Static Analysis with Goanna

Model checking for large code bases

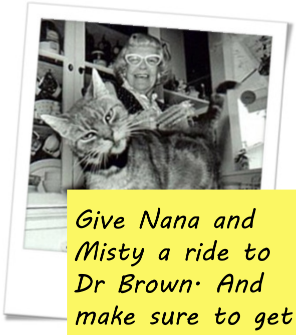
Ansgar Fehnker

About Us

- R&D spin-out **redlizards.com**
- 5 years technology research
- Funded and backed by NICTA

Mistakes are made

- Even good programmers make mistakes
- Bugs cost an average company US$50k per programmer per year (IDC)
- Finding bugs when testing up to 80 times more expensive than finding them when coding (IDC)



Give Nana and Misty a ride to Dr Brown. And make sure to get her vaccinated.

goanna NICTA

What We Do

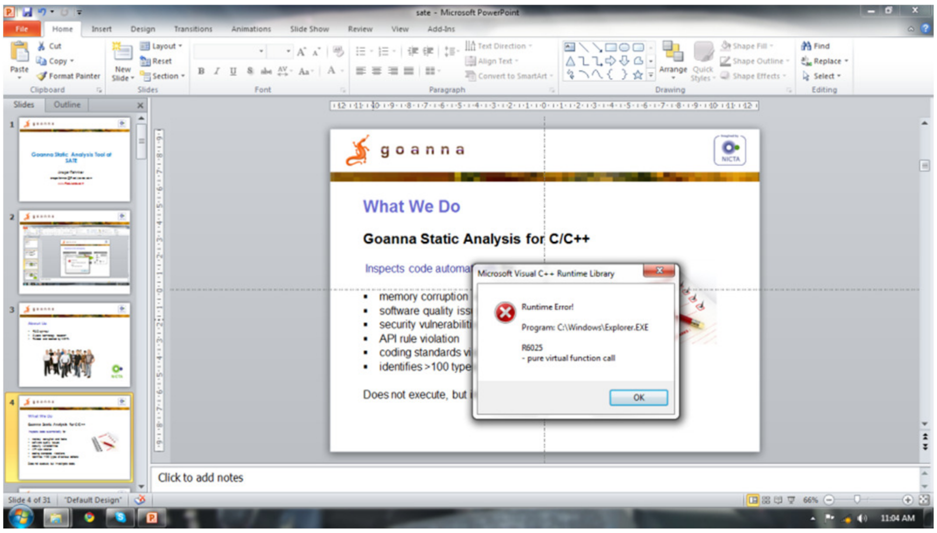
Goanna Static Analysis for C/C++

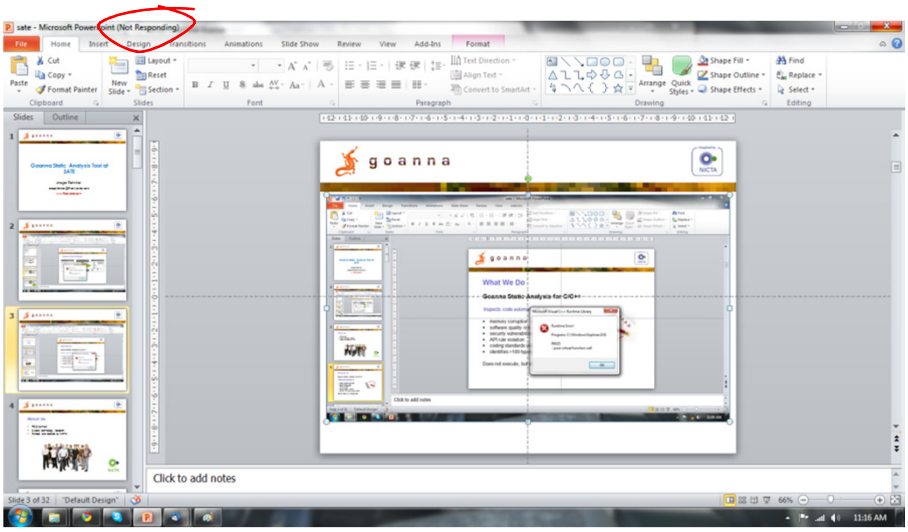
Inspects code automatically for

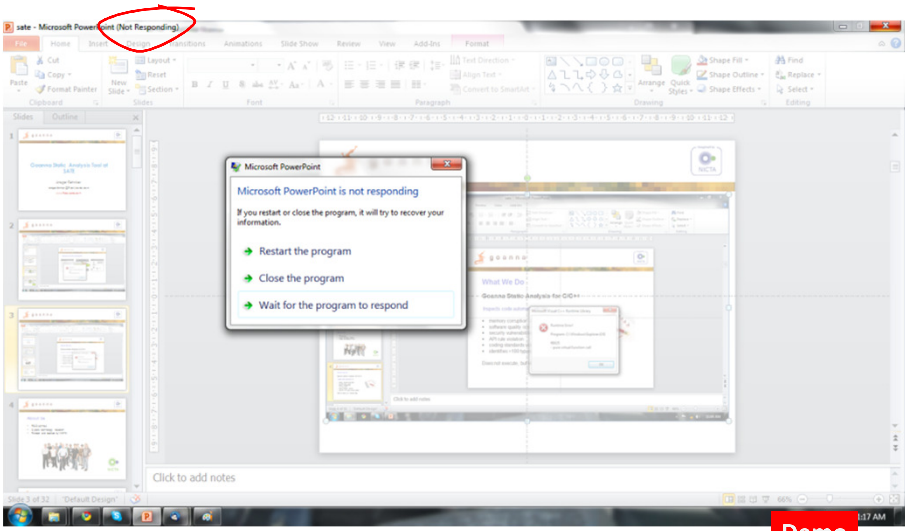
- memory corruption and leaks
- software quality issues
- security vulnerabilities
- API rule violation
- coding standards violations
- identifies >100 types of serious defects

Does not execute, but investigate code.

Demo


sate - Microsoft PowerPoint
What We Do
Goanna Static Analysis for C/C++
Inspects code automa
memory corruption
software quality iss
security vulnerabiliti
API rule violation
coding standards v
identifies >100 type
Does not execute, but
Microsoft Visual C++ Runtime Library
Runtime Error!
Program: C:\Windows\Explorer.EXE
R6025
- pure virtual function call
