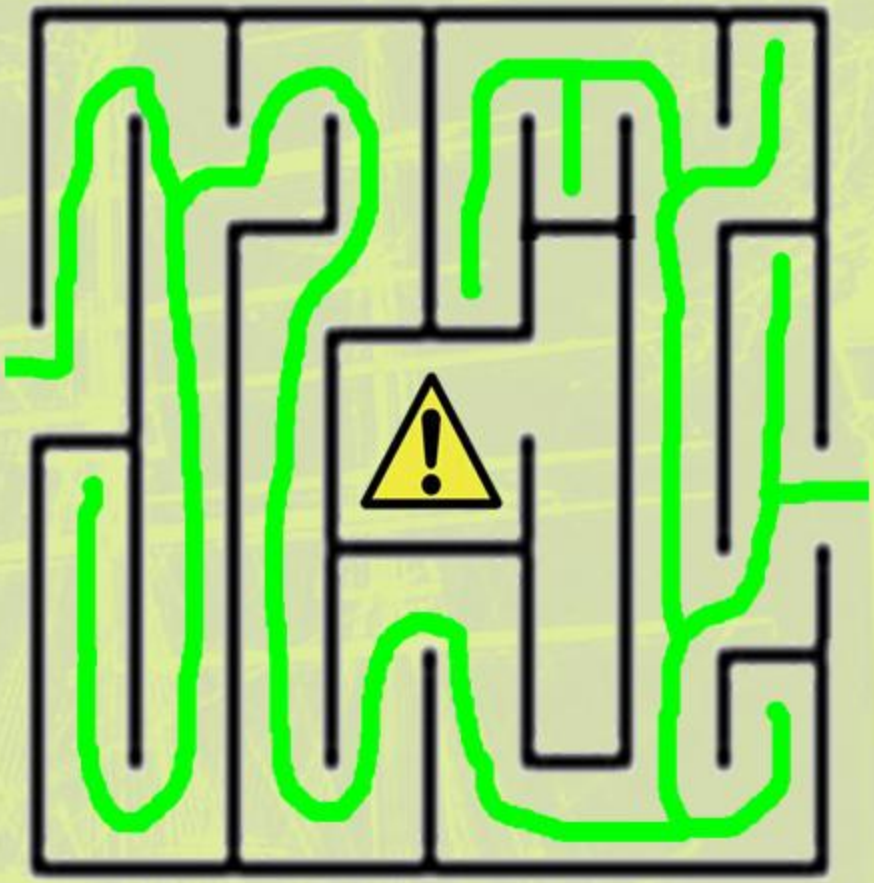
cupping

adam bakewell

university of birmingham

model checking for unsafety

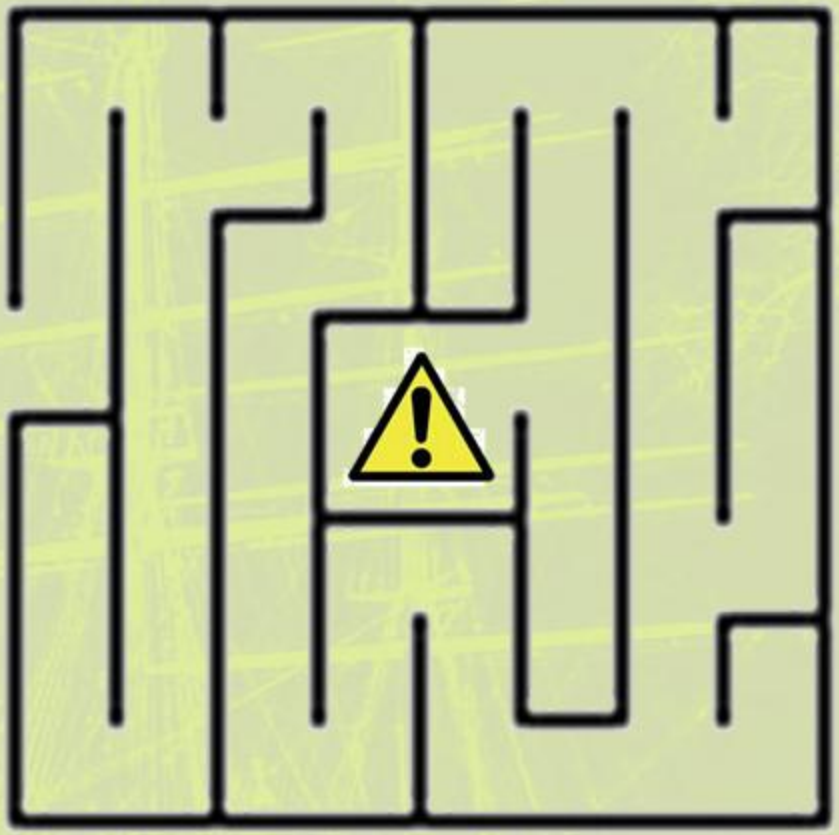
model checking for safety



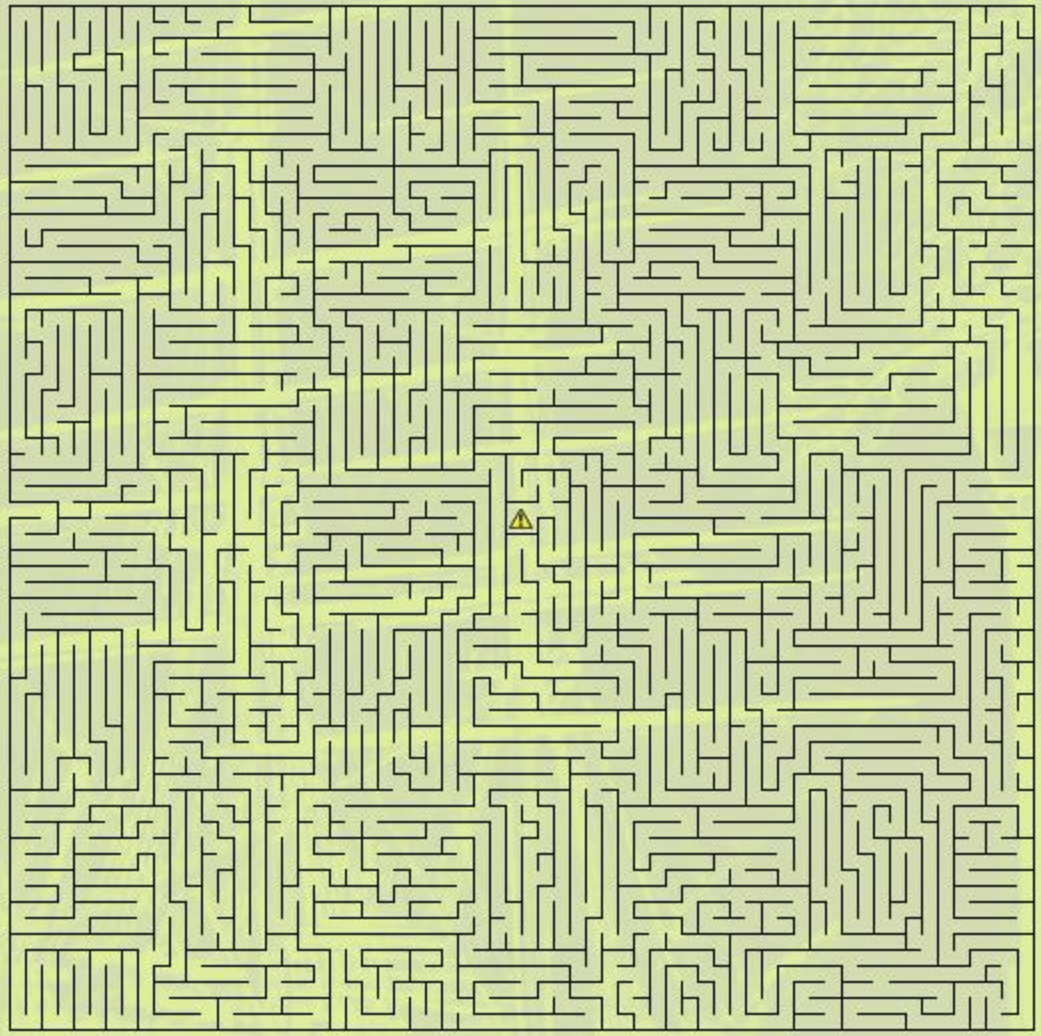

```
1.  button : bool,
2.  wait, crash : com
3.  |-
4.  macro NorthSouth 0 in macro EastWest 1 in
5.  new nat ns := 0 in new nat ew := 0 in

6.  let off be fun d:nat .  if d=NorthSouth then ns := 0 else ew := 0 in

7.  let on be fun d:nat . assert (if d=NorthSouth then ew=0 else ns=0) crash;
8.                        if d=NorthSouth then ns := 1 else ew := 1 in

9.  on(EastWest);

10. while 1 do {
11.   new nat timer := 10 in
12.   new nat zebra := 0 in
13.   while timer > 0 do {
14.     wait;
15.     timer := timer - 1;
16.     if button then {
17.       zebra := zebra + 1;
18.       new nat  timer1 := zebra in
19.       while timer > 0 & timer1 > 0 do {
20.         wait;
21.         timer1 := timer1 - 1;
22.         timer := timer – 1};
23.       if timer > 0 then {
24.         off(EastWest);
25.         new nat  timer2 := 3 in
26.         while timer2 > 0 do {
27.           wait;
28.           timer2 := timer2 – 1};
29.         on(EastWest) } };
30.   off(EastWest); on(NorthSouth);
31.   new nat timer := 10 in
32.   new nat zebra := 0 in
33.   while timer > 0 do {
34.     wait;
35.     timer := timer - 1;
36.     if button then {
37.       zebra := zebra + 1;
38.       new nat timer1 := zebra in
39.       while timer > 0 & timer1 > 0 do {
40.         wait;
41.         timer1 := timer1 - 1;
42.         timer := timer – 1 };
43.       if timer > 0 then {
44.         off(NorthSouth);
45.         new nat timer2 := 3 in
46.         while timer2 > 0 do {
47.           wait;
48.           timer2 := timer2 – 1 };
49.         on(NorthSouth) } };
50.   off(NorthSouth); on(EastWest) }
```

