Automatic discovery of fair paths in infinite-state transition systems

Alessandro Cimatti, Alberto Griggio, Enrico Magnago

Fondazione Bruno Kessler

Context

Problem
Does a transition system admit at least one fair path?
(Counterexample to liveness property).

- Undecidable in infinite-state systems.
- Techniques to prove the language empty (property holds) and techniques to prove the existence of a fair path (witness).
- Witnesses are often limited to lasso-shaped paths.
- Not sufficient in infinite-state, need to look for witnesses with different shapes.

How can we represent them?

Fair paths representation

Assume we want to prove the existence of an infinite run for the code below in which $c = 0$ infinitely often.

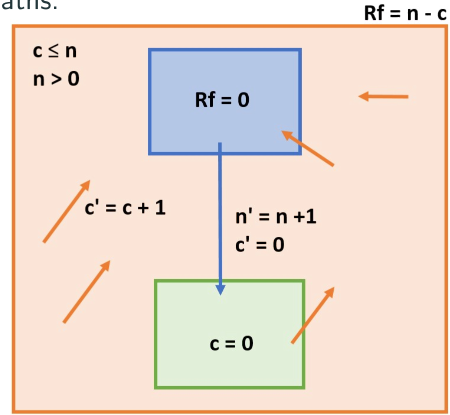
```
int c, n;
while ⊤ do
    while c < n do
        c ← c + 1
    end while
    c ← nondet()
    n ← n + 1
end while
```

```
VAR c: integer; n: integer;

TRANS
  (c < n ∧ next(c)=c + 1 ∧ next(n)=n) ∨
  (c ≥ n ∧ next(n)=n + 1);

FAIRNESS c = 0;
```

Funnel-loop: reachable, non-empty underapproximation with only fair paths.



Fair paths representation

Assume we want to prove the existence of an infinite run for the code below in which $c = 0$ infinitely often.

```
int c, n;
while ⊤ do
    while c < 
        c ← c + 1
    end while
    c ← nondet()
    n ← n + 1
end while
```

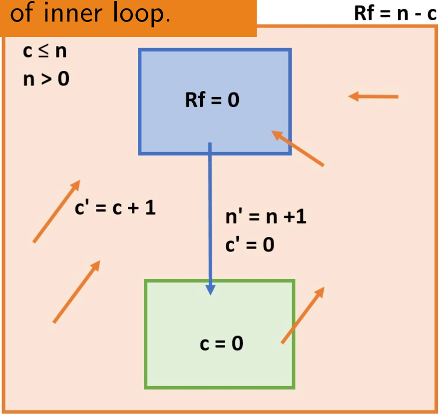
```
VAR c: integer; n: integer;

TRANS
  (c < n ∧ next(c)=c + 1 ∧ next(n)=n) ∨
  (c ≥ n ∧ next(n)=n + 1);

FAIRNESS c = 0;
```

Funnel-loop: reachable, non-empty
th only fair

Source region with ranking function Rf, ensures termination of inner loop.



Fair paths representation

Assume we want to prove the existence of an infinite run for the code below in which $c = 0$ infinitely often.

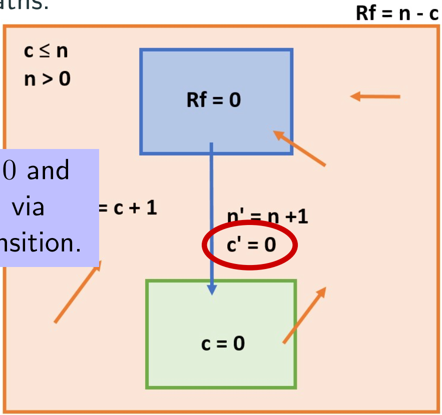
```
int c, n;
while ⊤ do
    while c < n do
        c ← c + 1
    end while
    c ← nondet()
    n ← n
end whil
```

```
VAR c : integer; n: integer;

TRANS
  (c < n ∧ next(c)=c + 1 ∧ next(n)=n) ∨
  (c ≥ n ∧ next(n)=n + 1);

FAIRNESS c = 0;
```

Funnel-loop: reachable, non-empty underapproximation with only fair paths.



Eventually **Rf** becomes 0 and reach destination region via underapproximating transition.

Fair paths representation

Assume we want to prove the existence of an infinite run for the code below in which $c = 0$ infinitely often.

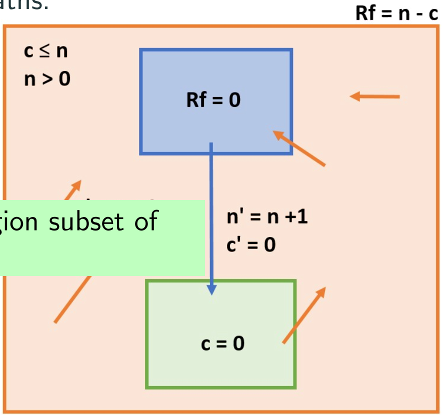
```
int c, n;
while ⊤ do
    while c < n do
        c ← c + 1
    end while
    c ← nondet()
    n ← n + 1
end while
```

```
VAR c : integer; n : intege

TRANS
  (c < n ∧ next(c)=c + 1 ∧ next(n)=n) ∨
  (c ≥ n ∧ next(n)=n + 1);

FAIRNESS c = 0;
```

Funnel-loop: reachable, non-empty underapproximation with only fair paths.