OK
Click to add notes
Slide 4 of 31
"Default Design"
11:04 AM


sate - Microsoft PowerPoint (Not Responding)
Click to add notes
Slide 3 of 32
"Default Design"
11:16 AM



Demo

Content

- Use Model Checking for Static Analysis of real code.
- Possible through the use of very coarse abstractions.
- Semantics added (only) if necessary
- Summaries for inter-procedural checking
- Able to find real bugs

goanna

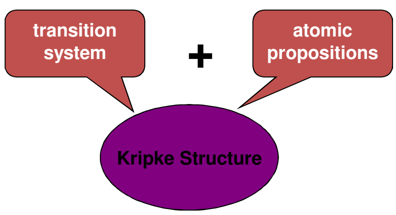
Under The Hood

Syntactic Model Checking

goanna

Syntactical Model Checking

- Discover syntactical structure of program by analysis of AST
- Map the syntactical structure of a program to a finite state model (Kripke Structure)
- Use temporal logic model checking to check for potential bugs and deficiencies in the code



goanna

Imagined by NICTA

Syntactical Model Checking

Source Code

```
int main(void) {
 int i,a=0;
 int *p = (int *)
     malloc(sizeof(int));
 for (i=1000; i > 0; i--){
     a = *p + i;
     i = i*2;
     …
```

goanna

Imagined by NICTA

Syntactical Model Checking

What happens?

Source Code

```
int main(void) {
 int i,a=0;
 int *p = (int *)
     malloc(sizeof(int));
 for (i=1000; i > 0; i--){
     a = *p + i;
     i = i*2;
     …
```

Syntactic Pattern

+

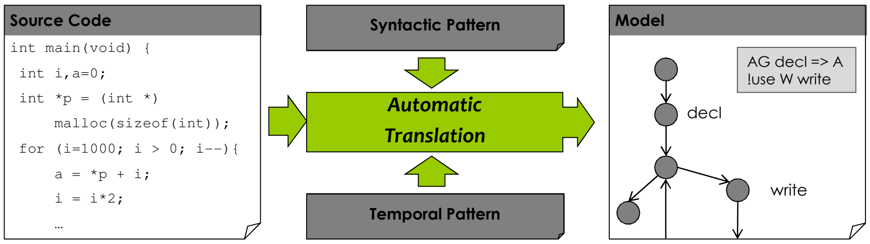
Temporal Pattern

When does it happen?

