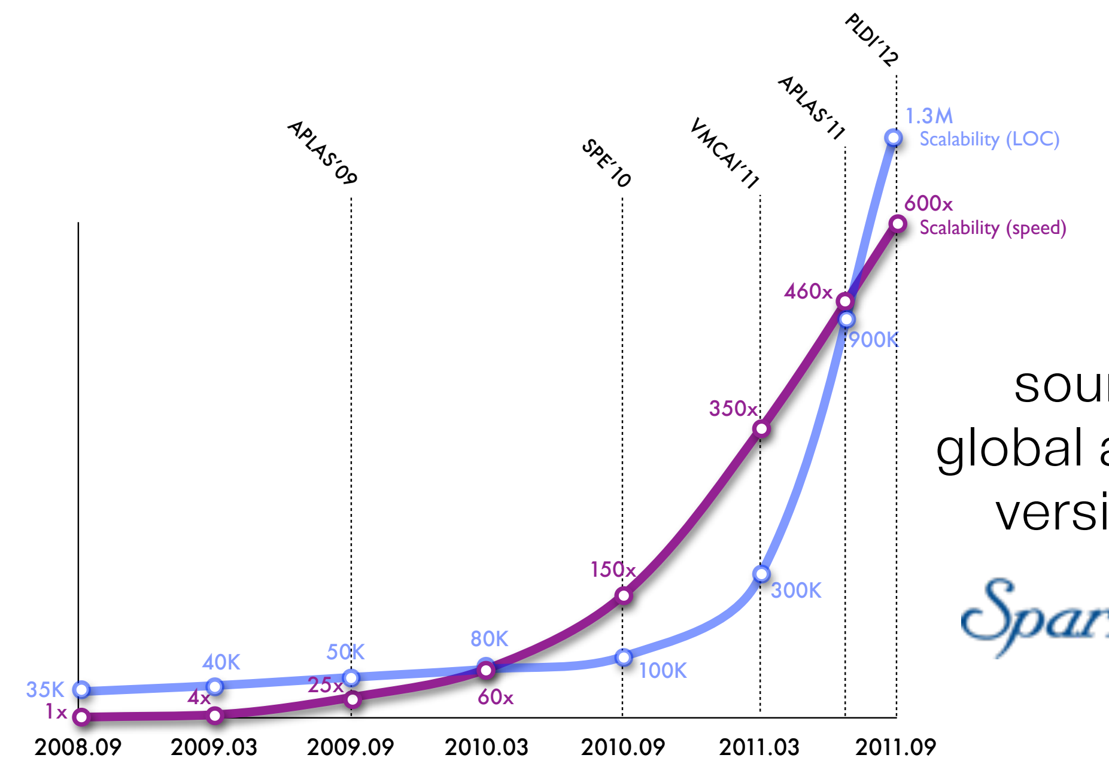
Towards Scalable Verified Validation of Static Analyzers

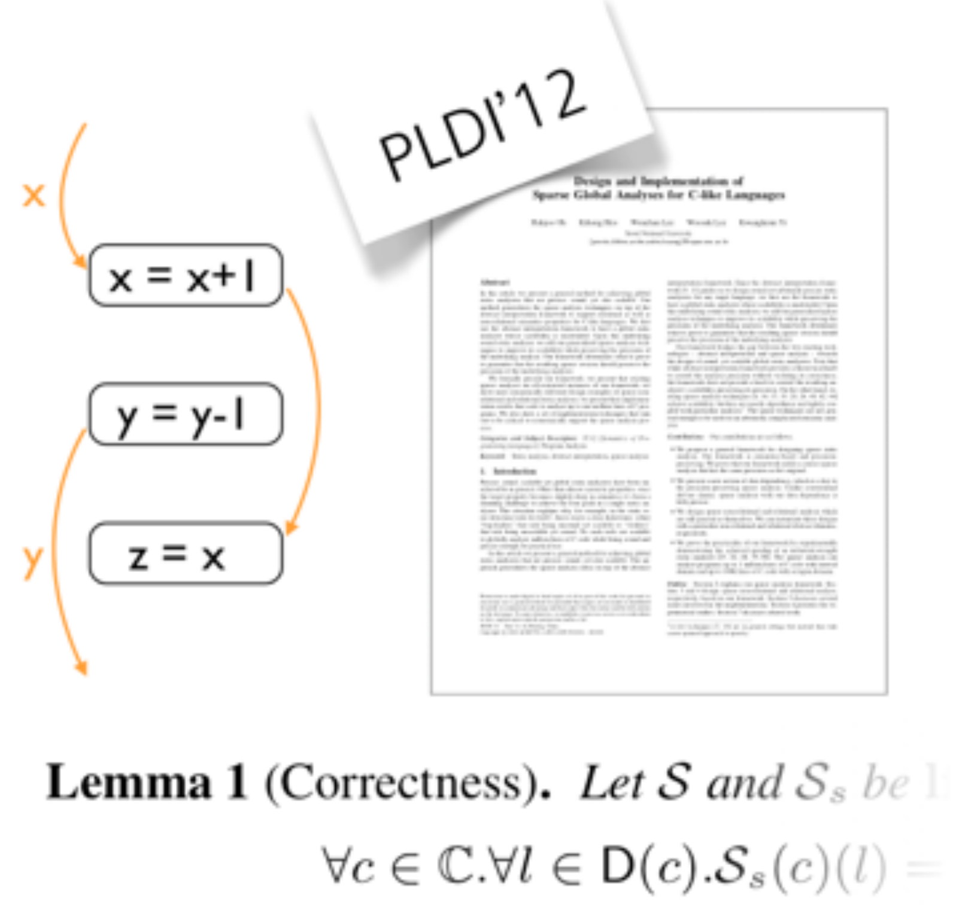
Jeehoon kang[1], Sungkeun Cho[1], Joonwon Choi[2], and Chung-Kil Hur[1]
[1] Seoul National University, Korea
[2] Massachusetts Institute of Technology, USA

Scalability has been improved.

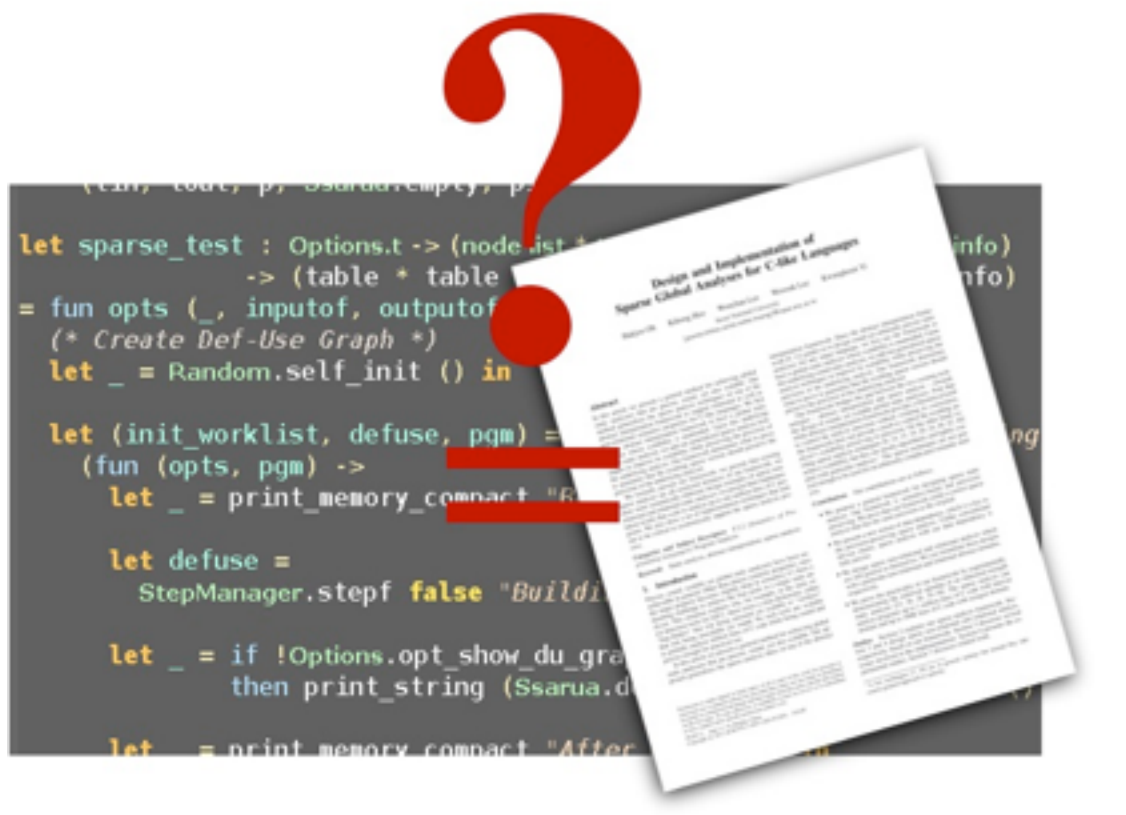


sound & global analysis version of Sparrow

Thanks to the correct analysis design.


x
y
x = x+1
y = y-1
z = x
PLDI'12
Lemma 1 (Correctness). Let $\mathcal{S}$ and $\mathcal{S}_s$ be
$\forall c \in \mathbb{C}. \forall l \in \mathrm{D}(c). \mathcal{S}_s(c)(l) =$

However, is the implementation correct?



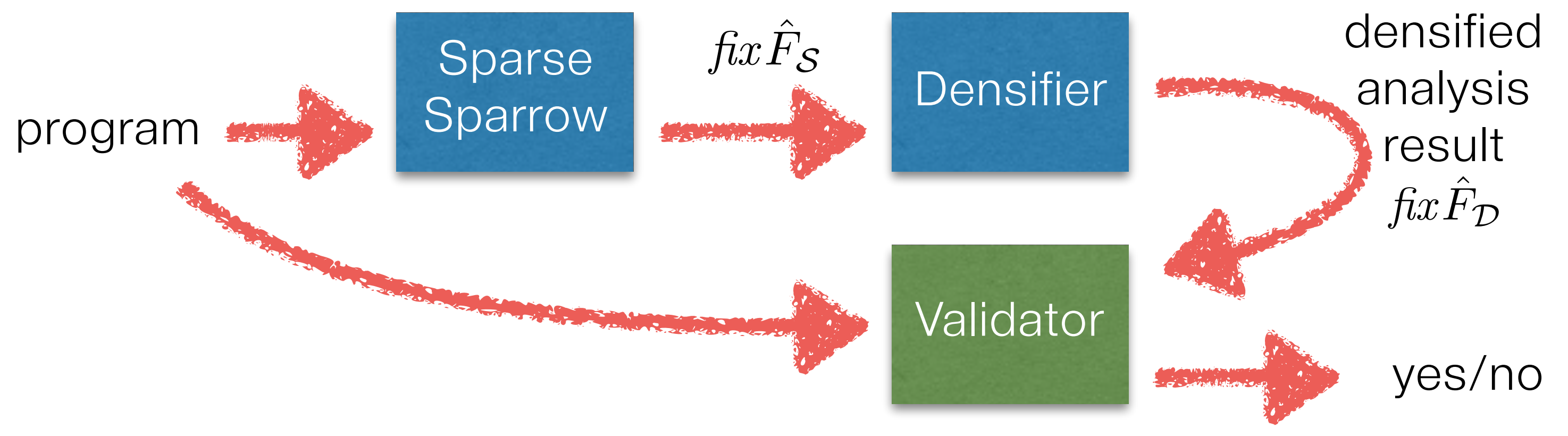
```
let sparse_test : Options.t -> (node list
            -> (table * table
= fun opts (_, inputof, outputof
  (* Create Def-Use Graph *)
  let _ = Random.self_init () in

  let (init_worklist, defuse, pgm) =
    (fun (opts, pgm) ->
      let _ = print_memory_compact "
      
      let defuse =
        StepManager.stepf false "Buildi

      let _ = if !Options.opt_show_du_gra
              then print_string (Ssarua.d
```

Static Analyzers as Verification Target

Static analyzers based on abstract interpretation are ideal target for formal software verification.

1. clear specification
2. importance of reliability
3. difficulties in testing
4. low verification cost by using translation validation

Big Picture



Trade-off

Development Scalability vs. Runtime Scalability

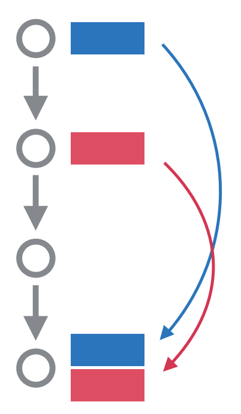
How to strike a balance between development cost and runtime cost of the validator?

Densifier verification approach: greatly reduces runtime cost of the validator, but high cost of proof.
Densifier validation approach: reduces proof effort, but not scalable. An early version of our validator was 100 times slower than the analysis time.

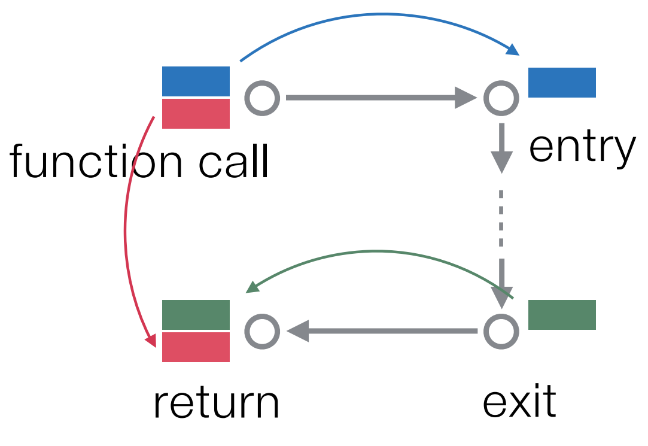
Hybrid approach(ours): splits the densifier into two ones.

Validated densifier
- low runtime(densification) cost
- high proof cost

Verified densifier
- low proof cost
- high runtime(densification) cost



e.g. sparse analysis



e.g. localization

Experiment Results

We found **13 bugs** from Sparrow.

Category	Found Bugs
drawing dependence graph	Dynamic locations were not included in a definition set when arrays are declared. Graph edges were not drawn correctly when weak-update occurs. Graph edges were not drawn correctly when an encoded library function is called. Graph edges for fields were not drawn correctly. Return edges should be definition points.
abstract semantics	Field values should be top if the struct itself is top. Local variables should not be removed on an exit node in some cases. Field values should not be declared as dynamic values. Typing errors on abstract interval operations. Zero and null worked inconsistently in some cases. Values from address-taken locations should not be removed on exit nodes. Weak update conditions for local variables were incorrect.
parser	Functions and local variables should be treated individually, even if their names are same.

Performance of the validator: times (in seconds) and memory consumptions (in megabytes) are represented for all benchmarks. The performance is evaluated for the analyzer that bugs are fixed by validation (**Analyzer**$_{\text{Fixed}}$).

Programs	LOC	Analyzer$_{\text{Fixed}}$		Validator					Cmp$_{\text{Time}}$	Cmp$_{\text{Mem}}$
		Time	Mem	Trs	Dns	Val	Time	Mem		
spell-1.0	2K	1.2	46	0.1	0.2	0.1	0.4	4	3.34 x	0.09 x
gzip-1.2.4a	7K	13	126	1	3	1	5	37	2.76 x	0.29 x
combine-0.3.3	11K	24	196	2	3	1	6	28	3.99 x	0.14 x
bc-1.06	13K	40	165	4	23	7	34	337	1.20 x	2.04 x
tar-1.13	20K	149	408	10	33	7	50	242	3.06 x	0.59 x
coan-4.2.2	22K	137	724	16	36	9	61	406	2.30 x	0.56 x
less-382	23K	280	479	45	133	24	201	718	1.43 x	1.50 x
make-3.76.1	27K	497	1299	30	106	10	146	496	3.49 x	0.38 x
cflow-1.3	34K	15	94	1	3	1	5	30	2.75 x	0.32 x
wget-1.9	35K	275	1041	24	51	8	83	458	3.43 x	0.44 x
screen-4.0.2	45K	1772	2899	184	389	28	600	1814	3.03 x	0.63 x
asn1c-0.9.21	50K	927	2185	76	320	96	493	2878	1.95 x	1.31 x
judy-1.0.5	87K	466	677	20	58	59	136	198	3.44 x	0.29 x
gsasl-1.6.1	91K	3493	754	828	342	82	1252	116	2.79 x	0.15 x
openssh-5.8p1	102K	4303	5485	1050	5060	650	6760	7308	0.66 x	1.33 x
lsh-2.0.4	111K	1714	2655	472	1972	461	2905	6768	0.62 x	2.55 x

on Linux 3.0, Intel quad-core 3.07GHz with 24GB memory

LOC: the number of lines of code, calculated with `wc`
Trs: the data translation time / **Dns**: the densification time
Val: the time for whole validations, including the prefixed point validation
Cmp$_{\text{Time}}$: how much the validator is faster than the analyzer
Cmp$_{\text{Mem}}$: how less the validator consumes memory than the analyzer

Verification cost: this project took 6 man-months, of which 5 man-months are done for proving the validator in Coq and 1 man-month for debugging the target analyzer once the validations failed.

ROSAEC center
Research On Software Analysis for Error-free Computing
SEOUL NATIONAL UNIVERSITY
Massachusetts Institute of Technology