Compositional May-Must Program Analysis
Unleashing the Power of Alternation
**Property checking**

**Question**
Does the assertion hold for all possible inputs?

```c
void f()
{
  0:  *p = 4;
  1:  *q = 5;
  2:  assert (¬φ_{error})
}
```

**Must analysis:** finds bugs, but can’t prove their absence
**May analysis:** can prove the absence of bugs, but can result in false errors

More generally, we are interested in the query

\[
\langle φ_{pre} \Rightarrow f φ_{error} \rangle
\]
SMASH = Compositional May-Must Analysis

- **May analysis** = predicate abstraction (**SLAM**)
- **Must analysis** = symbolic execution + tests (**DART**)
- **Compositional May-Must analysis**:
  - Interprocedural analysis
  - Memoize and re-use *may/must summaries*
  - Allows fine-grained coupling and *alternation*

SMASH » Compositional-May || Compositional-Must!
void f()
{
    0:  *p = 4;
    1:  *q = 5;
}

\langle T \Rightarrow_f (* p \neq 4) \rangle = yes

• Captures facts that are guaranteed to hold on particular executions of the program (under-approximation)
• Error condition is reachable by any input that satisfies \((p = q)\)
May information

\[ ((p \neq q) \Rightarrow f(* p \neq 4)) = \text{no} \]

```c
void f()
{
0:   *p = 4;
1:   *q = 5;
}
```

- Captures facts that are true for all executions of the program (over-approximation)
- Proof can be obtained by keeping track of the predicates \((p = q)\) and \((* p \neq 4)\)
Must analysis

\[
\frac{\langle \hat{\phi}_1 \Rightarrow^p \hat{\phi}_2 \rangle}{\Omega_{n_p}^0 := \hat{\phi}_1 \quad \forall n \in N_p \setminus \{n_p^0\}. \Omega_n := \emptyset} \quad \text{[INIT – OMEGA]}
\]

= \hat{\phi}_1

- Associate every program point \( n \) with a set of program states \( \Omega_n \subseteq \Sigma_P \) (under-approximation)
- Initialize \( \Omega_n \) sets at every program point \( n \):
  \[
  \Omega_{n_p}^0 := \hat{\phi}_1
  \]
Must analysis

\[
e = (n_1, n_2) \in E_p \quad \theta \subseteq Post(\Gamma_e, \Omega_{n_1}) \\
\Omega_{n_2} := \Omega_{n_2} \cup \theta
\] [MUST – POST]

- Extend $\Omega_n$ sets by forward (under-approximate) analysis
- In particular, use $\theta \subseteq Post(\Gamma_e, \Omega_{n_1})$
Must analysis

\[
\frac{\langle \hat{\phi}_1 \Rightarrow \hat{\phi}_2 \rangle \quad \Omega_{p\hat{\phi}} \cap \hat{\phi}_2 \neq \emptyset}{\langle \hat{\phi}_1 \Rightarrow \hat{\phi}_2 \rangle = yes} \quad [BUG - FOUND]
\]

- If an \( \Omega_n \) state satisfies error condition, \( \langle \hat{\phi}_1 \Rightarrow \hat{\phi}_2 \rangle = yes \)
- DART [PLDI ’05] is a specific instance
May analysis

\[ \langle \hat{\phi}_1 \Rightarrow_{p} \hat{\phi}_2 \rangle \]

\[ \Pi_{n_p^x} := \{ \hat{\phi}_2, \Sigma_{p} \setminus \hat{\phi}_2 \} \quad \forall n \in N_p \setminus \{ n_p^x \}. \quad \Pi_n := \{ \Sigma_{p} \} \quad \forall e \in E_p . \quad N_e := \emptyset \]

[Intit - PI - NE]