Destination region subset of the fair states.

Fair paths representation

Assume we want to prove the existence of an infinite run for the code below in which $c = 0$ infinitely often.

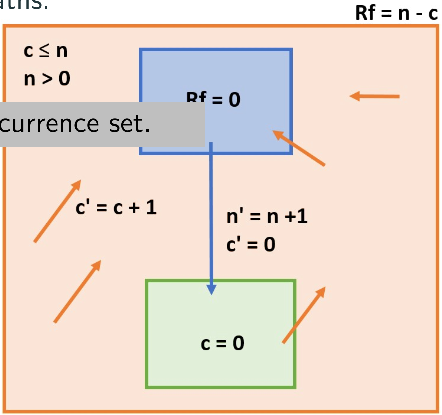
int c, n;
while $\top$ **do**
  while $c < n$ **do**
    $c \leftarrow c + 1$
  end while
  $c \leftarrow$
  $n \leftarrow n + 1$
end while

```
VAR c: integer; n: integer;

TRANS
  (c < n ∧ next(c)=c + 1 ∧ next(n)=n) ∨
  (c ≥ n ∧ next(n)=n + 1);

FAIRNESS c = 0;
```

Funnel-loop: reachable, non-empty underapproximation with only fair paths.

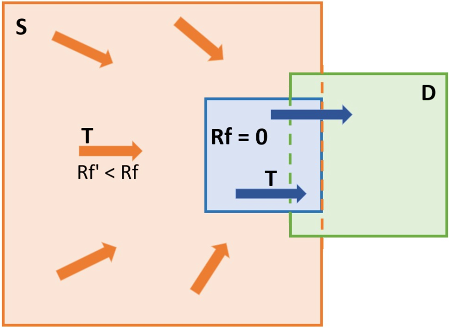


Source region is a closed recurrence set.

Funnels
Segment infinite paths into sequence of finite paths.

$fnl \doteq \langle S, T, D, \text{RF} \rangle$

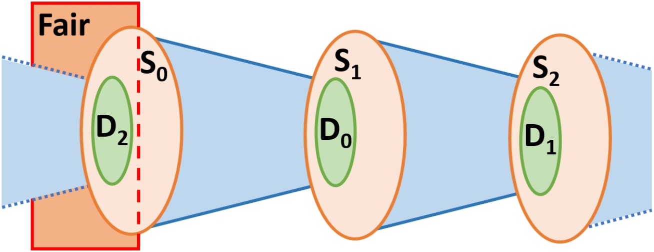
- T left-total w.r.t. S;
- $\text{RF} > 0 \land S \land T \to S' \land \text{RF}' < \text{RF}$;
- $\text{RF} = 0 \land S \land T \to D'$.



Funnel-loop

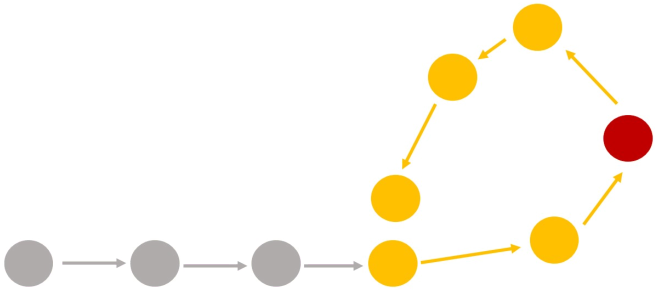
Funnel-loop: concatenation of n funnels $fnl_i \doteq \langle S_i, T_i, D_i, \mathrm{RF}_i \rangle$

- Regions reachable by transition system M.
- Funnels are correctly concatenated: $D_i \rightarrow S_{i+1\%n}$.
- Last destination region underapproximates the fair states: $D_{n-1} \rightarrow Fair$.



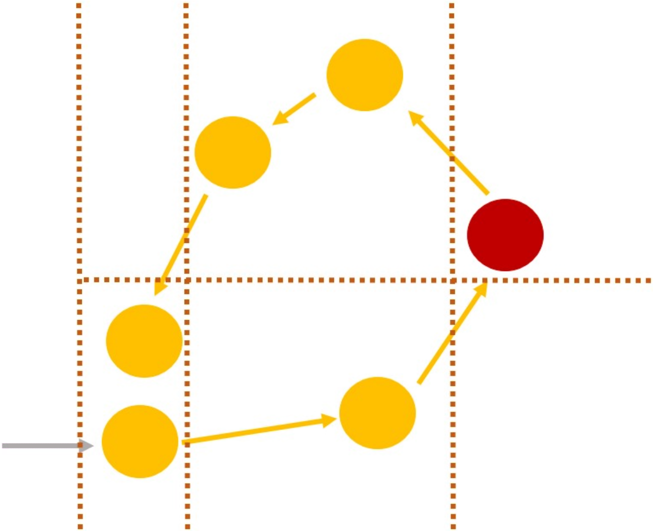
Funnel-loop search procedure overview

Enumerate paths that could correspond to an iteration over some funnel-loop.



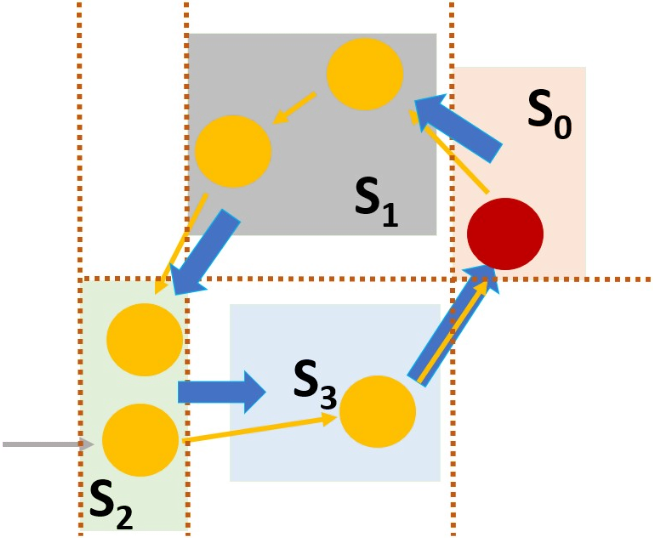
Funnel-loop search procedure overview

Consider sequence of states that correspond to a fair loop in some abstract space.



Funnel-loop search procedure overview

Define funnel-loop as strengthening of regions and transitions.



Funnel-loop search procedure

Algorithm SEARCH-FUNNEL-LOOP(M)

for $\langle v_0, abst\_s, abst\_t\rangle \in$ GENERATE-ABSTRACT-LOOPS(M) **do**
    for $fnl\_template \in$ GENERATE-TEMPLATES($v_0, abst\_s, abst\_t$) **do**
        $ef\_constrs \leftarrow fnl\_template.ef\_constraints()$
        $\langle found, model\rangle \leftarrow$ SEACH-PARAMETER-ASSIGNMENT($ef\_constrs$)
        if $found == \top$ **then**
            return $\langle model, fnl\_template\rangle$
        end if
    end for
end for
return $unknown$

Funnel-loop search procedure

Algorithm $\textsc{search-funnel-loop}(M)$

Given a fair transition system M

for $\langle v_0, abst\_s, abst\_t \rangle \in \textsc{generate-abstrac}$
    for $fnl\_template \in \textsc{generate-templates}(v_0, abst\_s, abst\_t)$ **do**
        $ef\_constrs \leftarrow fnl\_template.ef\_constraints()$
        $\langle found, model \rangle \leftarrow \textsc{seach-parameter-assignment}(ef\_constrs)$
        if $found == \top$ **then**
            return $\langle model, fnl\_template \rangle$
        end if
    end for
end for
return $unknown$

Funnel-loop search procedure

Algorithm $\textsc{search-funnel-loop}(M)$

for $\langle v_0, abst\_s, abst\_t \rangle \in \textsc{generate-abstract-loops}(M)$ **do**
  for $fnl\_template \in \textsc{generate-templates}(v_0, abst\_s, abst\_t)$ **do**
    $ef\_constrs \leftarrow fnl\_template.ef\_constraints()$
    $\langle found, model \rangle \leftarrow \textsc{seach-parameter-assignment}(ef\_constrs)$
    if $found == \top$ **then**
      return $\langle model, fnl\_template \rangle$
    end if
  end for
end for
return $unknown$

Enumerate underapproximations of M that represent fair loops over some predicates.

Funnel-loop search procedure

Algorithm SEARCH-FUNNEL-LOOP(M)

for $\langle v_0, abst\_s, abst\_t\rangle \in$ GENERATE-ABSTRACT-LOOPS(M) **do**
  for $fnl\_template \in$ GENERATE-TEMPLATES$(v_0, abst_s, abst_t)$ **do**
    $ef\_constrs \leftarrow fnl\_template.ef\_constraints()$
    …METER-ASSIGNMENT$(ef_constrs)$
  end for
end for
return $unknown$

$abst\_s$ is a sequence of regions, $abst\_t$ is a sequence of transitions, v_0 is a reachable state in the first regions.

Funnel-loop search procedure

Algorithm SEARCH-FUNNEL-LOOP(M)

```
for ⟨v_0, abst_s, abst_t⟩ ∈ GENERATE-ABSTRACT-LOOPS(M) do
    for fnl_template ∈ GENERATE-TEMPLATES(v_0, abst_s, abst_t) do
        ef_constrs ← fnl_template.ef_constraints()
        ⟨found, model⟩ ← SEACH-PARAMETER-ASSIGNMENT(ef_constrs)
        if found == ⊤ then
            return ⟨model, fnl_templ
        end if
    end for
end for
return unknown
```

Generate funnel-loop templates by strengthening $abst\_s$ and $abst\_t$ with parametric predicates.

Funnel-loop search procedure

Algorithm SEARCH-FUNNEL-LOOP(M)

for $\langle v_0, abst\_s, abst\_t\rangle \in$ GENERATE-ABSTRACT-LOOPS(M) **do**
  for $fnl\_template \in$ GENERATE-TEMPLATES($v_0, abst\_s, abst\_t$) **do**
    $ef\_constrs \leftarrow fnl\_template.ef\_constraints()$
    $\langle found, model\rangle \leftarrow$ SEACH-PARAMETER-ASSIGNMENT($ef\_constrs$)
    if $found == \top$ **then**
      return $\langle model, fnl\_$
    end if
  end for
end for
return $unknown$

Get $\exists\forall$ quantified formula: exists assignment to parameters such that template corresponds to funnel-loop.

Funnel-loop search procedure

Algorithm SEARCH-FUNNEL-LOOP(M)

```
for ⟨v_0, abst_s, abst_t⟩ ∈ GENERATE-ABSTRACT-LOOPS(M) do
    for fnl_template ∈ GENERATE-TEMPLATES(v_0, abst_s, abst_t) do
        ef_constrs ← fnl_template.ef_constraints()
        ⟨found, model⟩ ← SEACH-PARAMETER-ASSIGNMENT(ef_constrs)
        if found == ⊤ then
            return ⟨model, fnl_template⟩
        end if
    end for
end for
return unknown
```

Solve ∃∀ problem: find assignment for parameters

Funnel-loop search procedure

Algorithm SEARCH-FUNNEL-LOOP(M)

for $\langle v_0, abst\_s, abst\_t\rangle \in$ GENERATE-ABSTRACT-LOOPS(M) **do**
    for $fnl\_template \in$ GENERATE-TEMPLATES$(v_0, abst_s, abst_t)$ **do**
        $ef\_constrs \leftarrow fnl\_template.ef\_constraints()$
        $\langle found, model\rangle \leftarrow$ SEACH-PARAMETER-ASSIGNMENT$(ef_constrs)$
        if $found == \top$ **then**
            return $\langle model, fnl\_template\rangle$
        end if
    end for
end for
return unk

If successful return funnel-loop, otherwise analyse next template or candidate loop

Funnel-loop search: enumerate candidate loops

Algorithm GENERATE-ABSTRACT-LOOPS(M)

$\langle V, I, T, bad \rangle \leftarrow$ ENCODE-BMC-FAIR-ABSTRACT-LOOP(M)
for $k \in [0, 1, 2, \ldots]$ **do**
  $query \leftarrow I(V_0) \wedge \bigwedge_{i=0}^{k-1} T(V_i, V_{i+1}) \wedge bad(V_k)$
  $\langle sat, model \rangle \leftarrow$ SMT-SOLVE($query$)
  $refs \leftarrow []$
  while sat **do**
    $\langle abst\_s, abst\_t \rangle \leftarrow$ GET-IMPLICANT($model, query$)
    $\langle is\_ranked, rf \rangle \leftarrow$ RANK-LOOP($abst\_s, abst\_t$)
    if is_ranked **then**
      $\langle V, I, T, bad \rangle \leftarrow$ REMOVE-RANKED-LOOPS(V, I, T, bad, rf)
    else
      $v_0 \leftarrow$ GET-LOOPBACK-STATE($model$)
      yield $\langle v_0, abst\_s, abst\_t \rangle$
      $refs.append(\neg(\bigwedge_{s \in abst\_s} s \wedge \bigwedge_{t \in abst\_t} t))$
    end if
    $query \leftarrow I(V_0) \wedge \bigwedge_{i=0}^{k-1} T(V_i, V_{i+1}) \wedge bad(V_k) \wedge \bigwedge_{ref \in refs} ref$
    $\langle sat, model \rangle \leftarrow$ SMT-SOLVE($query$)
  end while
end for

Funnel-loop search: enumerate candidate loops

Algorithm GENERATE-ABSTRACT-LOOPS(M)

$\langle V, I, T, bad\rangle \leftarrow$ ENCODE-BMC-FAIR-ABSTRACT-LOOP(M)
for $k \in [0, 1, 2, \ldots]$ **do**
  $query \leftarrow I(V_0) \wedge \bigwedge_{i=0}^{k-1} T(V_i, V_{i+1}) \wedge bad(V_k)$
  $\langle sat, model\rangle \leftarrow$ SMT
  $refs \leftarrow []$
  while sat **do**
    $\langle abst\_s, abst\_t\rangle \leftarrow$ GET-IMPLICANT($model, query$)
    $\langle is\_ranked, rf\rangle \leftarrow$ RANK-LOOP($abst\_s, abst\_t$)
    if is_ranked **then**
      $\langle V, I, T, bad\rangle \leftarrow$ REMOVE-RANKED-LOOPS(V, I, T, bad, rf)
    else
      $v_0 \leftarrow$ GET-LOOPBACK-STATE($model$)
      yield $\langle v_0, abst\_s, abst\_t\rangle$
      $refs.append(\neg(\bigwedge_{s \in abst\_s} s \wedge \bigwedge_{t \in abst\_t} t))$
    end if
    $query \leftarrow I(V_0) \wedge \bigwedge_{i=0}^{k-1} T(V_i, V_{i+1}) \wedge bad(V_k) \wedge \bigwedge_{ref \in refs} ref$
    $\langle sat, model\rangle \leftarrow$ SMT-SOLVE($query$)
  end while
end for

L2S encoding for the search of fair loops. The loop-back is identified in the abstract space identified by a set of predicates.

Funnel-loop search: enumerate candidate loops

Algorithm GENERATE-ABSTRACT-LOOPS(M)

$\langle V, I, T, bad\rangle \leftarrow$ ENCODE-BMC-FAIR-ABSTRACT-LOOP(M)
for $k \in [0, 1, 2, \ldots]$ **do**
  $query \leftarrow I(V_0) \wedge \bigwedge_{i=0}^{k-1} T(V_i, V_{i+1}) \wedge bad(V_k)$ BMC unrolling.
  $\langle sat, model\rangle \leftarrow$ SMT-SOLVE$(query)$
  $refs \leftarrow []$
  while sat **do**
    $\langle abst\_s, abst\_t\rangle \leftarrow$ GET-IMPLICANT$(model, query)$
    $\langle is\_ranked, rf\rangle \leftarrow$ RANK-LOOP$(abst_s, abst_t)$
    if is_ranked **then**
      $\langle V, I, T, bad\rangle \leftarrow$ REMOVE-RANKED-LOOPS(V, I, T, bad, rf)
    else
      $v_0 \leftarrow$ GET-LOOPBACK-STATE$(model)$
      yield $\langle v_0, abst\_s, abst\_t\rangle$
      $refs.append(\neg(\bigwedge_{s \in abst\_s} s \wedge \bigwedge_{t \in abst\_t} t))$
    end if
    $query \leftarrow I(V_0) \wedge \bigwedge_{i=0}^{k-1} T(V_i, V_{i+1}) \wedge bad(V_k) \wedge \bigwedge_{ref \in refs} ref$
    $\langle sat, model\rangle \leftarrow$ SMT-SOLVE$(query)$
  end while
end for

Funnel-loop search: enumerate candidate loops

Algorithm $\textsc{generate-abstract-loops}(M)$

$\langle V, I, T, bad\rangle \leftarrow \textsc{encode-BMC-fair-abstract-loop}(M)$
for $k \in [0, 1, 2, \ldots]$ **do**
  $query \leftarrow I(V_0) \wedge \bigwedge_{i=0}^{k-1} T(V_i, V_{i+1}) \wedge bad(V_k)$
  $\langle sat, model\rangle \leftarrow \textsc{SMT-solve}(query)$
  $refs \leftarrow []$
  while sat **do**
    $\langle abst\_s, abst\_t\rangle \leftarrow \textsc{get-implicant}(model, query)$
    $\langle is\_ranked, rf\rangle \leftarrow \textsc{rank-loop}(abst\_s, abst\_t)$
    if is_ranked **then**
      $\langle V, I, T, bad\rangle \leftarrow \textsc{remove-ranked-loops}(V, I, T, bad, rf)$
    else
      $v_0 \leftarrow \textsc{get-loopback-state}(model)$
      yield $\langle v_0, abst\_s, abst\_t\rangle$
      $refs.append(\neg(\bigwedge_{s \in abst\_s} s \wedge \bigwedge_{t \in abst\_t} t))$
    end if
    $query \leftarrow I(V_0) \wedge \bigwedge_{i=0}^{k-1} T(V_i, V_{i+1}) \wedge bad(V_k) \wedge \bigwedge_{ref \in refs} ref$
    $\langle sat, model\rangle \leftarrow \textsc{SMT-solve}(query)$
  end while
end for

Iterate over candidate loops of length k.

Funnel-loop search: enumerate candidate loops

Algorithm GENERATE-ABSTRACT-LOOPS(M)

$\langle V, I, T, bad\rangle \leftarrow$ ENCODE-BMC-FAIR-ABSTRACT-LOOP(M)
for $k \in [0, 1, 2, \ldots]$ **do**
  $query \leftarrow I(V_0) \wedge \bigwedge_{i=0}^{k-1} T(V_i, V_{i+1}) \wedge bad(V_k)$
  $\langle sat, model\rangle \leftarrow$ SMT-SOLVE($query$)
  $refs \leftarrow []$
  while sat **do**
    $\langle abst\_s, abst\_t\rangle \leftarrow$ GET-IMPLICANT($model, query$)
    $\langle is\_ranked, rf\rangle \leftarrow$ RANK-LOOP($abst\_s, abst\_t$)
    if is_ranked **then**
      $\langle V, I, T, bad\rangle \leftarrow$
    else
      $v_0 \leftarrow$ GET-LOOPBACK-STATE($model$)
      yield $\langle v_0, abst\_s, abst\_t\rangle$
      $refs.append(\neg(\bigwedge_{s \in abst\_s} s \wedge \bigwedge_{t \in abst\_t} t))$
    end if
    $query \leftarrow I(V_0) \wedge \bigwedge_{i=0}^{k-1} T(V_i, V_{i+1}) \wedge bad(V_k) \wedge \bigwedge_{ref \in refs} ref$
    $\langle sat, model\rangle \leftarrow$ SMT-SOLVE($query$)
  end while
end for

Compute underapproximation of M corresponding to the loop described by $model$.

Funnel-loop search: enumerate candidate loops

Algorithm $\textsc{generate-abstract-loops}(M)$

$\langle V, I, T, bad\rangle \leftarrow \textsc{encode-}\mathrm{BMC}\textsc{-fair-abstract-loop}(M)$
for $k \in [0, 1, 2, \ldots]$ **do**
  $query \leftarrow I(V_0) \wedge \bigwedge_{i=0}^{k-1} T(V_i, V_{i+1}) \wedge bad(V_k)$
  $\langle sat, model\rangle \leftarrow \mathrm{SMT}\textsc{-solve}(query)$
  $refs \leftarrow []$
  while sat **do**
    $\langle abst\_s, abst\_t\rangle \leftarrow \textsc{get-implicant}(model, query)$
    $\langle is\_ranked, rf\rangle \leftarrow \textsc{rank-loop}(abst\_s, abst\_t)$
    if is_ranked **then**
      $\langle V, I, T, bad\rangle \leftarrow \textsc{remove-ranked-loops}(V, I, T, bad, rf)$
    else
      $v_0 \leftarrow \textsc{get-loo}$
      yield $\langle v_0, abst\_$
      $refs.append(\neg$
    end if
    $query \leftarrow I(V_0) \wedge \bigwedge_{i=0}^{k-1} T(V_i, V_{i+1}) \wedge bad(V_k) \wedge \bigwedge_{ref \in refs} ref$
    $\langle sat, model\rangle \leftarrow \mathrm{SMT}\textsc{-solve}(query)$
  end while
end for

Try synthesise ranking function for candidate, in case of success remove all loops ranked by the identified function.

Funnel-loop search: enumerate candidate loops

Algorithm GENERATE-ABSTRACT-LOOPS(M)

$\langle V, I, T, bad\rangle \leftarrow$ ENCODE-BMC-FAIR-ABSTRACT-LOOP(M)
for $k \in [0, 1, 2, \ldots]$ **do**
    $query \leftarrow I(V_0) \wedge \bigwedge_{i=0}^{k-1} T(V_i, V_{i+1}) \wedge bad(V_k)$
    $\langle sat, model\rangle \leftarrow$ SMT-SOLVE$(query)$
    $refs \leftarrow []$
    while sat **do**
        $\langle abst\_s, abst\_t\rangle \leftarrow$ GET-IMPLICANT$(model, query)$
        $\langle is\_ranked, rf\rangle \leftarrow$ RANK-LOOP$(abst_s, abst_t)$
        if is_ranked **then**
            $\langle V, I, T, bad\rangle \leftarrow$ REMOVE-RANKED-LOOPS(V, I, T, bad, rf)
        else
            $v_0 \leftarrow$ GET-LOOPBACK-STATE$(model)$
            yield $\langle v_0, abst\_s, abst\_t\rangle$
            $refs.append(\neg(\bigwedge \ldots s \wedge \bigwedge \ldots t))$
        end if
        $query \leftarrow I(V_0) \wedge \ldots$
        $\langle sat, model\rangle \leftarrow$ SMT-SOLVE$(query)$
    end while
end for

Get reachability witness from $model$ and return current candidate.

Funnel-loop templates

Algorithm GENERATE-TEMPLATES($\boldsymbol{v}_0, abst\_s, abst\_t$)

$ineqs \leftarrow$ HEURISTIC-PICK-NUM-INEQS($abst\_s, abst\_t$)
for $ineq \in ineqs$ **do**
    $n \leftarrow len(abst\_s)$
    $funnels \leftarrow []$
    for $i \in [0..n-2]$ **do**
        $src \leftarrow abst\_s[i] \wedge \bigwedge_{j=0}^{ineq-1}$ NEW-PARAMETRIC-EXPR(V) ≥ 0
        $rf \leftarrow$ NEW-PARAMETRIC-EXPR(V)
        $t \leftarrow \top$
        for $v_{i+1} \in V_{i+1}$ **do**
            if $v_{i+1} = f(V_i) \in abst\_t[i]$ for some function f **then**
                $t \leftarrow t \wedge v_{i+1} = f(V_i)$
            else
                $t \leftarrow t \wedge v_{i+1} =$ NEW-PARAMETRIC-EXPR(V_i)
            end if
        end for
        $dst(V) \leftarrow \exists V_0 \ : \ src(V_0) \wedge rf(V_0) = \mathbf{0} \wedge t(V_0, S)$
        $funnels.append(Funnel(src, t, rf, dst))$
    end for
    yield FUNNEL-LOOP($funnels, \boldsymbol{v}_0$)
end for

Funnel-loop templates

Algorithm $\textsc{Generate-Templates}(\boldsymbol{v}_0, abst\_s, abst\_t)$

$ineqs \leftarrow \textsc{Heuristic-Pick-Num-Ineqs}(abst\_s, abst\_t)$
for $ineq \in ineqs$ **do**
  $n \leftarrow len(abst\_s)$
  $funnels \leftarrow []$
  for $i \in [0..n-2]$ **do**
    $src \leftarrow abst\_s[i] \wedge \bigwedge_{j=0}^{ineq-1} \textsc{New-Parametric-Expr}(V) \geq 0$
    $rf \leftarrow \textsc{New-Parametric-Expr}(V)$
    $t \leftarrow \top$
    for $v_{i+1} \in V_{i+1}$ **do**
      if $v_{i+1} = f(V_i) \in abst\_t[i]$ for some function f **then**
        $t \leftarrow t \wedge v_{i+1} = f(V_i)$
      else
        $t \leftarrow t \wedge v_{i+1} = \textsc{New-Parametric-Expr}(V_i)$
      end if
    end for
    $dst(V) \leftarrow \exists V_0 \ : \ src(V_0) \wedge rf(V_0) = \mathbf{0} \wedge t(V_0, S)$
    $funnels.append(Funnel(src, t, rf, dst))$
  end for
  yield $\textsc{Funnel-loop}(funnels, \boldsymbol{v}_0)$
end for

Create a funnel template for every region in the candidate loop.

Funnel-loop templates

Algorithm $\textsc{generate-templates}(\boldsymbol{v}_0, abst\_s, abst\_t)$

$ineqs \leftarrow \textsc{heuristic-pick-num-ineqs}(abst\_s, abst\_t)$
for $ineq \in ineqs$ **do**
  $n \leftarrow len(abst\_s)$
  $funnels \leftarrow []$
  for $i \in [0..n-2]$ **do**
    $src \leftarrow abst\_s[i] \wedge \bigwedge_{j=0}^{ineq-1} \textsc{new-parametric-expr}(V) \geq 0$
    $rf \leftarrow \textsc{new-parametric-expr}(V)$
    $t \leftarrow \top$
    for $v_{i+1} \in V_{i+1}$ **d**
      if $v_{i+1} = f(V_i$
        $t \leftarrow t \wedge v_{i+1} = f(V_i)$
      else
        $t \leftarrow t \wedge v_{i+1} = \textsc{new-parametric-expr}(V_i)$
      end if
    end for
    $dst(V) \leftarrow \exists V_0 \;:\; src(V_0) \wedge rf(V_0) = \mathbf{0} \wedge t(V_0, S)$
    $funnels.append(Funnel(src, t, rf, dst))$
  end for
  yield $\textsc{Funnel-loop}(funnels, \boldsymbol{v}_0)$
end for

Strengthen region with $ineq$ new parametric predicates, e.g. linear combinations $\sum_{v \in V_i} \lambda_v v$.

Funnel-loop templates

Algorithm GENERATE-TEMPLATES$(\boldsymbol{v}_0, abst_s, abst_t)$

$ineqs \leftarrow$ HEURISTIC-PICK-NUM-INEQS$(abst_s, abst_t)$
for $ineq \in ineqs$ **do**
  $n \leftarrow len(abst\_s)$
  $funnels \leftarrow []$
  for $i \in [0..n-2]$ **do**
    $src \leftarrow abst\_s[i] \wedge \bigwedge$
    $rf \leftarrow$ NEW-PARAME
    $t \leftarrow \top$
    for $v_{i+1} \in V_{i+1}$ **do**
      if $v_{i+1} = f(V_i) \in abst\_t[i]$ for some function f **then**
        $t \leftarrow t \wedge v_{i+1} = f(V_i)$
      else
        $t \leftarrow t \wedge v_{i+1} =$ NEW-PARAMETRIC-EXPR(V_i)
      end if
    end for
    $dst(V) \leftarrow \exists V_0 \ : \ src(V_0) \wedge rf(V_0) = \mathbf{0} \wedge t(V_0, S)$
    $funnels.append(Funnel(src, t, rf, dst))$
  end for
  yield FUNNEL-LOOP$(funnels, \boldsymbol{v}_0)$
end for

Next assignments are function of current assignments: transition relation left-total by construction.

Algorithm $\textsc{Generate-Templates}(\boldsymbol{v}_0, abst\_s, abst\_t)$

$ineqs \leftarrow \textsc{Heuristic-Pick-Num-Ineqs}(abst\_s, abst\_t)$
for $ineq \in ineqs$ **do**
  $n \leftarrow len(abst\_s)$
  $funnels \leftarrow []$
  for $i \in [0..n-2]$ **do**
    $src \leftarrow abst\_s[i] \wedge \bigwedge_{j=0}^{ineq-1} \textsc{New-Parametric-Expr}(V) \geq 0$
    $rf \leftarrow \textsc{New-Parametric-Expr}(V)$
    $t \leftarrow \top$
    for $v_{i+1} \in V_{i+1}$ **do**
      if $v_{i+1} = f(V_i) \in abst\_t[i]$ for some function f **then**
        $t \leftarrow t \wedge v_{i+1} = f(V_i)$
      else
        $t \leftarrow t \wedge v_{i+1} =$ NE
      end if
    end for
    $dst(V) \leftarrow \exists V_0 \; : \; src(V_0) \wedge rf(V_0) = \mathbf{0} \wedge t(V_0, S)$
    $funnels.append(Funnel(src, t, rf, dst))$
  end for
  yield $\textsc{Funnel-loop}(funnels, \boldsymbol{v}_0)$
end for

Destination region implicitly defined.

Funnel-loop template example

Assume $abst\_s \doteq [\{k > 0\}, \{k < 0\}]$ and
$abst\_t \doteq [\{k' = k - n\}, \{k' = k + n\}]$.

