Effective Approaches to Abstraction Refinement for an Explicit Value Analysis

Stefan Löwe
Outline of my Thesis

Value Analysis
- Design of value domain
- Definition of value interpolation and CEGAR
- Optimization of interpolation and refinement components
- Implementation as independent components within CPAChecker

Predicate Analysis
- Design of predicate domain
- Definition of Craig interpolation and CEGAR
- Optimizations, e.g., adjustable block encoding
- Implementation as independent components within CPAChecker

Composite Analysis based on CEGAR
- Design of a precise and efficient composite analysis
- Contribution to SV-COMP'13 wins silver medal in "Overall"

CEGAR Analyses

Refinements over Infeasible Sliced Prefixes
- Definition of infeasible sliced prefixes and extraction algorithms
- Application of infeasible sliced prefixes to CEGAR-based analyses

Guided Refinement Selection
- Definition of heuristics for guided refinement selection
- Intra- and inter-analysis refinement selection for CEGAR-based analyses

Contribution to SV-COMP'16
- Contribution to SV-COMP'16 wins gold medal in "DeviceDriversLinux64"
- Further improvements over SV-COMP'16 contribution
Outline of my Talk

CEGAR Analyses

Value Analysis
- Design of value domain
- Definition of value interpolation and CEGAR
- Optimization of interpolation and refinement components
- Implementation as independent components within CPAchecker

Predicate Analysis
- Design of predicate domain
- Definition of Craig interpolation and CEGAR
- Optimizations, e.g., adjustable block encoding
- Implementation as independent components within CPAchecker

Composite Analysis based on CEGAR
- Design of a precise and efficient composite analysis
- Contribution to SV-COMP'13 wins silver medal in “Overall”

Refinements over Infeasible Sliced Prefixes
- Definition of infeasible sliced prefixes and extraction algorithms
- Application of infeasible sliced prefixes to CEGAR-based analyses

Guided Refinement Selection
- Definition of heuristics for guided refinement selection
- Intra- and inter-analysis refinement selection for CEGAR-based analyses

Contribution to SV-COMP’16
- Contribution to SV-COMP’16 wins gold medal in “DeviceDriversLinux64”
- Further improvements over SV-COMP’16 contribution
```c
#include <assert.h>

int main() {
    int a = 0;
    int b = 1;
    int c;
    b = a + b;

    if (c) {
        a = 1;
    } else {
        a = 2;
    }

    int f = a - b;
    if (f < 0) {
        assert(0);
    }
}
```
```c
#include "assert.h"
int main() {
    int a = 0;
    int b = 1;
    int c;
    b = a + b;
    if (c) {
        a = 1;
    } else {
        a = 2;
    }
    int f = a - b;
    if (f < 0) {
        assert(0);
    }
    return;
}
```

```text
#include "assert.h"
int main() {
    int a = 0;
    int b = 1;
    int c;
    b = a + b;
    if (! (c == 0)) {
        a = 1;
    } else {
        a = 2;
    }
    int f = a - b;
    if (! (f < 0)) {
        assert(0);
    }
    return;
}
```
Value Analysis by the Numbers

- Well over 4000 verification tasks from SV-COMP’16
- VA solves almost two thirds
- Under SV-COMP’16 rules, complete evaluation takes 440 hours
- 410 hours, or 93%, are wasted for unsolved verification tasks

State-space explosion is prime reason for extreme resource consumption
#include <assert.h>

extern int system_call();

int main(int x) {
    int flag = 0, ticks = 0;
    int result;

    while(1) {
        ticks = ticks + 1;
        result = system_call();

        if( result == 0 || ticks > x ) {
            break;
        }
    }

    if(flag > 0) {
        assert(0);
    }
}

int flag = 0;
int ticks = 0;
int result;

ticks = ticks + 1;
result = system_call();

[result == 0 || ticks > x]

break;

[flag > 0]

assert(0);

return;
Counterexample-Guided Abstraction Refinement

- Program source code
- Build & check abstract model
- Error path found
- Error path is infeasible
- Refine precision
- Precision is analysis dependent:
  - E.g., set of predicates for a predicate analysis
  - E.g., set of variable identifiers for a value analysis
- Is feasible?
- No error path
- Safe
- Unsafe

