SPL^{LIFT} - Transparent and Efficient Reuse of IFDS-based Static Program Analyses

Eric Bodden (TU Darmstadt), Claus Braband (ITU Copenhagen), Marcio Ribeiro, Tarsis Toledo, Paulo Borba (UFPE Brazil) and Mira Mezini (TU Darmstadt)

To appear at PLDI'13!

Motivation

- **Product lines** allow highly customizable products as specializations of a common platform; success stories exist in the car manufacturing and mobile devices industries
- **Software product lines** (SPLs) use conditional compilation (e.g. via `#ifdef`) to define many products as variations of a common code base
- **Problem: traditional static analyses** can only be applied to pre-processed software products
- **But even small product lines** can induce thousands of products: different combinations of features cause a combinatorial explosion
- **Result: existing approaches do not scale; SPLs are currently effectively unanalyzable**

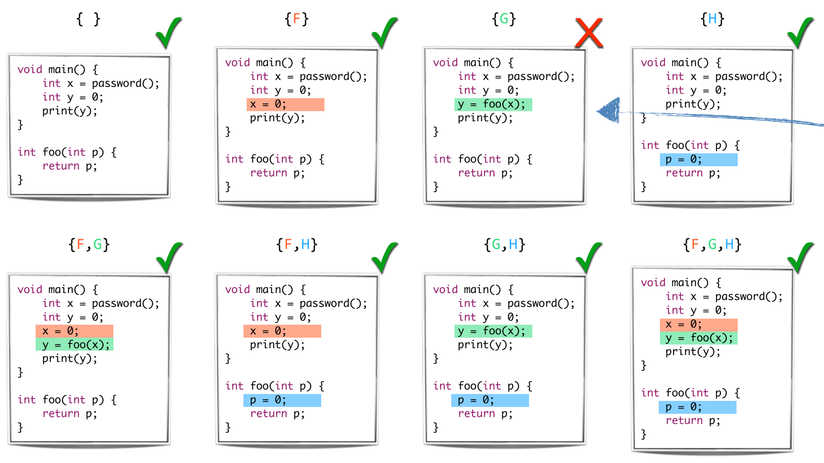
```
void main() {
    int x = password();
    int y = 0;
#ifdef F
    x = 0;          Feature F
#elif G
    y = foo(x);     Feature G
#endif
    print(y);
}

int foo(int p) {
#ifdef H
    p = 0;          Feature H
#endif
    return p;
}
```

?

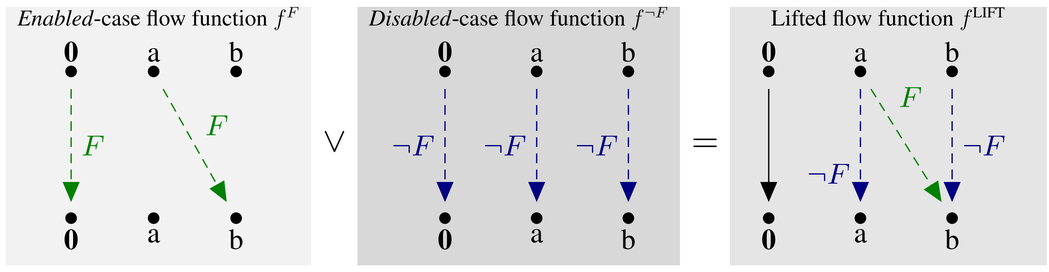
traditional approach: to decide whether the `password` may leak to the `print` statement, one must analyze all $2^{|\{F,G,H\}|}=8$ *possible products*

product {G} indeed contains a leak



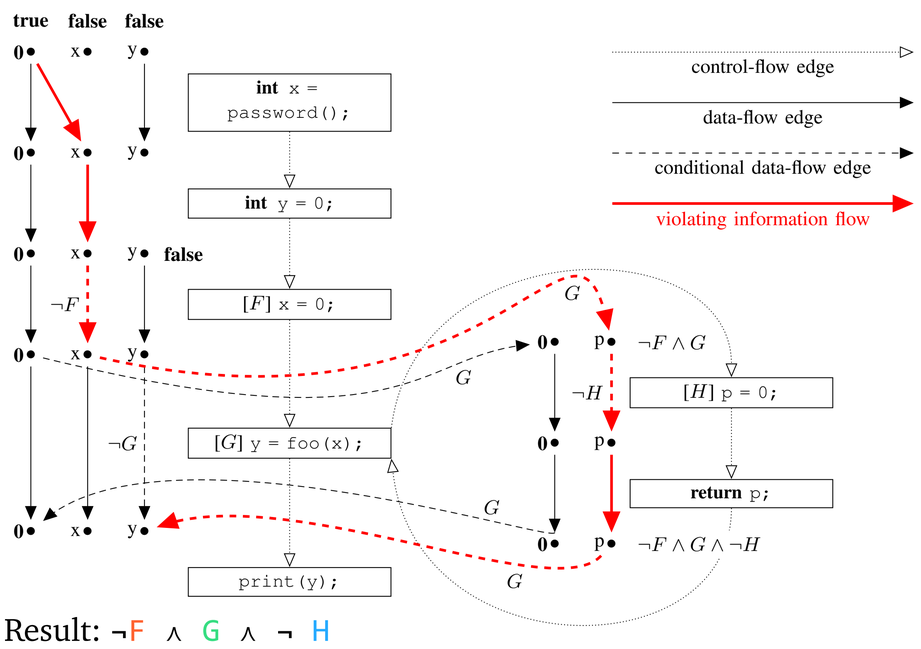
Methodology

- **Our approach SPL^{LIFT}** instead analyzes the entire product line at once, including all possible combinations
- **This is achieved by combining** flow functions for the case where an ifdef is disabled with the one for the case where it is enabled
- **The result is a lifted flow function**



lifting of a flow function that generates the data-flow fact b *after the statement if* a *is valid before the statement*

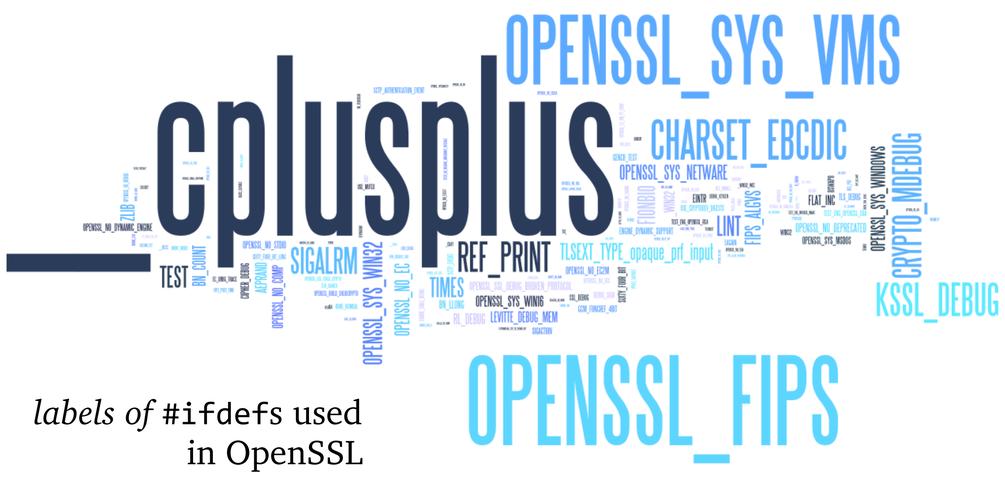
SPL^{LIFT} applied to example SPL



Result: ¬F ∧ G ∧ ¬ H

SPL^{LIFT} determines in a single pass that the `password` can only leak if G is enabled but F and H are disabled. This is consistent with the result determined by the traditional analysis.

Scale of the problem



labels of `#ifdefs` *used in OpenSSL*

- **Scale of the problem is immense:** as an example, we investigated the usage of `#ifdef` constructs in the OpenSSL crypto library
- **OpenSSL contains** 1874 `#ifdefs` with 391 different labels
- **This yields** $2^{391} \approx 5 \cdot 10^{117}$ combinations!
- **As comparison: The observable universe** has only about 10^{80} atoms.

Empirical Evaluation

- **SPL^{LIFT} shows remarkable performance**
- In some cases the **traditional approach** would have taken **days or years**, while **SPL^{LIFT}** only takes minutes to compute.
- **Hence solved a problem** to which previously no scalable solution existed

Benchmark	Possible Types SPL^{LIFT}	Possible Types *A2*	Reaching Definitions SPL^{LIFT}	Reaching Definitions *A2*	Uninitialized Variables SPL^{LIFT}	Uninitialized Variables *A2*
BerkeleyDB	24s	*years*	12m04s	*years*	10m18s	*years*
GPL	42s	9h03m39s	8m48s	*days*	7m09s	*days*
Lampiro	4s	13s	42s	3m30s	1m25s	3m09s
MM08	3s	2m06s	59s	24m29s	2m13s	27m39s

Performance comparison of SPL^{LIFT} with traditional approach "A2".

Implementation

- **Based on** CIDE, Soot, JavaBDD, and our IFDS implementation Heros
- **Available as open source at:** http://bodden.de/spllift/