An overview of SLAM

Albert Oliveras, Enric Rodríguez-Carbonell

Deduction and Verification Techniques
Session 5
Fall 2009, Barcelona

Overview of the session

- The SLAM loop
- SLAM components:
  - C2BP
  - Bebop
  - Newton
  - Constrain

Overview of SLAM

SLAM loop (for proving safety properties) [CEGAR loop]:

1. Run C2BP to construct Boolean Program BP
   [Predicate abstraction]
2. Run Bebop on BP. If no counterexample OK
   [Boolean techniques]
3. If counterexample run Newton to check feasibility.
   If feasible, then BUG
4. If infeasible, infer new predicates to rule out counterexample
   [Interpolants?]
5. If new predicates found, goto 1
6. Else, call Constrain and goto 2
   [Theorem proving]

Obvious remark: it may not terminate

Running example

Proving that the following program is correct

$\{Pre\text{:}\ x = 5 \wedge y > x\ \}$

x:= y;

$\{Post\text{:}\ x \neq 5\ \}$

is equivalent to proving that ERROR is not reachable in:

assume($x = 5 \wedge y > x$);

x:=y;

if x=5
  ERROR:;

Overview of the session

- The SLAM loop
- SLAM components:
  - C2BP
  - Bebop
  - Newton
  - Constrain

First step: C2BP (C to Boolean Program)

Underlying idea: PREDICATE ABSTRACTION

- Consider a C program with integer variables x, y, z.

First step: C2BP (C to Boolean Program)

Underlying idea: PREDICATE ABSTRACTION

- Consider a C program with integer variables x, y, z.
- Assume one is given the predicates
  $P = [x > 2, x + y < 1, z = x]$

First step: C2BP (C to Boolean Program)

Underlying idea: PREDICATE ABSTRACTION

- Consider a C program with integer variables x, y, z.
- Assume one is given the predicates
  $P = [x > 2, x + y < 1, z = x]$
CONCRETE STATE	ABSTRACT STATE
$\{x = 5, y = 2, z = -1\}$	$\{1, 0, 0\}$

First step: C2BP (C to Boolean Program)

Underlying idea: PREDICATE ABSTRACTION

- Consider a C program with integer variables x, y, z.
- Assume one is given the predicates
  $P = [x > 2, x + y < 1, z = x]$
- CONCRETE STATE ABSTRACT STATE
  $\{x = 5, y = 2, z = -1\}$ $\{1, 0, 0\}$
- Missing point: given the statement
  $x := x + 1;$
  how do the states change, i.e. which is the transition relation?

First step: C2BP (C to Boolean Program)

Underlying idea: PREDICATE ABSTRACTION

- Consider a C program with integer variables x, y, z.
- Assume one is given the predicates
  $P = [x > 2, x + y < 1, z = x]$
- CONCRETE STATE ABSTRACT STATE
  $\{x = 5, y = 2, z = -1\}$ $\{1, 0, 0\}$
- Missing point: given the statement
  $x := x + 1;$
  how do the states change, i.e. which is the transition relation?
- CONCRETE STATE ABSTRACT STATE
  $\{x = 6, y = 2, z = -1\}$ $\{?, ?, ?\}$

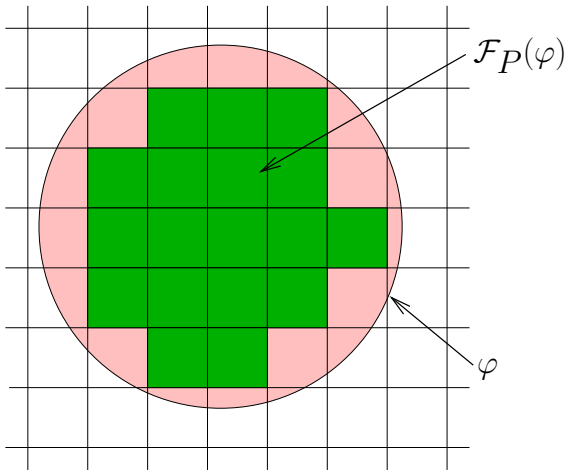
Predicate abstraction - Key operation

PREDICATE ABSTRACTION-KEY OPERATION:

- INPUT:
  - A theory T
  - A formula φ (representing, e.g., a set of concrete states)
  - A set of predicates $P = \{P_1, \ldots, P_n\}$ describing some set of properties of the system state
- OUTPUT: the most precise T-approximation of φ using P

This amounts to compute either

- $\mathcal{F}_P(\varphi)$: the weakest Boolean expression over P that T-implies φ, or
- $\mathcal{G}_P(\varphi)$: the strongest Boolean expression over P T-implied by φ



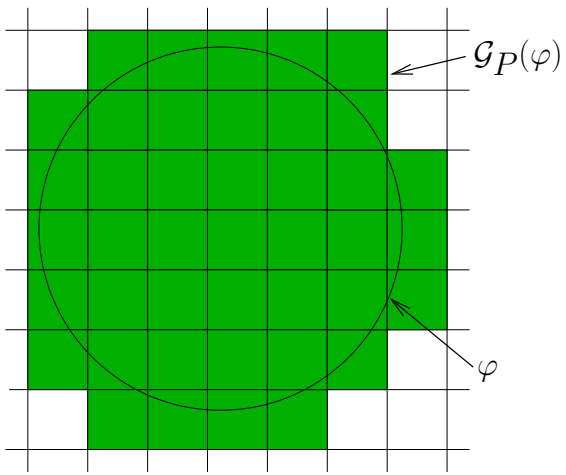
Predicate abstraction - Key operation

PREDICATE ABSTRACTION-KEY OPERATION:

- INPUT:
  - A theory T
  - A formula φ (representing, e.g., a set of concrete states)
  - A set of predicates $P = \{P_1, \ldots, P_n\}$ describing some set of properties of the system state
- OUTPUT: the most precise T-approximation of φ using P

This amounts to compute either

- $\mathcal{F}_P(\varphi)$: the weakest Boolean expression over P that T-implies φ, or
- $\mathcal{G}_P(\varphi)$: the strongest Boolean expression over P T-implied by φ



Predicate abstraction - Computation

Some notation:

- A cube is a conjunction of literals of P.
- A minterm is a cube of size $|P|$ with exactly one of P_i or $\neg P_i$.

The computation of $\mathcal{F}_P(\varphi)$ and $\mathcal{G}_P(\varphi)$ is given by:

- $\mathcal{F}_P(\varphi)$ is $\bigvee\{c \mid c$ is a minterm over P and $c \models_T \varphi\}$,
- $\mathcal{G}_P(\varphi)$ is $\neg\mathcal{F}_P(\neg\varphi)$.
- $\mathcal{G}_P(\varphi)$ is $\bigvee\{c \mid c$ is a minterm over P and $c \wedge \varphi$ is T-satisfiable$\}$,

ALGORITHM:

Check, for each minterm c, whether $c \wedge \varphi$ is T-satisfiable.

First step: C2BP (2)

INPUT:

- C program with a set of ERROR locations
  - loops are transformed to if + goto
  - procedure calls are allowed
  - pointers are allowed
- A set of predicates $P = [p_1, \ldots, p_n]$.