state explosion

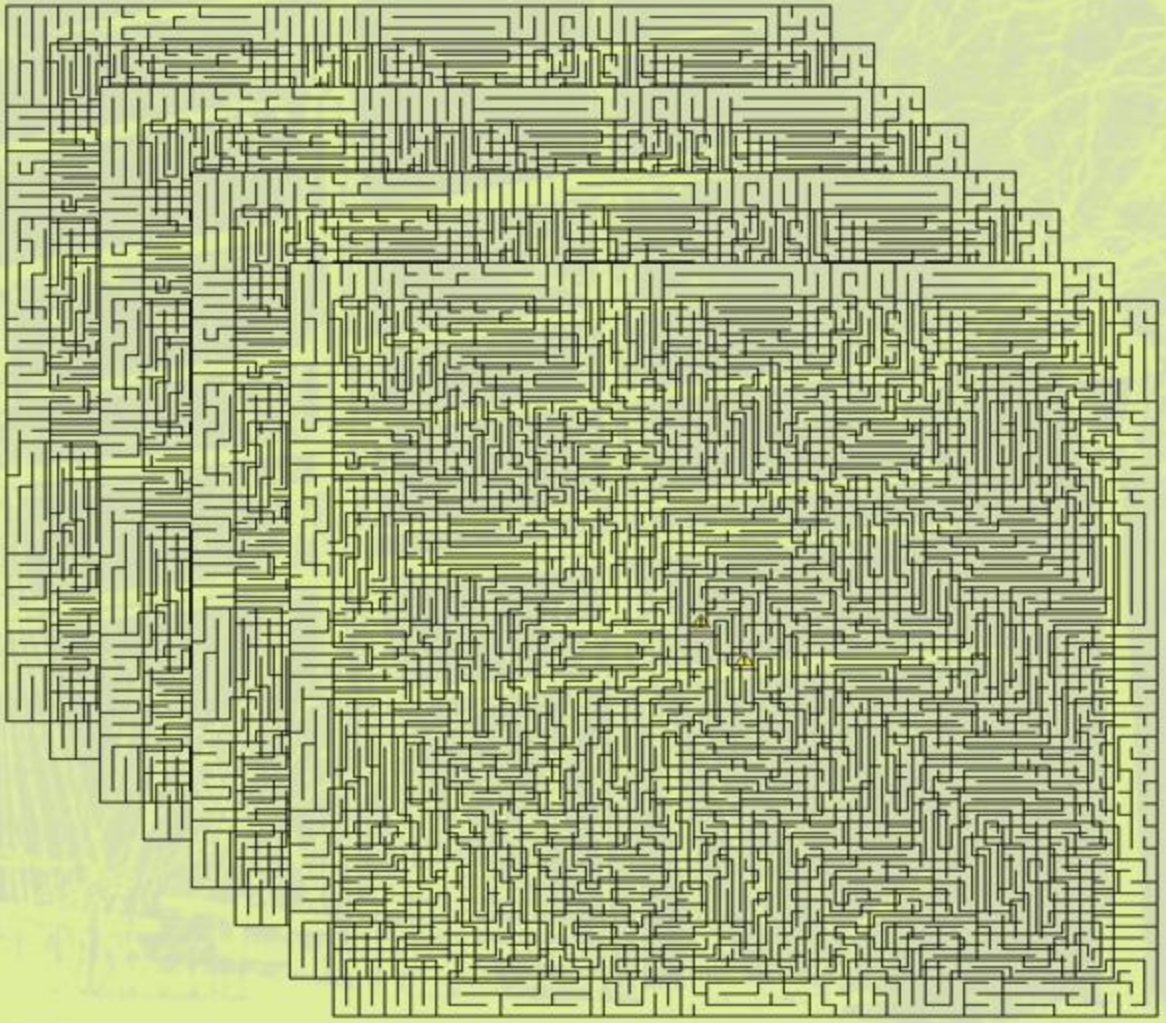
state explosion



state explosion



state explosion



```
1.   button : bool,
2.   wait, crash : com
3.   |-
4.   macro NorthSouth 0 in macro EastWest 1 in
5.   new nat ns := 0 in new nat ew := 0 in
6.   let off be fun d:nat .  if d=NorthSouth then ns := 0 else ew := 0 in
7.   let on be fun d:nat . assert (if d=NorthSouth then ew=0 else ns=0) crash;
8.                         if d=NorthSouth then ns := 1 else ew := 1 in
9.   on(EastWest);
10.  while 1 do {
11.   new nat timer := 10 in
12.   new nat zebra := 0 in
13.   while timer > 0 do {
14.    wait;
15.    timer := timer – 1 };
16.   off(EastWest); on(NorthSouth);
17.   off(NorthSouth); on(EastWest) }
```

0001 traces

0056 traces

```
1.   button : bool,
2.   wait, crash : com
3.   |-
4.   macro NorthSouth 0 in macro EastWest 1 in
5.   new nat ns := 0 in new nat ew := 0 in
6.   let off be fun d:nat .  if d=NorthSouth then ns := 0 else ew := 0 in
7.   let on be fun d:nat . assert (if d=NorthSouth then ew=0 else ns=0) crash;
8.                         if d=NorthSouth then ns := 1 else ew := 1 in
9.   on(EastWest);
10.  while 1 do {
11.   new nat timer := 10 in
12.   new nat zebra := 0 in
13.   while timer > 0 do {
14.    wait;
15.    timer := timer - 1;
16.    if button then {
17.     zebra := zebra + 1;
18.     if timer > 0 then {
19.      off(EastWest);
20.      on(EastWest) } };
21.   off(EastWest); on(NorthSouth);
22.   off(NorthSouth); on(EastWest) }
```

0075 traces

```
1.   button : bool,
2.   wait, crash : com
3.   |-
4.   macro NorthSouth 0 in macro EastWest 1 in
5.   new nat ns := 0 in new nat ew := 0 in
6.   let off be fun d:nat .  if d=NorthSouth then ns := 0 else ew := 0 in
7.   let on be fun d:nat . assert (if d=NorthSouth then ew=0 else ns=0) crash;
8.                         if d=NorthSouth then ns := 1 else ew := 1 in
9.   on(EastWest);
10.  while 1 do {
11.   new nat timer := 10 in
12.   new nat zebra := 0 in
13.   while timer > 0 do {
14.    wait;
15.    timer := timer - 1;
16.    if button then {
17.     zebra := zebra + 1;
18.     new nat  timer1 := zebra in
19.     while timer > 0 & timer1 > 0 do {
20.      wait;
21.      timer1 := timer1 - 1;
22.      timer := timer – 1 };
23.     if timer > 0 then {
24.      off(EastWest);
25.      on(EastWest) } };
26.   off(EastWest); on(NorthSouth);
27.   off(NorthSouth); on(EastWest) }
```

0095 traces

```
1.   button : bool,
2.   wait, crash : com
3.   |-
4.   macro NorthSouth 0 in macro EastWest 1 in
5.   new nat ns := 0 in new nat ew := 0 in
6.   let off be fun d:nat .  if d=NorthSouth then ns := 0 else ew := 0 in
7.   let on be fun d:nat . assert (if d=NorthSouth then ew=0 else ns=0) crash;
8.                         if d=NorthSouth then ns := 1 else ew := 1 in
9.   on(EastWest);
10.  while 1 do {
11.   new nat timer := 10 in
12.   new nat zebra := 0 in
13.   while timer > 0 do {
14.    wait;
15.    timer := timer - 1;
16.    if button then {
17.     zebra := zebra + 1;
18.     new nat  timer1 := zebra in
19.     while timer > 0 & timer1 > 0 do {
20.      wait;
21.      timer1 := timer1 - 1;
22.      timer := timer – 1 };
23.     if timer > 0 then {
24.      off(EastWest);
25.      new nat  timer2 := 3 in
26.      while timer2 > 0 do {
27.       wait;
28.       timer2 := timer2 – 1 };
29.      on(EastWest) } };
30.   off(EastWest); on(NorthSouth);
31.   off(NorthSouth); on(EastWest) }
```

3785 traces

```
1.  button : bool,
2.  wait, crash : com
3.  |-
4.  macro NorthSouth 0 in macro EastWest 1 in
5.  new nat ns := 0 in new nat ew := 0 in
6.  let off be fun d:nat .  if d=NorthSouth then ns := 0 else ew := 0 in
7.  let on be fun d:nat . assert (if d=NorthSouth then ew=0 else ns=0) crash;
8.                        if d=NorthSouth then ns := 1 else ew := 1 in
9.  on(EastWest);
10. while 1 do {
11.  new nat timer := 10 in
12.  new nat zebra := 0 in
13.  while timer > 0 do {
14.   wait;
15.   timer := timer - 1;
16.   if button then {
17.    zebra := zebra + 1;
18.    new nat  timer1 := zebra in
19.    while timer > 0 & timer1 > 0 do {
20.     wait;
21.     timer1 := timer1 - 1;
22.     timer := timer – 1 };
23.    if timer > 0 then {
24.     off(EastWest);
25.     new nat  timer2 := 3 in
26.     while timer2 > 0 do {
27.      wait;
28.      timer2 := timer2 – 1 };
29.     on(EastWest) } };
30.  off(EastWest); on(NorthSouth);
31.  new nat timer := 10 in
32.   new nat zebra := 0 in
33.   while timer > 0 do {
34.    wait;
35.    timer := timer - 1;
36.    if button then {
37.     zebra := zebra + 1;
38.     new nat timer1 := zebra in
39.     while timer > 0 & timer1 > 0 do {
40.      wait;
41.      timer1 := timer1 - 1;
42.      timer := timer – 1 };
43.     if timer > 0 then {
44.      off(NorthSouth);
45.      new nat timer2 := 3 in
46.      while timer2 > 0 do {
47.       wait;
48.       timer2 := timer2 – 1 };
49.      on(NorthSouth) } };
50.   off(NorthSouth); on(EastWest) }
```

abstraction

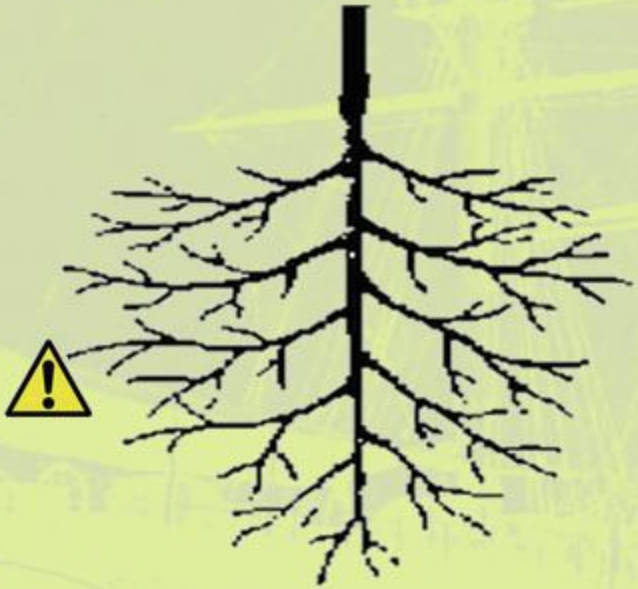
- The answer: abstract the model and check only the remaining tricky part not taken out by the general proven principle embodied by the abstraction
- The skill: abstraction design and application
- but, wait a minute…
- Could we just abstract the program and leave everything else the same?

Clipping

- Clip subterms not containing targets

Clipping

- Clip subterms not containing targets



Clipping

- Clip subterms not containing targets

Clipping

- Clip subterms not containing targets

Clipping

- Clip subterms not containing targets

Clipping

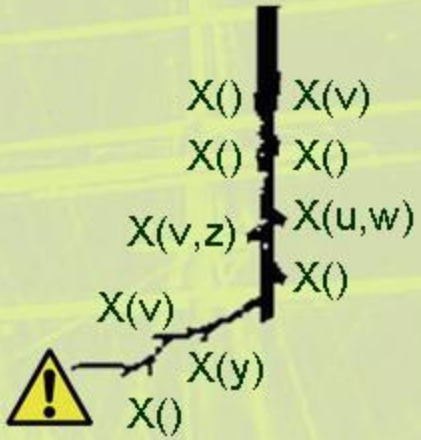
- Clip subterms not containing targets

Clipping

- Clip subterms not containing targets

stumping

- Replace clipped subterms with stump X(v1,..,vn)