- Unsafe
- Safe
Counterexample-Guided Abstraction Refinement

- Program source code

1. Build & check abstract model
2. Error path found
3. Interpolate infeasible error path to:
   - e.g., obtain set of predicates for a predicate analysis
   - e.g., obtain set of variable identifiers for a value analysis
4. Is feasible?
5. No error path
6. Safe/Unsafe
7. Refine precision

Error path is infeasible
Craig Interpolation

[Abstractions from Proofs, 2004, Henzinger, Jhala, Majumdar, McMillan]

\( \phi^- \)

\( \psi \)

the interpolant

\( \phi^+ \)

At \( L_{12} \) the interpolant \( \psi \) for \( \phi^- \) and \( \phi^+ \) could be: [flag = 0], or [flag \leq 0], or ...

int flag = 0;
int ticks = 0;
int result;
ticks = ticks + 1;
result = system_call();

[result == 0
|| ticks > x]

break;

[flag > 0]

assert(0);

return;

[!result == 0
|| ticks > x)]

[!(!result == 0
|| ticks > x)]
For a pair of constraint sequences $y^-$ and $y^+$, such that $y^- \land y^+$ is contradicting, an interpolant $\psi$ is a constraint sequence that fulfills the following requirements:

1) $y^-$ implies $\psi$
2) $\psi \land y^+$ is unsatisfiable
3) $\psi$ only contains symbols that are common to both $y^-$ and $y^+$

A the interpolant $\psi$ for $\phi^-$ and $\phi^+$ can only be:

[flag = 0]
Comparison to Plain Value Analysis

- Significant improvements in DeviceDrivers64Linux
- Significant regressions in ECA and ProductLines
- In total solves around 500 verification task less

High number of refinements is prime reason for overall regression
Inspecting Number of Refinements

At least three clusters distinguishable

- Solved by both #refinements < 200
- Solved only by VA-Cegar #refinements < 500
- Solved only by VA-Plain #refinements > 1000
Reducing Time for Refinements

- Optimized Interpolation
- Deepest Infeasible Suffix
- Interpolant-Equality
- Optimized Refinement
- “Scoped” Precision
- Eager Restart

➢ CEGAR pays off, solving well over 400 tasks more

➢ Lazy abstraction is not well-suited for the Value Analysis
Level of Non-Determinism

- Low level of non-determinism: Use Plain Value Analysis
- High level of non-determinism: Use Value Analysis with CEGAR

➢ Valid indicator whether to perform abstraction or not
Versatility of Value Interpolation

- Applicable to other analyses
- Octagon analysis
- Symbolic execution analysis
- Enables regression verification
- Parallel composition with Predicate Analysis
  - Availability of several effective analyses based on CEGAR

- Next: Techniques that may benefit all such analyses
#include <assert.h>
extern int f(int x);
int main() {
    int b = 0;
    int i = 0;

    while(1) {
        if (i > 9) {
            break;
        }
        f(i++);
    }

    if (b != 0) {
        if (i != 10) {
            assert(0);
        }
    }

    assert(0);

    (a) verification task
    (b) error path
    (c) bad sequence
    (d) good sequence
Extraction of Infeasible Sliced Prefixes

[Sliced Path Prefixes: An Effective Method to Enable Refinement Selection, 2015, Beyer, Löwe, Wendler]
Any infeasible sliced prefix \( \varphi \), that is extracted from an infeasible error path \( \sigma \), can be used for interpolation to exclude the original error path \( \sigma \) from subsequent iterations of CEGAR loop.

➢ We can use any prefix we want for interpolation!
Sliced Prefixes - Further Applications

- Enables guided refinement selection
- Improves effectiveness and efficiency of static refinement
- Speeds up Value Interpolation significantly
- Impressive results in combination with symbolic execution
- Better control for global refinement
  - All target states at once
  - Each target state with an unique refinement
- Infeasible Sliced Prefixes for ABE?
Infeasible Sliced Prefixes for ABE?