OUTPUT:

- A boolean program BP such that
  - the only variables are $b_1, \ldots, b_n$
  - maintains the flow structure of the C program
  - if ERROR is not reachable in BP, it is not reachable in the C program either

First step: C2BP (3)

We will make use of the function:

```
bool choose (bool b1, bool b2){
    if (b1) return true;
    if (b2) return false;
    return undeterministic_choise();
}
```

First step: C2BP (4)

Assume one is given the set of predicates $P = [x > 5, x < 5, y = 5]$ and the C program:

assume($x = 5 \wedge y > x$);
x:=y;
if x=5
  ERROR:;

The BP obtained is the following:

assume($\mathcal{G}_P(x = 5 \wedge y > x)$);
$< b_1, b_2, b_3 >:=<$ choose($\mathcal{F}_P(WP(x{:=}y, x{>}5))$, $\mathcal{F}_P(WP(x{:=}y, x{<=}5))$),
choose($\mathcal{F}_P(WP(x{:=}y, x{<}5))$, $\mathcal{F}_P(WP(x{:=}y, x{>=}5))$),
choose($\mathcal{F}_P(WP(x{:=}y, y{=}5))$, $\mathcal{F}_P(WP(x{:=}y, y{\neq}5))$) >
if (*){
  assume($\mathcal{G}_P(x = 5)$);
  ERROR:; }

First step: C2BP(5)

Assume one is given the set of predicates $P = [x > 5, x < 5, y = 5]$ and the C program:

```
assume(x = 5 ∧ y > x);
x:=y;
if x=5
   ERROR:;
```

The BP obtained is the following:

```
assume(¬b1 ∧ ¬b2 ∧ ¬b3);
< b1, b2, b3 >:=< choose(false, b3),
                  choose(false, b3),
                  choose(b3, ¬b3) >
if (*){
   assume(¬b1 ∧ ¬b2);
   ERROR:; }
```

Overview of the session

- The SLAM loop
- SLAM components:
  - C2BP
  - Bebop
  - Newton
  - Constrain

Second step: BEBOP - Model check BP

QUESTION: Is ERROR reachable in the boolean program?

assume($\neg b_1 \wedge \neg b_2 \wedge \neg b_3$);

$< b_1, b_2, b_3 >:=<$ choose(false, b_3),
choose(false, b_3),
choose($b_3, \neg b_3$) $>$

if (*){
assume($\neg b_1 \wedge \neg b_2$);
ERROR:; }

Second step: BEBOP - Model check BP

QUESTION: Is ERROR reachable in the boolean program?

assume($\neg b_1 \wedge \neg b_2 \wedge \neg b_3$);
$< b_1, b_2, b_3 >:=<$ choose(false, b_3),
choose(false, b_3),
choose($b_3, \neg b_3$) $>$
if (*){
assume($\neg b_1 \wedge \neg b_2$);
ERROR:; }

ANSWER: YES!!!!!

- Any program execution assigns b_3 to false, but arbitrary values to b_1 and b_2.
- Hence, a trace in which both b_1 and b_2 are assigned to *false* is possible and leads to ERROR

Overview of the session

- The SLAM loop
- SLAM components:
  - C2BP
  - Bebop
  - Newton
  - Constrain

Third step: Newton - path feasibility

Let's consider the path to ERROR in the original C program:

		VAR	VAL	CONSTR.
1.	$x{:=}x_\theta$	x	x_θ	

Third step: Newton - path feasibility

Let's consider the path to ERROR in the original C program:

		VAR	VAL	CONSTR.
1.	$x:=x_\theta$	x	x_θ	
2.	$y:=y_\theta$	y	y_θ	

Third step: Newton - path feasibility

Let's consider the path to ERROR in the original C program:

		VAR	VAL	CONSTR.
1.	x:=x_θ	x	x_θ	
2.	y:=y_θ	y	y_θ	
3.	assume(x=5)			$x_\theta = 5$

Third step: Newton - path feasibility

Let's consider the path to ERROR in the original C program:

		VAR	VAL	CONSTR.
1.	x:=x_θ	x	x_θ	
2.	y:=y_θ	y	y_θ	
3.	assume(x=5)			$x_\theta = 5$
4.	assume(y>x)			$y_\theta > x_\theta$

Third step: Newton - path feasibility

Let's consider the path to ERROR in the original C program:

		VAR	VAL	CONSTR.
1.	x:=x_θ	x	x_θ	
2.	y:=y_θ	y	y_θ	
3.	assume(x=5)			$x_\theta = 5$
4.	assume(y>x)			$y_\theta > x_\theta$
5.	x:=y;	x	y_θ	

Third step: Newton - path feasibility

Let's consider the path to ERROR in the original C program:

		VAR	VAL	CONSTR.
1.	x:=x_θ	x	x_θ	
2.	y:=y_θ	y	y_θ	
3.	assume(x=5)			$x_\theta = 5$
4.	assume(y>x)			$y_\theta > x_\theta$
5.	x:=y;	x	y_θ	
6.	assume(x=5)			$y_\theta = 5$

Third step: Newton - path feasibility

Let's consider the path to ERROR in the original C program:

		VAR	VAL	CONSTR.
1.	x:=x_θ	x	x_θ	
2.	y:=y_θ	y	y_θ	
3.	assume(x=5)			$x_\theta = 5$
4.	assume(y>x)			$y_\theta > x_\theta$
5.	x:=y;	x	y_θ	
6.	assume(x=5)			$y_\theta = 5$

NEWTON ACHIEVES TWO GOALS:

- Detects that the set of constrains is inconsistent (abstract path is infeasible)
- Generates the predicates $\{x = 5, y > x, y = 5\}$

First step again: C2BP

- Now the set of predicates is the original one
$$[x > 5, x < 5, y = 5]$$
plus the newly generated ones
$$[x = 5\,, y > x,\ y = 5].$$

First step again: C2BP

- Now the set of predicates is the original one
$$[x > 5, x < 5, y = 5]$$
plus the newly generated ones
$$[x = 5\,, y > x,\ y = 5].$$

- QUESTION: Do they rule out all the previous path?

First step again: C2BP

- Now the set of predicates is the original one
$$[x > 5, x < 5, y = 5]$$
plus the newly generated ones
$$[x = 5\,, y > x,\ y = 5].$$

- QUESTION: Do they rule out all the previous path?

- ANSWER: YES

First step again: C2BP

- Now the set of predicates is the original one
$$[x > 5, x < 5, y = 5]$$
plus the newly generated ones
$$[x = 5\,, y > x,\ y = 5].$$

- QUESTION: Do they rule out all the previous path?
- ANSWER: YES
- QUESTION: Are all of them necessary?

First step again: C2BP

- Now the set of predicates is the original one
$$[x > 5, x < 5, y = 5]$$
plus the newly generated ones
$$[x = 5\,, y > x,\ y = 5].$$

- QUESTION: Do they rule out all the previous path?
- ANSWER: YES

- QUESTION: Are all of them necessary?
- ANSWER: NO

Step 1 again: C2BP(2)

The abstraction w.r.t. the new predicates $P = [x = 5, y > x, y = 5]$ is

$$
\begin{array}{l}
\text{assume}(b_1 \wedge b_2 \wedge \neg b_3); \\
< b_1, b_2, b_3 > := < \text{choose}(b_3, \neg b_3), \\
\qquad\qquad\qquad\quad \text{choose}(\text{false}, \text{true}), \\
\qquad\qquad\qquad\quad \text{choose}(b_3, \neg b_3) > \\
\text{if } (*)\{ \\
\quad \text{assume}(b_1); \\
\quad \text{ERROR::} \}
\end{array}
$$

- Now, cleary ERROR is not reachable.
- Hence, not all predicates are needed to prove property!
- BLAST approach is better in this sense

Possible problems: No Difference Found

- Sometimes Newton is not able to generate new predicates

Possible problems: No Difference Found

- Sometimes Newton is not able to generate new predicates
- At that point, iterative refinement cannot make any more progress

Possible problems: No Difference Found

- Sometimes Newton is not able to generate new predicates
- At that point, iterative refinement cannot make any more progress
- What is the reason?