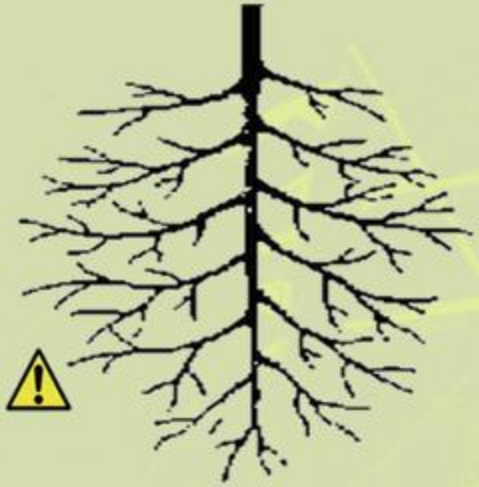


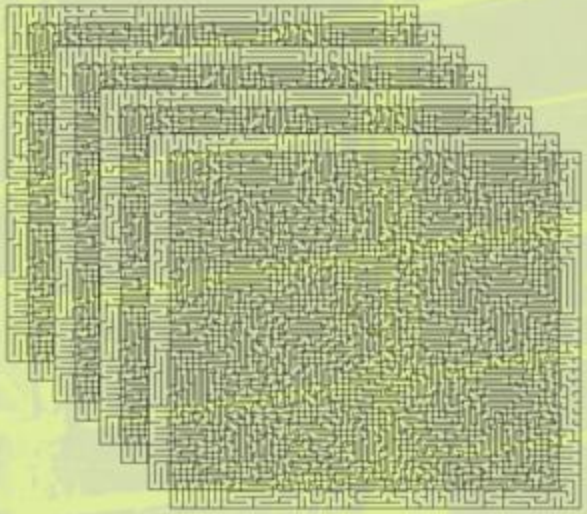
```
button : bool,
wait, crash : com
|-
macro NorthSouth 0 in macro EastWest 1 in
new bool ns . X1(ns); new bool ew . X2(ew);

let off be fun d:nat . if d=NorthSouth then ns := 0 else ew := 0 in

let on be fun d:nat . assert (if d=NorthSouth then ew=0 else ns=0) crash;
                      if d=NorthSouth then ns := 1 else ew := 1 in

if X3(ns) then crash else X4(ew)

while 1 do {
 new nat timer . X5(timer);
 new nat zebra . X6(zebra);
 while X7(timer) do {
  X8(timer,wait);
  timer := timer - 1;
  if button then {
   X9(zebra,wait);
   new nat timer1 := zebra in
   while timer > 0 & timer1 > 0 do {
    wait;
    timer1 := timer1 - 1;
    timer := timer – 1};
   if X10(timer) then {
    X11(ew,wait);
    new nat timer2 := 3 in
    while timer2 > 0 do {
     wait;
     timer2 := timer2 – 1};
    if X12(ns) then crash else X13(ew)}}};
 X14(ew); if X15(ew) then crash else X16(ns)
 new nat timer . X17(timer);
 new nat zebra . X18(zebra);
 while X19(timer) do {
  X20(timer,wait);
  timer := timer - 1;
  if button then {
   X21(zebra,wait);
   new nat timer1 := zebra in
   while timer > 0 & timer1 > 0 do {
    wait;
    timer1 := timer1 - 1;
    timer := timer – 1 };
   if X22(timer) then {
    X23(ns,wait);
    new nat timer2 := 3 in
    while timer2 > 0 do {
     wait;
     timer2 := timer2 – 1 };
    if X24(ew) then crash else X25(ns)}}};
 X26(ns); if X27(ns) then crash else X28(ew) }
```

```
1.  button : bool,
2.  wait, crash : com
3.  |-|-
4.  macro NorthSouth 0 in macro EastWest 1 in
5.  new nat ns := 0 in new nat ew := 0 in

6.  let off be fun d:nat .  if d=NorthSouth then ns := 0 else ew := 0 in

7.  let on be fun d:nat . assert (if d=NorthSouth then ew=0 else ns=0) crash;
8.                        if d=NorthSouth then ns := 1 else ew := 1 in

9.  on(EastWest);

10. while 1 do {
11.   new nat timer := 10 in
12.   new nat zebra := 0 in
13.   while timer > 0 do {
14.     wait;
15.     timer := timer - 1;
16.     if button then {
17.       zebra := zebra + 1;
18.       new nat  timer1 := zebra in
19.       while timer > 0 & timer1 > 0 do {
20.         wait;
21.         timer1 := timer1 - 1;
22.         timer := timer – 1};
23.       if timer > 0 then {
24.         off(EastWest);
25.         new nat  timer2 := 3 in
26.         while timer2 > 0 do {
27.           wait;
28.           timer2 := timer2 – 1};
29.         on(EastWest) } };
30.   off(EastWest); on(NorthSouth);

31.   new nat timer := 10 in
32.   new nat zebra := 0 in
33.   while timer > 0 do {
34.     wait;
35.     timer := timer - 1;
36.     if button then {
37.       zebra := zebra + 1;
38.       new nat timer1 := zebra in
39.       while timer > 0 & timer1 > 0 do {
40.         wait;
41.         timer1 := timer1 - 1;
42.         timer := timer – 1 };
43.       if timer > 0 then {
44.         off(NorthSouth);
45.         new nat timer2 := 3 in
46.         while timer2 > 0 do {
47.           wait;
48.           timer2 := timer2 – 1 };
49.         on(NorthSouth) } };
50.   off(NorthSouth); on(EastWest) }
```

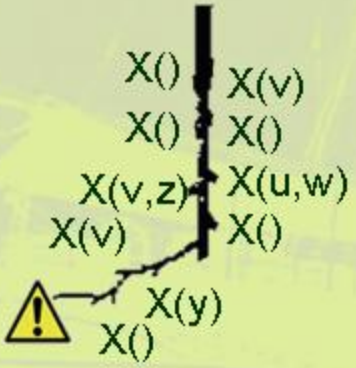

```
button : bool,
wait, crash : com
|-
new bool ns . X1(ns); new bool ew . X2(ew);
if X3(ns) then crash else X4(ew)

while 1 do {
  new nat timer . X5(timer);                                    new nat
  new nat zebra . X6(zebra);                                    new nat
  while X7(timer) do {                                          while X1
    X8(timer,wait);                                              X20(tir
    if button then {                                             if butto
      X9(zebra,wait);                                             X21(z
      if X10(timer) then {                                        if X22
        X11(ew,wait);                                              X23
        if X12(ns) then crash else X13(ew)}}};                      if X2
  X14(ew); if X15(ew) then crash else X16(ns)      X26(ns)
```