goanna

NICTA

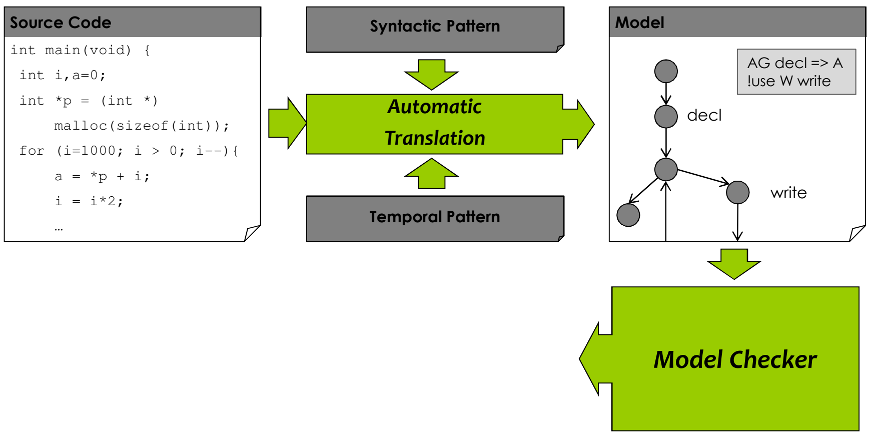
Syntactical Model Checking



goanna

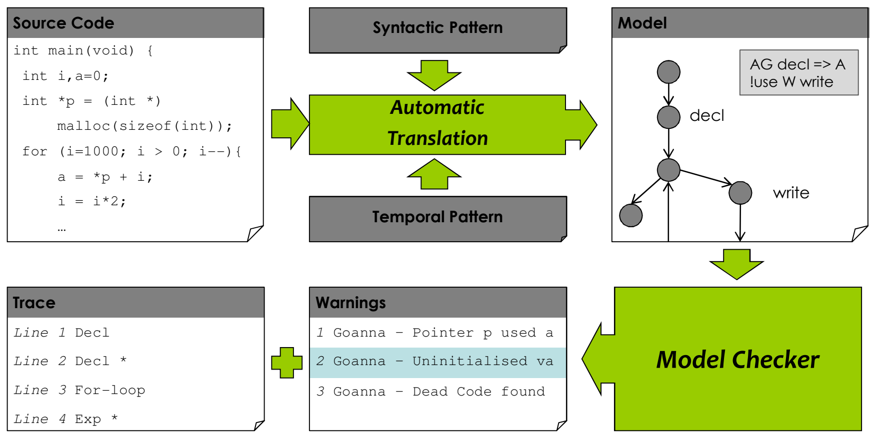
NICTA

Syntactical Model Checking



goanna NICTA

Syntactical Model Checking



goanna NICTA

Example: Uninitialized Variable

```
int foo(int n) {
 int x = 0, y = 1, q, i = 0;
 do {
      int oldy = y;
      y = x;
      q = x + oldy;
      x = q;
      i++;
  } while(i < n);
  return q;
}
```

goanna

Example: Uninitialized Variable

```
int foo(int n) {
 int x = 0, y = 1, q, i = 0;
 do {
      int oldy = y;
      y = x;
      q = x + oldy;
      x = q;
      i++;
  } while(i < n);
  return q;
}
```

Annotation
var_q

Annotation
write_q

Annotation
read_q

Annotation
read_q

goanna

Example: Uninitialized Variable

```
int foo(int n) {
 int x = 0, y = 1, q, i = 0;
 do {
      int oldy = y;
      y = x;
      q = x + oldy;
      x = q;
      i++;
  } while(i < n);
  return q;
}
```

Annotation
var_q

Annotation
write_q

Annotation
read_q

Annotation
read_q

Temporal Specification
Forall *var* **Never** *read* **Before** *write*

Example: Uninitialized Variable

```
int foo(int n) {
 int x = 0, y = 1, q, i = 0;
 do {
      int oldy = y;
      y = x;
      q = x + oldy;
      x = q;
      i++;
  } while(i < n);
  return q;
}
```

Annotation
var_q

Annotation
write_q

Annotation
read_q

Annotation
read_q

Temporal Specification

Forall *var* **Never** *read* **Before** *write*

Output

```
Goanna - analyzing file
Number of functions: 1
Total runtime : 0.01 second
```

Example: Uninitialized Variable

```
int foo(int n) {
 int x = 0, y = 1, q, i = 0;
 do {
      int oldy = y;
      y = x;
      q = x + oldy;
      x = q;
      i++;
  } while(i < n);
  return q;
}
```

Annotation
var_q

Annotation
write_q

Annotation
read_q

Annotation
read_q

Temporal Specification

Forall *var* **Never** *read* **Before** *write*

Output

```
Goanna - analyzing file
Number of functions: 1
Total runtime : 0.01 second
```


Note

Completely Automatic Analysis

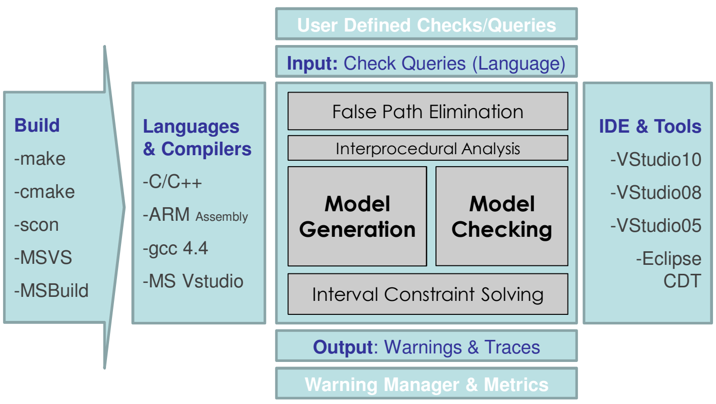
Syntactic Model Checking

- Uses a very coarse abstraction.
- Adds syntactic information as labels in Kripke structure
- Translates static analysis problems to CTL
- Uses model checking to analyse resulting model

Advantage: Very flexible

Challenge: Inter-procedural checking

Goanna Architecture



Demo

goanna

Ongoing Work

Inter-procedural Model Checking

goanna

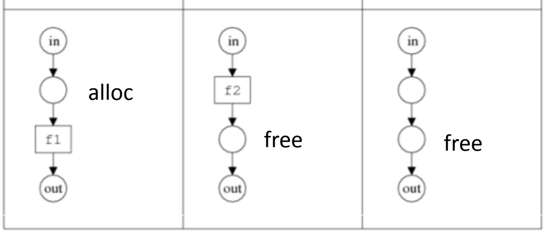
Problem

- Labels are distributed over functions
- Inlining not an option
  - Large models
  - Problem with recursion
  - Monolithic
- Example: double free

```
int main(){
  int *p=malloc();
  f1(p);
  return 1;
}
```

```
int f1(int *q){
  f2(q);
  free(q);
  return 1;
}
```

```
int f2 (int *r){
  *r =  4;
  free(r)
  return 1;
}
```



AG (free => not (EX E not alloc U free))

Solution

- Model programs as *Recursive State Machine*
- Use *3 valued logic* model checking to capture partial information
- Use *summaries* to avoid inlining

Advantages

- Local analysis
- Partial analysis
- Supports incremental analysis

Related Work

- Analysis of Recursive State Machines
  - By Rajeev Alur, Kousha Etessami, Mihalis Yannakakis.
- Model Checking Partial State Spaces with 3-Valued Temporal Logics
  - By Glenn Bruns and Patrice Godefroid
- Abstraction refinement for 3-valued-logic analysis
  - By Alexey Loginov, Thomas Reps, Mooly Sagiv
- Multi-valued symbolic model-checking
  - Marsha Chechik , Benet Devereux, Steve Easterbrook, Arie Gurfinkel

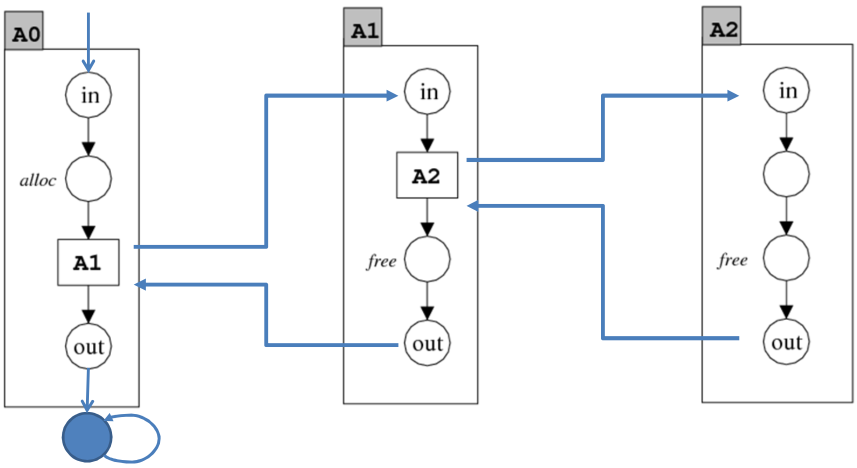
goanna

Recursive State Machine

- A collection of *state machines*
- Each state machine may contain states and "boxes"
- States are labelled.
- Each state machine has entry and exit states
- A box points to another state machine
- Can be recursive
- Semantics given by Kripke structure

goanna

Recursive State Machine



goanna

NICTA

3-valued Kleene Logic

- Extends binary logic with 3rd value: *M*
- Negation: not *T* = *F*, not *F* = *T*, not *M* = *M*
- x or y =
  - T if x = T or y = T,
  - F if x = F and y = F
  - M otherwise
- x join y =
  - T if x = T and y = T,
  - F if x = F and y = F
  - M otherwise

goanna

NICTA

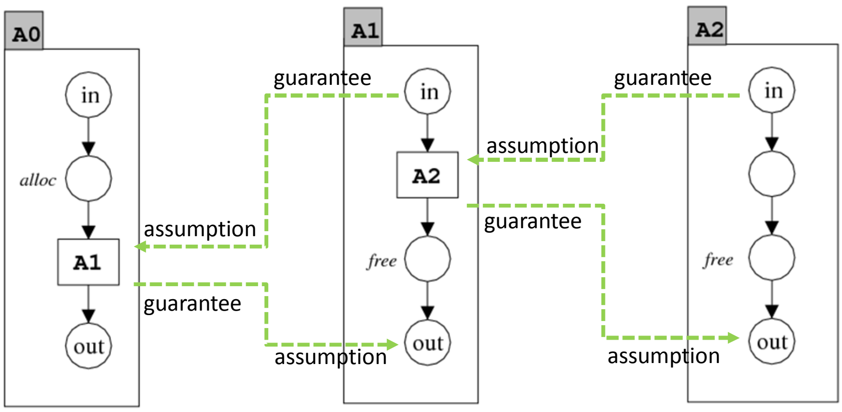
Summaries

- Given a property Φ of the form EG ϕ_1 or E ϕ_1 U ϕ_2
- External assumption
  - The external assumption refers to the caller of a sub-system
  - Says whether Φ is assumed to be T, F, or M when system returns
- A *summary* consists of
  - (internal) assumptions
  - (internal) guarantees

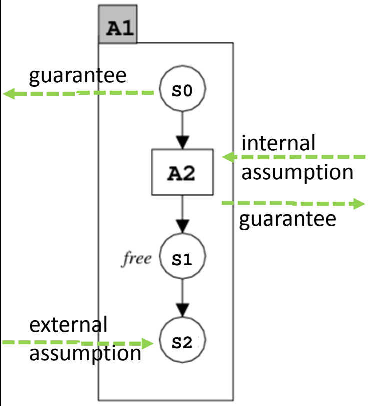
Summaries

- Internal assumption maps
  - External assumption to a mapping from boxes to T,F,M
- Internal guarantee maps
  - External assumption to a mapping from states to T,F,M
- Assumptions label boxes, guarantees label states
- An *assumption* refers to guarantees given by callees
- The *guarantee* to what a caller can assume about the system
- They state whether Φ is T, F, or M

Summaries



Example: Φ = E not alloc U free



Assumption A1			
if extA |= Φ	**T**	**M**	**F**
then A2 |=Φ	T	M	M

Guarantee A1			
if extA |=Φ	**T**	**M**	**F**
then S0|=Φ	M	M	M
then S1|=Φ	T	M	M
then S2|=Φ	T	M	M

goanna
NICTA

Coherence

- A summary is *coherent* if the *guarantees match the assumptions.*
- Matching means that for any external assumption extA
  - *a state s is labelled T if s |= Φ given the assumptions*
  - *a state s is labelled F if s |≠ Φ given the assumptions*
  - *and is labelled M otherwise*

g o a n n a

NICTA

Coherence

- A summary is *coherent* if the *guarantees match the assumptions.*
- Matching means that for any external assumption extA
  - *a state s is labelled T if s* $\models \Phi$ *given the assumptions*
  - *a state s is labelled F if s* $\not\models \Phi$ *given the assumptions*
  - *and is labelled M otherwise*
- This means for $\Phi = E\,\phi_1\; U\,\phi_2$ or $\Phi = EG\,\phi_1$
  - *s is labelled T if* $s \models \Phi$ or $E\,\phi_1\; U\;\mathbf{T}$

An assumption that Φ is true

g o a n n a

NICTA

Coherence

- A summary is *coherent* if the *guarantees match the assumptions.*
- Matching means that for any external assumption extA
  - *a state s is labelled T if s* $\models \Phi$ *given the assumptions*
  - *a state s is labelled F if s* $\not\models \Phi$ *given the assumptions*
  - *and is labelled M otherwise*
- This means for $\Phi = E\,\phi_1\; U\,\phi_2$ or $\Phi = EG\,\phi_1$
  - *s is labelled T if* $s \models \Phi$ or $E\,\phi_1\; U\;\mathbf{T}$
  - *s is labelled F if* $s \models$ not Φ and not $E\,\phi_1\; U\;\mathbf{T}$ and not $E\,\phi_1\; U\;\mathbf{M}$

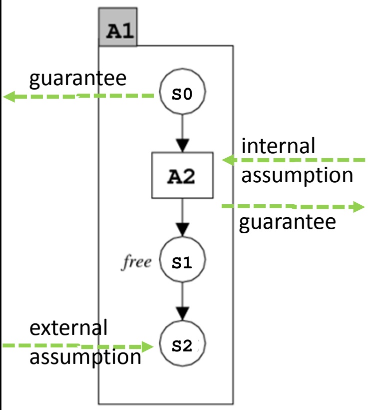
An assumption that Φ is true

An assumption that Φ might be true

Coherence

- A summary is *coherent* if the *guarantees match the assumptions.*
- Matching means that for any external assumption extA
  - *a state s is labelled T if s* $|= \Phi$ *given the assumptions*
  - *a state s is labelled F if s* $|\neq \Phi$ *given the assumptions*
  - *and is labelled M otherwise*
- This means for $\Phi = E\,\phi_1\ U\,\phi_2$ or $\Phi = EG\,\phi_1$
  - *s is labelled T if* $s \models \Phi$ or $E\,\phi_1\ U$ **T**
  - *s is labelled F if* $s \models$ not Φ and not $E\,\phi_1\ U$ **T** and not $E\,\phi_1\ U$ **M**
  - *and is labelled M otherwise*

Example: Φ = E not alloc U free

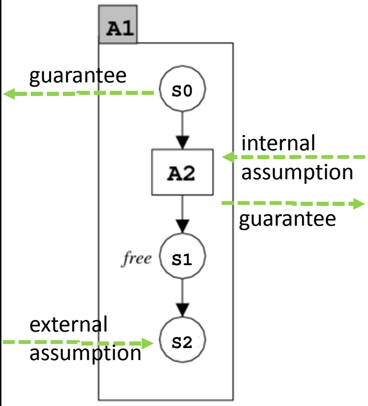


Assumption A1			
if extA |= Φ	**T**	**M**	**F**
then A2 |=Φ	T	M	M

If A2|=Φ then S0|=Φ

Guarantee A1			
if extA |=Φ	**T**	**M**	**F**
then S0|=Φ	M	M	M
then S1|=Φ	T	M	M
then S2|=Φ	T	M	M

Example: Φ = E not alloc U free



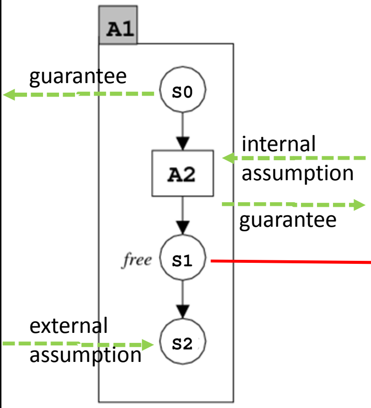
Assumption A1			
if extA |= Φ	T	M	F
then A2 |=Φ	T	M	M

Guarantee A1			
if extA |=Φ	T	M	F
then S0|=Φ	M	M	M
then S1|=Φ	T	M	M
then S2|=Φ	T	M	M

If ext A\|≠Φ then S2\|≠Φ

goanna NICTA

Example: Φ = E not alloc U free

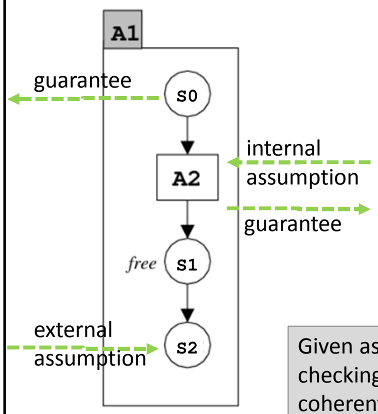


Assumption A1			
if extA |= Φ	T	M	F
then A2 |=Φ	T	M	M

Guarantee A1			
if extA |=Φ	T	M	F
then S0|=Φ	M	M	M
then S1|=Φ	T	M	M
then S2|=Φ	T	M	M

If S1\|= free then S1\|= Φ

Example: Φ = E not alloc U free



Assumption A1			
if extA $\models \Phi$	T	M	F
then A2 $\models \Phi$	T	M	M

Guarantee A1			
if extA $\models \Phi$	T	M	F
then S0 $\models \Phi$	T	M	M
then S1 $\models \Phi$	T	T	T
then S2 $\models \Phi$	T	M	F

Given assumptions, use CTL model checking to make guarantees coherent.

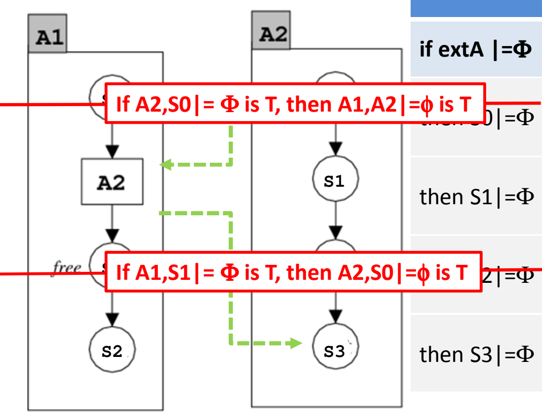
goanna NICTA

Consistency

- The summaries for all sub-structures are *consistent* if the *assumptions match the guarantees*
- Matching means that
  - *given the labelling in the immediate successors of a "box",*
  - *the labelling of the box (assumption) coincides with that of the guarantee of the callee.*