Possible problems: No Difference Found

- Sometimes Newton is not able to generate new predicates
- At that point, iterative refinement cannot make any more progress
- What is the reason?
- The set of predicates may suffice to rule out infeasible abstract path, but
  - Computation of $\mathcal{G}_P(\varphi)$ or $\mathcal{F}_P(\varphi)$ might be inaccurate
  - SLAM abstract transition relation computation is not exact (both in practice and in theory)

Cartesian approx. vs. exact computation

SLAM Boolean abstraction of the C program was the following:

$$
\begin{array}{l}
\text{assume}(\neg b_1 \wedge \neg b_2 \wedge \neg b_3); \\
< b_1, b_2, b_3 > := < \text{choose}(\text{false}, b_3), \\
\qquad\qquad\qquad\quad \text{choose}(\text{false}, b_3), \\
\qquad\qquad\qquad\quad \text{choose}(b_3, \neg b_3) > \\
\text{if } (*)\{ \\
\quad \text{assume}(\neg b_1 \wedge \neg b_2); \\
\quad \text{ERROR:;} \}
\end{array}
$$

- Which are the reachable abstract states before reaching **if**?
- Reachable states are:
  - $\{0,0,0\}$ (ERROR!)
  - $\{0,1,0\}$
  - $\{1,0,0\}$
  - $\{1,1,0\}$

Cartesian approx. vs. exact comput. (3)

Let's compute the reachable abstract states in a different way:

assume($x = 5 \wedge y > x$);

x:=y;

if x=5
  ERROR:;

Reachable concrete states before **if** can be described as

$$\{(x_1, y_0) | x_0 = 5 \wedge y_0 > x_0 \wedge x_1 = y_0\}$$

If we know take $P = \{x_1 < 5,\ x_1 > 5,\ y_0 = 5\}$ we can compute the reachable abstract states as

$$\mathcal{G}_P(x_0 = 5 \wedge y_0 > x_0 \wedge x_1 = y_0)$$

which is exactly $\{0, 1, 0\}$.
Hence, the abstract bad state $\{0, 0, 0\}$ is indeed NOT reachable.

Cartesian approx. vs. exact comput. (2)

- Cartesian approximation is not 100% precise
- However, it seems very good in practice(cheap and quite precise)
- In some cases, due to the loss of precision, Newton cannot make progress
- In those situations, Constrain is called

Overview of the session

- The SLAM loop
- SLAM components:
  - C2BP
  - Bebop
  - Newton
  - Constrain

Fourth step: Constrain - Tr. relat. refin.

INPUT:

- C program
- Boolean abstraction
- Trace through the Boolean abstraction (valuation of predicates at each step)

OUTPUT:

- Set of constrains to be added to the Boolean abstraction

IDEA:

Transition from abstract state A_i to A_{i+1} using instr. i spurious if

$$\gamma(A_i) \rightarrow \neg WP(I_i, \gamma(A_{i+1}))$$

is valid (where γ is the concretization function).

Fourth step: Constrain (2)

Let's revisit our example (predicates were $P = [x > 5, x < 5, y = 5]$):

C program

I_1 : assume($x = 5 \wedge y > x$);

I_2 x:=y;

I_3 **if** x=5
ERROR:;

Boolean trace

$A_0 = \{0, 0, 0\}$
(I_1)
$A_1 = \{0, 0, 0\}$
(I_2)
$A_2 = \{0, 0, 0\}$
(I_3) *//assume (x=5)*
$A_3 = \{0, 0, 0\}$

Fourth step: Constrain (2)

Let's revisit our example (predicates were $P = [x > 5, x < 5, y = 5]$):

C program	**Boolean trace**
	$A_0 = \{0, 0, 0\}$
I_1 : assume($x = 5 \wedge y > x$);	(I_1)
	$A_1 = \{0, 0, 0\}$
I_2 x:=y;	(I_2)
	$A_2 = \{0, 0, 0\}$
I_3 **if** x=5	(I_3) *//assume (x=5)*
ERROR:;	
	$A_3 = \{0, 0, 0\}$

$$\gamma(A_0) \rightarrow \neg WP(I_1, \gamma(A_1)) \text{ is valid?}$$

$x = 5 \wedge y \neq 5 \rightarrow \neg WP(assu.(x = 5 \wedge y > x), x = 5 \wedge y \neq 5)$ is valid?