- ABE: block size can have any size
- ABE-encoded path represent different paths
  - Simply pick one? No!
  - Simply pick all? No!
- Just think in blocks
  - SBE-encoded paths also are made of blocks
  - SBE: each block contains a single statement
- For ABE: apply same approach as for SBE / Value Analysis
Infeasible Sliced Prefixes for ABE
Elimination of Infeasible Sliced Prefixes!

(a) Source code of verification task

```c
extern void VERIFIER_error();
void VERIFIER_assert(int cond) {
    if (!(cond)) {
        ERROR: VERIFIER_error();
    }
    return;
}

int main(void) {
    unsigned int x = 1;
    unsigned int y = 0;
    while (y < 1024) {
        x = 0;
        y++;
    }
    VERIFIER_assert(x == 0);
}
```

(b) Error path over two loop iteration

```c
main();

x = 1;
y = 0;
[y < 1024]
x = 0;
y = y + 1;
[y < 1024] !y < 1024
VERIFIER_assert((x == 0)
? cond = 1
: cond = 0);
[cond == 0]
VERIFIER_error();
```

Verification task `const_true-unreach-call1.c` from the official SVCOMP’16 repository, and a possible infeasible error path when analyzing the task with ABE-If.
Elimination of Infeasible Sliced Prefixes!

(a) Source code of verification task

```c
extern void VERIFIER_error();

void VERIFIER_assert(int cond) {
    if (!cond) {
        ERROR: VERIFIER_error();
    }
    return;
}

int main(void) {
    unsigned int x = 1;
    unsigned int y = 0;
    while (y < 1024) {
        x = 0;
        y++;
    }
    VERIFIER_assert(x == 0);
}
```

(b) Error path over two loop iteration

```c
main();

x = 1;
y = 0;
[y < 1024]
x = 0;
y = y + 1;
[y < 1024]
![y < 1024]
x = 0;
y = y + 1;

[!y < 1024]
VERIFIER_assert((x == 0)
    ? cond = 1
    : cond = 0);
[cond == 0]
VERIFIER_error();
```

Verification task `const_true-unreach-call1.c` from the official SVCOMP’16 repository, and a possible infeasible error path when analyzing the task with ABE-lf.
Elimination of Infeasible Sliced Prefixes!

(a) Source code of verification task

```c
extern void VERIFIER_error();
void VERIFIER_assert(int cond) {
  if (!(cond)) {
    ERROR: VERIFIER_error();
  }
  return;
}

int main(void) {
  unsigned int x = 1;
  unsigned int y = 0;
  while (y < 1024) {
    x = 0;
    y++;
  }
  VERIFIER_assert(x == 0);
}
```

(b) Error path over two loop iteration

```c
main();
x = 1;
y = 0;
[y < 1024]
x = 0;
y = y + 1;
[y < 1024]
VERIFIER_assert((y == 0)
? cond = 1
: cond = 0);
[cond == 0]
VERIFIER_error();
```

➢ For ABE: this approach is also not perfect
➢ Any other ideas?
Quite good for LDV