Example: Φ = E not alloc U free

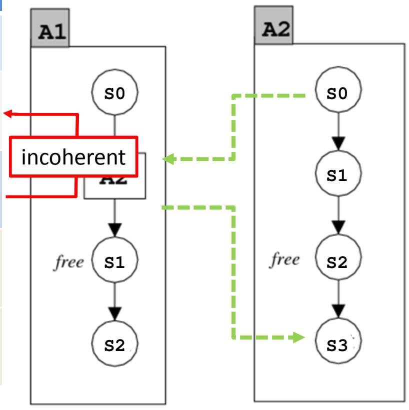
Summary A1			
if extA |= Φ	**T**	**M**	**F**
then S0|=Φ	T	M	M
then A2 |=Φ	T	M	M
then S1|=Φ	T	T	T
then S2|=Φ	T	M	F



Guarantee A2			
if extA |=Φ	**T**	**M**	**F**
then S0|=Φ	T	T	T
then S1|=Φ	T	T	T
then S2|=Φ	T	T	T
then S3|=Φ	T	M	F

Example: Φ = E not alloc U free

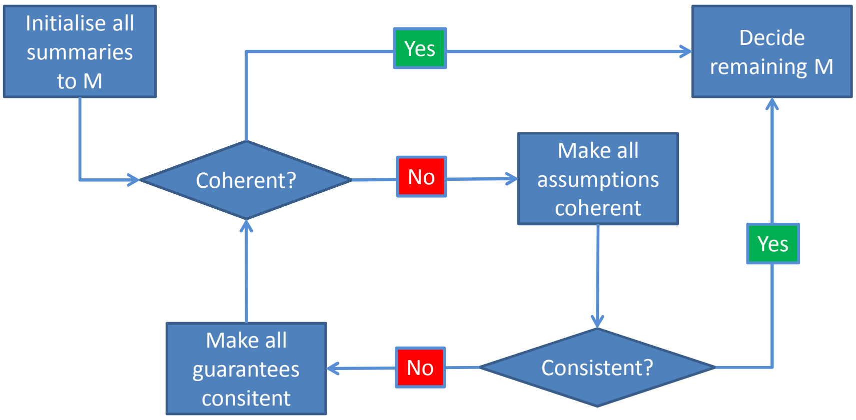
Summary A1			
if extA |= Φ	**T**	**M**	**F**
then S0|=Φ	T	M	M
then A2 |=Φ	T	T	T
then S1|=Φ	T	T	T
then S2|=Φ	T	M	F



Guarantee A2			
if extA |=Φ	**T**	**M**	**F**
then S0|=Φ	T	T	T
then S1|=Φ	T	T	T
then S2|=Φ	T	T	T
then S3|=Φ	T	M	F

goanna

Iterate



goanna

Deciding Remaining M

Theorem

If all summaries are coherent and consistent then

Case: Φ = EG ϕ_1

- If a state s is labelled M then s|= Φ

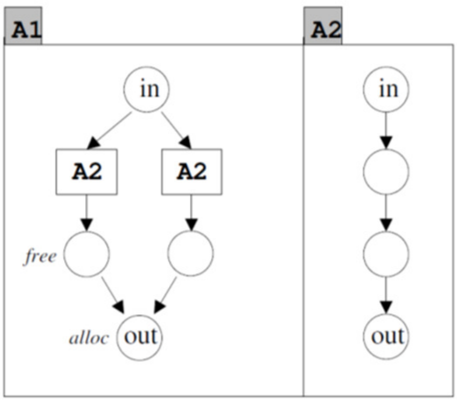
Case: Φ = E ϕ_1 U ϕ_2

- If a state s is labelled M then s|≠ Φ

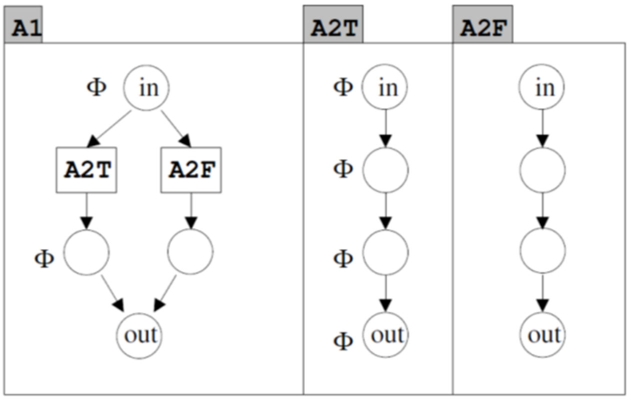
Iteration over sub-formulae

- Structural induction over CTL in ENF
    - Φ = p | not ϕ_1 | ϕ_1 or ϕ_2 | EX ϕ_1 | EG ϕ_1 | E ϕ_1 U ϕ_2
- Φ holds for the system if the initial state of the initial sub-system is labelled Φ
- Labelling with temporal operators EX, EG, EG might require to create a copy of a sub-system

Example: Φ = E not alloc U free



Example: Φ = E not alloc U free



Summary

- CTL model checking algorithm for recursive state machines
- Linear for each sub-formulae
- Exponential in the number of sub-formulae
- Uses sub-system summaries
  - Sub-systems are checked one-by-one
  - 3 valued summaries allow for partial evaluation
  - Coherence and consistency allow for incremental evaluation

goanna

NICTA

Back to Reality

Goanna Results in Practice

goanna

NICTA

Some SATE Results

Static Analysis Tool Exposition (NIST)

- NIST selected 5 code bases for analysis
- NIST selected known CVEs to be found
- NIST selected random warnings for manual evaluation

Goanna SATE participation

- We used the default checks (55 checks)
- Geared towards quality issues, omitted checks for “cosmetic issues”
- “Sanity” assumption

goanna

NICTA

Some SATE Results

Number of Warnings

Chrome 375.54	*1079*
Chrome 375.70	*1173*
Dovecot	*180*
Wireshark 2.0	*534*
Wireshark 2.9	*532*

Top10

PTR-null-pos-assign	952
RED-cmp-never	788
RED-cmp-always	448
PTR-null-cmp-aft	328
SPC-uninit-var-some	189
PTR-null-assign-fun-pos	144
RED-unused-val-ptr	124
PTR-param-unchk-some	94
RED-unused-var-all	67
RED-case-reach	46

PTR: Pointer misuse
RED: Redundant code
SPC: Unspecified behavior

Demo

goanna

NICTA

SEM-const-call

unichar.c:193: warning: Goanna[SEM-const-call] Non-const function `uint16_find' is called in const function `uni_ucs4_to_titlecase'

```
unichar_t uni_ucs4_to_titlecase(unichar_t chr)
{ (…)
  if (!uint16_find(titlecase16_keys,
      N_ELEMENTS(titlecase16_keys), chr, &idx))
        return chr; (…)
```

- Semantic attributes are a GNU language extension
- *uni_ucs4_to_titlecase* has `__attribute__ ((const))` (see *unichar.h*)
- *uint16_find* has not
- GNU says: *"(...) a function that calls a non-const function usually must not be const"*

goanna

RED-cmp-never

director-connection.c:655: warning: Goanna[RED-cmp-never] Comparison never holds

```
if (str_array_length(args) != 2 ||
    director_args_parse_ip_port(conn, args, &ip, &port) < 0) {
      i_error("director(%s): Invalid CONNECT args", conn->name);
      return FALSE;
}
```

- *director_args_parse_ip_port()* only returns *TRUE or FALSE*
- *director_args_parse_ip_port()<0* never true
- *ip* and *port* might not be assigned, but this failure is not detected

goanna

PTR-null-assign-fun-pos

director.c 180: warning: Goanna[PTR-null-assign-fun-pos] Dereference of `preferred_host' which may be NULL

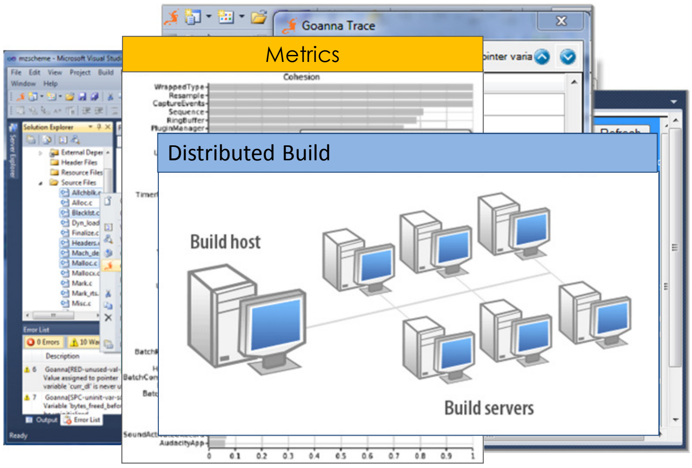
```
static void director_reconnect_timeout(struct director *dir){
  struct director_host *cur_host,
         *preferred_host = director_get_preferred_right_host(dir);
                 (…)
  if (cur_host != preferred_host)
         (void)director_connect_host(dir, preferred_host);
  else {(…)}
}
```

- *director_get_preferred_right_host* might NULL
- *director_connect_host dereferences preferred_host*
- Potential NULL deref

goanna

NICTA

Other Things



goanna

NICTA

Summary

- Goanna is a static analysis solution for C/C++
- Desktop and server version available at redlizards.com
- It uses a combination of model checking and static analysis to find serious bugs in real code
- It finds serious bugs in real code
- It is named after a bug-eating lizard

goanna

NICTA

Goanna at work

Next 500 MLoC

goanna

NICTA