proof.

Semantic Self Composition Function

- $f : S^k \rightarrow \mathbb{P}(1..k)$
- $f(s_1, ..., s_k) = M \iff (s_1, ..., s_k) \models C_M$

(NOTE: Syntactic Compositions only depend on control locations of the copies.)

Composed Program

Given: C_M for every set $M \subseteq 1..k$
$T^f = (S^{||k}, R^f, F^{||k})$

$$R^f = \bigvee_{\phi \neq M \subseteq 1..k} (C_M \wedge \varphi_M)$$

where

$$\varphi_M = \bigwedge_{j \in M} R(\vartheta^j, \vartheta^{j'}) \wedge \bigwedge_{j \notin M} \vartheta^j = \vartheta^{j'}$$

Examples : Semantic Self Composition Functions

- $f(s_1, .., s_k) = \{1..k\}$
  LOCKSTEP COMPOSITION

Examples : Semantic Self Composition Functions

- $f(s_1, .., s_k) = \{1..k\}$
  LOCKSTEP COMPOSITION

- $f(s_1, .., s_k) = \{i\}$ where i is minimal index of a non-terminal state in $\{s_1, ..., s_k\}$ and $\{k\}$ otherwise
  SEQUENTIAL COMPOSITION

PDSC Algorithm - Key property

T has *Inv* in $\mathcal{L}_{\mathcal{P}} \iff A_{\mathcal{P}}(T)$ is safe.

Algorithm

1. $f \leftarrow$ lockstep, $E \leftarrow \emptyset$, $Unreach \leftarrow false$
2. $res \leftarrow \mathrm{Safe}(Abstract_{\mathcal{P}}(T^f))$

Counterexample

f_{old}

$$s_1 \xrightarrow{M_1} s_2 \xrightarrow{M_2} s_3 \cdots\cdots\cdots\cdots\cdots\cdots s_{m-1} \xrightarrow{M_{m-1}} s_m \xrightarrow{M_m} s_{m+1}$$

f_{old}

$$s_1 \xrightarrow{M_1} s_2 \xrightarrow{M_2} s_3 \cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots s_{m-1} \xrightarrow{M_{m-1}} s_m \xrightarrow{M_m} {\color{red} s_{m+1}}$$

f_{new} in progress

$$s_1 \xrightarrow{M_1} s_2 \xrightarrow{M_2} s_3 \cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots s_{m-1} \xrightarrow{M_{m-1}} s_m \xrightarrow{\cancel{M_m}} {\color{red} s_{m+1}}$$

Algorithm

1. $f \leftarrow$ lockstep, $E \leftarrow \emptyset$, $Unreach \leftarrow false$
2. $res \leftarrow$ Safe$(Abstract_{\mathcal{P}}(T^f))$
3. $Update\_E(s)$

f_{old}

$$s_1 \xrightarrow{M_1} s_2 \xrightarrow{M_2} s_3 \cdots\cdots\cdots\cdots\cdots s_{m-1} \xrightarrow{M_{m-1}} s_m \xrightarrow{M_m} s_{m+1}$$

f_{new} in progress

$$s_1 \xrightarrow{M_1} s_2 \xrightarrow{M_2} s_3 \cdots\cdots\cdots\cdots\cdots s_{m-1} \xrightarrow{M_{m-1}} s_m \xrightarrow{\cancel{M_m}} s_{m+1} \qquad E = \{\langle s_m, M_m \rangle\}$$

Algorithm

1. $f \leftarrow$ lockstep, $E \leftarrow \emptyset$, $Unreach \leftarrow false$
2. while $true$
3. $res \leftarrow \mathrm{Safe}(Abstract_{\mathcal{P}}(T^f))$
4. $Update\_E(s)$
5.   if find_next(f)
6.   Update_f()

f_{old}

$$s_1 \xrightarrow{M_1} s_2 \xrightarrow{M_2} s_3 \cdots\cdots\cdots\cdots s_{m-1} \xrightarrow{M_{m-1}} s_m \xrightarrow{M_m} s_{m+1}$$