clipping safety property



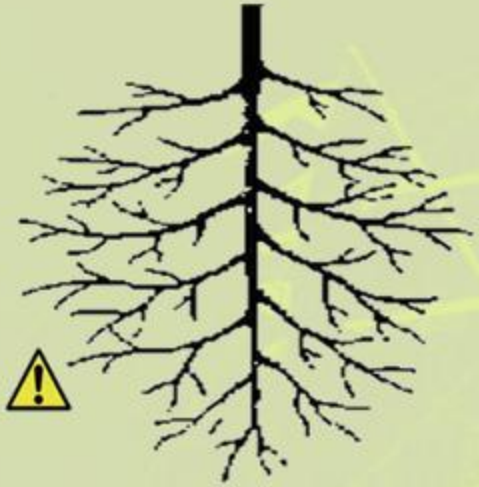


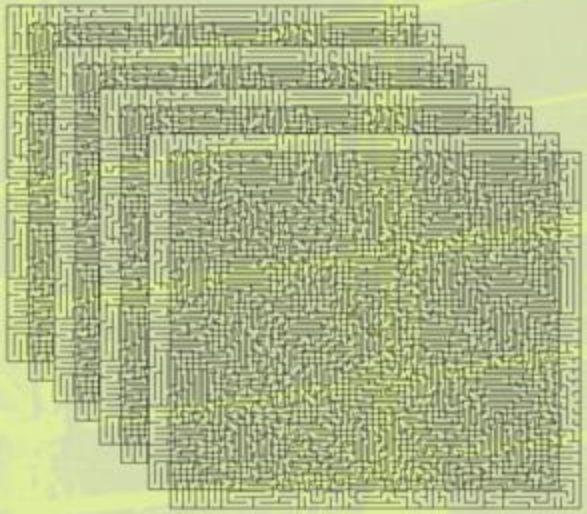
safe



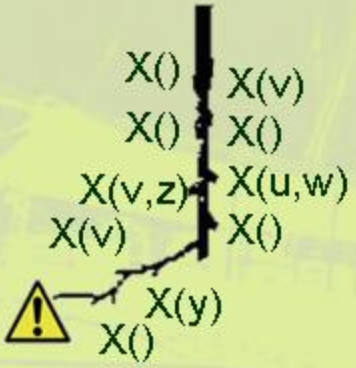
safe

clipping unsafety property



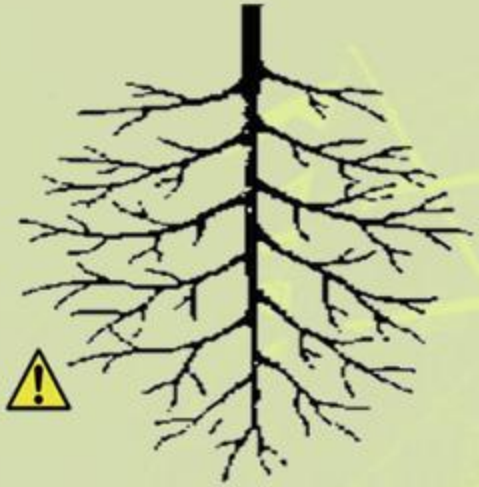


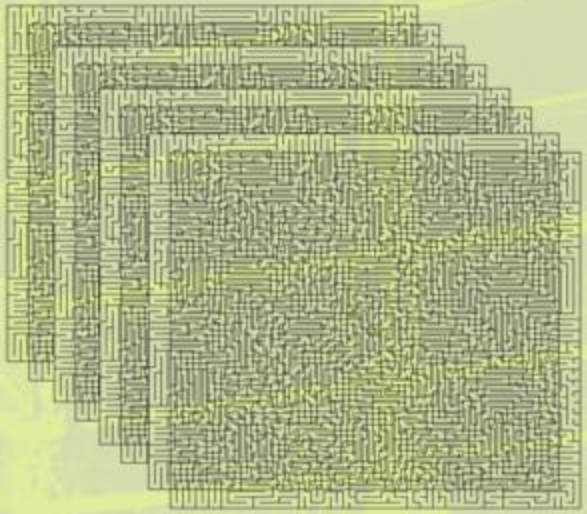
unsafe if path has no stump



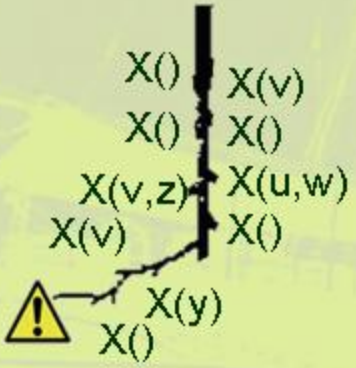
unsafe

clipping unknown safety property





unknown if path has a stump



unsafe

Clipping safety property proof

- Every trace to target assert in unclipped model – plus stump moves – appears in clipped model
- i.e. X(v1,..,vn) overapproximates
  (M where fv(M)={v1,..,vn})
- With non-denotational models the stumps would be a language extension – model pieces in the program - with denotational (e.g. game) models they are just id's

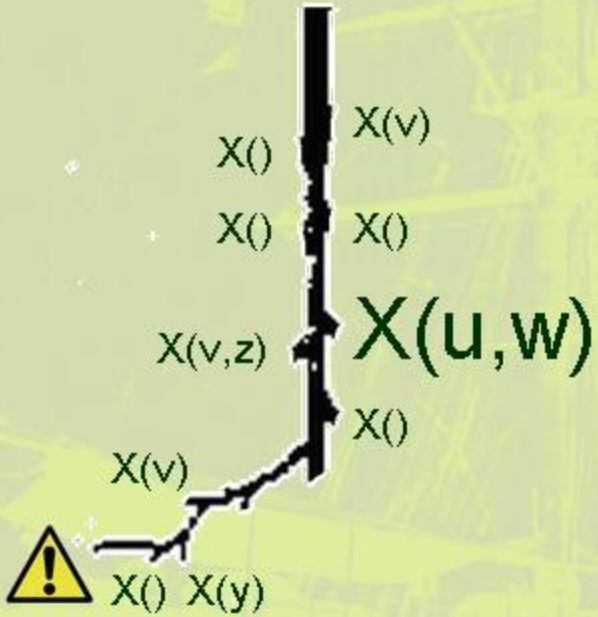
button : bool,
wait, crash : com
|-
new bool ns . X1(ns); new bool ew . X2(ew);
if X3(ns) then crash else X4(ew)

CEX: .q,X1.q,X1.ok,X2.q,X2.ok,X3.q,X3.tt,crash.q

unknown safety!

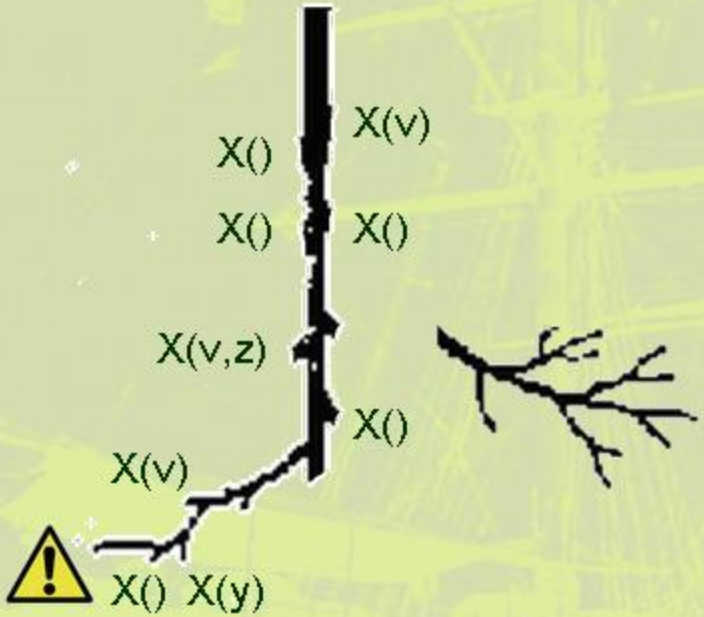
grow back

1. Choose stump $X(v_1,..,v_n)$ from CEX



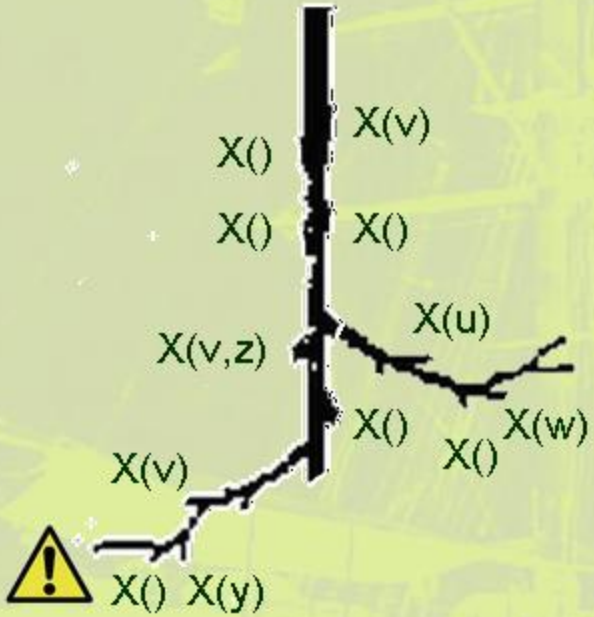
grow back

2. Get subterm M replaced by stump

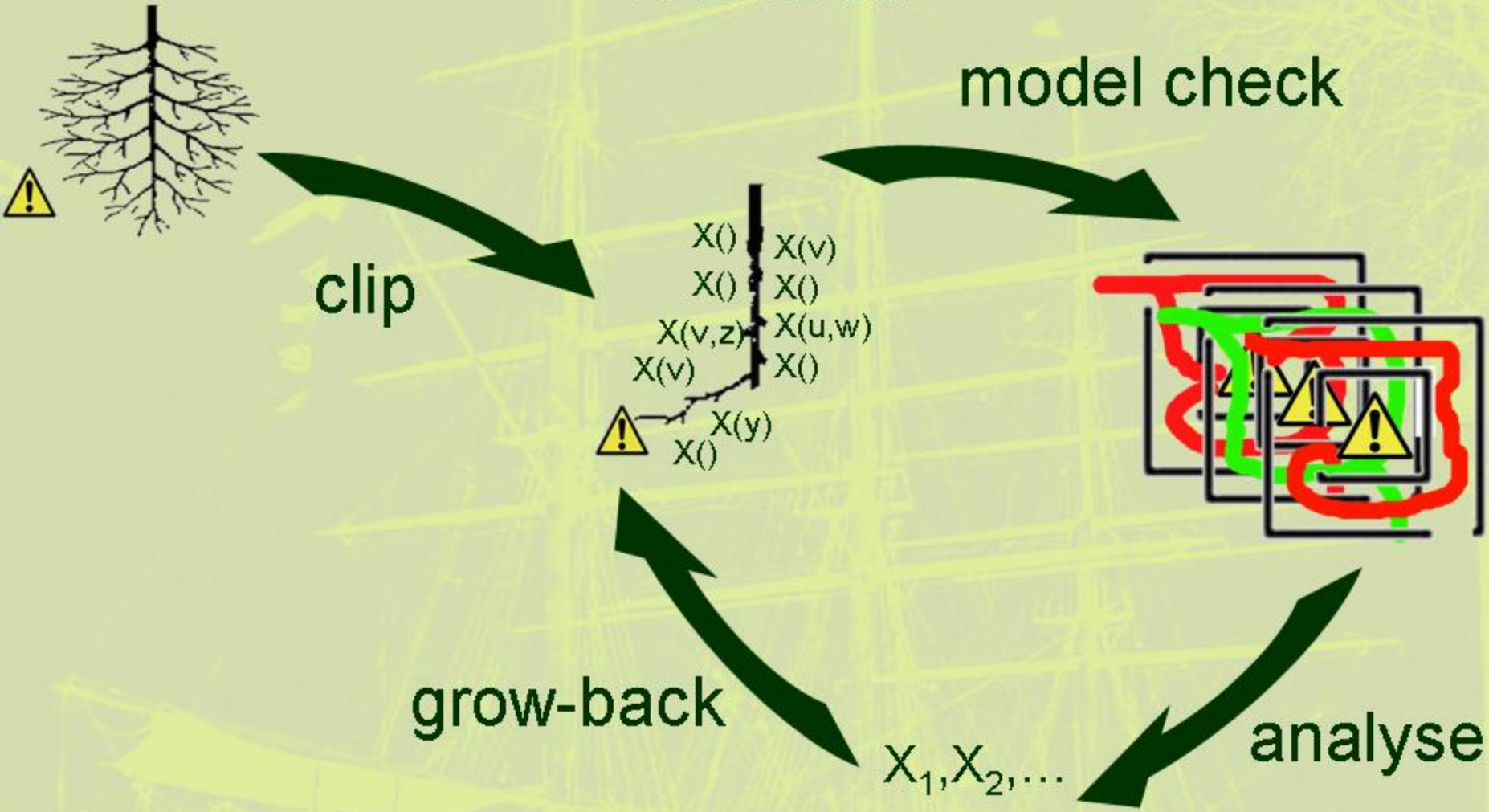


grow back

3. clip M (target some leaf) and graft



CEGAR


model check
clip
X()
X(v)
X()
X()
X(v,z)
X(u,w)
X(v)
X()
X(y)
X()
grow-back
$X_1, X_2, \ldots$
analyse

new bool ns . X1(ns); new bool ew . X2(ew);
if X3(ns) then crash else X4(ew)
CEX: .q,X1.q,X1.ok,X2.q,X2.ok,X3.q,X3.tt,crash.q

new bool ns . X1(ns); new bool ew . X2(ew);
if ns != 0 then crash else X4(ew)
CEX:X1.q,X1.0.1,ns.1,ns.ok,X1.0.ok,X1.ok,X2.q,X2.ok,X3.q,X3.tt,crash.q

new bool ns := 0; new bool ew . X2(ew);
if X3(ns) then crash else X4(ew)
CEX: X2.q,X2.ok,X3.q,X3.tt,crash.q

new bool ns . X1(ns); new bool ew := 0;
if X3(ns) then crash else X4(ew)
CEX: X1.q,X1.ok,X3.q,X3.tt,crash.q

new bool ns := 0; new bool ew := 0;
if X3(ns) then crash else X4(ew)
CEX: X3.q,X3.tt,crash.q

new bool ns := 0; new bool ew . X2(ew);
if ns != 0 then crash else X4(ew)
SAFE

new bool ns . X1(ns); new bool ew := 0;
if ns != 0 then crash else X4(ew)
CEX:X1.q,X1.0.1,ns.1,ns.ok,X1.0.ok,X1.ok,X3.q,X3.tt,crash.q

new bool ns := 0; new bool ew := 0;
if ns != 0 then crash else X4(ew)
SAFE

strategy

- Find minimal (un)safe clipping by BFS
- Optimize: prioritize grow-back for:
  - Conditional nearest crash
  - Assignment to ids in conditionals on crash path

SAFE with 8 grow-backs

```
button : bool,
wait, crash : com
|-
new nat ns := 0 in
new nat ew . X2(ew) in
if ns = 1 then crash else X4(ew);

while 1 do {
  new nat timer . X5(timer);
  new nat zebra . X6(zebra);
  while X7(timer) do {
    X8(timer,wait);
