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
Proving that the following program is correct

\{ \text{Pre}: x = 5 \land y > x \} 
\begin{align*}
\text{x:= y;}
\end{align*}
\{ \text{Post}: x \neq 5 \}

is equivalent to proving that \text{ERROR} is not reachable in:

\begin{align*}
\text{assume}(x = 5 \land y > x); \\
\text{x:=y;}
\text{if } x=5 \\
\text{ERROR::*;}
\end{align*}
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
  \[ P = [x > 2, x + y < 1, z = x] \]

<table>
<thead>
<tr>
<th>CONCRETE STATE</th>
<th>ABSTRACT STATE</th>
</tr>
</thead>
<tbody>
<tr>
<td>{x = 5, y = 2, z = -1}</td>
<td>{1, 0, 0}</td>
</tr>
</tbody>
</table>
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
  \[ P = [x > 2, x + y < 1, z = x] \]

<table>
<thead>
<tr>
<th>CONCRETE STATE</th>
<th>ABSTRACT STATE</th>
</tr>
</thead>
<tbody>
<tr>
<td>${x = 5, y = 2, z = -1}$</td>
<td>${1, 0, 0}$</td>
</tr>
</tbody>
</table>

**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
  \[ P = [x > 2, x + y < 1, z = x] \]

<table>
<thead>
<tr>
<th>CONCRETE STATE</th>
<th>ABSTRACT STATE</th>
</tr>
</thead>
<tbody>
<tr>
<td>{x = 5, y = 2, z = −1}</td>
<td>{1, 0, 0}</td>
</tr>
</tbody>
</table>

**Missing point:** given the statement
\[ x := x + 1; \]
how do the states change, i.e. which is the transition relation?

<table>
<thead>
<tr>
<th>CONCRETE STATE</th>
<th>ABSTRACT STATE</th>
</tr>
</thead>
<tbody>
<tr>
<td>{x = 6, y = 2, z = −1}</td>
<td>{?, ?, ?}</td>
</tr>
</tbody>
</table>
Predicate abstraction - Key operation

**PREDICATE ABSTRACTION-KEY OPERATION:**

- **INPUT:**
  - A theory $T$
  - A formula $\varphi$ (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 $\varphi$ using $P$

This amounts to compute either

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

**PREDICATE ABSTRACTION-KEY OPERATION:**

- **INPUT:**
  - A theory $T$
  - A formula $\varphi$ (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 $\varphi$ using $P$

This amounts to compute either

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

**ALGORITHM:**

Check, for each minterm $c$, whether $c \land \varphi$ 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:

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

The BP obtained is the following:

```
assume($G_P(x = 5 ∧ y > x))$;
$< b_1, b_2, b_3 > := < choose(F_P(WP(x:=y, x>5)) , F_P(WP(x:=y, x\leq 5))),$
  choose(F_P(WP(x:=y, x<5)) , F_P(WP(x:=y, x\geq 5))),
  choose(F_P(WP(x:=y, y=5)) , F_P(WP(x:=y, y\neq 5))) >$
if (*){
  assume($G_P(x = 5))$;
  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
**QUESTION:** Is ERROR reachable in the boolean program?

\[
\begin{align*}
\text{assume}(\neg b_1 \land \neg b_2 \land \neg b_3); \\
< b_1, b_2, b_3 >: = & < \text{choose}(\text{false}, b_3), \\
& \text{choose}(\text{false}, b_3), \\
& \text{choose}(b_3, \neg b_3) > \\
\end{align*}
\]

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

---

An overview of SLAM – p. 12
QUESTION: Is ERROR reachable in the boolean program?

\[
\text{assume}(\neg b_1 \land \neg b_2 \land \neg b_3);
\]
\[
< b_1, b_2, b_3 > : = < \text{choose}(\text{false}, b_3),
\]
\[
\text{choose}(\text{false}, b_3),
\]
\[
\text{choose}(b_3, \neg b_3) >
\]
\[
\text{if } (*){
\]
\[
\text{assume}(\neg b_1 \land \neg b_2);
\]
\[
\text{ERROR}; } 
\]

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 \textit{false} is possible and leads to ERROR
Overview of the session

- The SLAM loop

- SLAM components:
  - C2BP
  - Bebop
  - Newton
  - Constrain
Let's consider the path to ERROR in the original C program:

1. \( x := x_\theta \)

<table>
<thead>
<tr>
<th>VAR</th>
<th>VAL</th>
<th>CONSTR.</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>( x )</td>
<td>( x_\theta )</td>
</tr>
</tbody>
</table>
Third step: Newton - path feasibility

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

<table>
<thead>
<tr>
<th>VAR</th>
<th>VAL</th>
<th>CONSTR.</th>
</tr>
</thead>
<tbody>
<tr>
<td>x = x_θ</td>
<td>x</td>
<td>x_θ</td>
</tr>
<tr>
<td>y = y_θ</td>
<td>y</td>
<td>y_θ</td>
</tr>
</tbody>
</table>
Let’s consider the path to ERROR in the original C program:

1. \( x := x_\theta \)
2. \( y := y_\theta \)
3. \( \text{assume}(x=5) \)

<table>
<thead>
<tr>
<th>VAR</th>
<th>VAL</th>
<th>CONSTR.</th>
</tr>
</thead>
<tbody>
<tr>
<td>( x )</td>
<td>( x_\theta )</td>
<td></td>
</tr>
<tr>
<td>( y )</td>
<td>( y_\theta )</td>
<td></td>
</tr>
</tbody>
</table>

\[ x_\theta = 5 \]
Let’s consider the path to ERROR in the original C program:

<table>
<thead>
<tr>
<th></th>
<th>VAR</th>
<th>VAL</th>
<th>CONSTR.</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.</td>
<td>$x := x_\theta$</td>
<td>$x$</td>
<td>$x_\theta$</td>
</tr>
<tr>
<td>2.</td>
<td>$y := y_\theta$</td>
<td>$y$</td>
<td>$y_\theta$</td>
</tr>
<tr>
<td>3.</td>
<td>assume($x=5$)</td>
<td>$x_\theta = 5$</td>
<td></td>
</tr>
<tr>
<td>4.</td>
<td>assume($y&gt;x$)</td>
<td>$y_\theta &gt; x_\theta$</td>
<td></td>
</tr>
</tbody>
</table>
Let’s consider the path to ERROR in the original C program:

<table>
<thead>
<tr>
<th></th>
<th></th>
<th>VAR</th>
<th>VAL</th>
<th>CONSTR.</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.</td>
<td>x:=x_θ</td>
<td>x</td>
<td>x_θ</td>
<td></td>
</tr>
<tr>
<td>2.</td>
<td>y:=y_θ</td>
<td>y</td>
<td>y_θ</td>
<td></td>
</tr>
<tr>
<td>3.</td>
<td>assume(x=5)</td>
<td></td>
<td></td>
<td>x_θ = 5</td>
</tr>
<tr>
<td>4.</td>
<td>assume(y&gt;x)</td>
<td></td>
<td></td>
<td>y_θ &gt; x_θ</td>
</tr>
<tr>
<td>5.</td>
<td>x:=y;</td>
<td>x</td>
<td>y_θ</td>
<td></td>
</tr>
</tbody>
</table>
### Third step: Newton - path feasibility

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

<table>
<thead>
<tr>
<th></th>
<th>VAR</th>
<th>VAL</th>
<th>CONSTR.</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.</td>
<td>$x := x_\theta$</td>
<td>$x$</td>
<td>$x_\theta$</td>
</tr>
<tr>
<td>2.</td>
<td>$y := y_\theta$</td>
<td>$y$</td>
<td>$y_\theta$</td>
</tr>
<tr>
<td>3.</td>
<td><code>assume(x=5)</code></td>
<td></td>
<td>$x_\theta = 5$</td>
</tr>
<tr>
<td>4.</td>
<td><code>assume(y&gt;x)</code></td>
<td></td>
<td>$y_\theta &gt; x_\theta$</td>
</tr>
<tr>
<td>5.</td>
<td>$x := y$;</td>
<td>$x$</td>
<td>$y_\theta$</td>
</tr>
<tr>
<td>6.</td>
<td><code>assume(x=5)</code></td>
<td></td>
<td>$y_\theta = 5$</td>
</tr>
</tbody>
</table>
Third step: Newton - path feasibility

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

<table>
<thead>
<tr>
<th></th>
<th>VAR</th>
<th>VAL</th>
<th>CONSTR.</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.</td>
<td>x:=xₜ</td>
<td>x</td>
<td>xₜ</td>
</tr>
<tr>
<td>2.</td>
<td>y:=yₜ</td>
<td>y</td>
<td>yₜ</td>
</tr>
<tr>
<td>3.</td>
<td>assume(x=5)</td>
<td></td>
<td>xₜ = 5</td>
</tr>
<tr>
<td>4.</td>
<td>assume(y&gt;x)</td>
<td></td>
<td>yₜ &gt;xₜ</td>
</tr>
<tr>
<td>5.</td>
<td>x:=y;</td>
<td>x</td>
<td>yₜ</td>
</tr>
<tr>
<td>6.</td>
<td>assume(x=5)</td>
<td></td>
<td>yₜ = 5</td>
</tr>
</tbody>
</table>

NEWTON ACHIEVES TWO GOALS:

- **Detects** that the set of constraints 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?
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?
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

\[
\text{assume}(b_1 \land b_2 \land \lnot b_3);
\]
\[
< b_1, b_2, b_3 >:=< \text{choose}(b_3, \lnot b_3),
\]
\[
\text{choose}(\text{false, true}),
\]
\[
\text{choose}(b_3, \lnot b_3) >
\]

if (*){
\[
\text{assume}(b_1);
\]
\[
\text{ERROR;;}
\]
}

- Now, clearly \textbf{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)
SLAM Boolean abstraction of the C program was the following:

\[
\begin{align*}
\text{assume}(\neg b_1 \land \neg b_2 \land \neg b_3); \\
< b_1, b_2, b_3 > := < \text{choose}(\text{false}, b_3), \\
\quad \text{choose}(\text{false}, b_3), \\
\quad \text{choose}(b_3, \neg b_3) > \\
\text{if (*)}{ \\
\quad \text{assume}(\neg b_1 \land \neg b_2); \\
\quad \text{ERROR}; } \\
\end{align*}
\]

Which are the reachable abstract states before reaching \textbf{if}?

Reachable states are:

- \{0,0,0\} (ERROR!)
- \{1,0,0\}
- \{0,1,0\}
- \{1,1,0\}
Let’s compute the reachable abstract states in a different way:

```
assume(x = 5 ∧ y > x);

x:=y;

if x=5
    ERROR:;
```

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

\[
\{(x_1, y_0) | x_0 = 5 ∧ y_0 > x_0 ∧ x_1 = y_0\}
\]

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

\[
G_P(x_0 = 5 ∧ y_0 > x_0 ∧ x_1 = y_0)
\]

which is exactly \( \{0, 1, 0\} \).

Hence, the abstract bad state \( \{0, 0, 0\} \) is indeed NOT reachable.
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 $\gamma$ 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($x = 5 \land y > x$);

$I_2$ $x:=y$;

$I_3$ 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**

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

$I_2$ $x:=y$;

$I_3$ 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 ($x=5$)

$A_3 = \{0,0,0\}$

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

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

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

An overview of SLAM – p. 22
Fourth step: Constrain (2)

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

**C program**

$I_1 : \text{assume}(x = 5 \land y > x)$;

$I_2 x := y$;

$I_3 \textbf{if } x=5$

\[\text{ERROR;}\]

**Boolean trace**

$A_0 = \{0,0,0\}$

$(I_1)$

$A_1 = \{0,0,0\}$

$(I_2)$

$A_2 = \{0,0,0\}$

$(I_3) //\text{assume } (x=5)$

$A_3 = \{0,0,0\}$

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

\[x = 5 \land y \neq 5 \rightarrow \neg WP(x := y, x = 5 \land y \neq 5) \text{ is valid?}\]

\[x = 5 \land y \neq 5 \rightarrow \neg(y = 5 \land y \neq 5) \text{ valid? YES! } (A_1 \rightarrow A_2 \text{ is spur.}).\]

BEBOP will get the constrain $\neg(\neg b_1 \land \neg b_2 \land \neg b_3 \land \neg b'_1 \land \neg b'_2 \land \neg b'_3)$
Let’s revisit our example (predicates were $P = [x > 5, x < 5, y = 5]$):

**C program**

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

$I_2$: $x := y$;

$I_3$: 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 ($x = 5$)

$A_3 = \{0,0,0\}$

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

$x = 5 \land y \neq 5 \rightarrow \neg WP(\text{assu.}(x = 5 \land y > x), x = 5 \land y \neq 5)$ is valid?

$x = 5 \land y \neq 5 \rightarrow \neg(x = 5 \land y \neq 5)$ valid? NO! ($A_2 \rightarrow A_3$ not spur.).
After Constrain, BEBOP will get the Boolean program:

```plaintext
assume(¬b₁ ∧ ¬b₂ ∧ ¬b₃);
< b₁, b₂, b₃ >:=< choose(false, b₃),
    choose(false, b₃),
    choose(b₃, ¬b₃) >; //with constrain
    ¬( {0,0,0} → {0,0,0} )
```

```plaintext
if (*){
    assume(¬b₁ ∧ ¬b₂);
    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: