CMPUT 663

Variability-aware Analysis

Sarah Nadi
University of Alberta
www.sarahnadi.org

[w/ slides by Christian Kästner & Sven Apel]

Variability = Complexity

33 optional, independent features

a unique variant for every

person on this planet

320 optional, independent features

more variants than estimated

atoms in the universe

Variability in Real-life

2,000 configurable options

12,000 configurable options

~2.2M apps

So how can we analyze/ test these products?

First.. What to Check For?

- Usually check for some particular specification
- A *specification* is typically a **global** property x that should hold for all variants. Examples:
  - Syntactically correct
  - Well-typed
  - No access to file system
  - Terminates within 10 seconds
  - …

Specification Example: No Syntax Errors

```
1. // Other definitions..
2. #ifdef SPLT
3. void png_handle_sPLT(){
4.     #ifdef POINTER
5.         png_sPLT_entryp p;
6.     #endif
7.     // Lines of code..
8.     #ifdef POINTER
9.         p = palette + i;
10.        p->red = *start++;
11.    #else
12.        p = new_palette;
13.        p[i].red = *start++;
14.    #endif
15.}
16.#endif
17.// More definitions..
```

Configuration 1

```
#define SPLT
#define POINTER
```

Configuration 2

```
#undef SPLT
#define POINTER
```

Configuration 3

```
#define SPLT
#undef POINTER
```

Configuration 4

```
#undef SPLT
#undef POINTER
```

✓ Compilation succeed ⊗ Compilation error

Specification Example: No Syntax Errors

```
1. // Other definitions..
2. #ifdef SPLT
3. void png_handle_sPLT(){
4.     #ifdef POINTER
5.         png_sPLT_entryp p;
6.     #endif
7.     // Lines of code..
8.     #ifdef POINTER
9.         p = palette + i;
10.        p->red = *start++;
11.    #else
12.        p = new_palette;
13.        p[i].red = *start++;
14.    #endif
15.}
16.#endif
17.// More definitions..
```

Configuration 1

```
#define SPLT
#define POINTER
```

Configuration 2

```
#undef SPLT
#define POINTER
```

Configuration 3

```
#define SPLT
#undef POINTER
```

Configuration 4

```
#undef SPLT
#undef POINTER
```

Compilation succeed Compilation error

Specification Example: No Syntax Errors

```
1. // Other definitions..
2. #ifdef SPLT
3. void png_handle_sPLT(){
4.     #ifdef POINTER
5.         png_sPLT_entryp p;
6.     #endif
7.     // Lines of code..
8.     #ifdef POINTER
9.         p = palette + i;
10.        p->red = *start++;
11.    #else
12.        p = new_palette;
13.        p[i].red = *start++;
14.    #endif
15.}
16.#endif
17.// More definitions..
```

Configuration 1

```
#define SPLT
#define POINTER
```

✓

Configuration 2

```
#undef SPLT
#define POINTER
```

✓

Configuration 3

```
#define SPLT
#undef POINTER
```

Configuration 4

```
#undef SPLT
#undef POINTER
```

✓ Compilation succeed ⊗ Compilation error

Specification Example: No Syntax Errors

```
1. // Other definitions..
2. #ifdef SPLT
3. void png_handle_sPLT(){
4.     #ifdef POINTER
5.         png_sPLT_entryp p;
6.     #endif
7.     // Lines of code..
8.     #ifdef POINTER
9.         p = palette + i;
10.        p->red = *start++;
11.    #else
12.        p = new_palette;
13.        p[i].red = *start++;
14.    #endif
15.}
16.#endif
17.// More definitions..
```

Configuration 1

```
#define SPLT
#define POINTER
```

Configuration 2

```
#undef SPLT
#define POINTER
```

Configuration 3

```
#define SPLT
#undef POINTER
```

Configuration 4

```
#undef SPLT
#undef POINTER
```

Compilation succeed Compilation error

Specification Example: No Syntax Errors

```
1. // Other definitions..
2. #ifdef SPLT
3. void png_handle_sPLT(){
4.     #ifdef POINTER
5.         png_sPLT_entryp p;
6.     #endif
7.     // Lines of code..
8.     #ifdef POINTER
9.         p = palette + i;
10.        p->red = *start++;
11.    #else
12.        p = new_palette;
13.        p[i].red = *start++;
14.    #endif
15.}
16.#endif
17.// More definitions..
```

Configuration 1

```
#define SPLT
#define POINTER
```

Configuration 2

```
#undef SPLT
#define POINTER
```

Configuration 3

```
#define SPLT
#undef POINTER
```

Configuration 4

```
#undef SPLT
#undef POINTER
```

Compilation succeed Compilation error

Challenge: how to test ***all*** *configurations?*

Brute Force

One Configuration At a Time

Product Configuration

Conventional Analysis

One Configuration At a Time

Product Configuration

Conventional Analysis

Analyze all, make statements about all

320 optional, independent features

more variants than estimated

atoms in the universe

Sampling

Sampling

Product Configuration

Conventional Analysis

Sampling

Product Configuration

Analyze few, make statements about all

Conventional Analysis

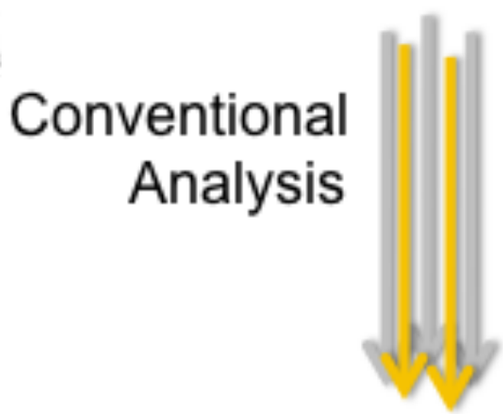
Sampling

Product Configuration

Analyze few, make statements about all

- Check random configurations
- Check representative configurations
- Check maximum configuration
- Check configs used by customers
- Combinatorial interaction testing (pairwise etc.)
- Code coverage heuristics

Conventional Analysis



Various Sampling Strategies

```
#ifdef A
  // code 1
#endif

#ifdef B
  // code 2
#else
  // code 3
#endif

#ifdef C
  // code 4
#endif
```

pair-wise

config-1:	!A	!B	C
config-2:	!A	B	!C
config-3:	A	!B	!C
config-4:	A	B	C

one-enabled

config-1:	A	!B	!C
config-2:	!A	B	!C
config-3:	!A	!B	C

one-disabled

config-1:	!A	B	C
config-2:	A	!B	C
config-3:	A	B	!C

most-enabled-disabled

config-1:	A	B	C
config-2:	!A	!B	!C

statement-coverage

config-1:	A	B	C
config-2:	A	!B	C

[Medeiros et al., *A Comparison of 10 Sampling Algorithms for Configurable Systems.* ICSE '16]

Various Sampling Strategies

```
#ifdef A
  // code 1
#endif

#ifdef B
  // code 2
#else
  // code 3
#endif

#ifdef C
  // code 4
#endif
```

pair-wise

config-1:	!A	!B
config-2:	!A	B
config-3:	A	!B
config-4:	A	B

one-enabled

config-1:	A	!B	!C
config-2:	!A	B	!C
config-3:	!A	!B	C

one-disabled

config-1:	!A	B	C
config-2:	A	!B	C
config-3:	A	B	!C

most-enabled-disabled

config-1:	A	B	C
config-2:	!A	!B	!C

statement-coverage

config-1:	A	B	C
config-2:	A	!B	C

[Medeiros et al., *A Comparison of 10 Sampling Algorithms for Configurable Systems.* ICSE '16]

Various Sampling Strategies

```
#ifdef A
  // code 1
#endif

#ifdef B
  // code 2
#else
  // code 3
#endif

#ifdef C
  // code 4
#endif
```

pair-wise

config-1:	!B	C
config-2:	B	!C
config-3:	!B	!C
config-4:	B	C

one-enabled

config-1:	A	!B	!C
config-2:	!A	B	!C
config-3:	!A	!B	C

one-disabled

config-1:	!A	B	C
config-2:	A	!B	C
config-3:	A	B	!C

most-enabled-disabled

config-1:	A	B	C
config-2:	!A	!B	!C

statement-coverage

config-1:	A	B	C
config-2:	A	!B	C

[Medeiros et al., *A Comparison of 10 Sampling Algorithms for Configurable Systems.* ICSE '16]

Various Sampling Strategies

```
#ifdef A
  // code 1
#endif

#ifdef B
  // code 2
#else
  // code 3
#endif

#ifdef C
  // code 4
#endif
```

pair-wise

config-1:	!A		C
config-2:	!A		!C
config-3:	A		!C
config-4:	A		C

one-enabled

config-1:	A	!B	!C
config-2:	!A	B	!C
config-3:	!A	!B	C

one-disabled

config-1:	!A	B	C
config-2:	A	!B	C
config-3:	A	B	!C

most-enabled-disabled

config-1:	A	B	C
config-2:	!A	!B	!C

statement-coverage

config-1:	A	B	C
config-2:	A	!B	C

[Medeiros et al., *A Comparison of 10 Sampling Algorithms for Configurable Systems.* ICSE '16]

Various Sampling Strategies

```
#ifdef A
  // code 1
#endif

#ifdef B
  // code 2
#else
  // code 3
#endif

#ifdef C
  // code 4
#endif
```

pair-wise

config-1:	!A	!B	C
config-2:	!A	B	!C
config-3:	A	!B	!C
config-4:	A	B	C

one-enabled

config-1:	A	!B	!C
config-2:	!A	B	!C
config-3:	!A	!B	C

one-disabled

config-1:	!A	B	C
config-2:	A	!B	C
config-3:	A	B	!C

most-enabled-disabled

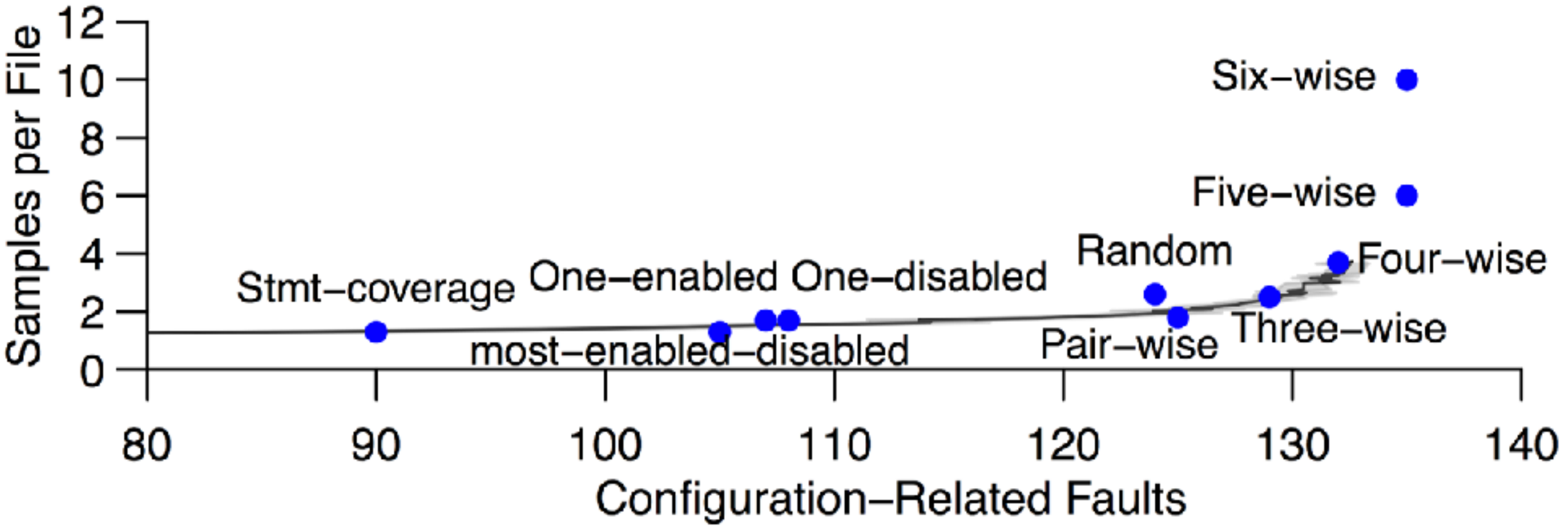
config-1:	A	B	C
config-2:	!A	!B	!C

statement-coverage

config-1:	A	B	C
config-2:	A	!B	C

[Medeiros et al., *A Comparison of 10 Sampling Algorithms for Configurable Systems.* ICSE '16]

Comparison of Sampling Strategies



total configuration-related faults = 135

[Medeiros et al., *A Comparison of 10 Sampling Algorithms for Configurable Systems.* ICSE '16]

Sampling Challenges

- Always incomplete
- Considering constraints complicates things
- Global vs local analysis
    - local considers fewer options, but can only detect bugs within individual files
    - global needed for testing and inter-file analysis
- Best practical technique to date

Variability-aware Analysis (a.k.a family-based analysis)

Variability-aware Analysis

Product Configuration

Variability-Aware Analysis

Conventional Analysis

Variability-aware Analysis

Product Configuration

Variability-Aware Analysis

Conventional Analysis

Analyze “one”, make statements about all

Applications of Variability-aware Analysis

- Lexing
- Parsing
- Type checking
- Linker
- Data-flow
- Testing
- …

Applications of Variability-aware Analysis

- Lexing
- Parsing
- Type checking
- Linker
- Data-flow
- Testing
- …

Example

```
#include <stdio.h>

#ifdef WORLD
char* msg = “Hello World”;
#endif

#ifdef BYE
char* msg = “Bye bye!\n”;
#endif

main() {
  printf(%s, msg);
}
```

Example

References

```
#include <stdio.h>

#ifdef WORLD
char* msg = “Hello World”;
#endif

#ifdef BYE
char* msg = “Bye bye!\n”;
#endif

main() {
  printf(%s, msg);
}
```

Conflicts

Example

Presence Condition (pc)

true

WORLD

BYE

true

```
#include <stdio.h>

#ifdef WORLD
char* msg = “Hello World”;
#endif

#ifdef BYE
char* msg = “Bye bye!\n”;
#endif

main() {
  printf(%s, msg);
}
```

Type Checking

References: pc(caller) => pc(target)

Conflicts: !(pc(def1) && pc(def2))

Type Checking

References: pc(caller) => pc(target)

Conflicts: !(pc(def1) && pc(def2))

```
#include <stdio.h>

#ifdef WORLD
char* msg = “Hello World”;
#endif

#ifdef BYE
char* msg = “Bye bye!\n”;
#endif

main() {
  printf(%s, msg);
}
```

!(WORLD && BYE)

true => BYE || WORLD

true => true

Type Checking

References: pc(caller) => pc(target)

Conflicts: !(pc(def1) && pc(def2))

```
#include <stdio.h>

#ifdef WORLD
char* msg = “Hello World”;
#endif

#ifdef BYE
char* msg = “Bye bye!\n”;
#endif

main() {
  printf(%s, msg);
}
```

!(WORLD && BYE)

true => BYE || WORLD

true => true

Type Checking

References: pc(caller) => pc(ta

Conflicts: !(pc(def1) && pc(def2

```
Found 2 type errors:
 - [WORLD & BYE] file greet.c:7:8
       redefinition of msg
 - [!WORLD & !BYE] file greet.c:11:8
       msg undeclared
```

!(WORLD && BYE)

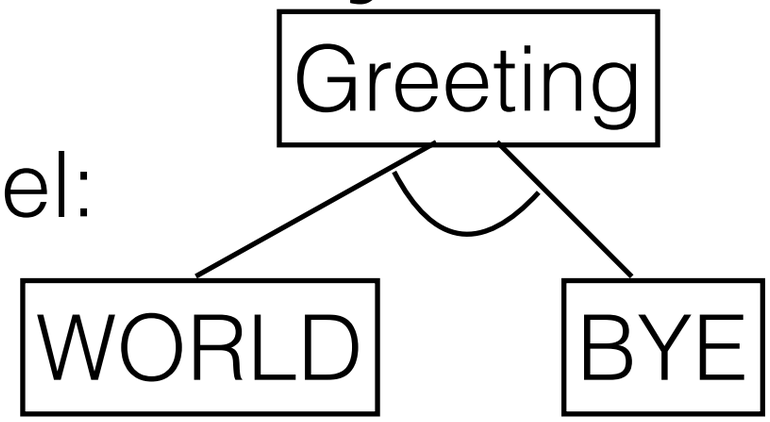
```
#include <stdio.h>

#ifdef WORLD
char* msg = “Hello World”;
#endif

#ifdef BYE
char* msg = “Bye bye!\n”;
#endif

main() {
  printf(%s, msg);
}
```

true => BYE || WORLD

true => true

Exploiting Variability Model

Variability Model:

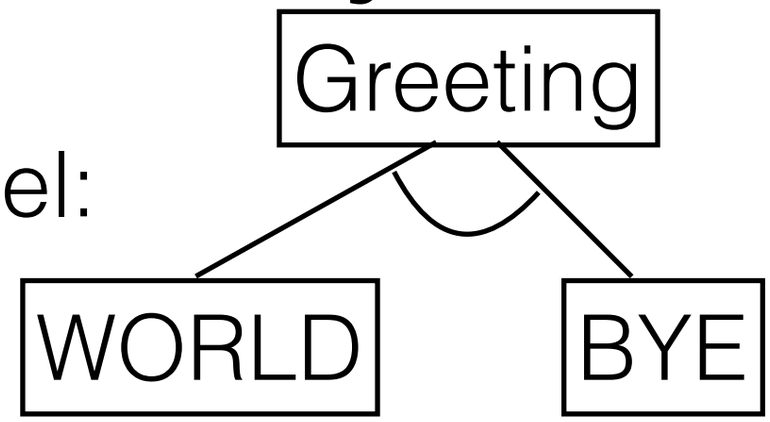


WORLD	BYE
T	T
T	F
F	T
F	F

```
#include <stdio.h>

#ifdef WORLD
char* msg = “Hello World”;
#endif

#ifdef BYE
char* msg = “Bye bye!\n”;
#endif

main() {
  printf(%s, msg);
}
```

Exploiting Variability Model

Variability Model:

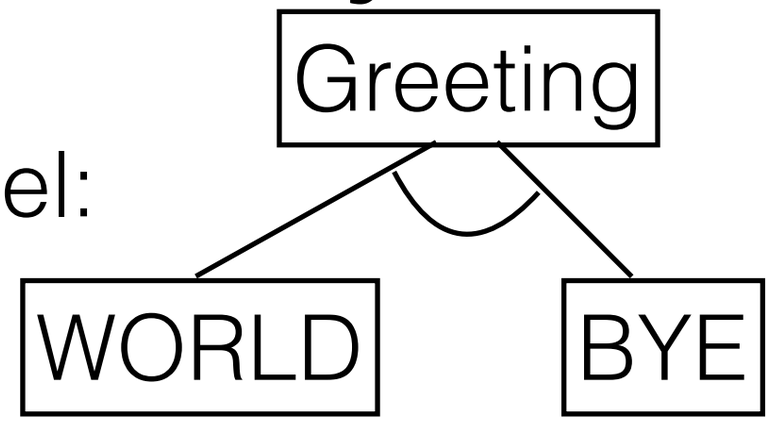


WORLD	BYE
T	T
T	F
F	T
F	F

```
#include <stdio.h>

#ifdef WORLD
char* msg = “Hello World”;
#endif

#ifdef BYE
char* msg = “Bye bye!\n”;
#endif

main() {
  printf(%s, msg);
}
```

Exploiting Variability Model

Variability Model:



WORLD	BYE
T	T
T	F
F	T
F	F

consistent?

```
#include <stdio.h>

#ifdef WORLD
char* msg = “Hello World”;
#endif

#ifdef BYE
char* msg = “Bye bye!\n”;
#endif

main() {
  printf(%s, msg);
}
```

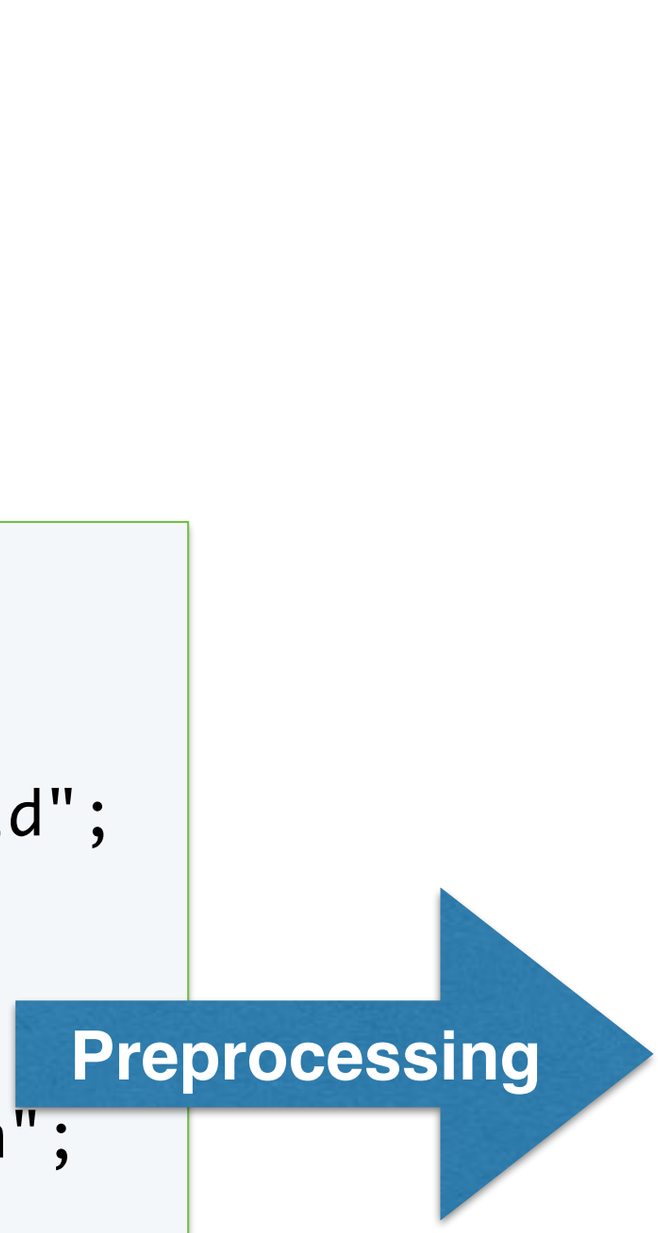
So how does variability-aware type checking work in the background?

So how does variability-aware type checking work in the background?

Variability-aware ASTs

```
#include <stdio.h>

#ifdef WORLD
char* msg = "Hello World";
#endif

#ifdef BYE
char* msg = "Bye bye!\n";
#endif

main() {
  printf("%s", msg);
}
```

```c
#include <stdio.h>

#ifdef WORLD
char* msg = "Hello World";
#endif

#ifdef BYE
char* msg = "Bye bye!\n";
#endif

main() {
  printf("%s", msg);
}
```



```c
#include <stdio.h>

#ifdef WORLD
char* msg = "Hello World";
#endif

#ifdef BYE
char* msg = "Bye bye!\n";
#endif

main() {
  printf("%s", msg);
}
```

WORLD
&&
!BYE

Preprocessing

```c
#include <stdio.h>

char * msg = “Hello World”;

main() {
  printf(%s, msg);
}
```

```c
#include <stdio.h>

#ifdef WORLD
char* msg = "Hello World";
#endif

#ifdef BYE
char* msg = "Bye bye!\n";
#endif

main() {
  printf("%s", msg);
}
```

WORLD && !BYE

BYE && !WORLD

Preprocessing

```c
#include <stdio.h>

char * msg = “Hello World”;

main() {
  printf(%s, msg);
}
```

```c
#include <stdio.h>

char * msg = “Bye bye!\n”;

main() {
  print(msg);
}
```

Abstract Syntax Trees (AST)

WORLD
&&
!BYE

eprocessing

BYE
&&
!WORLD

```
#include <stdio.h>

char * msg = “Hello World”;

main() {
  printf(%s, msg);
}
```

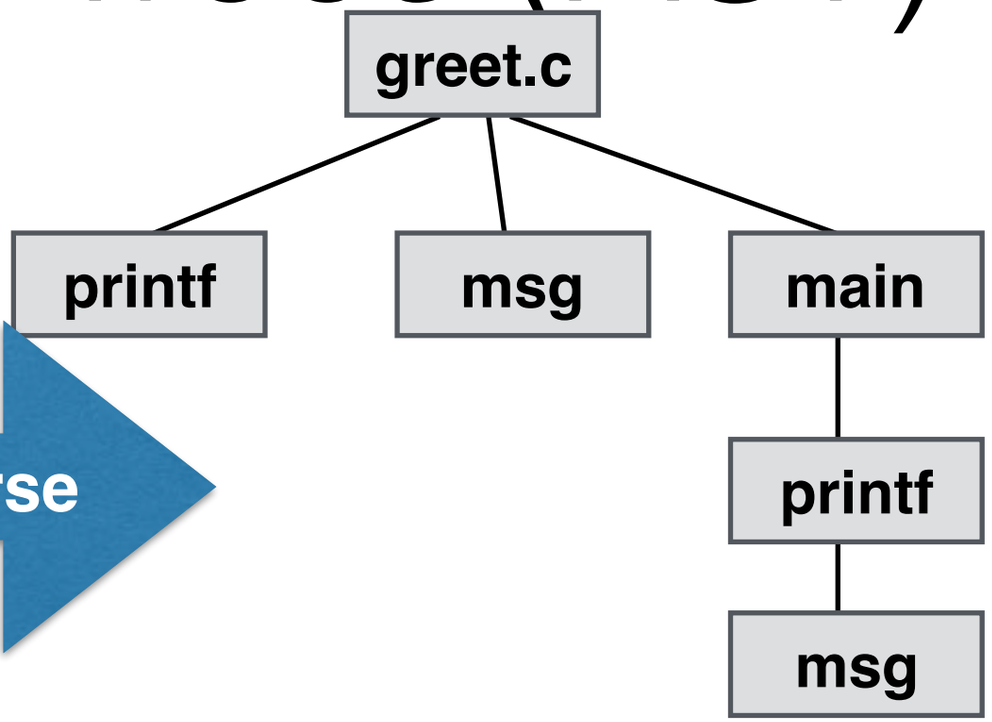
```
#include <stdio.h>

char * msg = “Bye bye!\n”;

main() {
  print(%s, msg);
}
```

Abstract Syntax Trees (AST)

WORLD
&&
!BYE

```
#include <stdio.h>

char * msg = “Hello World”;

main() {
  printf(%s, msg);
}
```

Parse



eprocessing

BYE
&&
!WORLD

```
#include <stdio.h>

char * msg = “Bye bye!\n”;

main() {
  print(%s, msg);
}
```

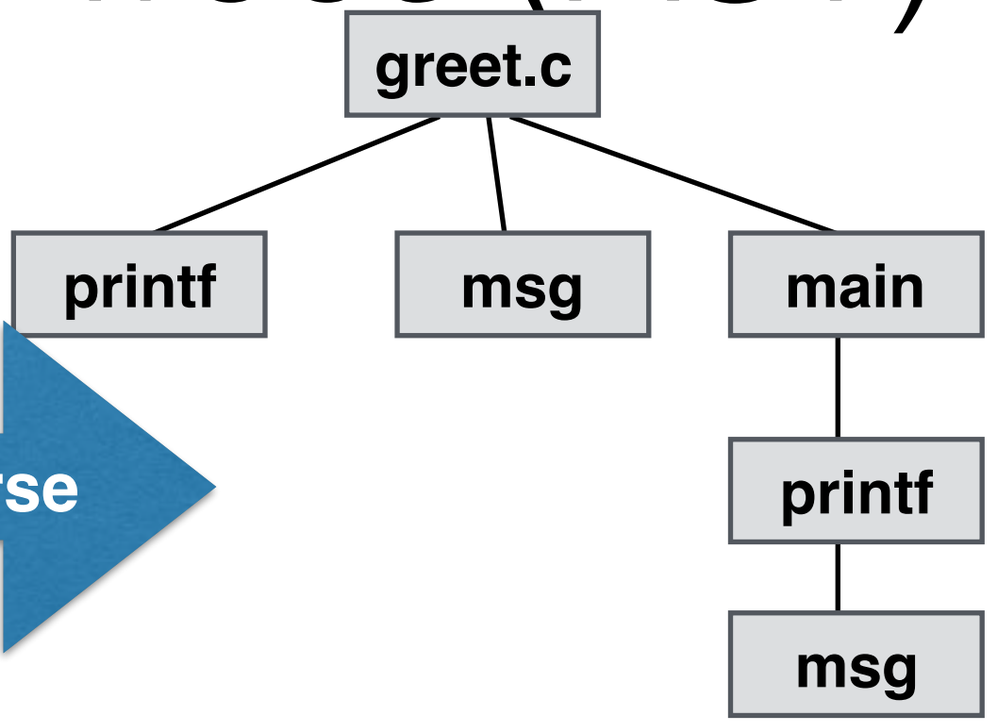
Abstract Syntax Trees (AST)

WORLD
&&
!BYE

eprocessing

```
#include <stdio.h>

char * msg = “Hello World”;

main() {
  printf(%s, msg);
}
```

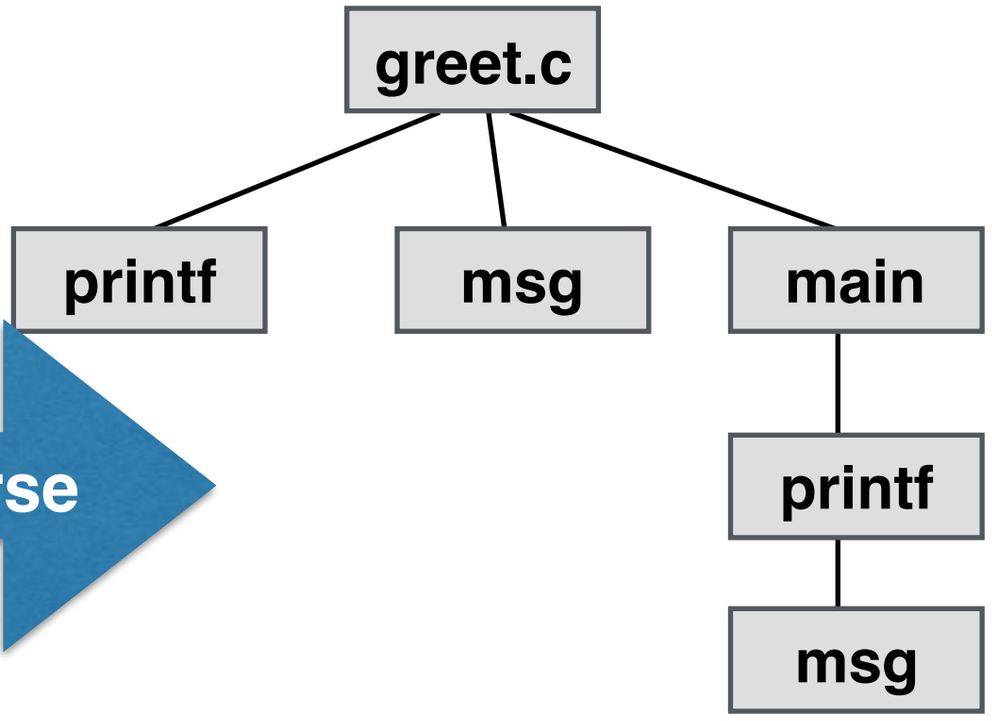
Parse



BYE
&&
!WORLD

```
#include <stdio.h>

char * msg = “Bye bye!\n”;

main() {
  print(%s, msg);
}
```

Parse



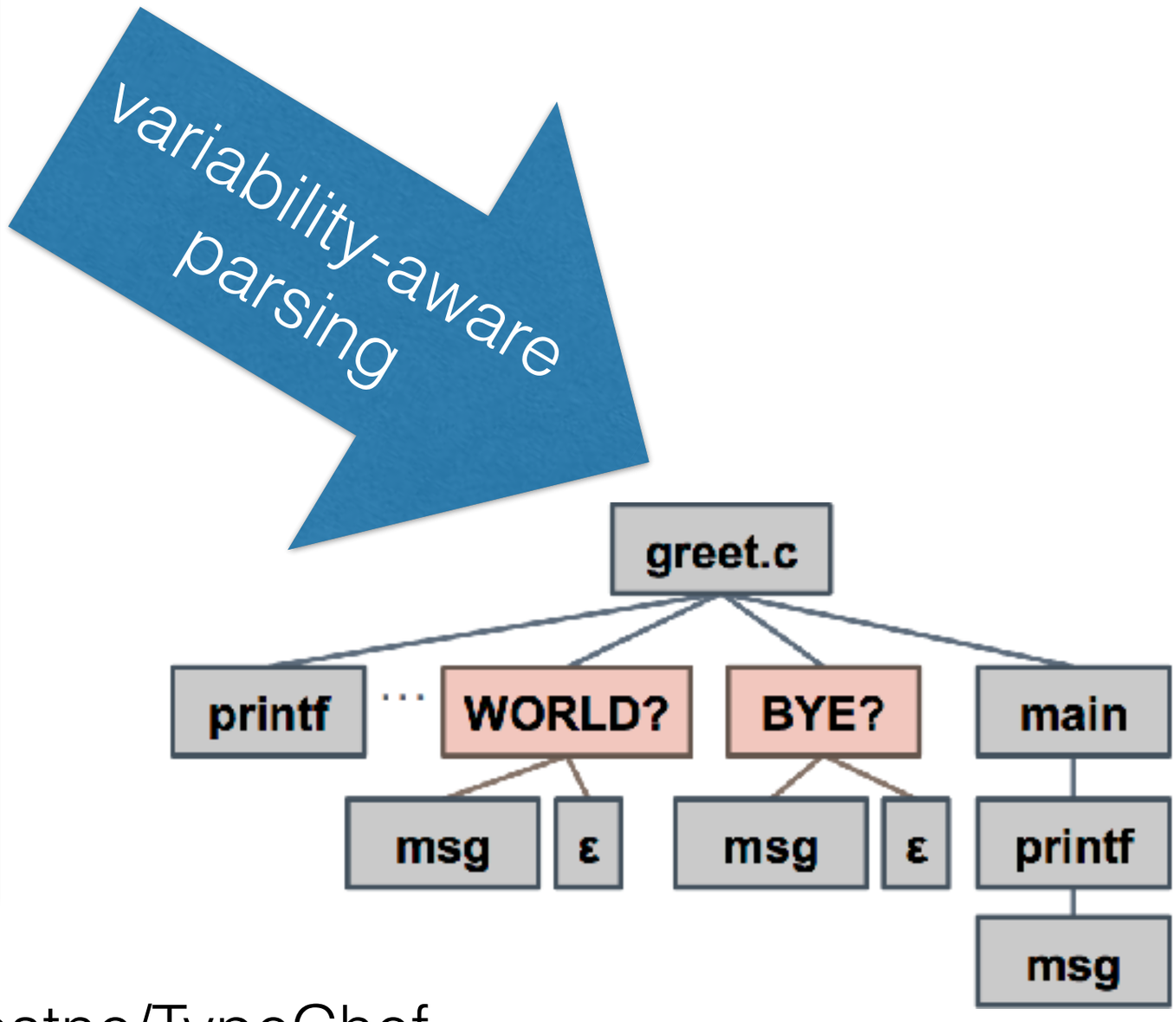
Variability-aware AST

```
#include <stdio.h>

#ifdef WORLD
char* msg = "Hello World";
#endif

#ifdef BYE
char* msg = "Bye bye!\n";
#endif

main() {
  printf("%s", msg);
}
```

Check https://github.com/ckaestne/TypeChef

Variability-aware AST

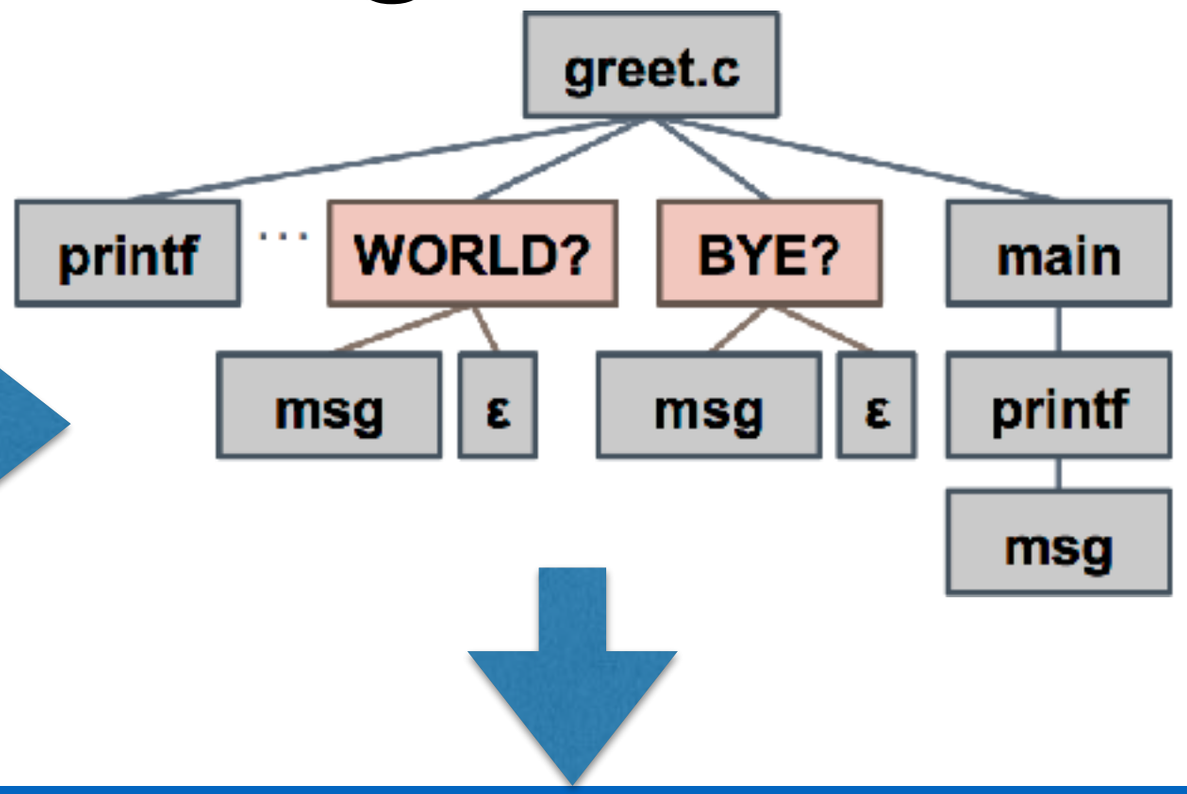
```
#include <stdio.h>

#ifdef WORLD
char* msg = "Hello World";
#endif

#ifdef BYE
char* msg = "Bye bye!\n";
#endif

main() {
  printf("%s", msg);
}
```



Check https://github.com/ckaestne/TypeChef

Variability-aware Type Checking

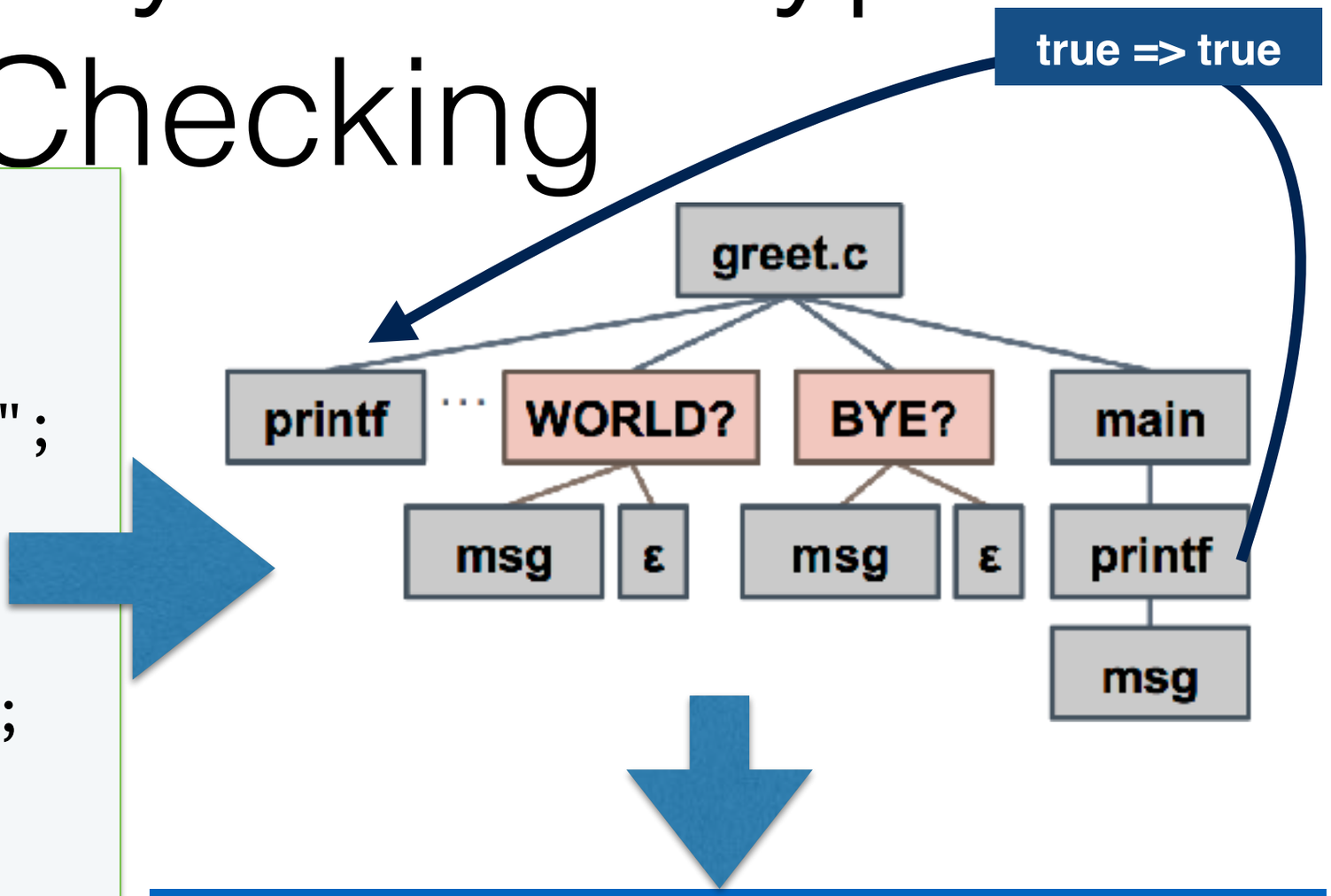
```
#include <stdio.h>

#ifdef WORLD
char* msg = "Hello World";
#endif

#ifdef BYE
char* msg = "Bye bye!\n";
#endif

main() {
  printf("%s", msg);
}
```



Name	Type	PC
printf	char * –> int	TRUE
msg	char *	WORLD
msg	char *	BYE

Variability-aware Type Checking

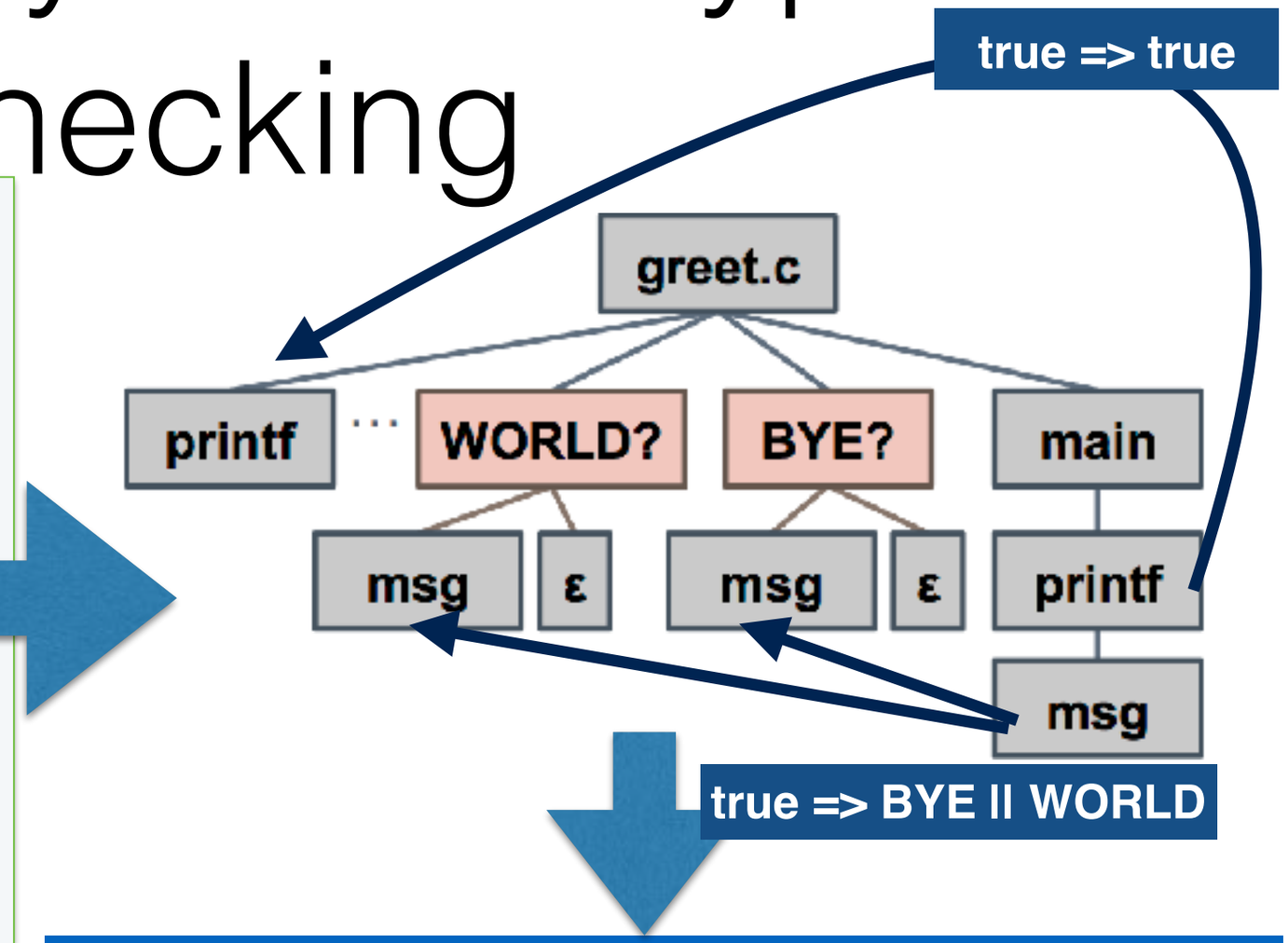
```
#include <stdio.h>

#ifdef WORLD
char* msg = "Hello World";
#endif

#ifdef BYE
char* msg = "Bye bye!\n";
#endif

main() {
  printf("%s", msg);
}
```



Name	Type	PC
printf	char * –> int	TRUE
msg	char *	WORLD
msg	char *	BYE

Variability-aware Type Checking

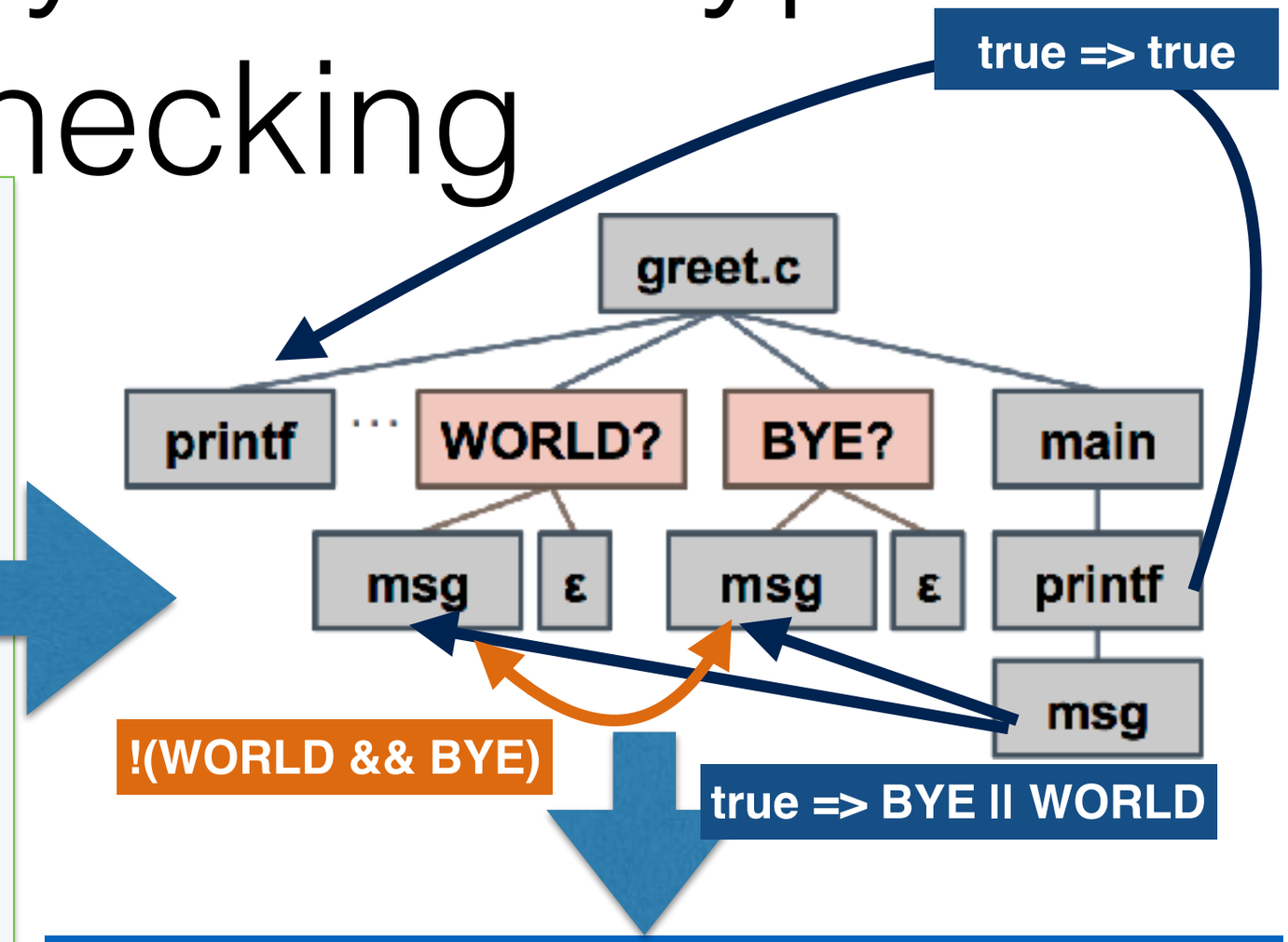
```
#include <stdio.h>

#ifdef WORLD
char* msg = "Hello World";
#endif

#ifdef BYE
char* msg = "Bye bye!\n";
#endif

main() {
  printf("%s", msg);
}
```



Name	Type	PC
printf	char * –> int	TRUE
msg	char *	WORLD
msg	char *	BYE

Variability-aware Type Checking

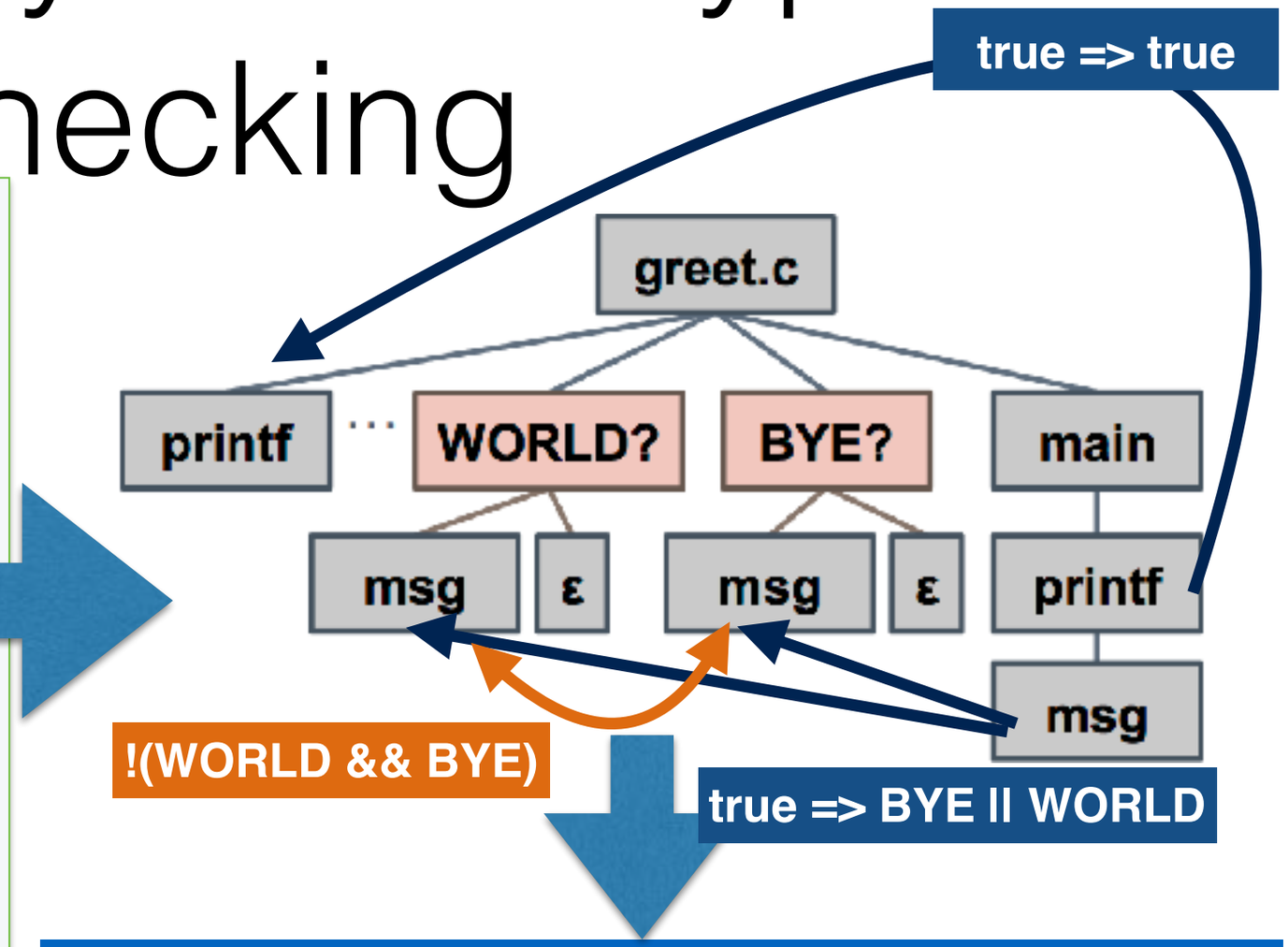
```
#include <stdio.h>

#ifdef WORLD
char* msg = "Hello World";
#endif

#ifdef BYE
char* msg = "Bye bye!\n";
#endif

main() {
  printf("%s", msg);
}
```



Name	Type	PC
printf	char * –> int	TRUE
msg	char *	WORLD
msg	char *	BYE

Variability-aware Type Checking

```
#include <stdio.h>

#ifdef WORLD
char* msg = "Hello World";
#endif

#ifdef BYE
char* msg = "Bye bye!\n";
#endif

main() {
  printf("%s", msg);
```



Name	Type	PC
printf	char * –> int	TRUE
msg	char *	WORLD
msg	char *	BYE

Found 2 type errors:
- [WORLD & BYE] file greet.c:7:8
  redefinition of msg
- [!WORLD & !BYE] file greet.c:11:8
  msg undeclared

Variability-aware … (anything!)

- Idea is to keep variability information in the data structure
- Most of the code is shared so you do not need to do anything special there
- Once you hit a decision point, you can **split** the analysis results
- Once you are out of the scope of the decision point you can **join** the analysis results

Variability-aware Parsing

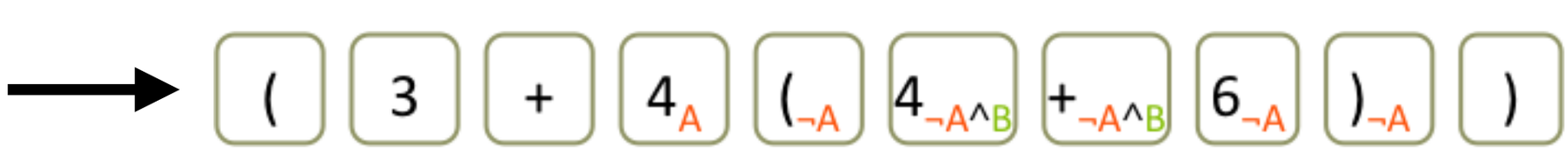
- Challenge is to simultaneously parse all possible paths
- Key-ideas
  - Maximize benefit of shared code
    - Split the parse tree as late as possible
    - Join the parse tree as early as possible

TypeChef


#ifdef A
#define X 4
#else
#define X 5
#endif
(2*3)+X
Variability-Aware
Lexer
Variability-Aware
Parser
+
*
V_A
2
3
4
5
$5_{\neg A}$

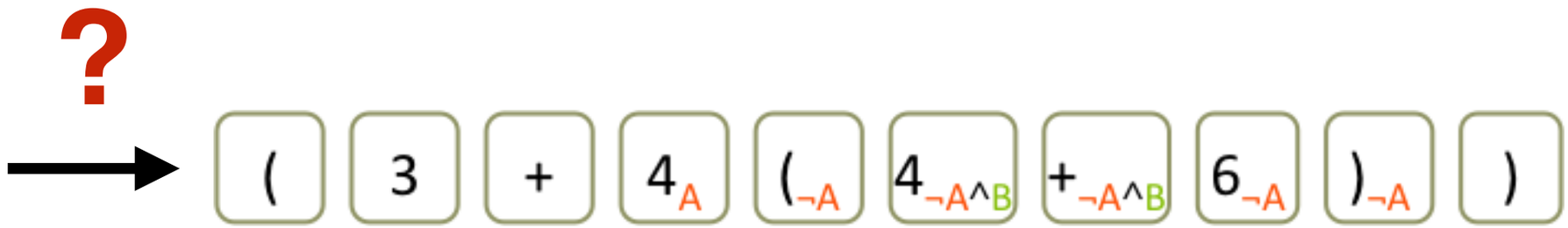
First.. Variability-aware Lexing

```
#ifdef A
  #define X 4
#else
  #ifdef B
    #define X 4+6
  #else
    #define X 6
  #endif
#endif

3+X;
```



First.. Variability-aware Lexing

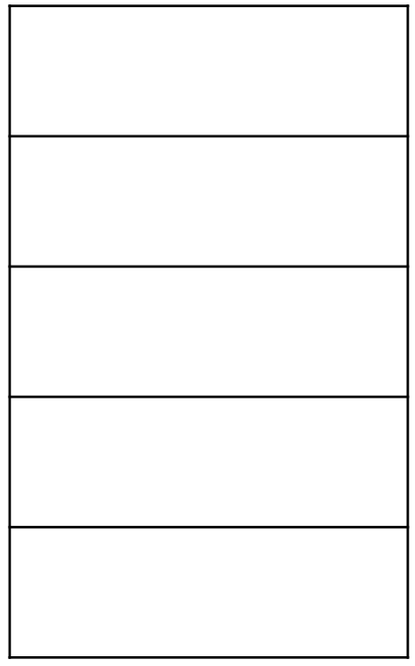
```
#ifdef A
  #define X 4
#else
  #ifdef B
    #define X 4+6
  #else
    #define X 6
  #endif
#endif

3+X;
```



First.. Variability-aware Lexing

```
#ifdef A
  #define X 4
#else
  #ifdef B
    #define X 4+6
  #else
    #define X 6
  #endif
#endif

3+X;
```

Conditional Stack



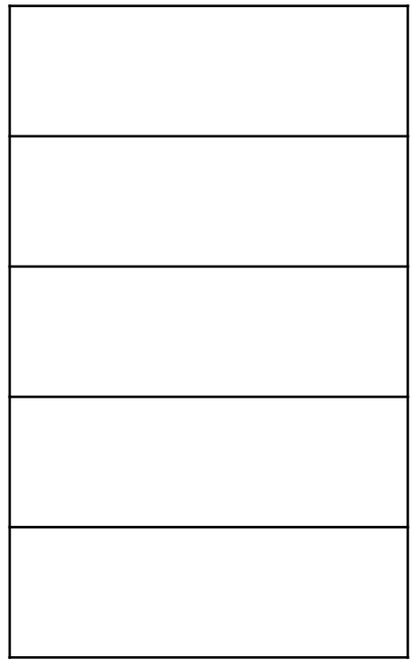
Macro Table

Macro	Expansion	PC

First.. Variability-aware Lexing

```
#ifdef A
  #define X 4
#else
  #ifdef B
    #define X 4+6
  #else
    #define X 6
  #endif
#endif

3+X;
```

Conditional Stack



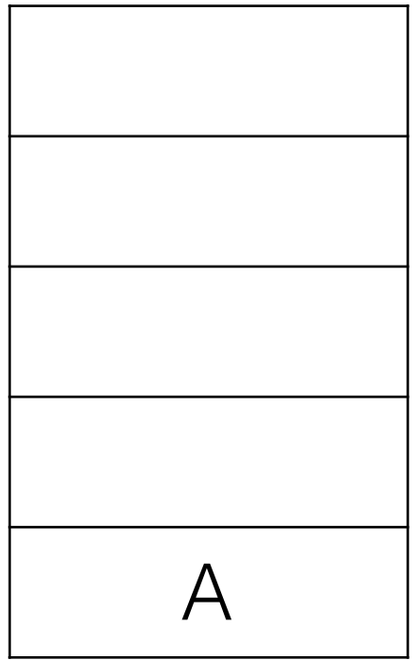
Macro Table

Macro	Expansion	PC

First.. Variability-aware Lexing

```
#ifdef A
 #define X 4
#else
 #ifdef B
  #define X 4+6
 #else
  #define X 6
 #endif
#endif

3+X;
```

Conditional Stack



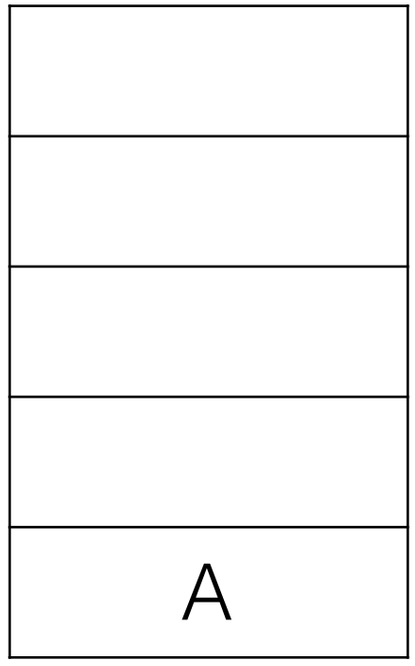
Macro Table

Macro	Expansion	PC

First.. Variability-aware Lexing

```
#ifdef A
  #define X 4
#else
  #ifdef B
    #define X 4+6
  #else
    #define X 6
  #endif
#endif

3+X;
```

Conditional Stack



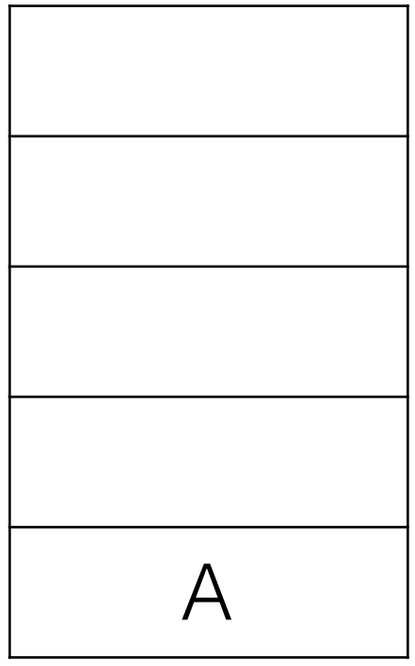
Macro Table

Macro	Expansion	PC

First.. Variability-aware Lexing

```
#ifdef A
 #define X 4
#else
 #ifdef B
  #define X 4+6
 #else
  #define X 6
 #endif
#endif

3+X;
```

Conditional Stack



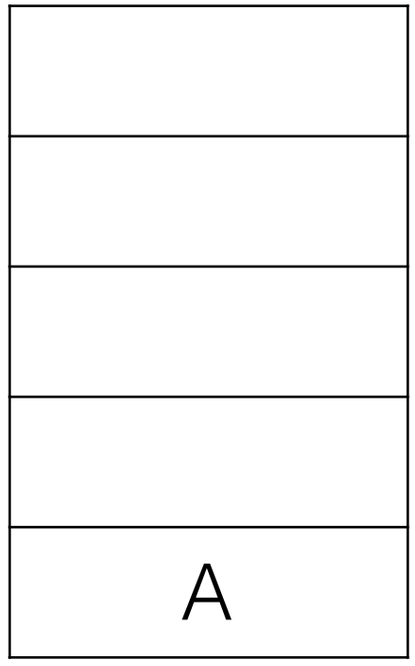
Macro Table

Macro	Expansion	PC
X	4	A

First.. Variability-aware Lexing

```
#ifdef A
 #define X 4
#else
 #ifdef B
  #define X 4+6
 #else
  #define X 6
 #endif
#endif

3+X;
```

Conditional Stack



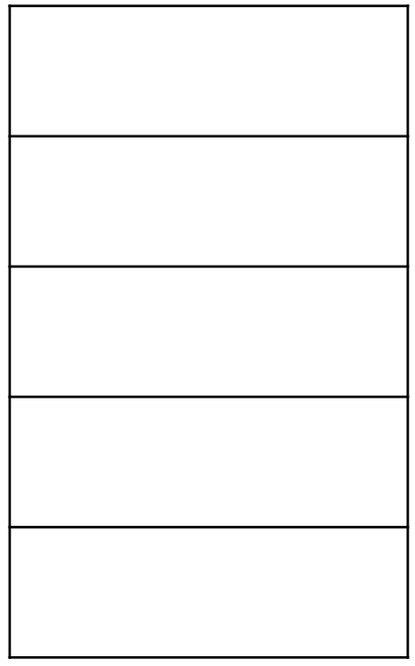
Macro Table

Macro	Expansion	PC
X	4	A

First.. Variability-aware Lexing

```
#ifdef A
 #define X 4
#else
 #ifdef B
  #define X 4+6
 #else
  #define X 6
 #endif
#endif

3+X;
```

Conditional Stack



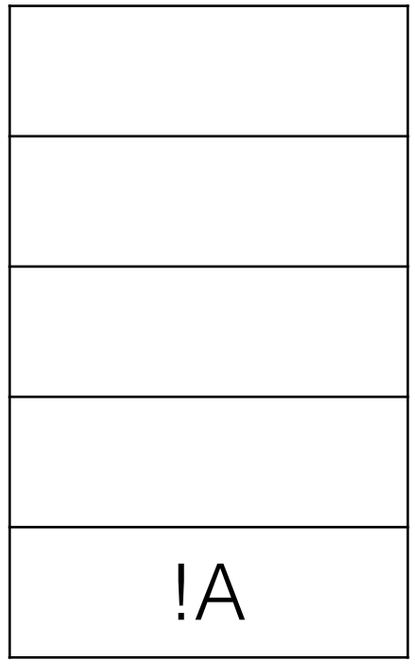
Macro Table

Macro	Expansion	PC
X	4	A

First.. Variability-aware Lexing

```
#ifdef A
 #define X 4
#else
 #ifdef B
  #define X 4+6
 #else
  #define X 6
 #endif
#endif

3+X;
```

Conditional Stack



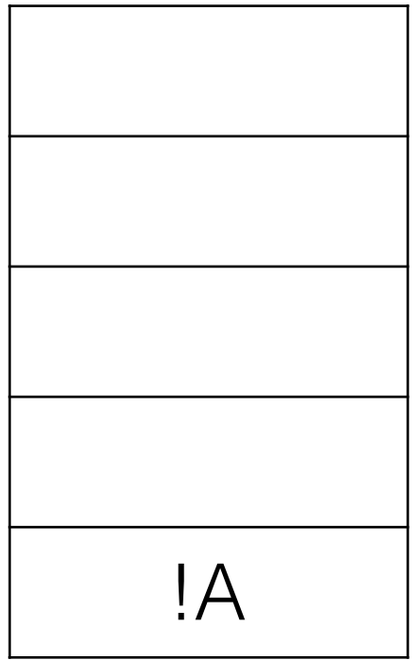
Macro Table

Macro	Expansion	PC
X	4	A

First.. Variability-aware Lexing

```
#ifdef A
 #define X 4
#else
 #ifdef B
  #define X 4+6
 #else
  #define X 6
 #endif
#endif

3+X;
```

Conditional Stack



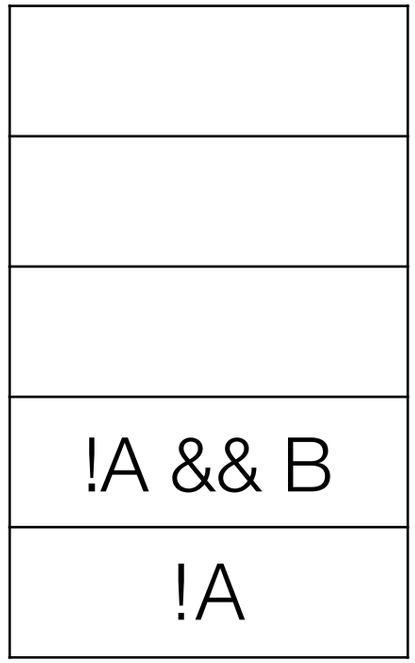
Macro Table

Macro	Expansion	PC
X	4	A

First.. Variability-aware Lexing

```
#ifdef A
 #define X 4
#else
 #ifdef B
  #define X 4+6
 #else
  #define X 6
 #endif
#endif

3+X;
```

Conditional Stack



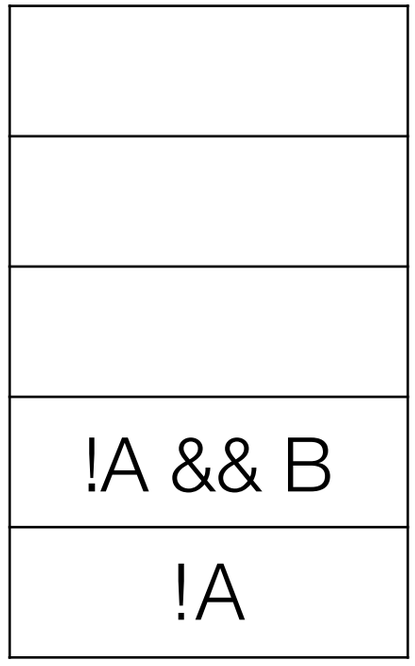
Macro Table

Macro	Expansion	PC
X	4	A

First.. Variability-aware Lexing

```
#ifdef A
 #define X 4
#else
 #ifdef B
  #define X 4+6
 #else
  #define X 6
 #endif
#endif

3+X;
```

Conditional Stack



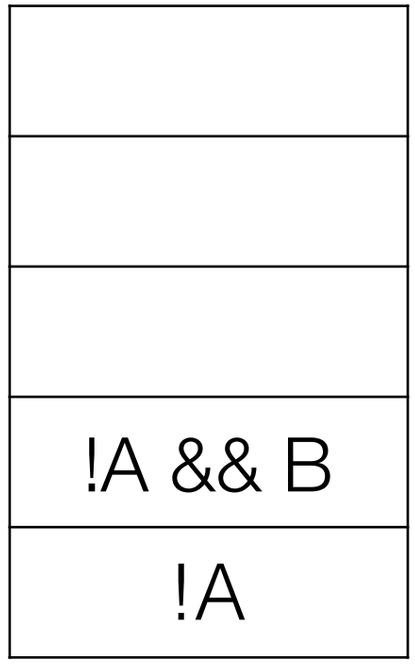
Macro Table

Macro	Expansion	PC
X	4	A

First.. Variability-aware Lexing

```
#ifdef A
 #define X 4
#else
 #ifdef B
  #define X 4+6
 #else
  #define X 6
 #endif
#endif

3+X;
```

Conditional Stack



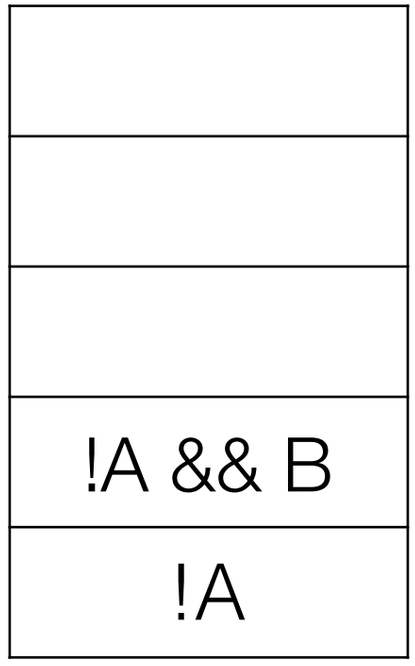
Macro Table

Macro	Expansion	PC
X	4	A
X	4 + 6	!A && B

First.. Variability-aware Lexing

```
#ifdef A
 #define X 4
#else
 #ifdef B
  #define X 4+6
 #else
  #define X 6
 #endif
#endif

3+X;
```

Conditional Stack



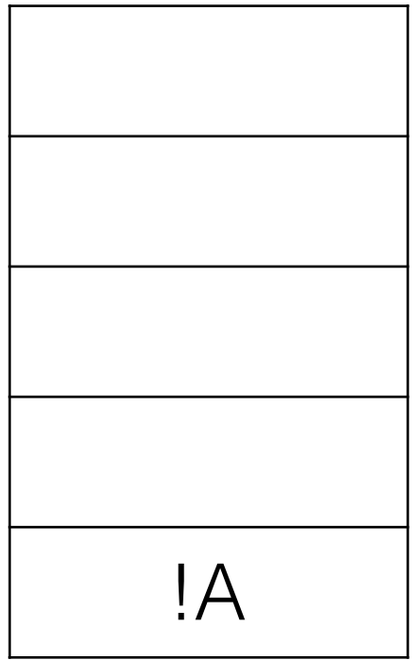
Macro Table

Macro	Expansion	PC
X	4	A
X	4 + 6	!A && B

First.. Variability-aware Lexing

```
#ifdef A
 #define X 4
#else
 #ifdef B
  #define X 4+6
 #else
  #define X 6
 #endif
#endif

3+X;
```

Conditional Stack



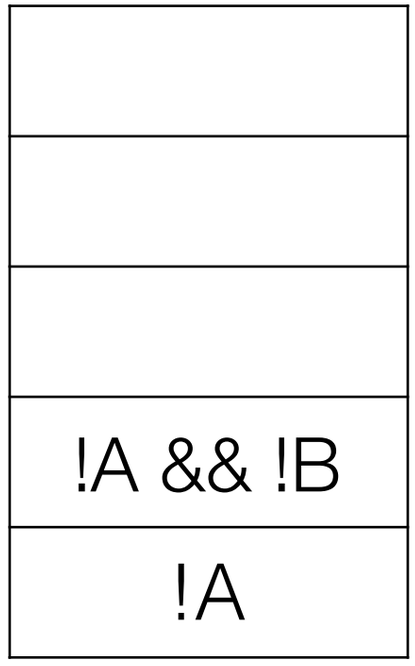
Macro Table

Macro	Expansion	PC
X	4	A
X	4 + 6	!A && B

First.. Variability-aware Lexing

```
#ifdef A
 #define X 4
#else
 #ifdef B
  #define X 4+6
 #else
  #define X 6
 #endif
#endif

3+X;
```

Conditional Stack



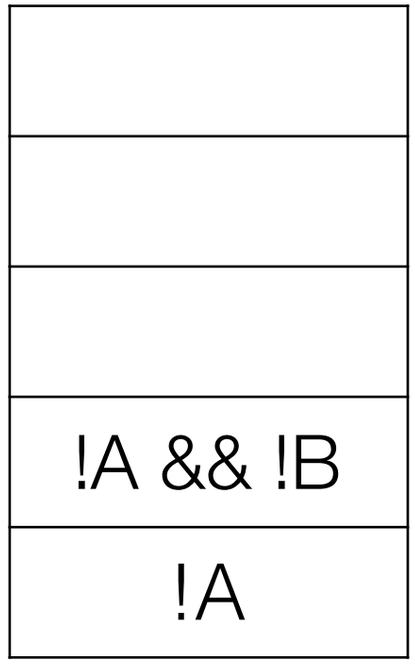
Macro Table

Macro	Expansion	PC
X	4	A
X	4 + 6	!A && B

First.. Variability-aware Lexing

```
#ifdef A
 #define X 4
#else
 #ifdef B
  #define X 4+6
 #else
  #define X 6
 #endif
#endif

3+X;
```

Conditional Stack



Macro Table

Macro	Expansion	PC
X	4	A
X	4 + 6	!A && B

First.. Variability-aware Lexing

```
#ifdef A
 #define X 4
#else
 #ifdef B
  #define X 4+6
 #else
  #define X 6
 #endif
#endif

3+X;
```

Conditional Stack

!A && !B
!A

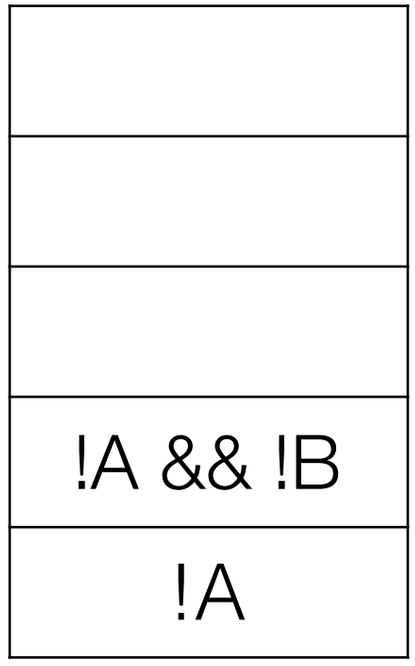
Macro Table

Macro	Expansion	PC
X	4	A
X	4 + 6	!A && B
X	6	!A && !B

First.. Variability-aware Lexing

```
#ifdef A
 #define X 4
#else
 #ifdef B
  #define X 4+6
 #else
  #define X 6
 #endif
#endif

3+X;
```

Conditional Stack



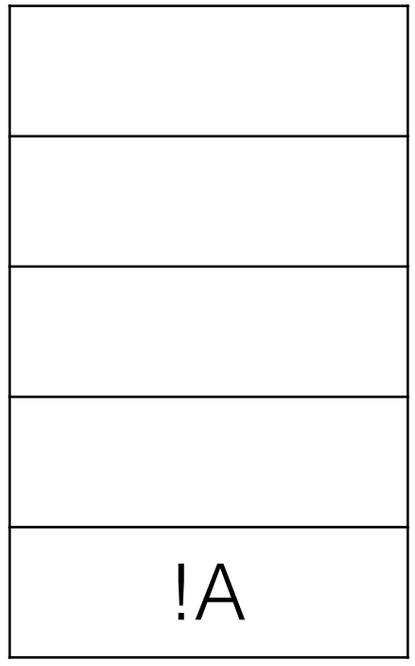
Macro Table

Macro	Expansion	PC
X	4	A
X	4 + 6	!A && B
X	6	!A && !B

First.. Variability-aware Lexing

```
#ifdef A
  #define X 4
#else
  #ifdef B
    #define X 4+6
  #else
    #define X 6
  #endif
#endif

3+X;
```

Conditional Stack



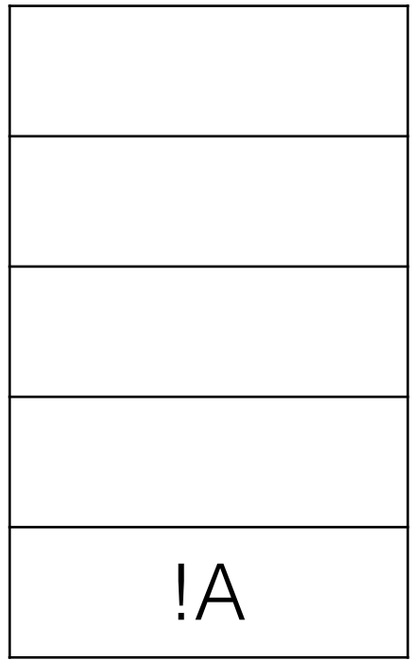
Macro Table

Macro	Expansion	PC
X	4	A
X	4 + 6	!A && B
X	6	!A && !B

First.. Variability-aware Lexing

```
#ifdef A
  #define X 4
#else
  #ifdef B
    #define X 4+6
  #else
    #define X 6
  #endif
#endif

3+X;
```

Conditional Stack



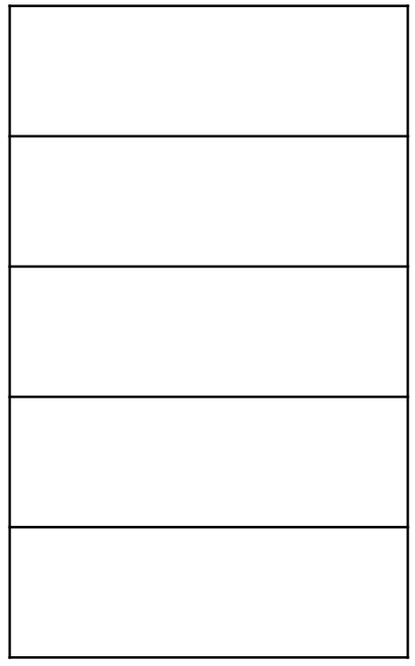
Macro Table

Macro	Expansion	PC
X	4	A
X	4 + 6	!A && B
X	6	!A && !B

First.. Variability-aware Lexing

```
#ifdef A
  #define X 4
#else
  #ifdef B
    #define X 4+6
  #else
    #define X 6
  #endif
#endif

3+X;
```

Conditional Stack



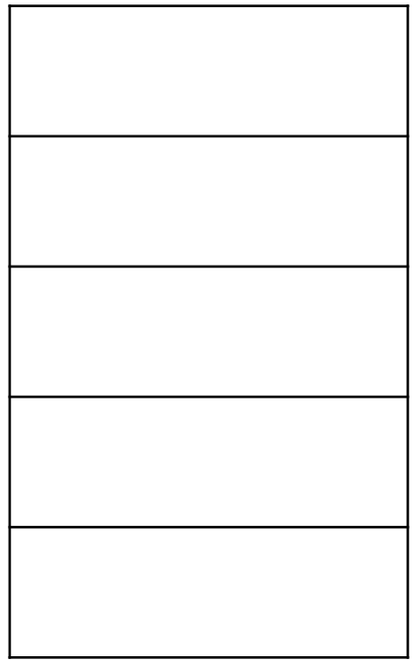
Macro Table

Macro	Expansion	PC
X	4	A
X	4 + 6	!A && B
X	6	!A && !B

First.. Variability-aware Lexing

```
#ifdef A
  #define X 4
#else
  #ifdef B
    #define X 4+6
  #else
    #define X 6
  #endif
#endif

3+X;
```

Conditional Stack

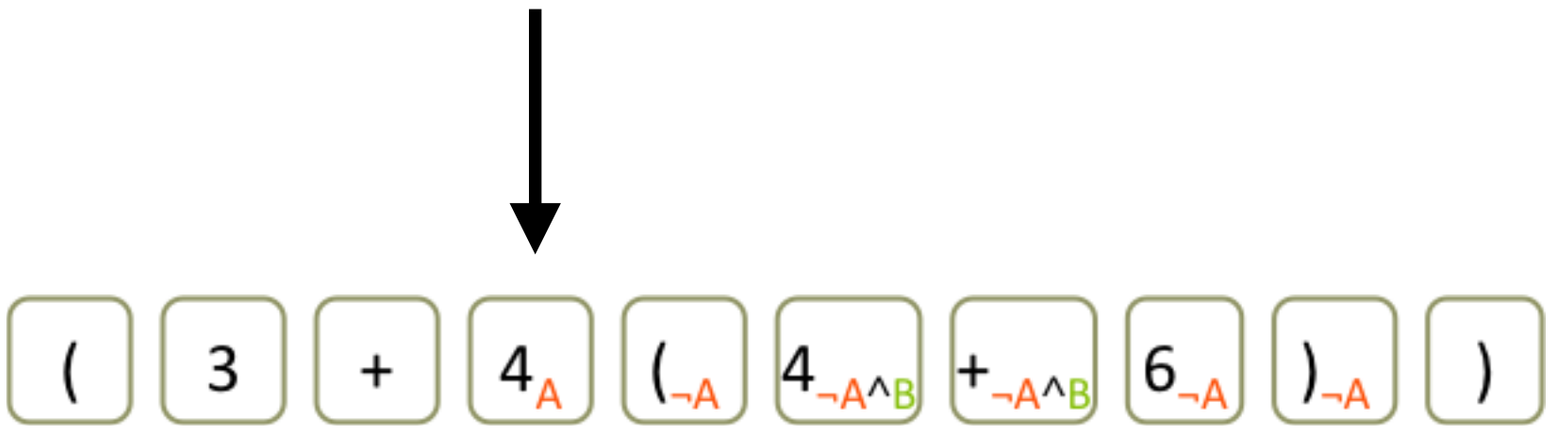


Macro Table

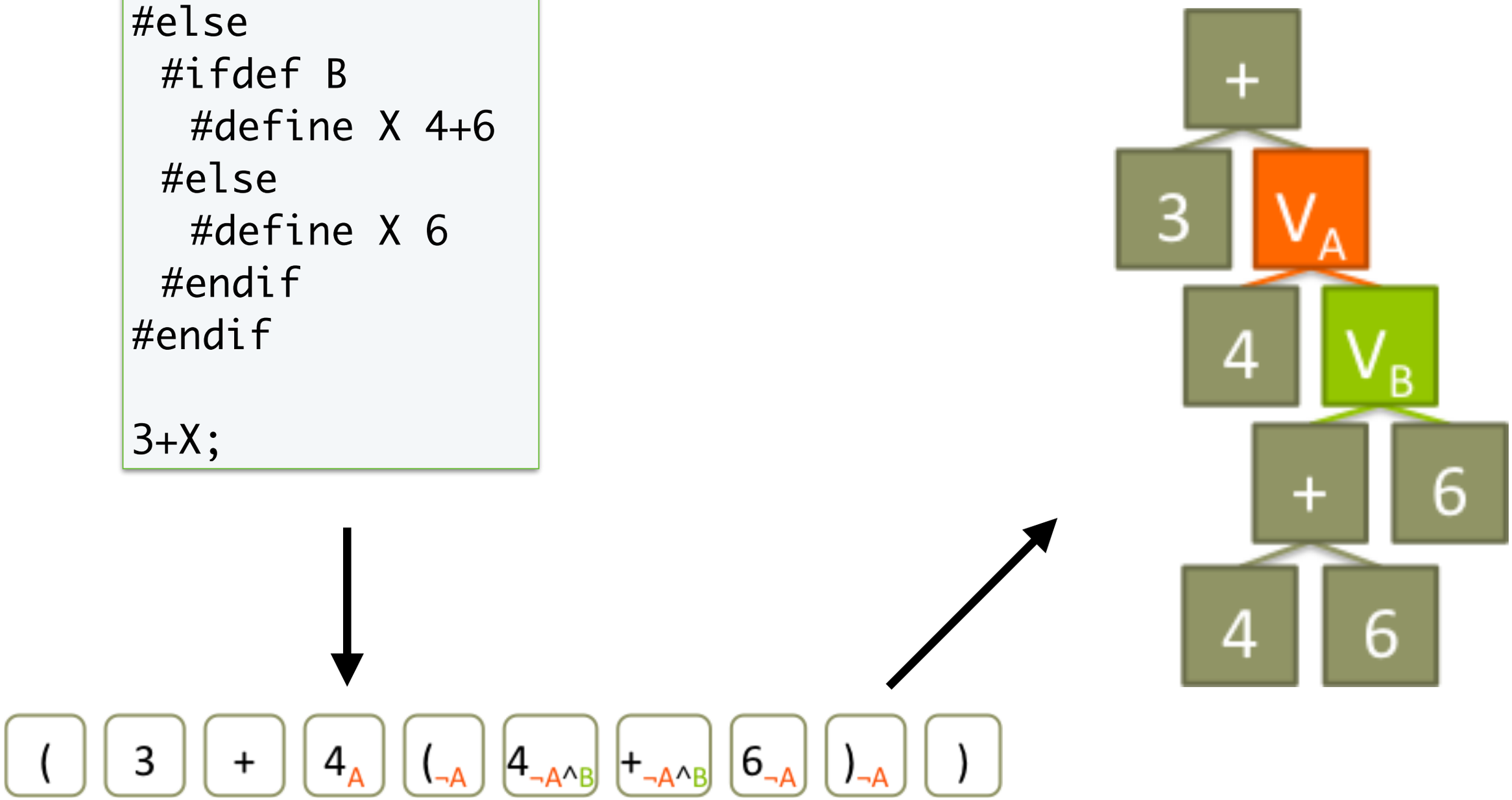
Macro	Expansion	PC
X	4	A
X	4 + 6	!A && B
X	6	!A && !B

```
#ifdef A
 #define X 4
#else
 #ifdef B
  #define X 4+6
 #else
  #define X 6
 #endif
#endif

3+X;
```

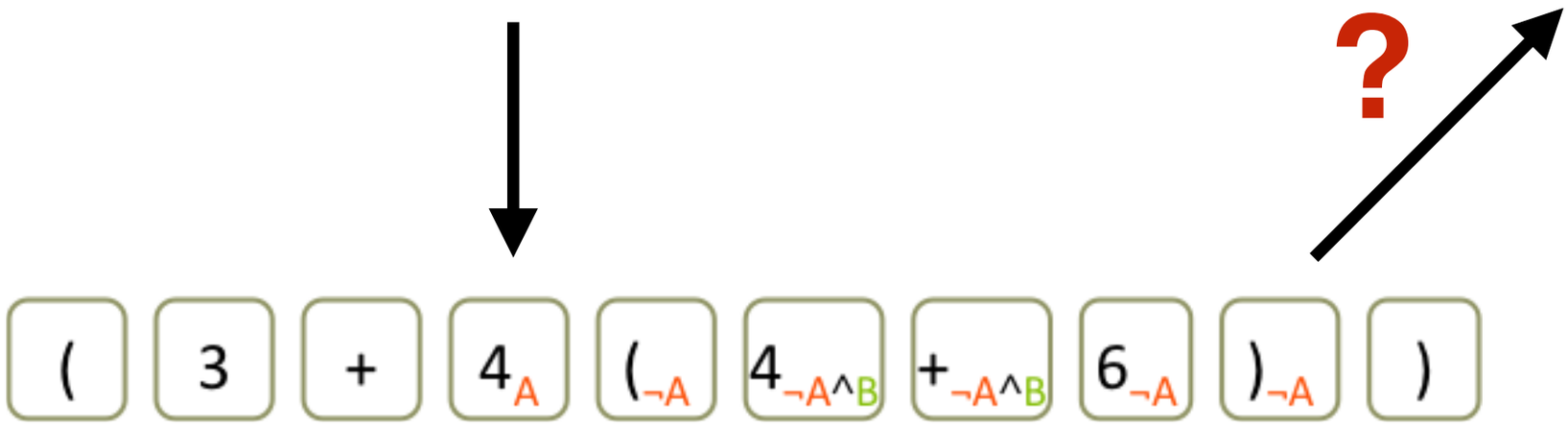
```
#ifdef A
 #define X 4
#else
 #ifdef B
  #define X 4+6
 #else
  #define X 6
 #endif
#endif

3+X;
```

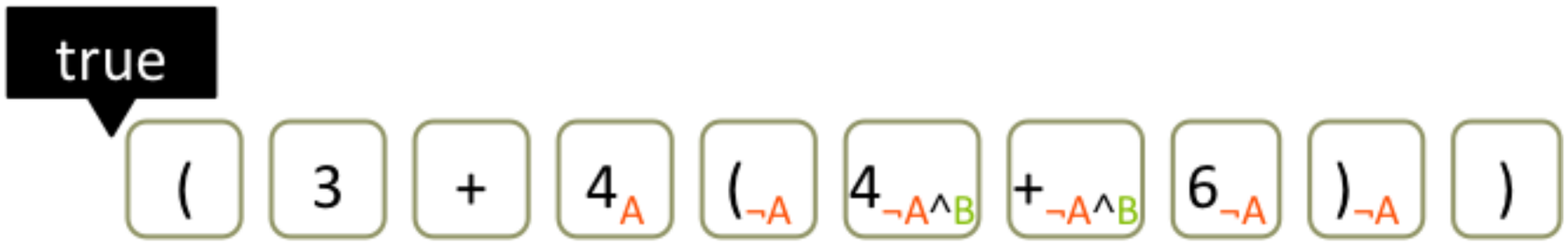


```
#ifdef A
 #define X 4
#else
 #ifdef B
  #define X 4+6
 #else
  #define X 6
 #endif
#endif

3+X;
```

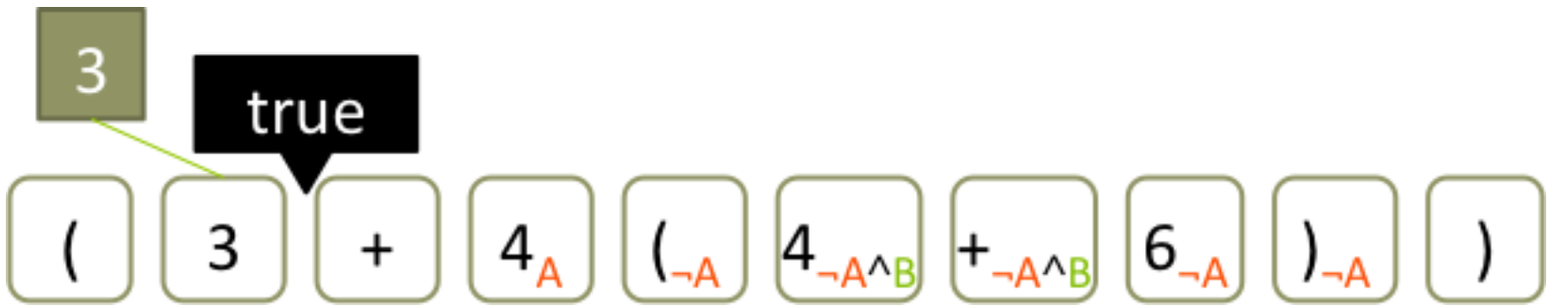


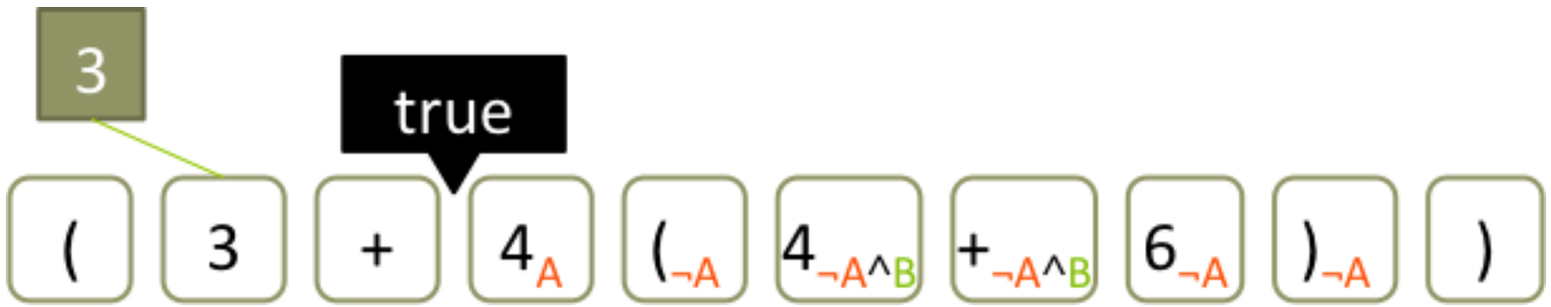
```
#ifdef A
 #define X 4
#else
 #ifdef B
  #define X 4+6
 #else
  #define X 6
 #endif
#endif

3+X;
```