f_{new} in progress

$$\boxed{\langle s_m, M_m^1 \rangle \notin E}$$

$$s_1 \xrightarrow{M_1} s_2 \xrightarrow{M_2} s_3 \cdots\cdots\cdots\cdots s_{m-1} \xrightarrow{M_{m-1}} s_m \xrightarrow{M_m^1}$$

$$E = \{\langle s_m, M_m \rangle\}$$

f_{old}

$$s_1 \xrightarrow{M_1} s_2 \xrightarrow{M_2} s_3 \cdots\cdots\cdots\cdots s_{m-1} \xrightarrow{M_{m-1}} s_m \xrightarrow{M_m} s_{m+1}$$

f_{new} in progress

$$s_1 \xrightarrow{M_1} s_2 \xrightarrow{M_2} s_3 \cdots\cdots\cdots\cdots s_{m-1} \xrightarrow{M_{m-1}} s_m$$

$E = \{<s_m, M_m>\}$

Unreach = $\{s_m\}$

f_{old}

$$s_1 \xrightarrow{M_1} s_2 \xrightarrow{M_2} s_3 \cdots\cdots\cdots\cdots\cdots s_{m-1} \xrightarrow{M_{m-1}} s_m \xrightarrow{M_m} s_{m+1}$$

f_{new} in progress

$$s_1 \xrightarrow{M_1} s_2 \xrightarrow{M_2} s_3 \cdots\cdots s_i \xrightarrow{M_i^1}$$

$E = \{<s_m, M_m>, <s_{m-1}, M_{m-1}>, .., <s_{i+1}, M_{i+1}>\}$
$Unreach = \{s_m, s_{m-1}, .., s_{i+1}\}$

cex has been fixed

- $f_{new}(s_i) = M_i^1$

f_{old}

$$s_1 \xrightarrow{M_1} s_2 \xrightarrow{M_2} s_3 \cdots\cdots\cdots\cdots\cdots s_{m-1} \xrightarrow{M_{m-1}} s_m \xrightarrow{M_m} s_{m+1}$$

f_{new}

s_1

$$E = \{\langle s_m, M_m\rangle, \langle s_{m-1}, M_{m-1}\rangle \ldots \langle s_1, M_1\rangle, \langle s_1, {M_1}^1\rangle, \langle s_1, {M_1}^2\rangle ..\}$$

$$\mathrm{Unreach} = \{s_m, s_{m-1}, \ldots, s_2, s_1\}$$

No solution in $\mathcal{L}_{\mathcal{P}}$

Algorithm

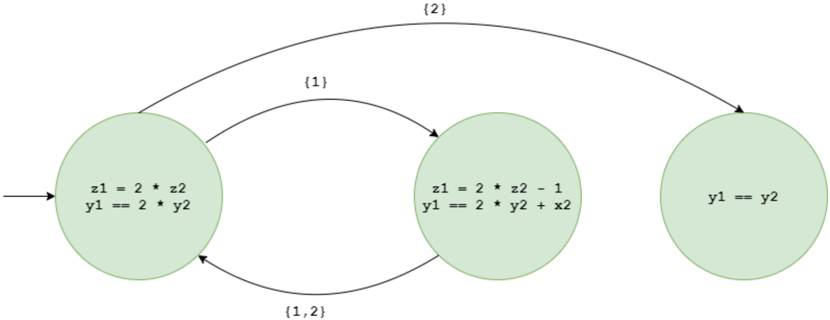
1. $f \leftarrow$ lockstep, $E \leftarrow \emptyset$, $Unreach \leftarrow false$
2. $res \leftarrow$ Safe$(Abstract_{\mathcal{P}}(T^f))$
3. $Update\_E(s)$
4. while $true$
5. if find_next(f)
6. Update_f()
7. repeat 2
8. return "no solution in $\mathcal{L}_{\mathcal{P}}$"

