Model Checking II

Emina Torlak
emina@cs.washington.edu
Today

Last lecture
  • Model checking basics

Today
  • Software model checking with SLAM

Reminders
  • Homework 3 is due on today at 11pm
  • Homework 4 is out

Based on lectures by Tom Ball and Sriram K. Rajamani. See the SLAM project webpage for details.
Overview of SLAM

SLAM
Software, programming Languages, Abstraction, and Model checking

Program P

Safety property S

A trace of P that violates S
Overview of SLAM

A sequential program (device driver) implemented in C.

Program P

Safety property S

SLAM
Software, programming Languages, Abstraction, and Model checking

A trace of P that violates S
Overview of SLAM

A sequential program (device driver) implemented in C.

Program P

Safety property S

Temporal property (an API usage rule) written in SLIC, such as “a lock should be alternatively acquired and released.”

SLAM
Software, programming Languages, Abstraction, and Model checking

A trace of P that violates S
Overview of SLAM

Most influential PLDI paper award and the 2011 CAV award.

Program P

Safety property S

Ships in Microsoft’s Static Driver Verifier (SDV) tool.

A trace of P that violates S
The SLAM process

Program P → Instrumentation → P' → Safety property S
The SLAM process

Program P

Instrumentation

P'

Abstraction

boolean program B

Safety property S
The SLAM process

Program P → Instrumentation → P' → Model checking

Abstraction

boolean program B

Safety property S
The SLAM process

Program P → Instrumentation → \( P' \) → Abstraction → Model checking → ✓

Safety property S
The SLAM process

- **Program P**
- **Instrumentation**
- **Safety property S**
- **Abstraction**
  - boolean program B
- **Model checking**
  - error trace for B
- **Trace validation**

✓
The SLAM process

Program P

Instrumentation

P'

Abstraction

boolean program B

error trace for B

Model checking

Trace validation

A trace of P that violates S

Safety property S
The SLAM process

Program P → Instrumentation → P' → Abstraction → Model checking → Trace validation → A trace of P that violates S

- boolean program B
- error trace for B
- new predicates

Safety property S
The SLAM process

Program P

Instrumentation

P'

C2BP

boolean program B

error trace for B

Bebop

Newton

new predicates

A trace of P that violates S
The SLAM process: specifying safety properties

Program P → Instrumentation → P' → Bebop → C2BP → Newton

Safety property S

A trace of P that violates S

new predicates

boolean program B

error trace for B
Specification Language for Interface Checking

A finite state language for stating rules for API usage

- Temporal safety properties expressed as safety automata that monitor program’s execution behavior at the level of function calls and returns.
- Familiar C syntax.

Suitable for control-dominated properties

- E.g., ordering of function calls with associated constraints on data values at the API boundary.
A locking protocol in SLIC
The global state structure defines a static set of state variables.
A locking protocol in SLIC

State:

```
enum {Locked, Unlocked}
state = Unlocked;
```

KeAcquireSpinLock:

```
return {
if (state == Locked)
    abort;
else
    state = Locked;
}
```

KeReleaseSpinLock:

```
return {
if (state == Unlocked)
    abort;
else
    state = Unlocked;
}
```

Transfer functions define events and event handlers that describe state transitions on events.

Locked

Error

Unlocked

 acquire

 release

 acquire

 release

 acquire
The SLAM process: instrumentation

Program $P$

Instrumentation

Safety property $S$

$P' \rightarrow C2BP$

Boolean program $B$

Error trace for $B$

Newton

New predicates

A trace of $P$ that violates $S$
Instrumentation by example: 2 steps

state {
    enum {Locked, Unlocked}
    state = Unlocked;
}

KeAcquireSpinLock.return {
    if (state ==Locked)
        abort;
    else
        state = Locked;
}

KeReleaseSpinLock.return {
    if (state == Unlocked)
        abort;
    else
        state = Unlocked;
}

void example() {
    do {
        KeAcquireSpinLock();
        nOld = nPackets;
        if (request) {
            request = request->next;
            KeReleaseSpinLock();
            nPackets++;
        }
    } while (nPackets != nOld);
    KeReleaseSpinLock();
}

Safety property S

Program P

Simplified code for a PCI device driver.
Step 1: translate the SLIC spec S to C

```c
state {
    enum {Locked, Unlocked}
    state = Unlocked;
}

KeAcquireSpinLock.return {
    if (state == Locked)
        abort;
    else
        state = Locked;
}

KeReleaseSpinLock.return {
    if (state == Unlocked)
        abort;
    else
        state = Unlocked;
}
```

```c
enum {Locked=0, Unlocked=1}
    state = Unlocked;

void slic_abort() {
    SLIC_ERROR:
}

void KeAcquireSpinLock_return {
    if (state == Locked)
        slic_abort();
    else
        state = Locked;
}

void KeReleaseSpinLock_return {
    if (state == Unlocked)
        slic_abort();
    else
        state = Unlocked;
}
```

Distinguished error label.

Safety property S
Step 2: insert calls to SLIC functions into $P$

```c
void example() {
    do {
        KeAcquireSpinLock();
        nOld = nPackets;
        if (request) {
            request = request->next;
            KeReleaseSpinLock();
            nPackets++;
        }
    } while (nPackets != nOld);
    KeReleaseSpinLock();
}
```

```c
void example() {
    do {
        KeAcquireSpinLock();
        KeAcquireSpinLock_return();
        nOld = nPackets;
        if (request) {
            request = request->next;
            KeReleaseSpinLock();
            KeReleaseSpinLock_return();
            nPackets++;
        }
    } while (nPackets != nOld);
    KeReleaseSpinLock();
    KeReleaseSpinLock_return();
}
```

Program $P$

Program $P'$
P satisfies S iff SLIC_ERROR is unreachable in P'

```c
void example() {
    do {
        KeAcquireSpinLock();
        KeAcquireSpinLock_return();

        nOld = nPackets;

        if (request) {
            request = request->next;
            KeReleaseSpinLock();
            KeReleaseSpinLock_return();
            nPackets++;
        }
    } while (nPackets != nOld);

    KeReleaseSpinLock();
    KeReleaseSpinLock_return();
}
```

```c
enum {Locked=0, Unlocked=1}
    state = Unlocked;

void slic_abort() {
    SLIC_ERROR: ;
}

void KeAcquireSpinLock_return {
    if (state == Locked)
        slic_abort();
    else
        state = Locked;
}

void KeReleaseSpinLock_return {
    if (state == Unlocked)
        slic_abort();
    else
        state = Unlocked;
}
```
The SLAM process: predicate abstraction

Program P → Instrumentation → P' → Bebop → C2BP → Newton → A trace of P that violates S

Safety property S

boolean program B

error trace for B

ew predicates

✓
Predicate abstraction of C Programs

Given a program $P$ and a finite set $E$ of predicates, C2BP creates a boolean program $B$ that is a sound over-approximation of $P$.

- $B$ has the same control-flow structure as $P$, but only $|E|$ boolean variables.
- For any path $p$ feasible in $P$, there is a corresponding feasible path in $B$.

Suitable abstraction for checking control-dominated properties (such as SLIC rules).

- Models control flow in $P$ precisely.
- Models only a few predicates about data relevant to each rule being checked (so limits state explosion).
Predicate abstraction by example: 5 steps

```c
void example() {
    do {
        KeAcquireSpinLock();
        KeAcquireSpinLock_return();

        nOld = nPackets;

        if (request) {
            request = request->next;
            KeReleaseSpinLock();
            KeReleaseSpinLock_return();
            nPackets++;
        }
    } while (nPackets != nOld);

    KeReleaseSpinLock();
    KeReleaseSpinLock_return();
}
```

```c
enum {Locked=0, Unlocked=1}
    state = Unlocked;

void slic_abort() {
    SLIC_ERROR: ;
}

void KeAcquireSpinLock_return {
    if (state == Locked)
        slic_abort();
    else
        state = Locked;
}

void KeReleaseSpinLock_return {
    if (state == Unlocked)
        slic_abort();
    else
        state = Unlocked;
}
```
void example() {
    do {
        KeAcquireSpinLock();
        KeAcquireSpinLock_return();

        nOld = nPackets;

        if (request) {
            request = request->next;
            KeReleaseSpinLock();
            KeReleaseSpinLock_return();
            nPackets++;
        }
    } while (nPackets != nOld);
}

KeReleaseSpinLock();
KeReleaseSpinLock_return();

enum {Locked=0, Unlocked=1}
    state = Unlocked;

void slic_abort() {
    SLIC_ERROR: ; }

void KeAcquireSpinLock_return {
    if (state == Locked)
        slic_abort();
    else
        state = Locked; }

d(void KeReleaseSpinLock_return {
    if (state == Unlocked)
        slic_abort();
    else
        state = Unlocked; }

    (state == Locked)
    (state == Unlocked)
Step 2: introduce boolean variables for E

```c
void example() {
    do {
        KeAcquireSpinLock();
        KeAcquireSpinLock_return();
        nOld = nPackets;
        if (request) {
            request = request->next;
            KeReleaseSpinLock();
            KeReleaseSpinLock_return();
            nPackets++;
        }
    } while (nPackets != nOld);
    KeReleaseSpinLock();
    KeReleaseSpinLock_return();
}
```

```c
b(state==Locked), b(state==Unlocked) := F, T;
```

```c
void slic_abort() {
    SLIC_ERROR: ; }
```

```c
void KeAcquireSpinLock_return {
    if b(state==Locked)
        slic_abort();
    else
        state = Locked;
}
```

```c
void KeReleaseSpinLock_return {
    if b(state==Unlocked)
        slic_abort();
    else
        state = Unlocked;
}
```
void example() {
    do {
        KeAcquireSpinLock();
        KeAcquireSpinLock_return();

        nOld = nPackets;

        if (request) {
            request = request->next;
            KeReleaseSpinLock();
            KeReleaseSpinLock_return();
            nPackets++;
        }
    } while (nPackets != nOld);

    KeReleaseSpinLock();
    KeReleaseSpinLock_return();
}

// Encode effects of assignments on E
b(state==Locked), b(state==Unlocked) := F, T;

void slic_abort() {
    SLIC_ERROR: ; }

void KeAcquireSpinLock_return {
    if b(state==Locked)
        slic_abort();
    else
        b(state==Locked),
        b(state==Unlocked) := T, F; }

void KeReleaseSpinLock_return {
    if b(state==Unlocked)
        slic_abort();
    else
        b(state==Locked),
        b(state==Unlocked) := F, T; }

(state == Locked)
(state == Unlocked)
Step 4: skip statements with no effect on E

```c
void example() {
    do {
        skip;
        KeAcquireSpinLock_return();
        skip;
        if (request) {
            skip;
            skip;
            KeReleaseSpinLock_return();
            skip;
        }
    } while (nPackets != nOld);
    skip;
    KeReleaseSpinLock_return();
}
```

```c
b(state==Locked), b(state==Unlocked) := F, T;

void slic_abort() {
    SLIC_ERROR: ; }

void KeAcquireSpinLock_return {
    if b(state==Locked)
        slic_abort();
    else
        b(state==Locked),
        b(state==Unlocked) := T, F; }

void KeReleaseSpinLock_return {
    if b(state==Unlocked)
        slic_abort();
    else
        b(state==Locked),
        b(state==Unlocked) := F, T; }

(state == Locked)
(state == Unlocked)
```
void example() {
    do {
        skip;
        KeAcquireSpinLock_return();
        skip;
        if (*) {
            skip;
            skip;
            KeReleaseSpinLock_return();
            skip;
        }
    } while (*);
}

void slic_abort() {
    SLIC_ERROR: ; }

void KeAcquireSpinLock_return {
    if b(state==Locked)
        slic_abort();
    else
        b(state==Locked),
        b(state==Unlocked) := T, F; }

void KeReleaseSpinLock_return {
    if b(state==Unlocked)
        slic_abort();
    else
        b(state==Locked),
        b(state==Unlocked) := F, T; }

(b(state == Locked)
(b(state == Unlocked))
Step 5: use non-determinism for conditions

```c
void example() {
  do {
    skip;
    KeAcquireSpinLock_return();
    skip;
    if (*) {
      skip;
      skip;
      KeReleaseSpinLock_return();
      skip;
    }
  } while (*);
  skip;
  KeReleaseSpinLock_return();
}
```

This is a highly simplified example of predicate abstraction. The process is much more complex in reality. For details, see [Automatic predicate abstraction of C programs](#).
The SLAM process: model checking

- Program P
- Instrumentation
- Safety property S
- P'
- C2BP
  - boolean program B
  - error trace for B
- Bebop
- Newton
  - new predicates
- A trace of P that violates S

✓
Model checking of boolean programs

Given a boolean program $B$ and a statement $s$ in $B$, Bebop determines if $s$ is reachable in $B$.

- Produces a shortest trace in $B$ (if any) leading to $s$.

Performs symbolic reachability analysis using BDDs.

- Adapts the interprocedural dataflow analysis of Reps, Horwitz and Sagiv (RHS) to decide the reachability of $s$ in $B$.
- Uses BDDs to represent the procedure summaries in RHS, which are binary relations between sets of states.

For details, see Bebop: A Symbolic Model Checker for Boolean Programs.
Model checking of the example program

```c
void example() {
    do {
        skip;
        KeAcquireSpinLock_return();
        skip;
        if (*) {
            skip;
            skip;
            KeReleaseSpinLock_return();
            skip;
        }
    } while (*);
}

void slic_abort() {
    SLIC_ERROR: ;
}

void KeAcquireSpinLock_return {
    if b(state==Locked)
        slic_abort();
    else
        b(state==Locked), b(state==Unlocked) := T, F; }

void KeReleaseSpinLock_return {
    if b(state==Unlocked)
        slic_abort();
    else
        b(state==Locked), b(state==Unlocked) := F, T; }
```
The SLAM process: trace validation

Program P → Instrumentation → P’ → Bebop → C2BP

Safety property S

error trace for B

new predicates

new predicates

A trace of P that violates S
Error trace validation & abstraction refinement

Given a program $P'$ and a candidate error trace, Newton determines if the trace is feasible.

- Uses verification condition generation for feasibility checking.
- If feasible, the error trace corresponds to a real bug.
- If not, returns a small set of predicates that explain why the path is infeasible. Based on greedy minimal unsatisfiable core computation.

For details, see Generating Abstract Explanations of Spurious Counterexamples in C Programs.
void example() {
    do {
        KeAcquireSpinLock();
        KeAcquireSpinLock_return();
        nOld = nPackets;

        if (request) {
            request = request->next;
            KeReleaseSpinLock();
            KeReleaseSpinLock_return();
            nPackets++;
        }
    } while (nPackets != nOld);

    KeReleaseSpinLock();
    KeReleaseSpinLock_return();
}

Validation & refinement for the example

enum {Locked=0, Unlocked=1}
    state = Unlocked;

void slic_abort() {
    SLIC_ERROR: ; }

void KeAcquireSpinLock_return {
    if (state == Locked)
        slic_abort();
    else
        state = Locked;
}

void KeReleaseSpinLock_return {
    if (state == Unlocked)
        slic_abort();
    else
        state = Unlocked;
}

(state == Locked)
(state == Unlocked)
Validation & refinement for the example

```c
void example() {
    do {
        KeAcquireSpinLock();
        KeAcquireSpinLock_return();

        nOld =nPackets;

        if (request) {
            request = request->next;
            KeReleaseSpinLock();
            KeReleaseSpinLock_return();
            nPackets++;
        }
    } while (nPackets != nOld);

    KeReleaseSpinLock();
    KeReleaseSpinLock_return();
}
```

```c
enum {Locked=0, Unlocked=1}
state = Unlocked;

void slic_abort() {
    SLIC_ERROR: ; }

void KeAcquireSpinLock_return {
    if (state == Locked)
        slic_abort();
    else
        state = Locked; }

void KeReleaseSpinLock_return {
    if (state == Unlocked)
        slic_abort();
    else
        state = Unlocked; }

(nPackets == nOld)
(state == Locked)
(state == Unlocked)
```
void example() {
    do {
        skip;
        KeAcquireSpinLock_return();
        b(nOld==nPackets) := T;
        if (*) {
            skip;
            skip;
            KeReleaseSpinLock_return();
            b(nOld==nPackets) :=
                b(nOld==nPackets) ? F : *;
        }
    } while (!b(nOld==nPackets));
    skip;
    KeReleaseSpinLock_return();
}
void example() {
    do {
        skip;
        KeAcquireSpinLock_return();
        b(nOld==nPackets) := T;
        if (*) {
            skip;
            skip;
            KeReleaseSpinLock_return();
            b(nOld==nPackets) := b(nOld==nPackets) ? F : *;
        }
    } while (!b(nOld==nPackets));
    skip;
    KeReleaseSpinLock_return();
}

b(state==Locked), b(state==Unlocked) := F, T;

void slic_abort() {
    SLIC_ERROR: ; }

void KeAcquireSpinLock_return {
    if b(state==Locked)
        slic_abort();
    else
        b(state==Locked),
        b(state==Unlocked) := T, F; }

void KeReleaseSpinLock_return {
    if b(state==Unlocked)
        slic_abort();
    else
        b(state==Locked),
        b(state==Unlocked) := F, T; }

Back to C2BP and Bebop ...
Summary

Today

• Software model checking with SLAM
  • Predicate abstraction of C programs
  • Model checking of boolean programs
  • Trace validation and abstraction refinement

Next lecture

• Program synthesis