    if button then
      X9(zebra,timer,wait);
      if X10(timer) then {
        X11(ew,wait);
        if ns = 1 then crash else X13(ew) } };
  ew := 0; if ew = 1 then crash else X16(ns);
```

```
new nat ti
new nat ze
while X19(
 X20(time
 if button
  X21(zeb
  if X22(ti
   X23(ns
   if ew =
ns := 0; if
```

variant: UNSAFE with 9 grow-backs

```
button : bool,
wait, crash : com
|-
new nat ns := 0 in
new nat ew . X2(ew) in
if ns = 1 then crash else ew := 1;

while 1 do {
  new nat timer := 10;
  new nat zebra := 0;
  while timer > 0 do {
    wait; timer := timer – 1;
    if button then
      X9(zebra,timer,wait);
      if X10(timer) then {
        X11(ew,wait);
        if X12(ns) then crash else X13(ew) } };
  if ew = 1 then crash else X15(ns); X16(ew);
```

```
new nat ti
new nat ze
while X19(
 X20(time
 if button
  X21(zeb
  if X22(ti
   X23(ns
   if ew =
ns := 0; if
```


```
button : bool,
wait, crash : com
|-
new nat ns := 0 in
new nat ew := 0 in
if ns = 1 then crash else ew := 1;

while 1 do {
  new nat timer . X5(timer);
  new nat zebra . X6(zebra);
  while X7(timer) do {
    X8(timer,wait);
    if button then
      X9(zebra,timer,wait);
      if X10(timer) then {
        ew := 0; wait;
        if ns = 1 then crash else ew := 1 } };
  ew := 0; if ew = 1 then crash else ns := 1;
```

cf. ns/ew clipping -
extra detail: good or bad?

```
new nat ti
new nat ze
while X19(
  X20(time
  if button
    X21(zeb
    if X22(ti
      ns := 0
      if ew =
ns := 0; if
```

to do

- Stumping is useful only if
  - model#(X(v_1,..,v_n)) << model#(M)
- Conditionals over ints have big models
  - Predicate abstract them?
- Non-bool stumps have big models
  - Data abstract them?