Extracting Configuration Knowledge from Build Files with Symbolic Analysis

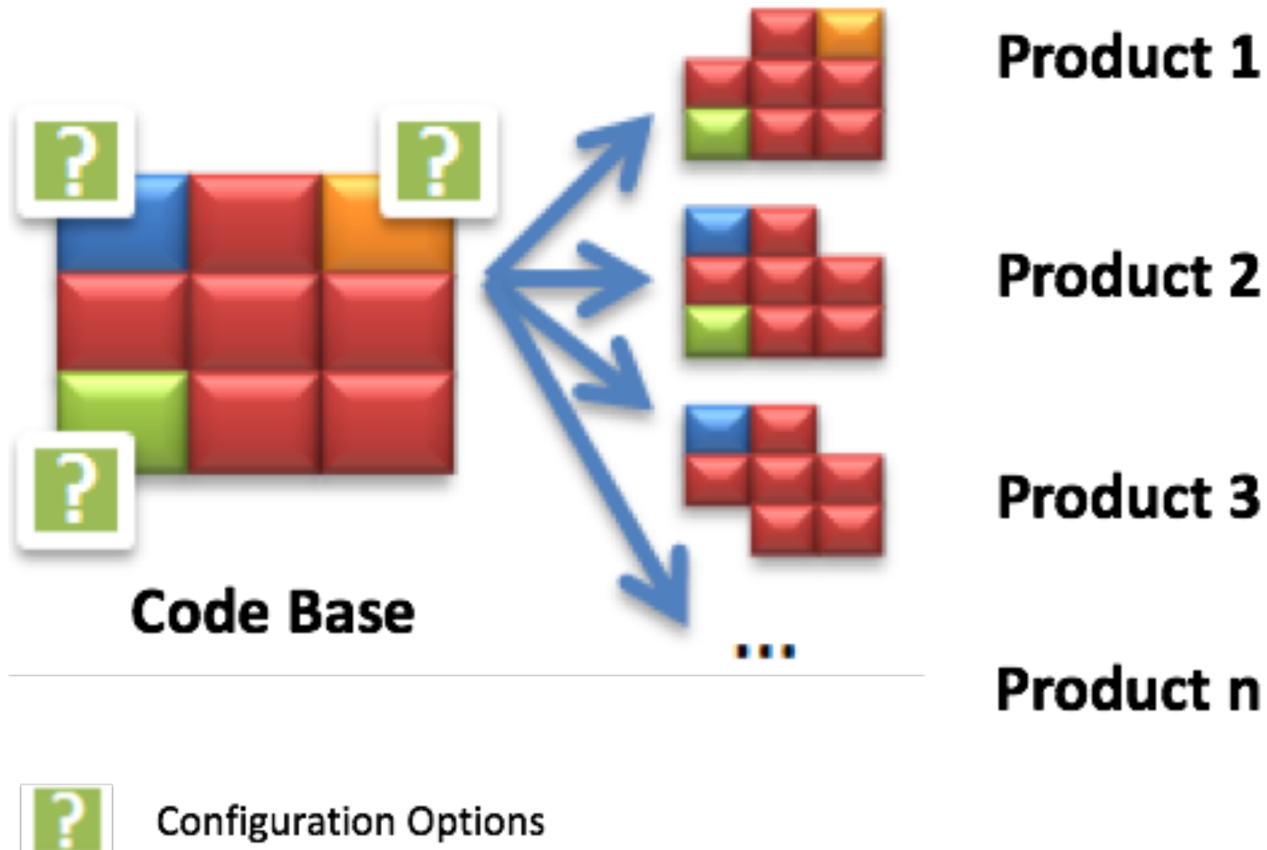
Shurui Zhou[1], Jafar Al-Kofahi[2], Tien N. Nguyen[2],
Christian Kästner[1], Sarah Nadi[3]
[1] Carnegie Mellon University
[2] Iowa State University
[3] Technische Universität Darmstadt