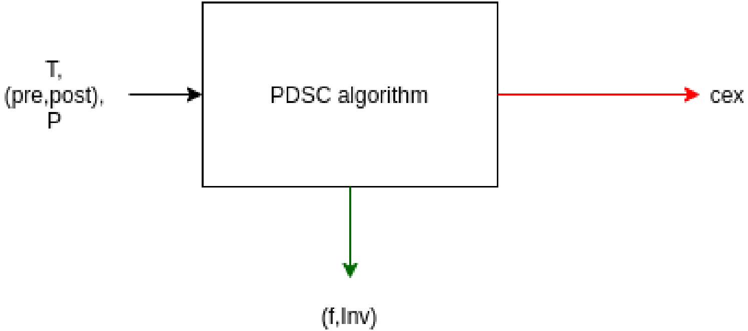
1. Given : T, k - safety property, a finite $\mathcal{P}$
   Output : (f, Inv) in $\mathcal{L}_{\mathcal{P}}$

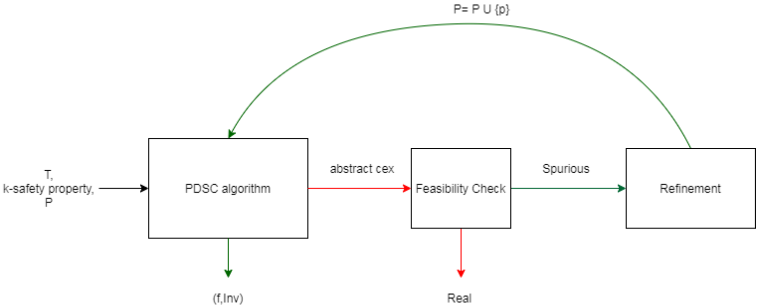
Non trivial composition with a “simple” proof

```
main(int x1, int x2){

    int z1, z2, y1=0, y2=0;
    assume(x1 == x2);

    z1 = 2*x1;
    z2 = x2;                                   y1 = 2*y2
    while (z1>0 && z2>0) {
        z1--; y1 = y1+x1;
        z1--; y1 = y1+x1;
        z2 --;
        y2 = y2+x2;

    }
    y2 = 2*y2;

    assert(y1 == y2);
}
```

Figure: Composition has simpler inductive invariants


{2}
{1}
z1 = 2 * z2
y1 == 2 * y2
z1 = 2 * z2 - 1
y1 == 2 * y2 + x2
y1 == y2
{1,2}


T,
(pre,post),
P
PDSC algorithm
cex
(f,Inv)


P= P U {p}
T,
k-safety property,
P
PDSC algorithm
abstract cex
Feasibility Check
Spurious
Refinement
(f,Inv)
Real

S. No.	Benchmark	Source	Safe/Unsafe	SyGuS(CVC 4.1.8) (#pred, time)	Abduction(Z3) (#preds, time)
1.	sum_to_n	crafted	safe	timeout	8, 1m30s
2.	sum_to_n_err	crafted	unsafe	0, 1.1s	0, 0.8s
3.	inc-dec	crafted	safe	5, 39 s	8, 35.8 s
4.	squareSum	cav19	safe	0, 2.2 s	0, 1.1 s
5.	sum-pc	cav19	safe	5, 4m5.3s	1, 11.9 s
6.	fig4_1	icse16	unsafe	timeout	2, 7.63 s
7.	fig4_2	icse16	unsafe	timeout	2, 7.65 s
8.	fig4_ref_ref	icse16	safe	0, 0.8 s	0, 0.6 s
9.	subsume_1	icse16	unsafe	timeout	3, 13 s
10.	subsume_2	icse16	unsafe	timeout	2, 8.8 s
11.	subsume_ref_ref	icse16	safe	timeout	1, 3.9 s
12.	puzzle_1	derived from icse16	unsafe	timeout	4, 26.8 s
13.	puzzle_2	derived from icse16	unsafe	timeout	8, 2m25s
14.	puzzle_3	derived from icse16	safe	timeout	2, 11.9s
15.	halfSquare	cav19	safe	timeout	4, 1m10s
16.	doubleSquare_1	derived from cav19	safe	timeout	6, 1m55s
17.	doubleSquare_2	derived from cav19	safe	timeout	3, 43.8s
18.	doubleSquare_3	derived from cav19	safe	timeout	5, 1m29s

Thank you!

shenoy.akshatha@tcs.com