For $ineq$ equal to 1 we generate a funnel-loop described by the following components:

- $S_0 \doteq k > 0 \wedge \lambda_0 k + \lambda_1 n + \lambda_2 \geq 0$;
- $t_0 \doteq k' = k - n \wedge n' = \lambda_3 k + \lambda_4 n + \lambda_5$;
- $\mathrm{R}\textsc{f}_0 \doteq \lambda_6 n + \lambda_7 k + \lambda_8$;

- $S_1 \doteq k < 0 \wedge \lambda_9 k + \lambda_{10} n + \lambda_{11} \geq 0$;
- $t_1 \doteq k' = k + n \wedge n' = \lambda_{12} k + \lambda_{13} n + \lambda_{14}$;
- $\mathrm{R}\textsc{f}_1 \doteq \lambda_{15} n + \lambda_{16} k + \lambda_{17}$;

Objective: find assignment to the $\{\lambda_i\}_{i=0}^{17}$.

Funnel-loop template example

Assume $abst\_s \doteq [\{k > 0\}, \{k < 0\}]$ and $abst\_t \doteq [\{k' = k - n\}, \{k' = k + n\}]$.

For $ineq$ equal to 1 we generate a funnel-loop described by the following components:

- $S_0 \doteq k > 0 \wedge \lambda_0 k + \lambda$
- $t_0 \doteq k' = k - n \wedge n' =$
- $\mathrm{RF}_0 \doteq \lambda_6 n + \lambda_7 k + \lambda$

- $S_1 \doteq k < 0 \wedge \lambda_9 k + \lambda$
- $t_1 \doteq k' = k + n \wedge n' =$
- $\mathrm{RF}_1 \doteq \lambda_{15} n + \lambda_{16} k +$

Solution:

- $S_0 \doteq k > 0 \wedge n \geq k + 1$;
- $t_0 \doteq k' = k - n \wedge n' = n + 1$;
- $\mathrm{RF}_0 = 0$

- $S_1 \doteq k < 0 \wedge n \geq -k + 1$;
- $t_1 \doteq k' = k + n \wedge n' = n + 1$;
- $\mathrm{RF}_1 = 0$

Objective: find assignment to the $\{\lambda_i\}_{i=0}^{17}$.

Funnel-loop synthesis problem

For a funnel-loop template of length n, search for an assignment to the parameters P such that the following hold:

- $\boldsymbol{v}_0$ is in the first region (i.e. funnel-loop is reachable):

$$\exists P \ : \ S_0(\boldsymbol{v}_0, P)$$

- Remain in the same region as long as the ranking function is positive:

$$\exists P \forall V, V' : S_i(V, P) \wedge \mathrm{RF}_i(V, P) > \mathbf{0} \wedge T_i(V, V', P) \rightarrow S_i(V', P) \wedge \mathrm{RF}_i(V', P) < \mathrm{RF}_i(V, P)$$

- Reach next region when ranking function is 0:

$$\exists P \forall V, V' : S_i(V, P) \wedge \mathrm{RF}_i(V, P) = \mathbf{0} \wedge T_i(V, V', P) \rightarrow S_{i+1\%n}(V', P)$$

- Every step underapproximates the transition relation of M:

$$\exists P \forall V, V' : S(V, P) \wedge T(V, V', P) \rightarrow T_M(V, V')$$

Prototype implementation

F3 implements the search procedure on top of PYSMT using MathSAT5 and Z3.

Supports LTL via the tableau construction and timed systems by removing zeno-paths and ensuring the divergence of *time*.

We considered benchmarks from the following categories:

- **LS** : linear software,
- **NS** : nonlinear software,
- **ITS** : LTL on infinite state transition systems,
- **TA** : LTL on timed automata,
- **TTS** : LTL on timed transition systems,
- **HS** : LTL on hybrid systems.

Experimental evaluation

Table reports number of solved instances per benchmark family.
F3 can only identify counterexamples, most of the other tools can also prove properties.

Benchmark family	F3	Anant	AProVe	DiVinE3	MITLBMC	nuXmv	T2	Ultimate	Uppaal
LS (52)	**52**	38	43	–	–	28	38	49	–
NS (30)	**29**	25	5	–	–	14	2	–	–
ITS (70)	**57**	–	–	–	–	4	–	8	–
TA (174)	137	–	–	43	**151**	90	–	0	103*
TTS (120)	**55**	–	–	–	–	8	–	1	–
HS (9)	0	–	–	–	–	0	–	–	–
Total (455)	**330**	63	48	43	151	144	40	58	103

* UPPAAL does not handle full LTL: ran only on 116 instances.

Entries marked with "–" denote that the tool cannot handle the given benchmarks.

Conclusions

Summary

- representation of fair paths via funnel-loops;
- automated search of funnel-loop via reduction to SMT;
- appears to be effective on a wide range of benchmark categories.

Future work

- integration with techniques to prove LTL properties;
- define more incremental procedure, possibly exploiting a decomposition of the system.

The End

**Thank you for your attention,
questions?**

Inductive Reachability Witness

"Funnel" also called Inductive Reachability Witness in "Polynomial Reachability Witnesses via Stellensätze", Asadi et al., PLDI-2021