SAT Solving in the Serverless Cloud

Alex Ozdemir*, Haoze Wu*, Clark Barrett

Stanford University

Background

- Cloud computing services
  - On-demand parallelism at lost cost.
  - Increasingly fine granularity and low latency.

amazon web services

salesforce

Azure

ORACLE

- Serverless computing (e.g., AWS Lambda)
  - Rapidly and plentifully provisioned at a low price.
  - Successfully scaled up many tasks:
    - video processing
    - neural network training
    - compiling large programs (e.g., gcc)

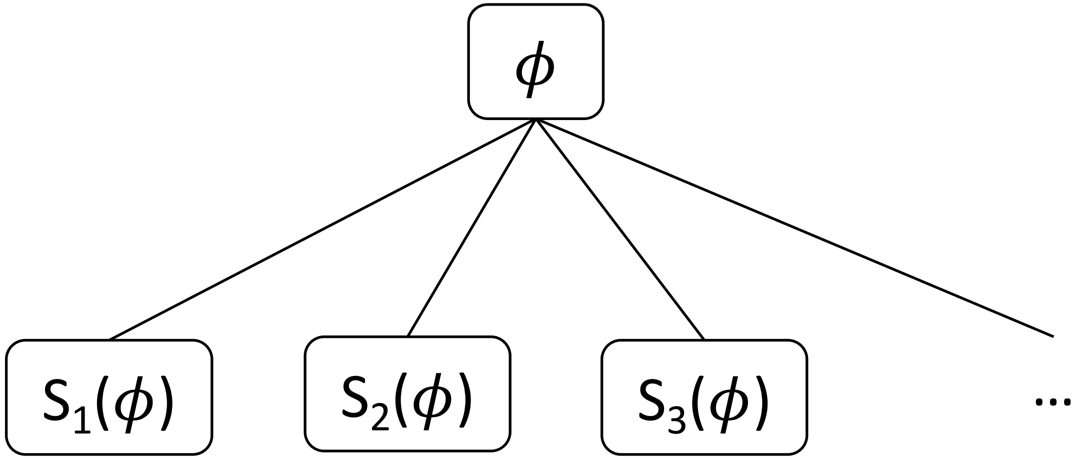
AWS Lambda

Google Cloud Functions

Azure Functions

Can serverless computing be leveraged for massively parallel SAT-solving?

Parallel SAT: the portfolio approach



Unsuitable for serverless cloud

- Short runtime (e.g., 15 minutes)
- Large number of executors
- Limited communications

Parallel SAT: Divide-and-Conquer

Solver S

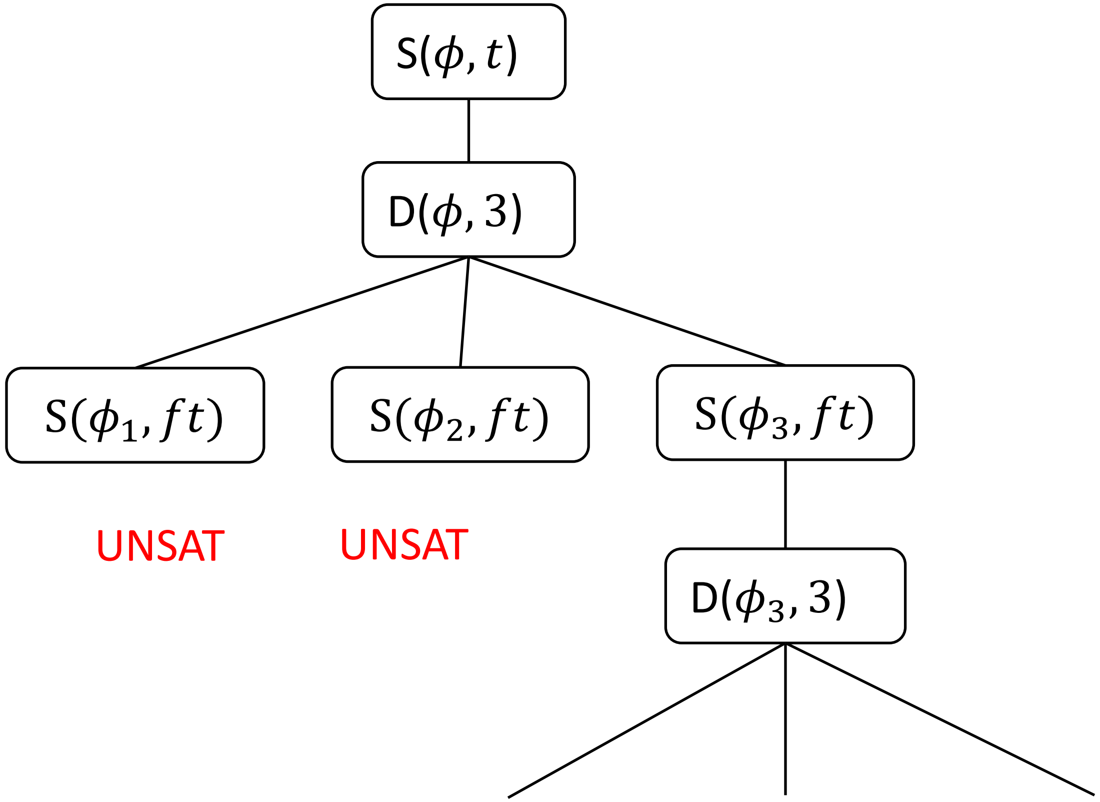
- solves a SAT (sub)-formula

Divider D

- splits a formula into (easier) sub-formulas.

Other parameters:

- Initial timeout t
- The timeout growth factor f
- The number of partitions



S(ϕ, t)

D(ϕ, 3)

S(ϕ_1, ft)

S(ϕ_2, ft)

S(ϕ_3, ft)

UNSAT

UNSAT

D(ϕ_3, 3)

Implementation

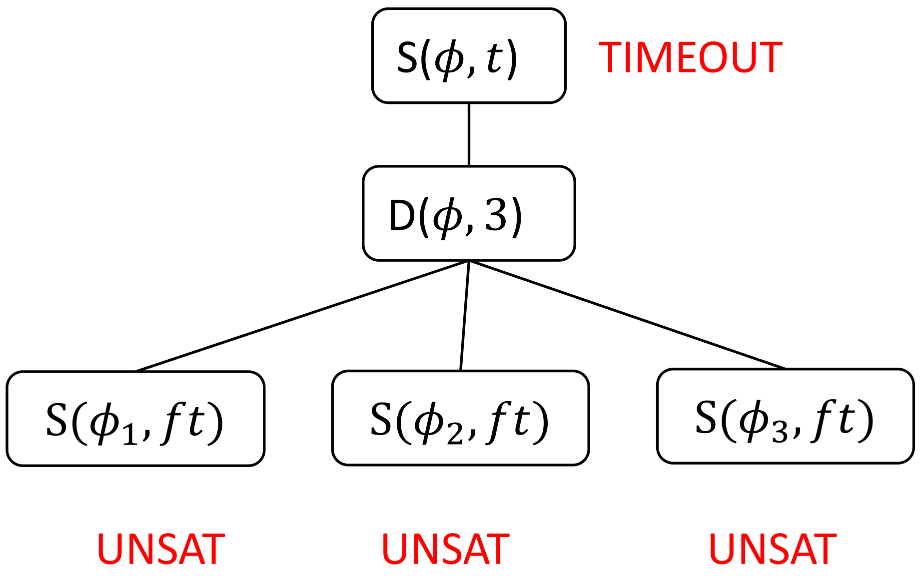
Use a CDCL solver (CaDiCaL) for solving and a lookahead solver (march) for splitting.

Use the gg framework for managing the D&C execution.

- Express the computation as a dependency graph of *thunks*.
- Thunks are executed by user-specified backends.
- Three types of thunks
  - Solve
  - Divide
  - Merge

A program that take files as inputs and outputs either files or new thunks.

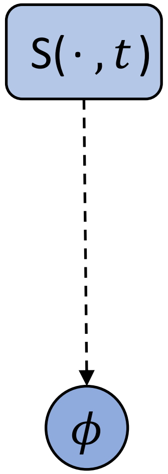
D&C search as gg dependency graph



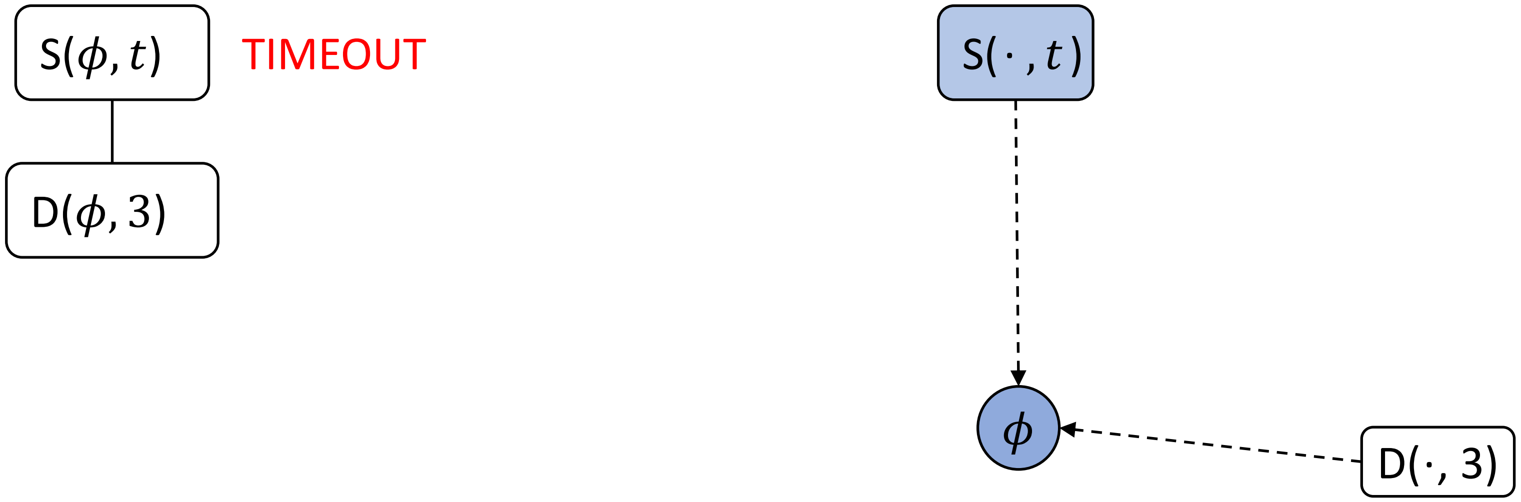
D&C search as gg dependency graph

D&C search as gg dependency graph

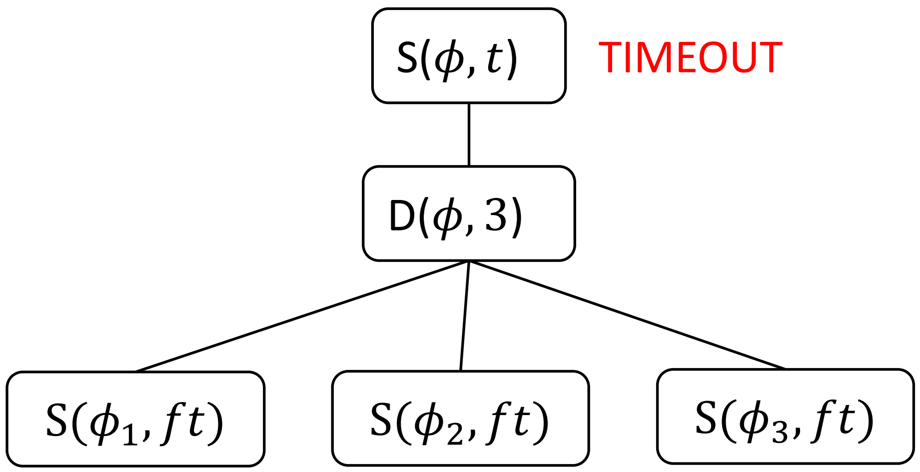
$S(\phi, t)$ TIMEOUT



D&C search as gg dependency graph



D&C search as gg dependency graph



S(ϕ, t)

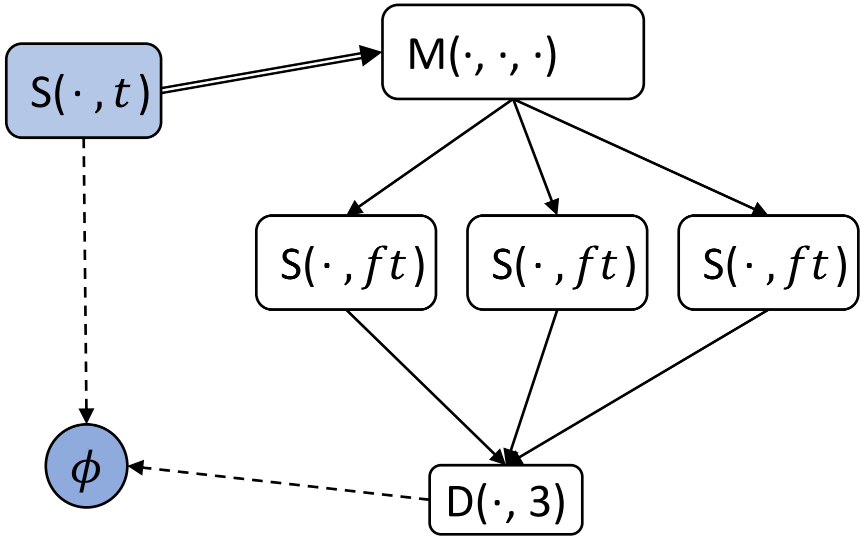
TIMEOUT

D(ϕ, 3)

S(ϕ_1, ft)

S(ϕ_2, ft)

S(ϕ_3, ft)



S($\cdot, t$)

M($\cdot, \cdot, \cdot$)

S($\cdot, ft$)

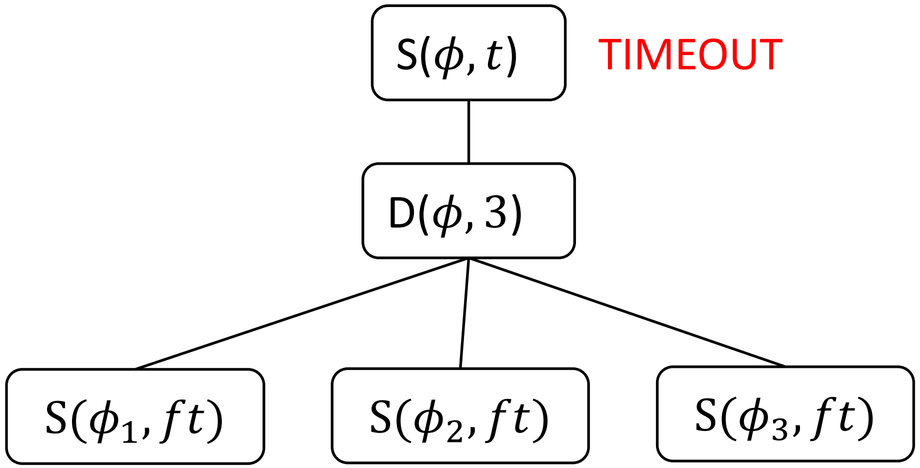
S($\cdot, ft$)

S($\cdot, ft$)

ϕ

D($\cdot$, 3)

D&C search as gg dependency graph



S(ϕ, t)

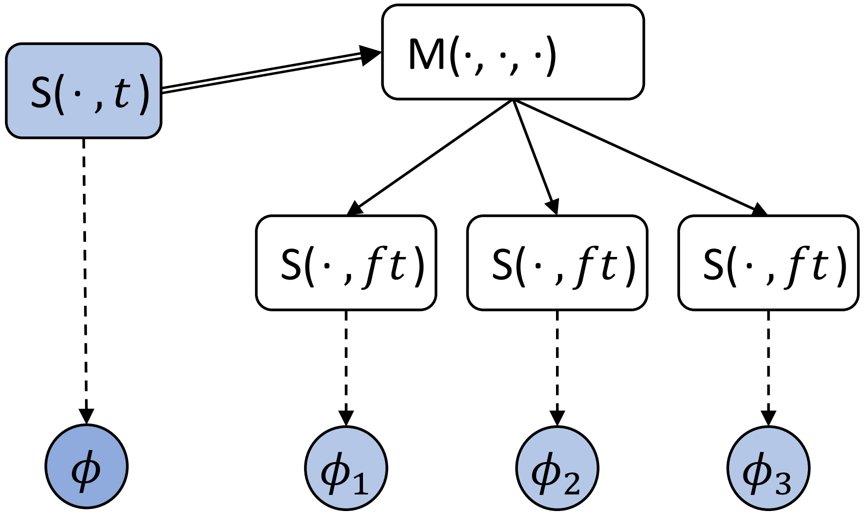
TIMEOUT

D($\phi, 3$)

S(ϕ_1, ft)

S(ϕ_2, ft)

S(ϕ_3, ft)



S($\cdot, t$)

M($\cdot, \cdot, \cdot$)

S($\cdot, ft$)

S($\cdot, ft$)

S($\cdot, ft$)

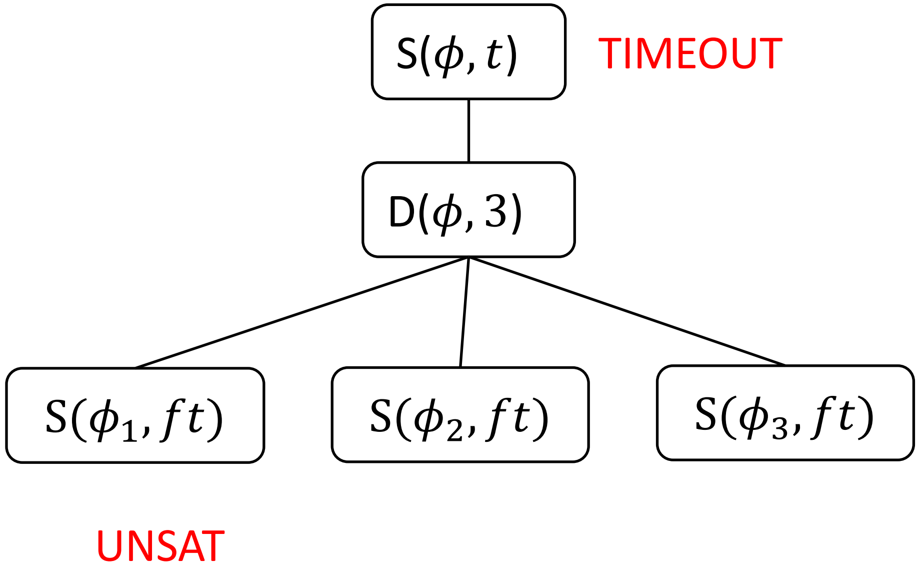
ϕ

ϕ_1

ϕ_2

ϕ_3

D&C search as gg dependency graph



S(ϕ, t)

TIMEOUT

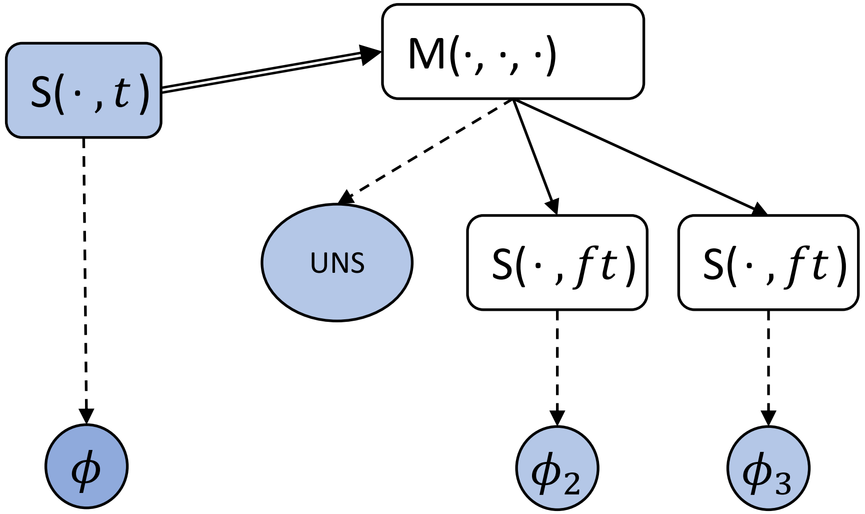
D(ϕ, 3)

S(ϕ_1, ft)

S(ϕ_2, ft)

S(ϕ_3, ft)

UNSAT



S($\cdot, t$)

M($\cdot, \cdot, \cdot$)

UNS

S($\cdot, ft$)

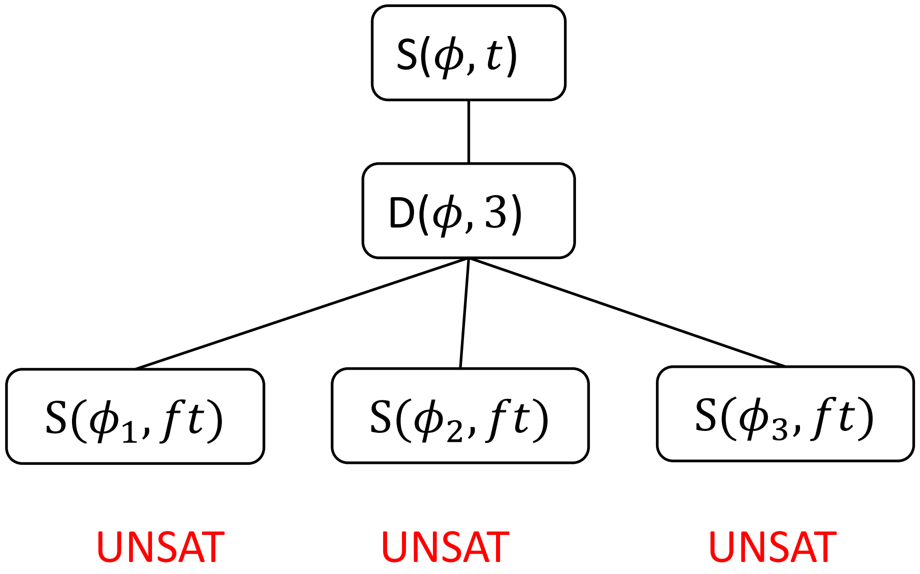
S($\cdot, ft$)

ϕ

ϕ_2

ϕ_3

D&C search as gg dependency graph



S(ϕ, t)

D($\phi, 3$)

S(ϕ_1, ft)

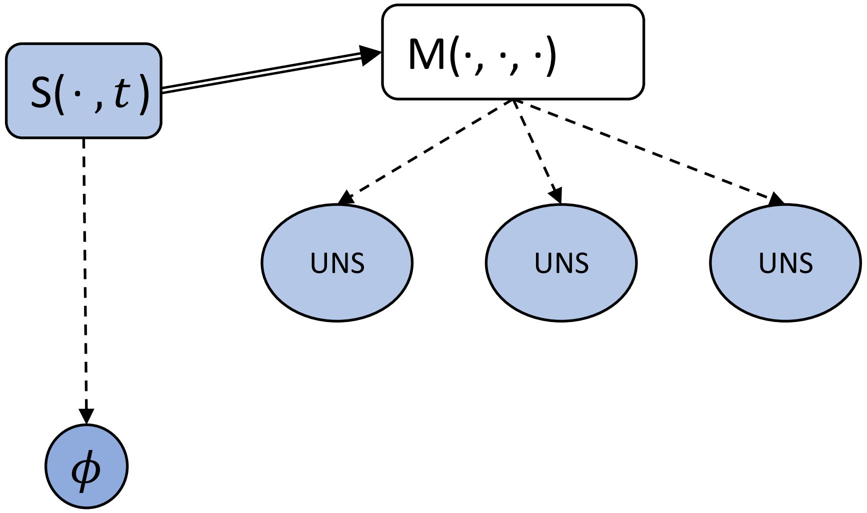
S(ϕ_2, ft)

S(ϕ_3, ft)

UNSAT

UNSAT

UNSAT



S($\cdot, t$)

M($\cdot, \cdot, \cdot$)

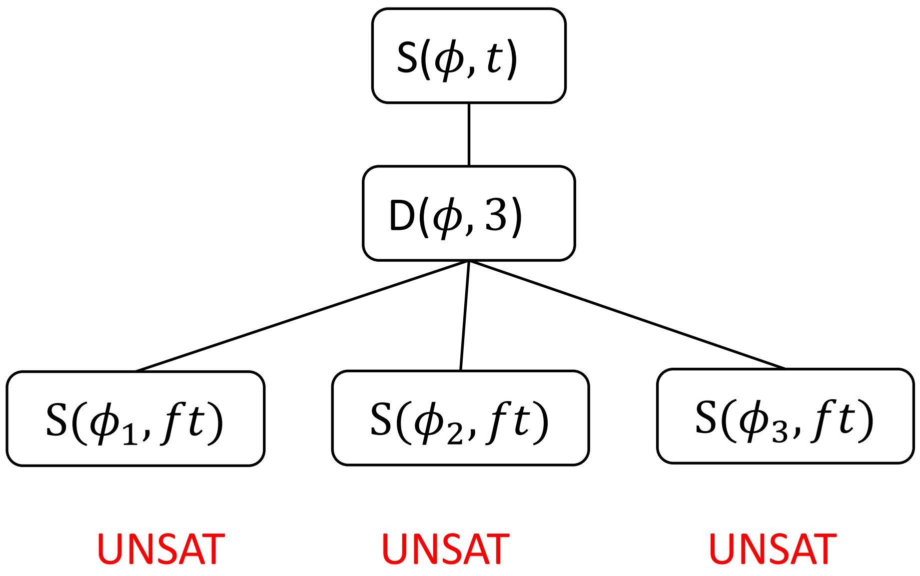
UNS

UNS

UNS

ϕ

D&C search as gg dependency graph



S(ϕ, t)

D($\phi, 3$)

S(ϕ_1, ft)

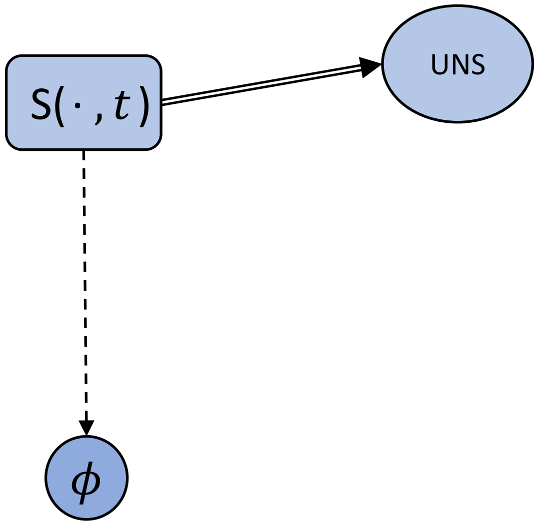
S(ϕ_2, ft)

S(ϕ_3, ft)

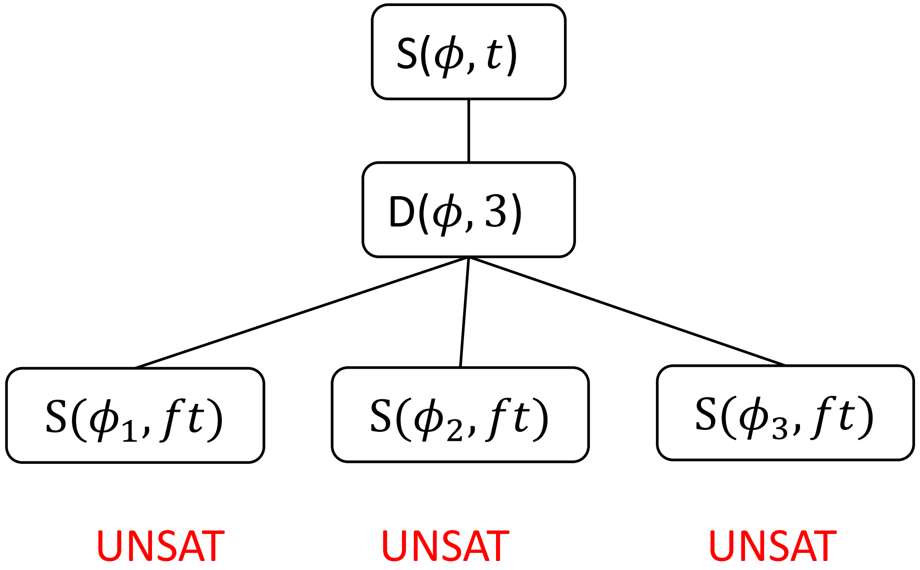
UNSAT

UNSAT

UNSAT



D&C search as gg dependency graph



S(ϕ, t)

D($\phi, 3$)

S(ϕ_1, ft)

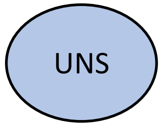
S(ϕ_2, ft)

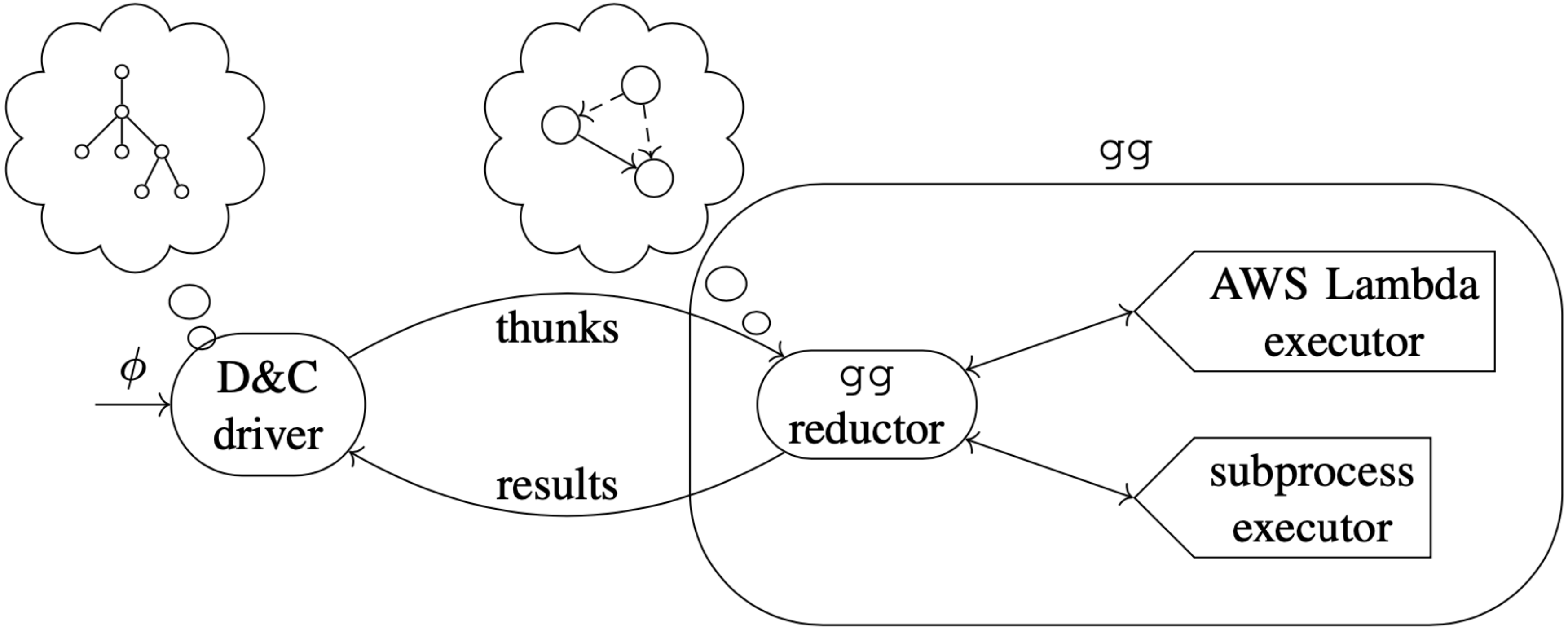
S(ϕ_3, ft)

UNSAT

UNSAT

UNSAT




ϕ
D&C driver
thunks
results
gg
gg reductor
AWS Lambda executor
subprocess executor

Experiments

Two experiments

Local Experiment

- Testbed:
  - Single host
  - Multithreaded (64)
- Moderately hard benchmarks
  - From Paracooba, CnC papers
- Systems
  - gg-SAT
  - Paracooba
  - CnC
  - Treengeling
  - Plingeling
- Question: **Is gg-SAT's algorithm reasonable?**

Serverless Experiment

- Primary Testbed:
  - 1000 AWS Lambda workers
- Hard benchmarks
  - Random unsolved instances from SAT'20
- Systems
  - gg-SAT / 1000x AWS Lambda (1hr)
  - All systems / 64x threads (4hrs)
- Question: **Is 1000-way parallelism useful?**

Local Experiment

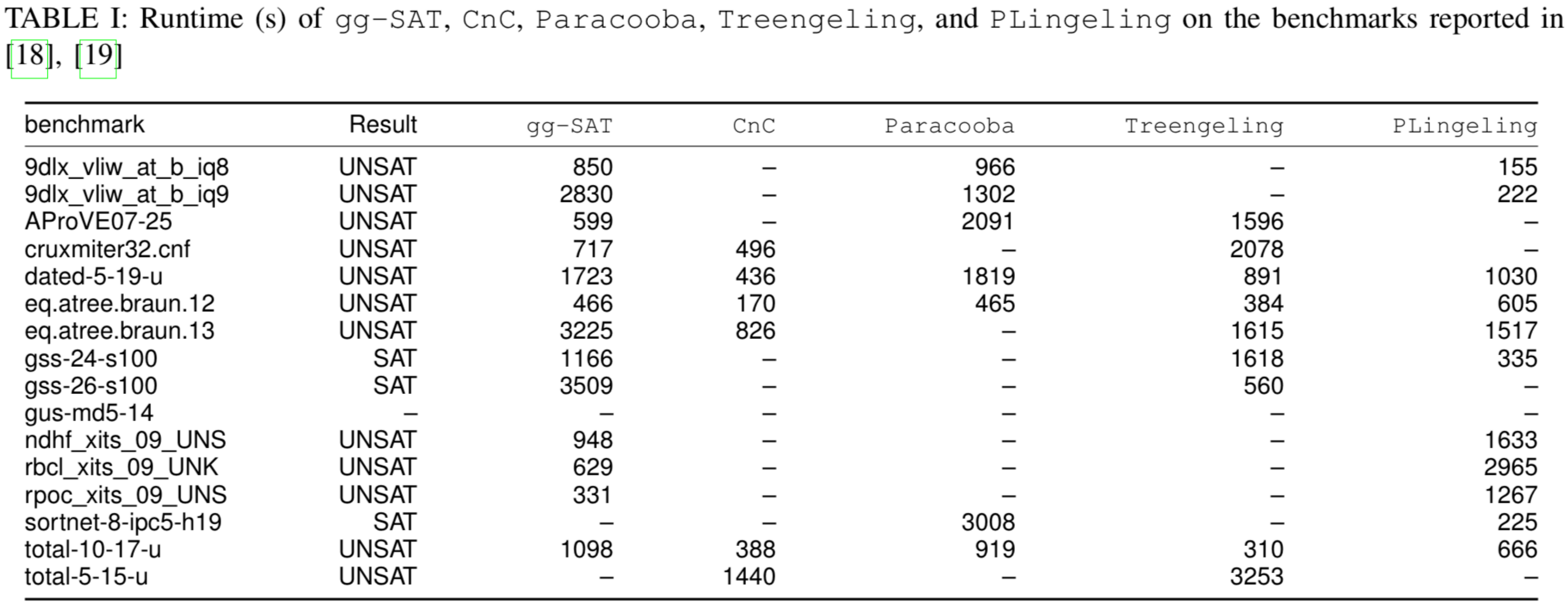


TABLE I: Runtime (s) of `gg-SAT`, `CnC`, `Paracooba`, `Treengeling`, and `PLingeling` on the benchmarks reported in [18], [19]

benchmark	Result	`gg-SAT`	`CnC`	`Paracooba`	`Treengeling`	`PLingeling`
9dlx_vliw_at_b_iq8	UNSAT	850	–	966	–	155
9dlx_vliw_at_b_iq9	UNSAT	2830	–	1302	–	222
AProVE07-25	UNSAT	599	–	2091	1596	–
cruxmiter32.cnf	UNSAT	717	496	–	2078	–
dated-5-19-u	UNSAT	1723	436	1819	891	1030
eq.atree.braun.12	UNSAT	466	170	465	384	605
eq.atree.braun.13	UNSAT	3225	826	–	1615	1517
gss-24-s100	SAT	1166	–	–	1618	335
gss-26-s100	SAT	3509	–	–	560	–
gus-md5-14	–	–	–	–	–	–
ndhf_xits_09_UNS	UNSAT	948	–	–	–	1633
rbcl_xits_09_UNK	UNSAT	629	–	–	–	2965
rpoc_xits_09_UNS	UNSAT	331	–	–	–	1267
sortnet-8-ipc5-h19	SAT	–	–	3008	–	225
total-10-17-u	UNSAT	1098	388	919	310	666
total-5-15-u	UNSAT	–	1440	–	3253	–

Serverless Experiment

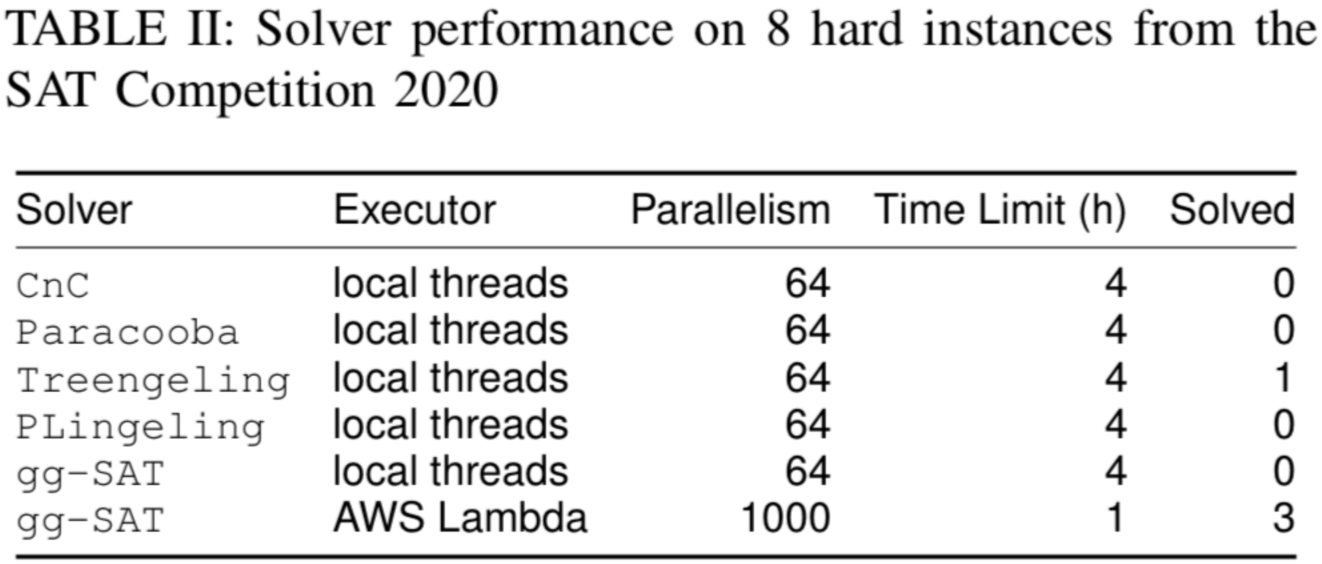


TABLE II: Solver performance on 8 hard instances from the SAT Competition 2020

Solver	Executor	Parallelism	Time Limit (h)	Solved
CnC	local threads	64	4	0
Paracooba	local threads	64	4	0
Treengeling	local threads	64	4	1
PLingeling	local threads	64	4	0
gg-SAT	local threads	64	4	0
gg-SAT	AWS Lambda	1000	1	3

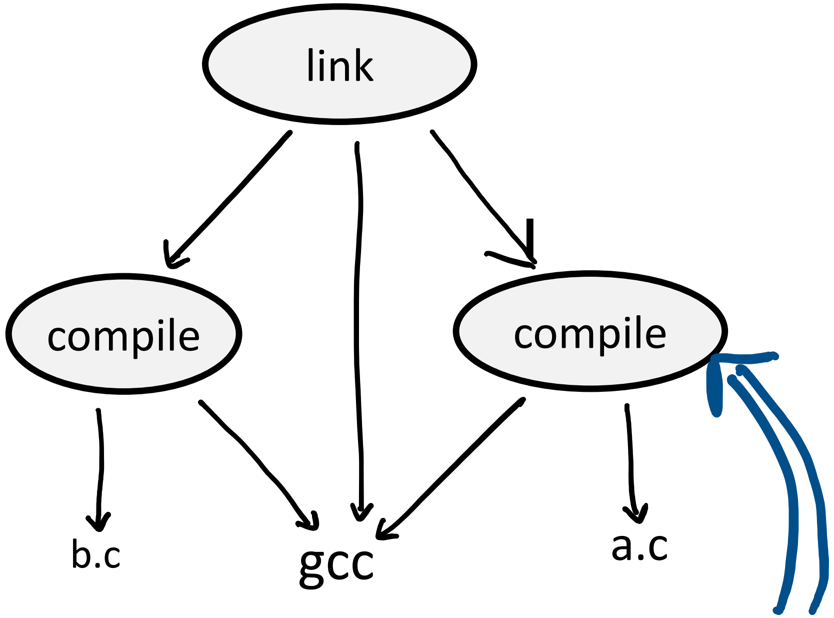
pygg

Improving the interface to gg

gg Interface

- Thunk representation
  - Structures serialized as files
  - Made using gg-create-thunk
  - Referenced with content hashes

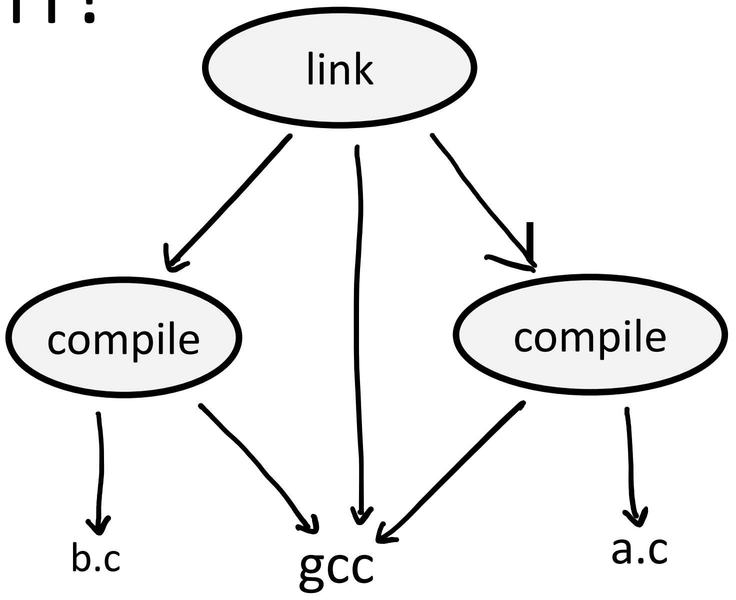
- Consequence: lots of glue code
  - Typically, shell scripts
  - Continuations are especially ☹



```
Thunk (simplified) {
  Command: [
    Va429b…5,
    “-o”,
    “out.o”,
    “-c”
    V98e10…e,
  ]
  Outputs: [“out.o”]
}
```

Graphs as function application?

- Could dependency graphs be built with **function application expressions**?



Goal:

```
Graph =
  gcc(“-o”, main,
    gcc(“-o”, “out”, “-c”, “a.c”).out,
    gcc(“-o”, “out”, “-c”, “b.c”).out)
```

pygg: Thunks as functions

```
import gg
@gg.thunk()
def add(a: gg.Value,
        b: gg.Value)
        -> gg.Value:
    A = int(a.as_str())
    B = int(b.as_str())
    return gg.str_value(
      str(A + B))
gg.main()
```

Essentially a file. Has:

- Contents
- Path
- Hash (if you’re curious!)

```
$ python init add \
   <(echo 1) <(echo 2)
$ gg-force thunk.out
…
$ cat out
3
```

pygg: Binary management

```
import gg
gg.install(“sum”)
@gg.thunk()
def add(a: gg.Value,
        b: gg.Value)
        -> gg.Value:
    subprocess.run(
      [gg.bin(“sum”),
       a.path(), b.path(),
       “-o”, “out”])
    return gg.file_value(“out”)
gg.main()
```

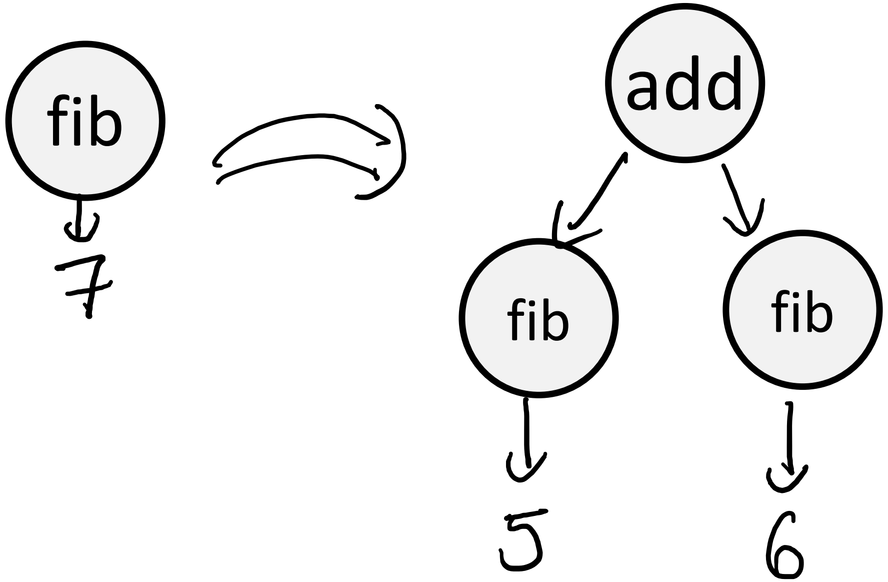
```
$ sum X Y –o SUM
```

Searches on PATH

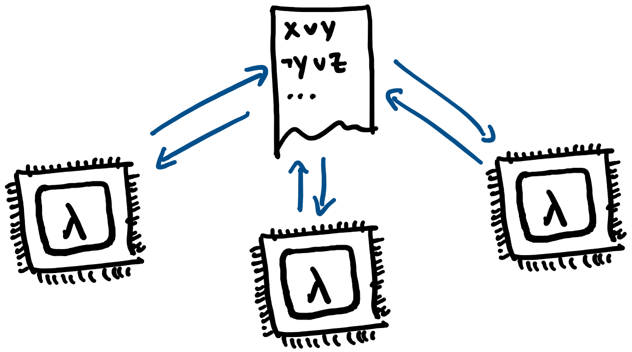
Gets installed binary

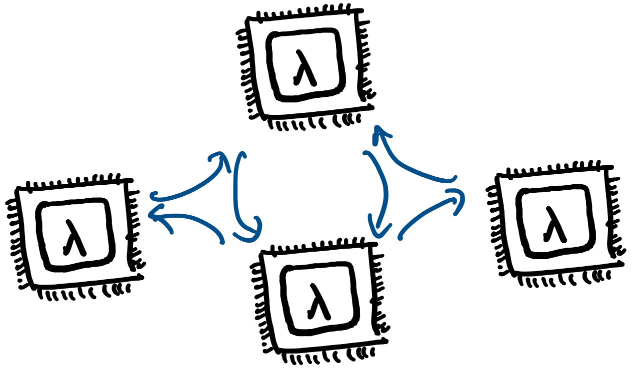
pygg: Continuations via deferred calls

```
import gg
@gg.thunk()
def fib(n: int):
  if n < 2:
    return gg.int_value(n)
  else:
    return gg.thunk(add,
      gg.thunk(fib, n – 1),
      gg.thunk(fib, n – 2))
gg.main()
```



Future Directions

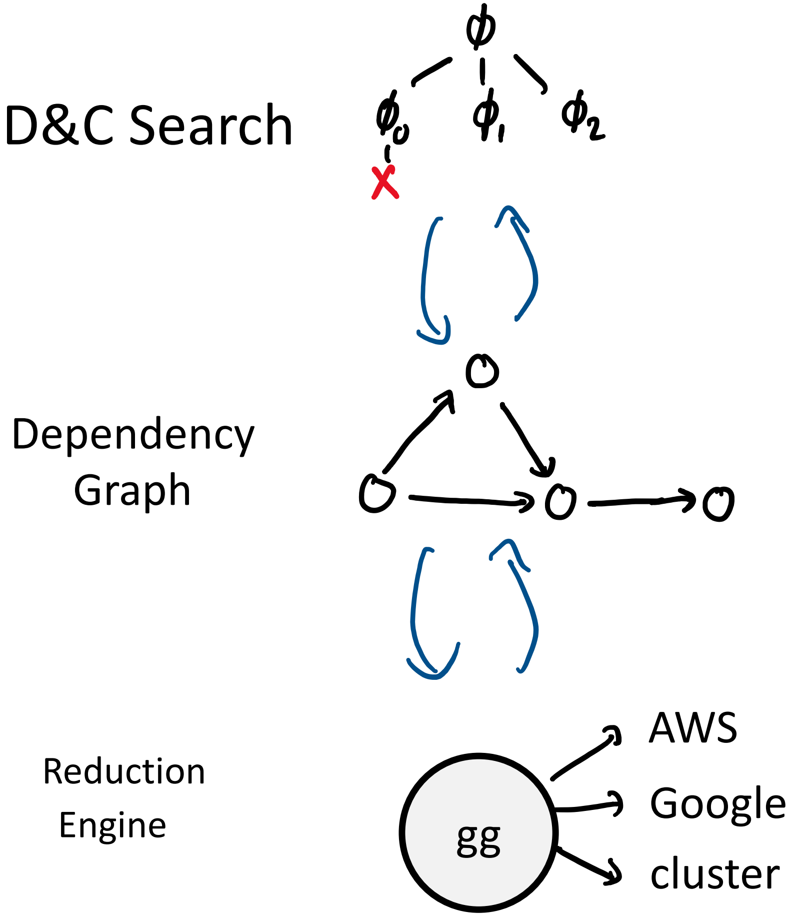




- gg & pygg: more automated reasoning?
  - Perhaps SMT?
  - Interesting relaxations to explore
    - Status Quo: purely functional computations
    - Previous additions: short-circuiting
    - Future additions:
      - Append-only log?
      - Gossip network?
      - …?
- Alternate serverless strategies
  - Save & restore

“SAT solving in the serverless cloud”

- D&C solving on serverless executors
- Tools:
  - gg-SAT
    - D&C SAT solver compatible with
      - Multithreading
      - A cluster
      - Serverless computing
  - Pygg
    - Ergonomic gg interface
    - github.com/gg-project



D&C Search

Dependency Graph

Reduction Engine

gg

AWS

Google

cluster