Carnegie Mellon University

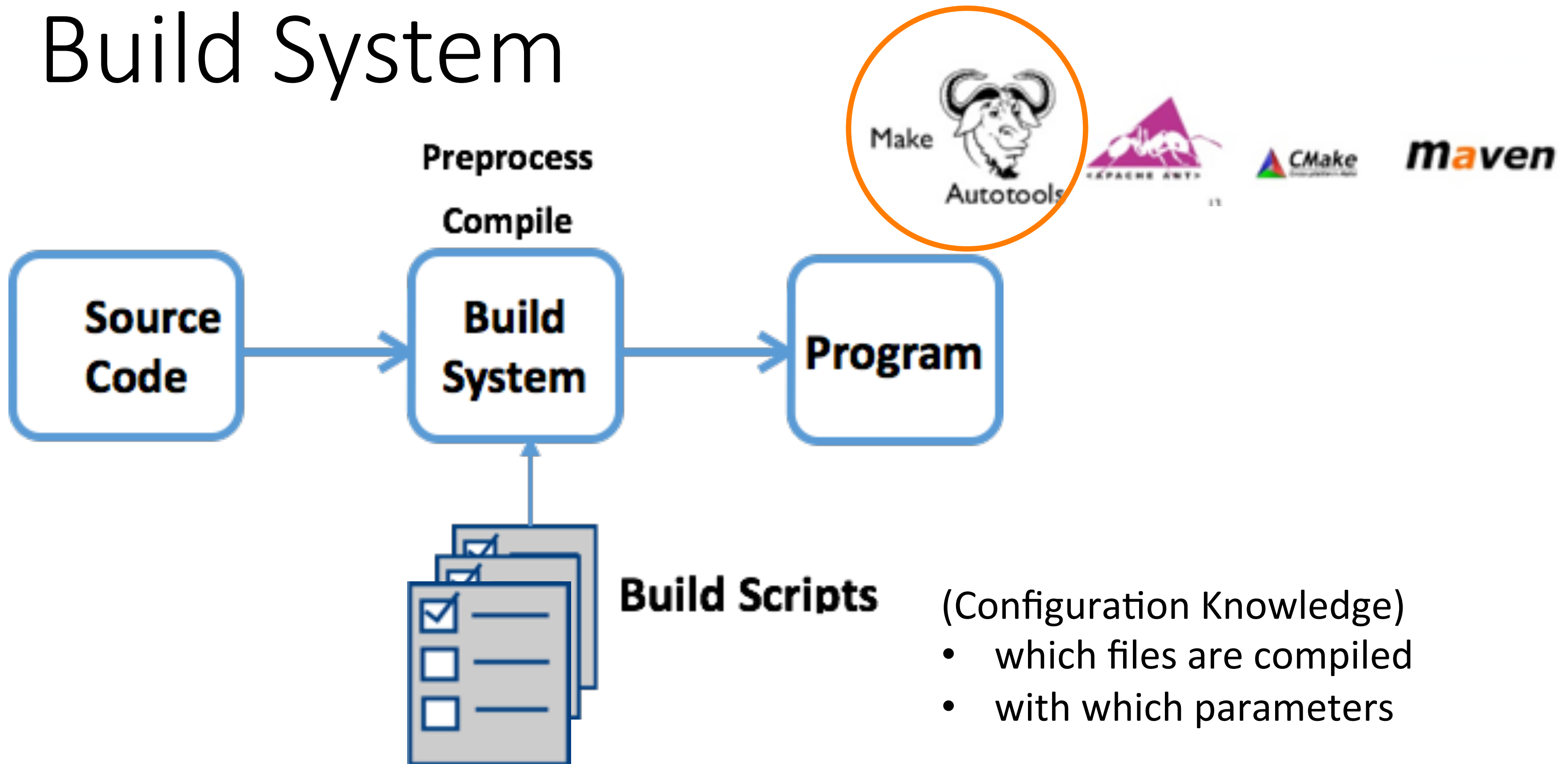
IOWA STATE UNIVERSITY

TECHNISCHE UNIVERSITÄT DARMSTADT

Highly Configurable System



Build System



(Configuration Knowledge)

- which files are compiled
- with which parameters

```
/*lib1.c*/  : foo(){}
/*lib2.c*/  : foo(){}
/*main.c*/
  #ifdef X
      main(){ foo();)
  #endif
```

File Presence Condition

lib1.c	IF	LIB1
lib2.c	IF	! LIB1

Additional Parameters

main.c	-UX

Make / Makefiles

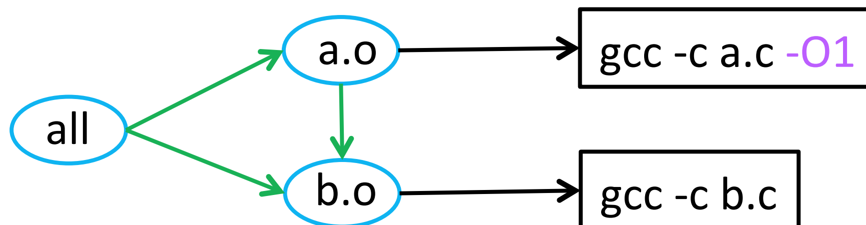
```
CFLAGS = -O
all : a.o  b.o
a.o : b.o
      gcc -c a.c  $(CFLAGS)
b.o :
      gcc –c b.c
```

Variables =

Target: Prerequisites

Command

Dependency Graph

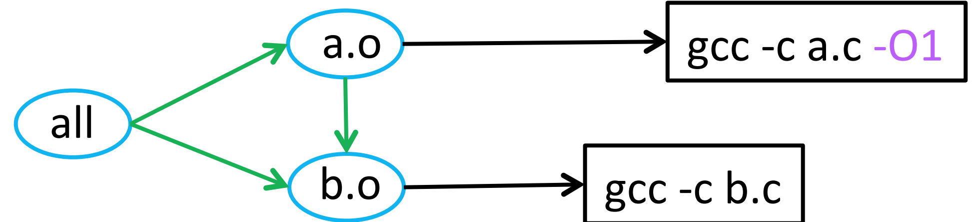


Conditional Directives of Make

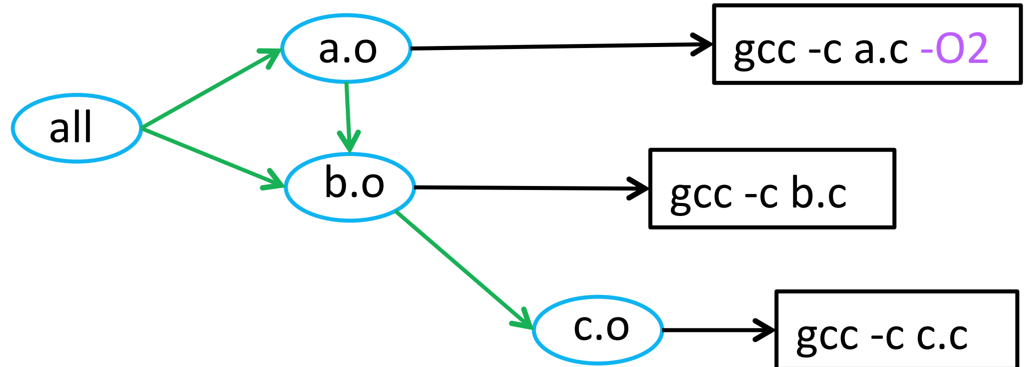
```
1   OS = $(shell uname)
2   ifeq ($(OS),Linux)
3     CFLAGS = -O1
4   else
5     CFLAGS = -O2
6   endif
7   all: a.o  b.o
8   a.o: b.o
9       gcc -c a.c $(CFLAGS)
10  b.o:
11      gcc –c b.c
12  ifdef X
13  b.o: c.o
14  endif
15  c.o:
16      gcc -c c.c
```

Dependency Graph

$(OS)=Linux ^ !X

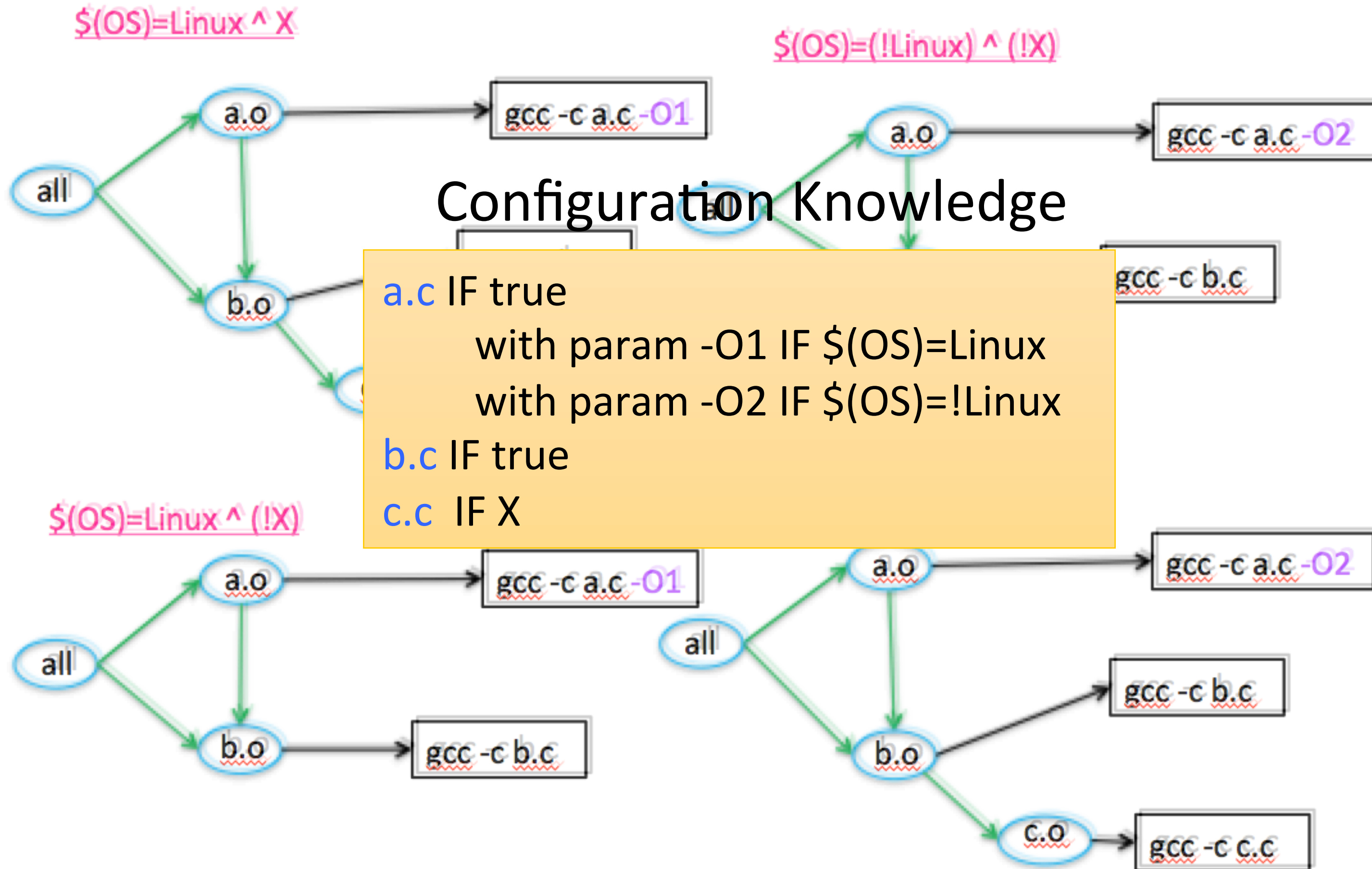


$(OS)=!Linux ^ X



Expect Result

Dependency Graph



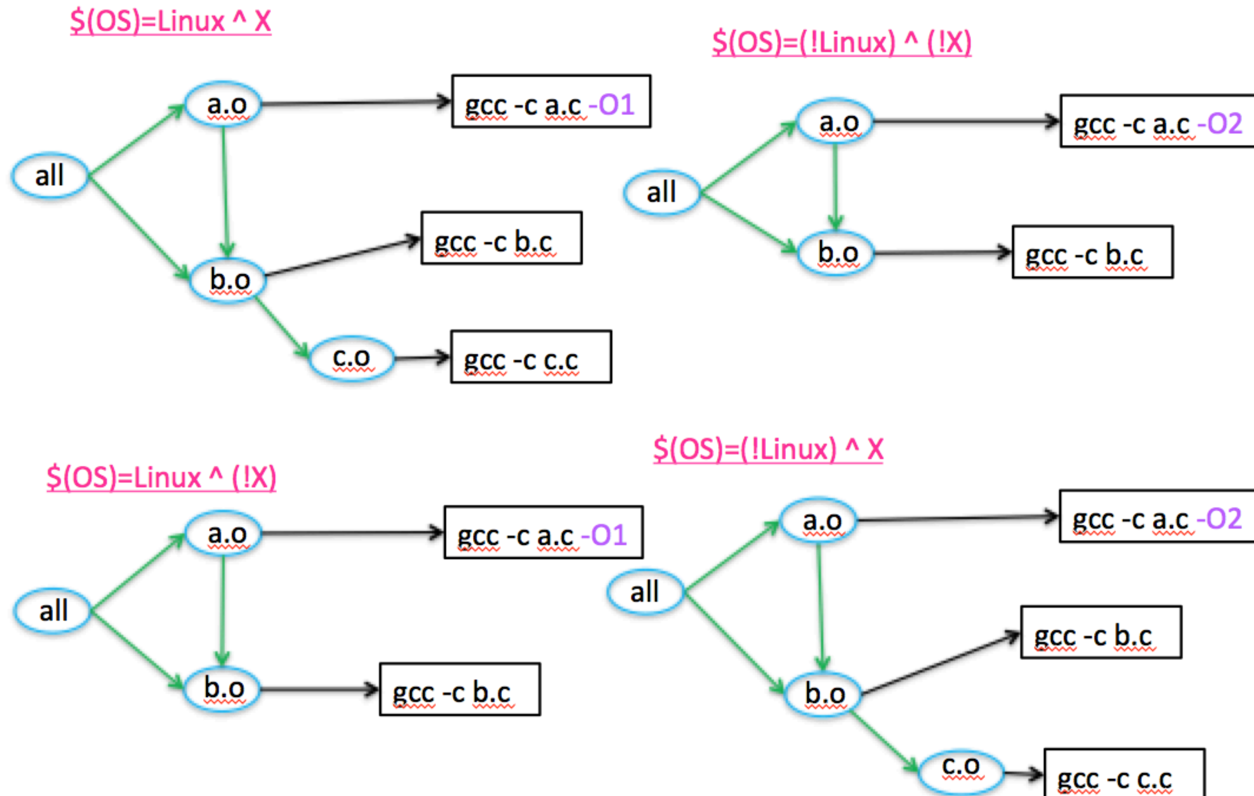
Configuration Knowledge

a.c IF true
 with param -O1 IF $(OS)=Linux
 with param -O2 IF $(OS)=!Linux
b.c IF true
c.c IF X

Challenge

- Arbitrary computations
- Explod

OS = $(shell uname)



Related Work

- Analyze Build Files
  - Dynamic

    Only analyze one configuration at a time

    [van der Burg et al, ASE'14 ,
    Milos Gligoric et al, OOPSLA'14,
    Jafar Al-Kofahi et al, ICSE'14]
  - Static [Ahmed Tamrawi et al, ICSE'12]

    SYMake : Symbolic Execution
- Extracting Variability Information
  - Dynamic
    - Sampling [Christian Dietrichet al, SPLC2012]

      imprecise
  - Static Approximation [Thorsten Berger et al,SPLC 2010, Sarah Nadi et al, 2014]

- Only fit for Linux kernel that build files follow certain patterns
- Ignoring variables

Our Approach

Step 1

Symbolically evaluate variables and collect rules

SYMake tool

- Analyzing all configurations at once
- Abstracting over unknown values

Step2

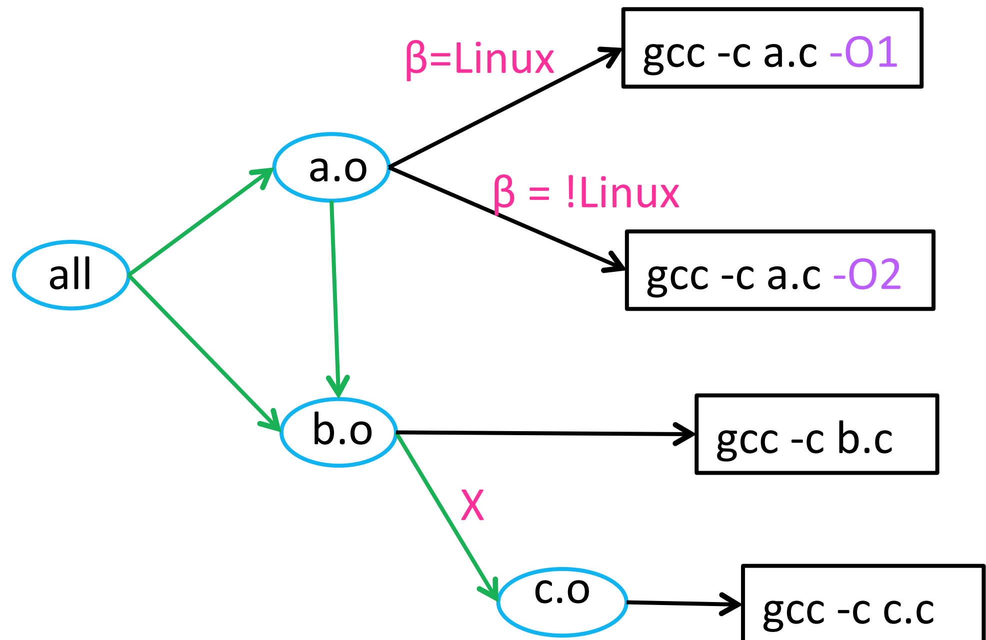
Extract Presence Conditions from graph

Step 1(SYMake)
Symbolically evaluate variables and collect rules

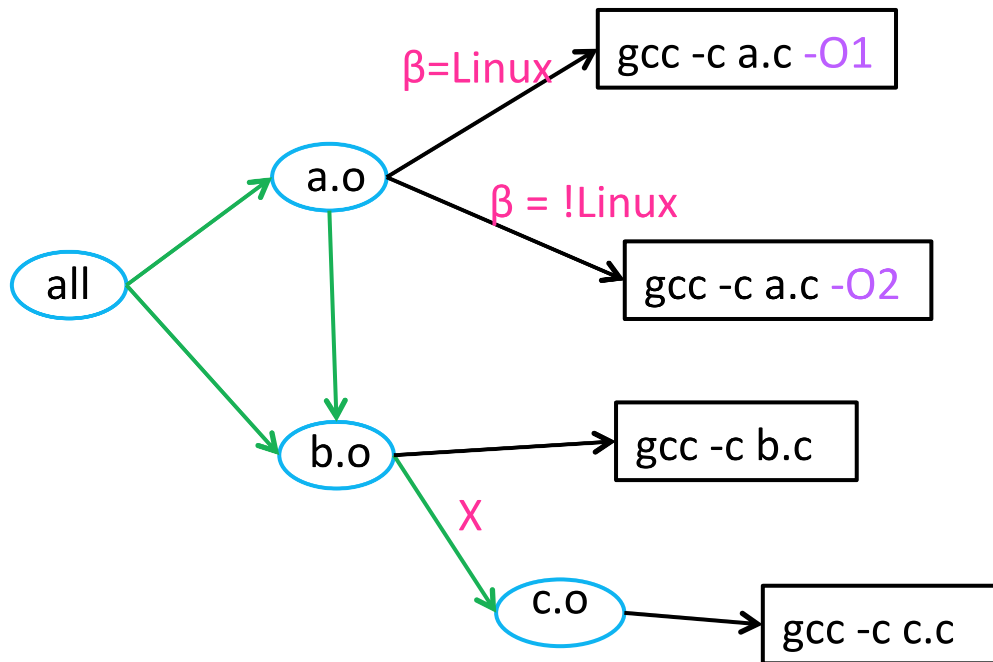
```
1   OS = $(shell uname)
2   ifeq ($(OS),Linux)
3       CFLAGS= -O1
4   else
5       CFLAGS= -O2
6   endif
7   all: a.o  b.o
8   a.o: b.o
9           gcc -c a.c $(CFLAGS)
10  b.o:
11          gcc –c b.c
12  ifdef X
13  b.o: c.o
14  endif
15  c.o:
16        gcc -c c.c
```

OS = β

CFLAGS = ITE(β=Linux, -O1, -O2)



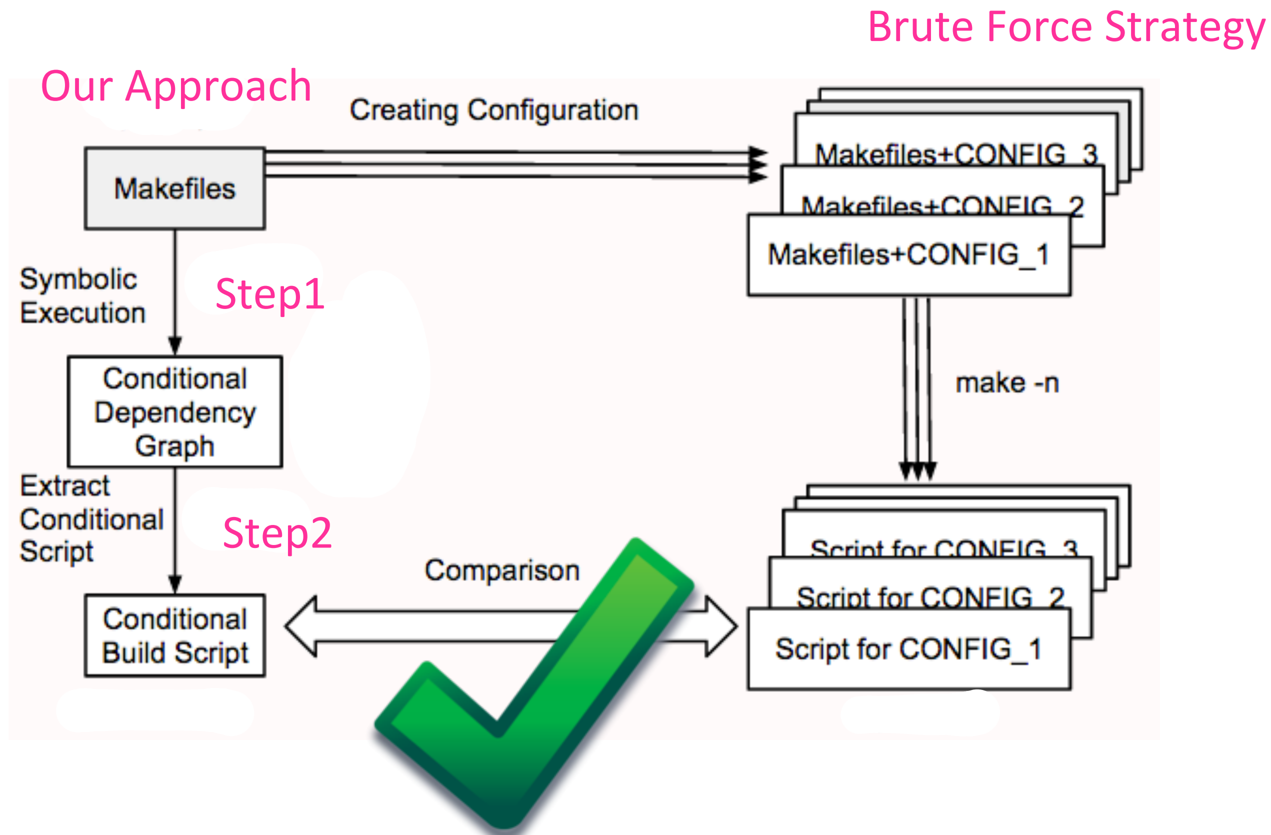
Step2
Extract Presence Conditions from graph



a.c IF true
  with param -O1 IF β = Linux
  with param -O2 IF β = !Linux
b.c IF true
c.c IF X

Implementation

- Extract configuration knowledge
- Test mechanism



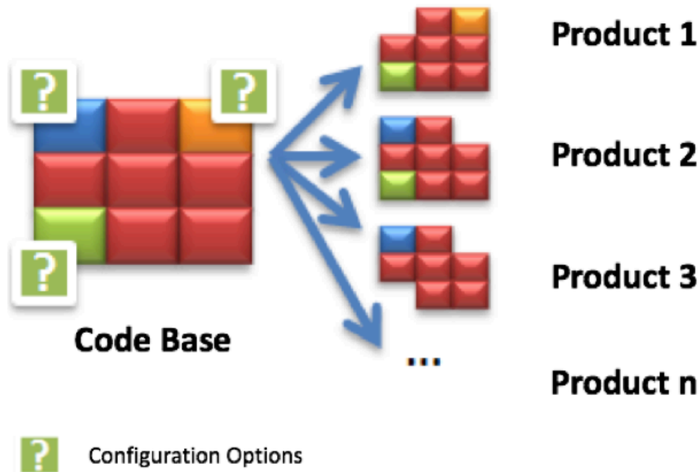
Limitation

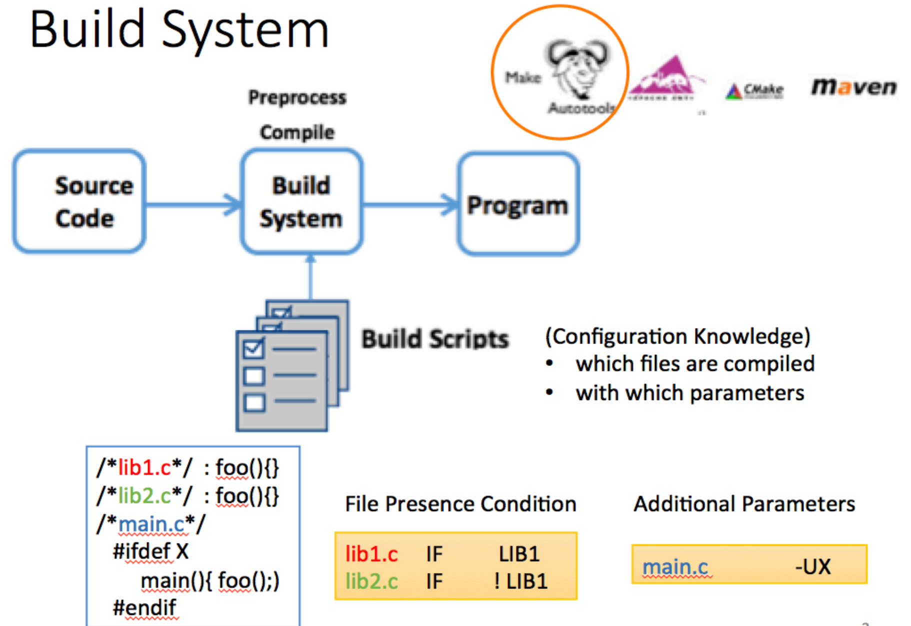
- Cannot deal with arbitrary symbolic values
- Do not support submake right now

Future Work

- Extend our analysis to support real-world systems
- Integrate with TypeChef

Highly Configurable System

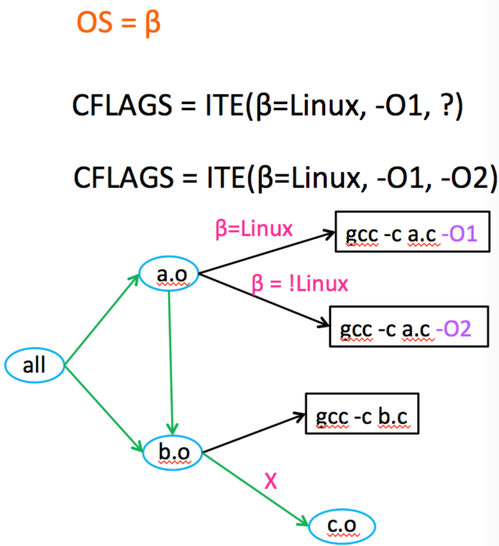




Step 1

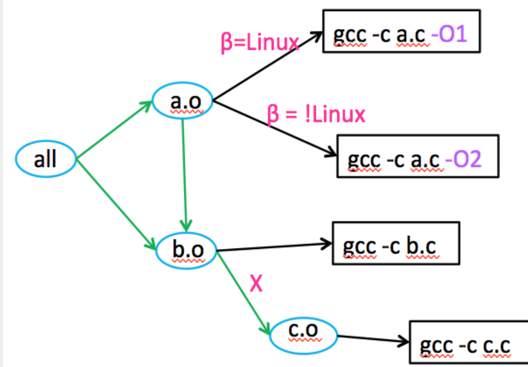
Symbolically evaluate variables and collect rules

```
1  OS = $(shell uname)
2  ifeq ($(OS),Linux)
3     CFLAGS= -O1
4  else
5     CFLAGS= -O2
6  endif
7  all: a.o b.o
8  a.o: b.o
9       gcc -c a.c $(CFLAGS)
10 b.o:
11      gcc –c b.c
12 ifdef X
13 b.o: c.o
14 endif
```



Step2

Extract Presence Conditions from graph



a.c IF true
    with param -O1 IF β = Linux
    with param -O2 IF β = !Linux
b.c IF true
c.c IF X

Discussion

- How analyzable are common build systems?
- How analyzable should they be?