$x = 5 \wedge y \neq 5 \rightarrow \neg(x = 5 \wedge y \neq 5)$ valid? NO! ($A_0 \rightsquigarrow A_1$ not spur.).

Fourth step: Constrain (2)

Let's revisit our example (predicates were $P = [x > 5, x < 5, y = 5]$):

C program

I_1 : assume($x = 5 \wedge y > x$);

I_2 x:=y;

I_3 **if** x=5
ERROR:;

Boolean trace

$A_0 = \{0, 0, 0\}$
(I_1)
$A_1 = \{0, 0, 0\}$
(I_2)
$A_2 = \{0, 0, 0\}$
(I_3) *//assume (x=5)*
$A_3 = \{0, 0, 0\}$

$$\gamma(A_1) \rightarrow \neg WP(I_2, \gamma(A_2)) \text{ is valid?}$$

$x = 5 \wedge y \neq 5 \rightarrow \neg WP(x := y, x = 5 \wedge y \neq 5)$ is valid?

$x = 5 \wedge y \neq 5 \rightarrow \neg(y = 5 \wedge y \neq 5)$ valid? YES! ($A_1 \rightarrow A_2$ is spur.).
BEBOP will get the constrain $\neg(\neg b1 \wedge \neg b_2 \wedge \neg b_3 \wedge \neg b'_1 \wedge \neg b'_2 \wedge \neg b'_3)$

Fourth step: Constrain (2)

Let's revisit our example (predicates were $P = [x > 5, x < 5, y = 5]$):

C program

I_1 : assume($x = 5 \wedge y > x$);

I_2 x:=y;

I_3 **if** x=5
ERROR:;

Boolean trace

$A_0 = \{0, 0, 0\}$
(I_1)
$A_1 = \{0, 0, 0\}$
(I_2)
$A_2 = \{0, 0, 0\}$
(I_3) *//assume (x=5)*
$A_3 = \{0, 0, 0\}$

$$\gamma(A_2) \rightarrow \neg WP(I_3, \gamma(A_3)) \text{ is valid?}$$

$x = 5 \wedge y \neq 5 \rightarrow \neg WP(assu.(x = 5 \wedge y > x), x = 5 \wedge y \neq 5)$ is valid?

$x = 5 \wedge y \neq 5 \rightarrow \neg(x = 5 \wedge y \neq 5)$ valid? NO! ($A_2 \rightarrow A_3$ not spur.).

BEBOP again

After Constrain, BEBOP will get the Boolean program:

assume($\neg b_1 \wedge \neg b_2 \wedge \neg b_3$);
$< b_1, b_2, b_3 >:=<$ choose(false, b_3),
choose(false, b_3),
choose($b_3, \neg b_3$) $>$ //with constrain
$\neg(\{0,0,0\} \rightarrow \{0,0,0\})$
if (*){
assume($\neg b_1 \wedge \neg b_2$);
ERROR:; }

- Now clearly ERROR is not reachable
- We have proven the program correct WITHOUT adding new predicates

Overview of SLAM

SLAM loop (for proving safety properties):

1. Run C2BP to construct Boolean Program BP
   [Predicate abstraction]
2. Run Bebop on BP. If no counterexample OK
   [Boolean techniques]
3. If counterexample run Newton to check feasibility.
   If feasible, then BUG
4. If infeasible, infer new predicates to rule out counterexample
   [Interpolants?]
5. If new predicates found, goto 1
6. Else, call Constrain and goto 2
   [Theorem proving]

Obvious remark: it may not terminate

Bibliography - Some further reading

All papers can be found at:

- `http://research.microsoft.com/en-us/projects/slam/`