- Associate every program point \( n \) with a finite partition \( \Pi_n \) of \( \Sigma_p \) (over-approximation)
- Initialize regions \( \Pi_n \) at every program point \( n \):
  \[ \Pi_{n_p^x} := \{ \hat{\phi}_2, \Sigma_{p} \setminus \hat{\phi}_2 \} \]
May analysis

\[
\begin{align*}
\varphi_1 & \in \Pi_{n_1} \quad \varphi_2 \in \Pi_{n_2} \\
\quad e = (n_1, n_2) \in E_p \\
\theta & \supseteq Pre(\Gamma_e, \varphi_2) \\
\Pi_{n_1} & := (\Pi_{n_1} \setminus \{\varphi_1\}) \cup \{\varphi_1 \land \theta, \varphi_1 \land \neg \theta\} \\
N_e & := N_e \cup \{(\varphi_1 \land \neg \theta, \varphi_2)\}
\end{align*}
\]

[NOTMAY – PRE]

- Refine abstraction via a backward (over-approximate) analysis
- In particular, use \( \theta \supseteq Pre(\Gamma_e, \varphi_2) \) for refinement and record deleted abstract edge in \( N_e \)
May analysis

\[ \langle \hat{\phi}_1 \Rightarrow_p \hat{\phi}_2 \rangle \]
\[ \forall n_0, ..., n_k . \forall \varphi_0, ..., \varphi_k . n_0 = n_P^0 \land n_k = n_P^x \land \varphi_0 \in \Pi_{n_0} \land \cdots \land \varphi_k \in \Pi_{n_k} \land \varphi_0 \land \varphi_k \neq \emptyset \land \varphi_0 \land \varphi_k \neq \emptyset \]
\[ \Rightarrow \exists i \in [0, k) . e = (n_i, n_{i+1}) \in E_P \Rightarrow (\varphi_i, \varphi_{i+1}) \in N_e \]
\[ \langle \hat{\phi}_1 \Rightarrow_p \hat{\phi}_2 \rangle = \text{no} \]

- If the error is unreachable in the abstraction, \[ \langle \hat{\phi}_1 \Rightarrow_p \hat{\phi}_2 \rangle = \text{no} \]
- SLAM [POPL 02] is a specific instance
May-Must analysis

\[
\varphi_1 \in \Pi_{n_1}, \quad \varphi_2 \in \Pi_{n_2}, \quad e = (n_1, n_2) \in E_P \\
\Omega_{n_1} \cap \varphi_1 \neq \emptyset, \quad \Omega_{n_2} \cap \varphi_2 = \emptyset \quad \theta \subseteq Post(\Gamma_e, \Omega_{n_1} \cap \varphi_1) \quad \varphi_2 \cap \theta \neq \emptyset \\
\Omega_{n_2} := \Omega_{n_2} \cup \theta
\]

\[\text{[MUST – POST]}\]

- Check if frontier \((n_1, n_2)\) can be extended by an \(\Omega_{n_2}\) set
May-Must analysis

\[
\begin{align*}
\varphi_1 \in \Pi_{n_1} & \quad \varphi_2 \in \Pi_{n_2} & \quad e = (n_1, n_2) \in E_P \\
\Omega_{n_1} \cap \varphi_1 \neq \emptyset & \quad \Omega_{n_2} \cap \varphi_2 = \emptyset & \quad \theta \subseteq \text{Post} \left( \Gamma_e, \Omega_{n_1} \cap \varphi_1 \right) & \quad \varphi_2 \cap \theta \neq \emptyset \\
\Omega_{n_2} := & \quad \Omega_{n_2} \cup \theta
\end{align*}
\]

[MUST – POST]

- Check if frontier \((n_1, n_2)\) can be extended by an \(\Omega_{n_2}\) set
- If yes, grow \(\Omega_{n_2}\) with \(\theta \subseteq \text{Post} \left( \Gamma_e, \Omega_{n_1} \cap \varphi_1 \right)\)
May-Must analysis

\[ \varphi_1 \in \Pi_{n_1} \quad \varphi_2 \in \Pi_{n_2} \quad e = (n_1, n_2) \in E_p \]
\[ \Omega_{n_1} \cap \varphi_1 \neq \emptyset \quad \Omega_{n_2} \cap \varphi_2 = \emptyset \quad \theta \supseteq \text{Pre}(\Gamma_e, \varphi_2) \quad \theta \cap \Omega_{n_1} = \emptyset \]
\[ \Pi_{n_1} := (\Pi_{n_1} \setminus \{\varphi_1\}) \cup \{\varphi_1 \cap \theta, \varphi_1 \cap \neg \theta\} \quad N_e := N_e \cup \{(\varphi_1 \cap \neg \theta, \varphi_2)\} \]

[NOTMAY − PRE]

- Check if frontier \((n_1, n_2)\) can be extended by an \(\Omega_{n_2}\) set
- If not, refine \(\Pi_{n_1}\) with \(\theta \supseteq \text{Pre}(\Gamma_e, \varphi_2)\) and record deleted abstract edge in \(N_e\)
- Synergy/Dash [FSE ’06, ISSTA ’08] are specific instances
A \textit{must summary} for a procedure $\mathcal{P}_i$ is of the form $(\varphi_1, \varphi_2) \in \xrightarrow{\text{must}} \mathcal{P}_i$

- $\forall t \in \varphi_2 . \exists s \in \varphi_1 . t$ can be obtained by executing $\mathcal{P}_i$ from an initial state $s$
Compositional Must analysis

\[ e = (n_1, n_2) \in E_{\mathcal{P}_i} \text{ is a call to procedure } \mathcal{P}_j \]

\[
(\varphi_1, \varphi_2) \in \xrightarrow{\text{must}}_{\mathcal{P}_j} \quad \begin{align*}
\Omega_{n_1} \supseteq \varphi_1 & \quad \theta \subseteq \varphi_2 \\
\Omega_{n_2} & := \Omega_{n_2} \cup \theta
\end{align*}
\]

\[ [\text{MUST – POST – USESUM}] \]

- Generate post states by using must summaries
Compositional Must analysis

\[ e = (n_1, n_2) \in E_{P_i} \text{ is a call to procedure } P_j \]
\[ (\varphi_1, \varphi_2) \in \xrightarrow{\text{must}} P_j \quad \Omega_{n_1} \supseteq \varphi_1 \quad \theta \subseteq \varphi_2 \]
\[ \Omega_{n_2} := \Omega_{n_2} \cup \theta \]

**must summary**

\[ \varphi_1 \subseteq \Omega_{n_1} \]
\[ \varphi_2 \supseteq \theta \]

- Generate post states by using **must** summaries
  - If **must summary** \((\varphi_1, \varphi_2)\) is applicable, use \(\theta \subseteq \varphi_2\) to extend \(\Omega_{n_2}\) set
- If no **must** summaries are available for procedure \(P_j\), analyze \(P_j\)
- **SMART** [POPL '07] is a specific instance

**procedure** \(P_i\)
Compositional May analysis

- A $\neg$may summary for a procedure $P_i$ is of the form $(\varphi_1, \varphi_2) \in \neg\text{may } P_i$
- $\forall s \in \varphi_1 \forall t \in \varphi_2 . t$ cannot be obtained by executing $P_i$ starting in state $s$
Compositional May analysis

$\varphi_1 \in \Pi_{n_1}$  $\varphi_2 \in \Pi_{n_2}$  $e = (n_1, n_2) \in E_{\mathcal{P}_i}$ is a call to procedure $\mathcal{P}_j$

$\langle \hat{\varphi}_1, \hat{\varphi}_2 \rangle \in \underset{\text{may}}{\Rightarrow}_{\mathcal{P}_j}$  $\varphi_2 \subseteq \hat{\varphi}_2$  $\theta \subseteq \hat{\varphi}_1$

$\Pi_{n_1} := (\Pi_{n_1} \setminus \{\varphi_1\}) \cup \{\varphi_1 \cap \theta, \varphi_1 \cap \neg \theta\}$  $N_e := N_e \cup \{(\varphi_1 \cap \theta, \varphi_2)\}$

[NMAY – PRE – USESUM]

\[\text{\textit{may summary}}\]

- Refine the abstraction for procedure $\mathcal{P}_i$ by using the \textit{may summary} for $\mathcal{P}_j$
Compositional May analysis

\[ \varphi_1 \in \Pi_{n_1} \quad \varphi_2 \in \Pi_{n_2} \quad e = (n_1, n_2) \in E_{\mathcal{P}_i} \text{ is a call to procedure } \mathcal{P}_j \]

\[ (\hat{\varphi}_1, \hat{\varphi}_2) \in \xrightarrow{\text{may}}_{\mathcal{P}_j} \varphi_2 \subseteq \hat{\varphi}_2 \quad \theta \subseteq \hat{\varphi}_1 \]

\[ \Pi_{n_1} \coloneqq (\Pi_{n_1} \setminus \{\varphi_1\}) \cup \{\varphi_1 \land \theta, \varphi_1 \land \neg \theta\} \quad N_e \coloneqq N_e \cup \{(\varphi_1 \land \theta, \varphi_2)\} \]

\[ \neg \text{may summary} \]

\[ \varphi_1 \supseteq \theta \]

\[ \varphi_2 \supseteq \varphi_2 \]

---

\[ \text{procedure } \mathcal{P}_i \]

\[ 0 \quad T \]

\[ 1 \quad T \]

\[ 2 \quad \varphi_1 \land \neg \theta \]

\[ 3 \quad T \]

\[ 4 \quad \varphi_1 \land \theta \]

\[ 5 \quad T \]

\[ 6 \quad \varphi_2 \]

\[ 7 \]

- Refine the abstraction for procedure \( \mathcal{P}_i \) by using the \( \neg \text{may summary} \) for \( \mathcal{P}_j \)
  - If \( \neg \text{may summary} \) (\( \hat{\varphi}_1, \hat{\varphi}_2 \)) is applicable, use \( \theta \subseteq \hat{\varphi}_1 \) to refine the abstraction
- If \( \neg \text{may} \) summaries are not available for procedure \( \mathcal{P}_j \), analyze \( \mathcal{P}_j \)
- SLAM [POPL ’02] is a specific instance
\( \varphi_1 \in \Pi_{n_1} \quad \varphi_2 \in \Pi_{n_2} \quad \varphi_1 \cap \Omega_{n_1} \neq \emptyset \quad \varphi_2 \cap \Omega_{n_2} = \emptyset \)

\( e = (n_1, n_2) \in E_{P_i} \) is a call to procedure \( P_j \)

\[
(\hat{\varphi}_1, \hat{\varphi}_2) \in \xrightarrow{\text{must}}_{P_j} \quad \Omega_{n_1} \supseteq \hat{\varphi}_1 \quad \theta \subseteq \hat{\varphi}_2 \quad \varphi_2 \cap \theta \neq \emptyset
\]

\[
\Omega_{n_2} := \Omega_{n_2} \cup \theta
\]

[\text{MUST – POST – USESUM}]

- Base analysis is a may-must analysis (Dash)
- Check if frontier \((n_1, n_2)\) can be extended by a \textit{must summary} \((\hat{\varphi}_1, \hat{\varphi}_2)\)
\[ \varphi_1 \in \Pi_{n_1} \quad \varphi_2 \in \Pi_{n_2} \quad \varphi_1 \cap \Omega_{n_1} \neq \emptyset \quad \varphi_2 \cap \Omega_{n_2} = \emptyset \]

\( e = (n_1, n_2) \in E_{P_i} \) is a call to procedure \( P_j \)

\[ (\hat{\varphi}_1, \hat{\varphi}_2) \in \frac{\text{must}}{P_j} \quad \Omega_{n_1} \ni \hat{\varphi}_1 \quad \theta \subseteq \hat{\varphi}_2 \quad \varphi_2 \cap \theta \neq \emptyset \]

\[ \Omega_{n_2} := \Omega_{n_2} \cup \theta \] [MUST – POST – USESUM]

**must summary**

- \( \hat{\varphi}_1 \subseteq \Omega_{n_1} \)
- \( P_j \)
- \( (\hat{\varphi}_2 \ni \theta) \wedge (\varphi_2 \cap \theta \neq \emptyset) \)

- Check if frontier \((n_1, n_2)\) can be extended by a *must summary* \((\hat{\varphi}_1, \hat{\varphi}_2)\)
- If yes, grow \( \Omega_{n_2} \) with \( \theta \subseteq \hat{\varphi}_2 \)
\( \varphi_1 \in \Pi_{n_1} \quad \varphi_2 \in \Pi_{n_2} \quad \varphi_1 \cap \Omega_{n_1} \neq \emptyset \quad \varphi_2 \cap \Omega_{n_2} = \emptyset \)

\( e = (n_1, n_2) \in E_{P_i} \) is a call to procedure \( P_j \)

\( (\hat{\varphi}_1, \hat{\varphi}_2) \in \xrightarrow{\text{may}} P_j \quad \varphi_2 \subseteq \hat{\varphi}_2 \quad \theta \subseteq \hat{\varphi}_1 \quad \neg \theta \cap \Omega_{n_1} = \emptyset \)

\( \Pi_{n_1} := (\Pi_{n_1} \setminus \{\varphi_1\}) \cup \{\varphi_1 \cap \theta, \varphi_1 \cap \neg \theta\} \quad N_e := N_e \cup \{(\varphi_1 \cap \theta, \varphi_2)\} \)

- may summary

\[ (\hat{\varphi}_1 \supseteq \theta) \land (\neg \theta \cap \Omega_{n_1} = \emptyset) \]

\[ \hat{\varphi}_2 \supseteq \varphi_2 \]

- Check if frontier \((n_1, n_2)\) can be refined by a
  - may summary \((\hat{\varphi}_1, \hat{\varphi}_2)\)
- If yes, use \( \theta \subseteq \hat{\varphi}_1 \) to refine the abstraction
- If both must and may summaries are not available, analyze procedure \( P_j \)
  - yes \( \Rightarrow \) must summary for \( P_j \)
  - no \( \Rightarrow \) may summary for \( P_j \)
Interplay between \( \neg \text{may} \) and \( \text{must} \) summaries
The **SMASH** implementation is a deterministic realization of the declarative rules

Input C program is first abstractly interpreted
- No pointer arithmetic -- \(*(p+i)\) is treated as \*p
- Logic encoding -- propositional logic, linear arithmetic and uninterpreted functions

Theorem prover: **Z3**
Evaluation on Windows 7 drivers

<table>
<thead>
<tr>
<th>Statistics</th>
<th>Dash</th>
<th>SMASH</th>
</tr>
</thead>
<tbody>
<tr>
<td>Average ¬<em>may summaries</em>/driver</td>
<td>0</td>
<td>39</td>
</tr>
<tr>
<td>Average <em>must summaries</em>/driver</td>
<td>0</td>
<td>12</td>
</tr>
<tr>
<td>Number of proofs</td>
<td>2176</td>
<td>2228</td>
</tr>
<tr>
<td>Number of bugs</td>
<td>64</td>
<td>64</td>
</tr>
<tr>
<td>Time-outs</td>
<td>61</td>
<td>9</td>
</tr>
<tr>
<td>Time (hours)</td>
<td>117</td>
<td>44</td>
</tr>
</tbody>
</table>

69 drivers (342000 LOC) and 85 properties

We have unleashed the power of alternation!
SMASH is a unified framework for compositional may-must program analysis.

We have explained SMASH in the context of existing analyses (SLAM, DART, Synergy/Dash ...) in the area.

Empirical evaluation shows that SMASH can significantly outperform may-only, must-only and non-compositional may-must algorithms.