<table>
<thead>
<tr>
<th>Tool</th>
<th>CPAchecker 1.6.1-svn 23191M</th>
</tr>
</thead>
<tbody>
<tr>
<td>Date of execution</td>
<td>2016-09-22 23:40:40 CEST</td>
</tr>
<tr>
<td>Run set</td>
<td>original</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Options</th>
<th>status</th>
<th>cputime (s)</th>
<th>status</th>
<th>cputime (s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>test/programs/benchmarks/ldv-linux-4.2-rcl/linux-4.2-rcl.tar.xz-43 2a-drivers--tty--serial--jsm.cil.out.c</td>
<td>timeout</td>
<td>1000</td>
<td>true</td>
<td>48.9</td>
</tr>
<tr>
<td>test/programs/benchmarks/ldv-linux-4.2-rcl/linux-4.2-rcl.tar.xz-43 2a-drivers--tty--synclink_gtout.c</td>
<td>timeout</td>
<td>929</td>
<td>true</td>
<td>94.3</td>
</tr>
<tr>
<td>test/programs/benchmarks/ldv-linux-4.2-rcl/linux-4.2-rcl.tar.xz-43 2a-drivers--tty--synclinkmp.ut.c</td>
<td>timeout</td>
<td>916</td>
<td>true</td>
<td>81.1</td>
</tr>
<tr>
<td>test/programs/benchmarks/ldv-linux-4.2-rcl/linux-4.2-rcl.tar.xz-43 2a-drivers--usb--host--ehci-ll.out.c</td>
<td>timeout</td>
<td>921</td>
<td>true</td>
<td>158</td>
</tr>
<tr>
<td>test/programs/benchmarks/ldv-linux-4.2-rcl/linux-4.2-rcl.tar.xz-43 2a-drivers--usb--host--fotg2l.cil.out.c</td>
<td>timeout</td>
<td>1000</td>
<td>true</td>
<td>132</td>
</tr>
<tr>
<td>test/programs/benchmarks/ldv-linux-4.2-rcl/linux-4.2-rcl.tar.xz-43 2a-drivers--usb--host--ohci-ll.out.c</td>
<td>timeout</td>
<td>1000</td>
<td>true</td>
<td>139</td>
</tr>
<tr>
<td>test/programs/benchmarks/ldv-linux-4.2-rcl/linux-4.2-rcl.tar.xz-43 2a-drivers--usb--host--ohci2ll.cil.out.c</td>
<td>timeout</td>
<td>1000</td>
<td>true</td>
<td>141</td>
</tr>
<tr>
<td>test/programs/benchmarks/ldv-linux-4.2-rcl/linux-4.2-rcl.tar.xz-43 2a-drivers--usb--host--r8a66ll.cil.out.c</td>
<td>timeout</td>
<td>916</td>
<td>true</td>
<td>471</td>
</tr>
<tr>
<td>test/programs/benchmarks/ldv-linux-4.2-rcl/linux-4.2-rcl.tar.xz-43 2a-drivers--usb--host--serial--diganh-call.cil.out.c</td>
<td>timeout</td>
<td>924</td>
<td>true</td>
<td>101</td>
</tr>
<tr>
<td>test/programs/benchmarks/ldv-linux-4.2-rcl/linux-4.2-rcl.tar.xz-43 2a-drivers--usb--storage--uwa.out.c</td>
<td>timeout</td>
<td>1000</td>
<td>true</td>
<td>157</td>
</tr>
<tr>
<td>test/programs/benchmarks/ldv-challenges/linux-3.8-rcl-32 7a-drivers--md--md-mod.ko--ldv main0 see-unreach-call.cil.out.c</td>
<td>timeout</td>
<td>907</td>
<td>true</td>
<td>46.8</td>
</tr>
<tr>
<td>test/programs/benchmarks/ldv-challenges/linux-3.8-rcl-32 7a-drivers--video--fbdev--ath-call.cil.out.c</td>
<td>timeout</td>
<td>922</td>
<td>true</td>
<td>311</td>
</tr>
<tr>
<td>test/programs/benchmarks/ldv-challenges/linux-3.8-rcl-32 7a-drivers--video--fbdev--viall.cil.out.c</td>
<td>timeout</td>
<td>982</td>
<td>true</td>
<td>140</td>
</tr>
<tr>
<td>test/programs/benchmarks/ldv-challenges/linux-3.8-rcl-32 7a-drivers--xen--xen--pcibackach-call.cil.out.c</td>
<td>timeout</td>
<td>1000</td>
<td>true</td>
<td>82.4</td>
</tr>
<tr>
<td>test/programs/benchmarks/ldv-challenges/linux-3.8-rcl-32 7a-drivers--xen--xen--pcibackach-call.cil.out.c</td>
<td>timeout</td>
<td>1000</td>
<td>true</td>
<td>88.5</td>
</tr>
<tr>
<td>test/programs/benchmarks/ldv-challenges/linux-3.8-rcl-32 7a-drivers--md--md-mod.ko--ldv main0 see-unreach-call.cil.out.c</td>
<td>timeout</td>
<td>912</td>
<td>true</td>
<td>405</td>
</tr>
</tbody>
</table>

| total tasks | 267 | 239000 |
| score (267 tasks, max score: 496) | -19 | 411 |
Questions ?