Taming Win32 Threads with Static Analysis

Jason Yang

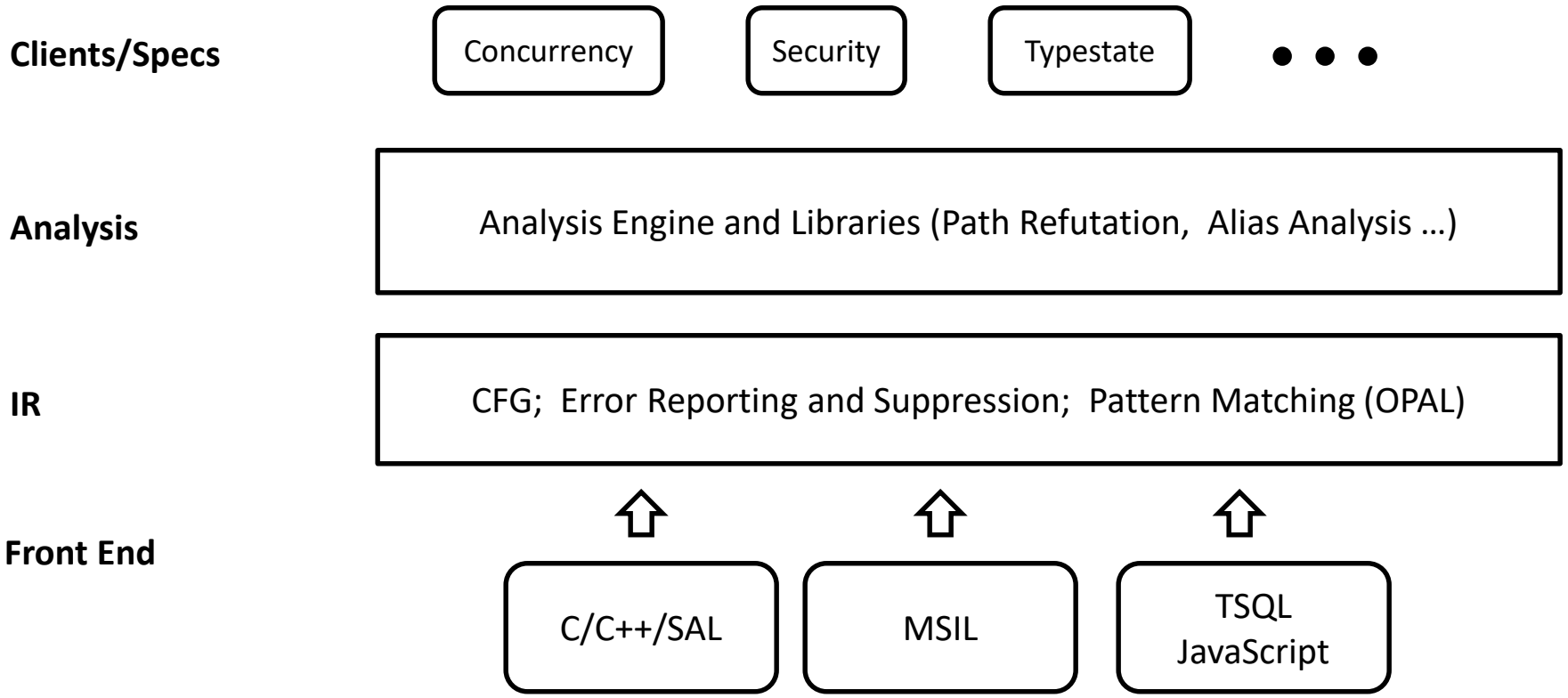
Program Analysis Group
Center for Software Excellence (CSE)
Microsoft Corporation

CSE Program Analysis Group

- Technologies
  - PREfix, SAL, ESP, EspX, EspC, …
- Defect detection tools
  - Buffer overrun, Null deref, SQL injection, programmable specification checker, concurrency, …

http://www.microsoft.com/windows/cse/pa_home.mspx

The Esp Analysis Platform



Locking Problems

- Insufficient lock protection
- Lock order violation
- Forgetting to release
- Ownership violation
- No-suspend guarantee violation
- UI thread blocking due to SendMessage
- And more …

Win32 Multithreading: Plenty of Challenges

- Rich set of lock primitives
  - Critical Section, Mutex, Event, Semaphore, …
- "Free-style" acquire/release
  - Not syntactically scoped
- Implicit blocking semantics
  - SendMessage, LoaderLock, …

- How to ensure the intended locking discipline?
  - Who guards what, who needs to acquire, lock order, …

Central Issue: Locking Disciplines

- Essential for avoiding threading errors
- Surprisingly hard to enforce in practice
  - "We have a set of locking conventions to follow"
  - "They are informally documented"
  - "We don't have a way to check against the violations"

Checking Concurrency Properties via Sequential Analysis

- Insight: Developers mostly reason about concurrent code “sequentially” following locking disciplines
- Approach: Mimic developer’s reasoning
  - Track locking behavior via sequential analysis
  - Simulate interleaving effects afterwards

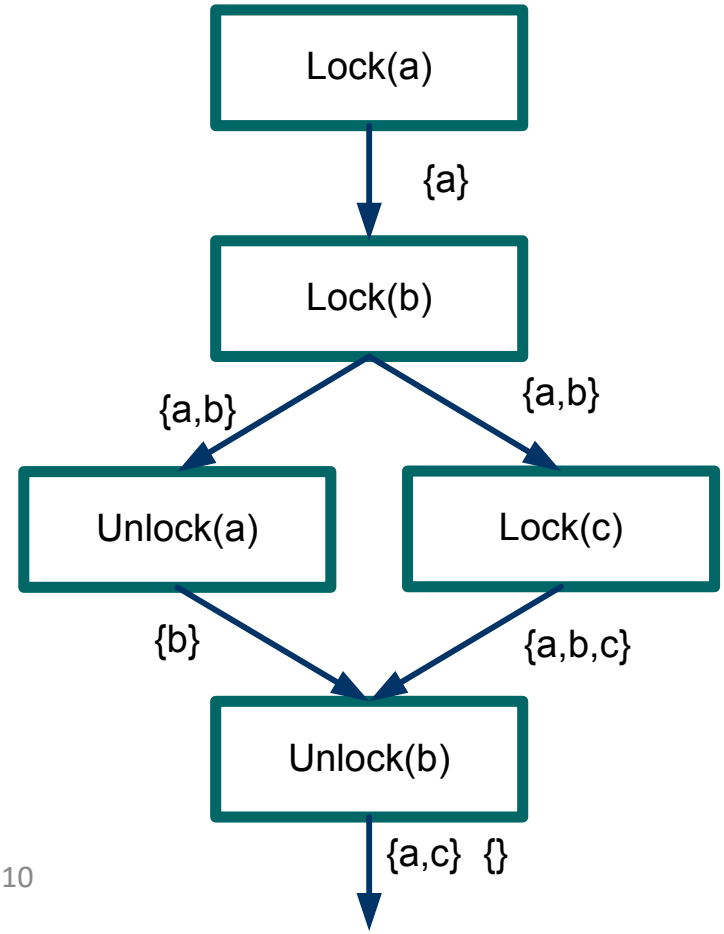
EspC Concurrency Toolset

- Global lock analysis – Global EspC
- Concurrency annotation – Concurrency SAL
- Local lock analysis – EspC
- Lock annotation inference – CSALInfer

Global EspC

- Global lock analyzer
  - Deadlock detection
  - Code mining
- Based on ESP
  - Inter-procedural analysis with function summaries
  - Path-sensitive, context-sensitive
  - Selective merge on dataflow facts
  - Symbolic simulation for path feasibility
- “Understands” Win32 threading semantics
  - Use OPAL specification to capture Win32 locking APIs

Phase 1: Lock Sequence Computation



- Start from root functions
- Track lock sequences at each program point
- Do not merge if lock sequences are different
- Identify locks syntactically based on their types

Phase 2: Defect Detection

- Deadlock detection
  - Cyclic locking $\Rightarrow$ deadlock!
- Race detection
  - Insufficient locking $\Rightarrow$ race condition!
- Lock misuse patterns
  - E.g., exit while holding a lock $\Rightarrow$ orphaned lock!

Concurrency SAL

- Extension to SAL
  - Covers concurrency properties
- Example lock annotations
  - Lock/data protection relation:

    __guarded_by(cs) int balance;
  - Caller/callee locking responsibility:

    __requires_lock_held(cs) void Deposit (int amount);
  - Locking side effect:

    __acquires_lock(cs) void Enter();__releases_lock(cs) void Leave();
  - Lock acquisition order:

    __lock_level_order(TunnelLockLevel, ChannelLockLevel);
  - Threading context:

    __no_competing_thread void Init();

CSALInfer

- Concurrency SAL inference engine
- Statistics-based inference for in-scope locks
  - Tracks lock statistics for accessing shared variables
  - Computes “dominant lock” for shared variable
- Constraint-based inference for out-of-scope locks
  - Generates constraints based on locking events
  - Translates constraints to propositional formula
  - Solves constraints via SAT solving
- Heuristics-based inference
  - Looks for strong evidence along a path

EspC

- Local static lock analyzer
  - Understands lock annotations
  - Check violations of Concurrency SAL
- Runs on developer's desktop
  - PREfast plugin

Subset of EspC Warnings

- **26100**: EspC: race condition. Variable ‘VarExpr’ should be protected by ‘LockExpr’.
- **26110**: EspC: caller failing to hold ‘LockExpr’ before calling ‘FunctionName’.
- **26111**: EspC: caller failing to release ‘LockExpr’ before calling ‘FunctionName’.
- **26112**: EspC: caller cannot hold any lock before calling ‘FuncName’.
- **26115**: EspC: failing to release ‘LockExpr’ in ‘FunctionName’.
- **26116**: EspC: failing to acquire or to hold ‘LockExpr’ in ‘FunctionName’.
- **26117**: EspC: releasing unheld lock ‘LockExpr’ in ‘FunctionName’.
- **26140**: EspC: error in Concurrency SAL annotation.

Summary

- Covers a variety of concurrency issues
  - Deadlocks
  - Race conditions
  - Win32 locking errors
  - Atomicity violations
- Tackles from different angles
  - Global analysis: Global EspC
  - Annotations: Concurrency SAL
  - Local analysis: EspC
  - Annotation inference: CSALInfer