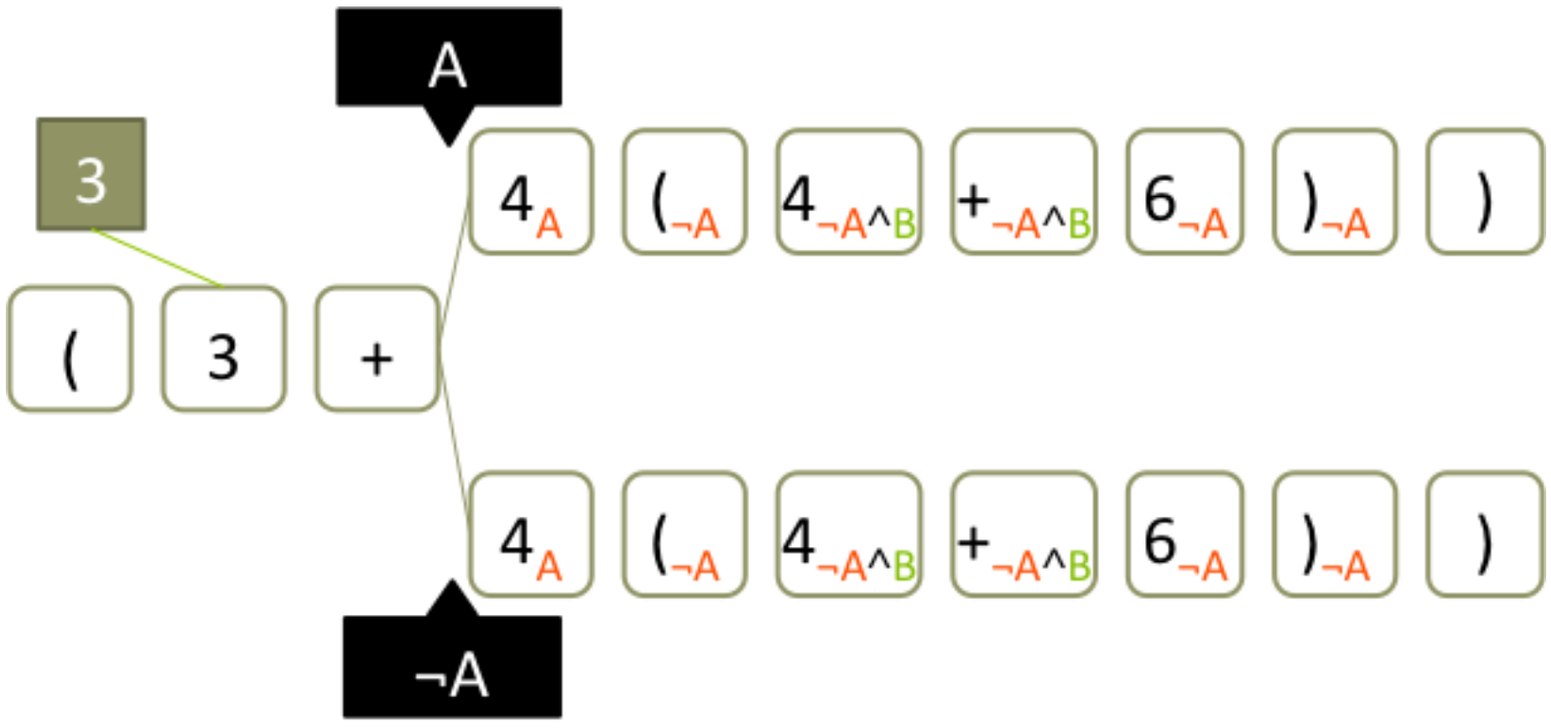


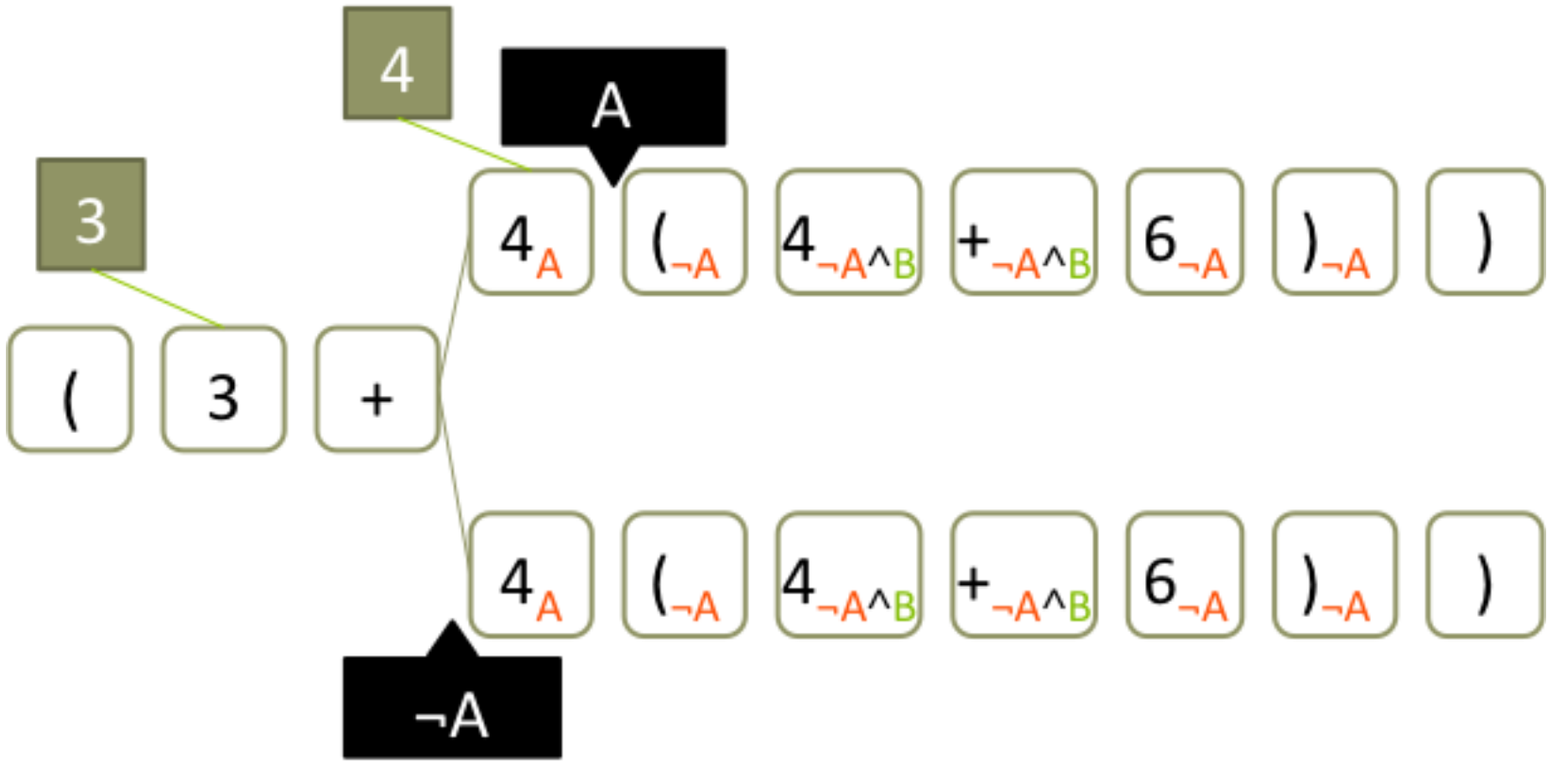

true
(3 + 4A (¬A 4¬A^B +¬A^B 6¬A)¬A)

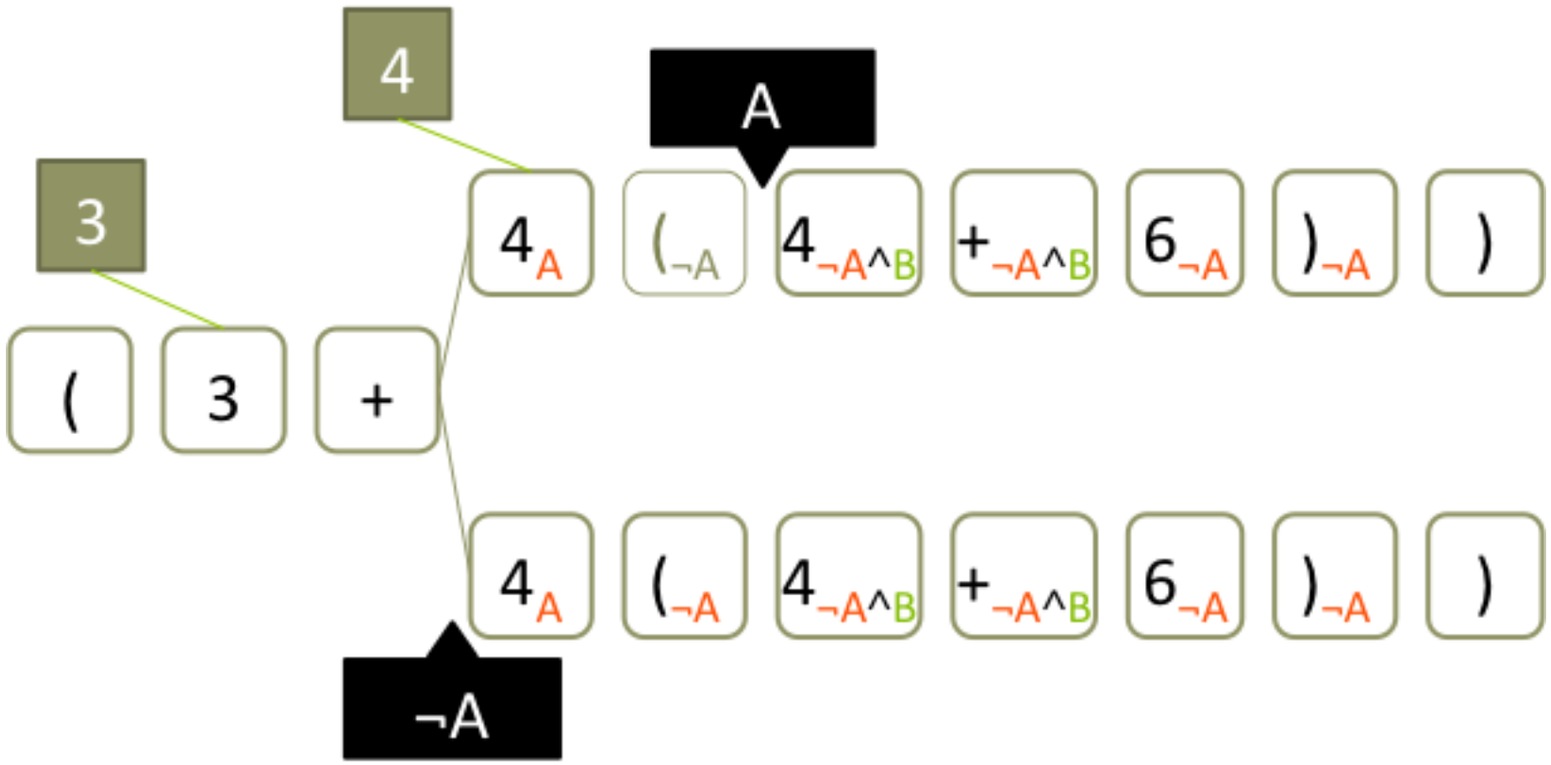

true
(
3
+
4A
(¬A
4¬A^B
+¬A^B
6¬A
)¬A
)

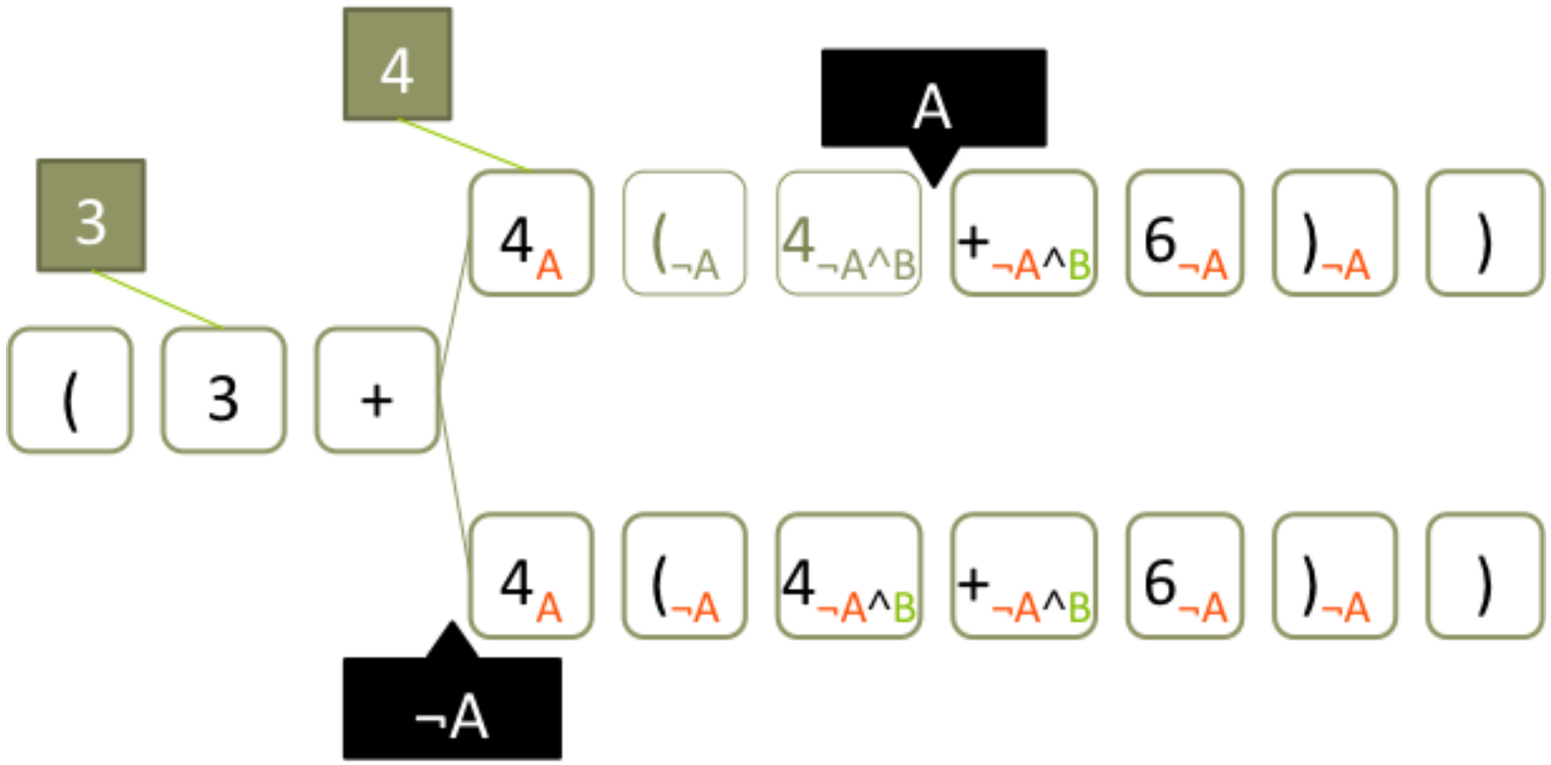

3
true
(
3
+
4A
(¬A
4¬A^B
+¬A^B
6¬A
)¬A
)

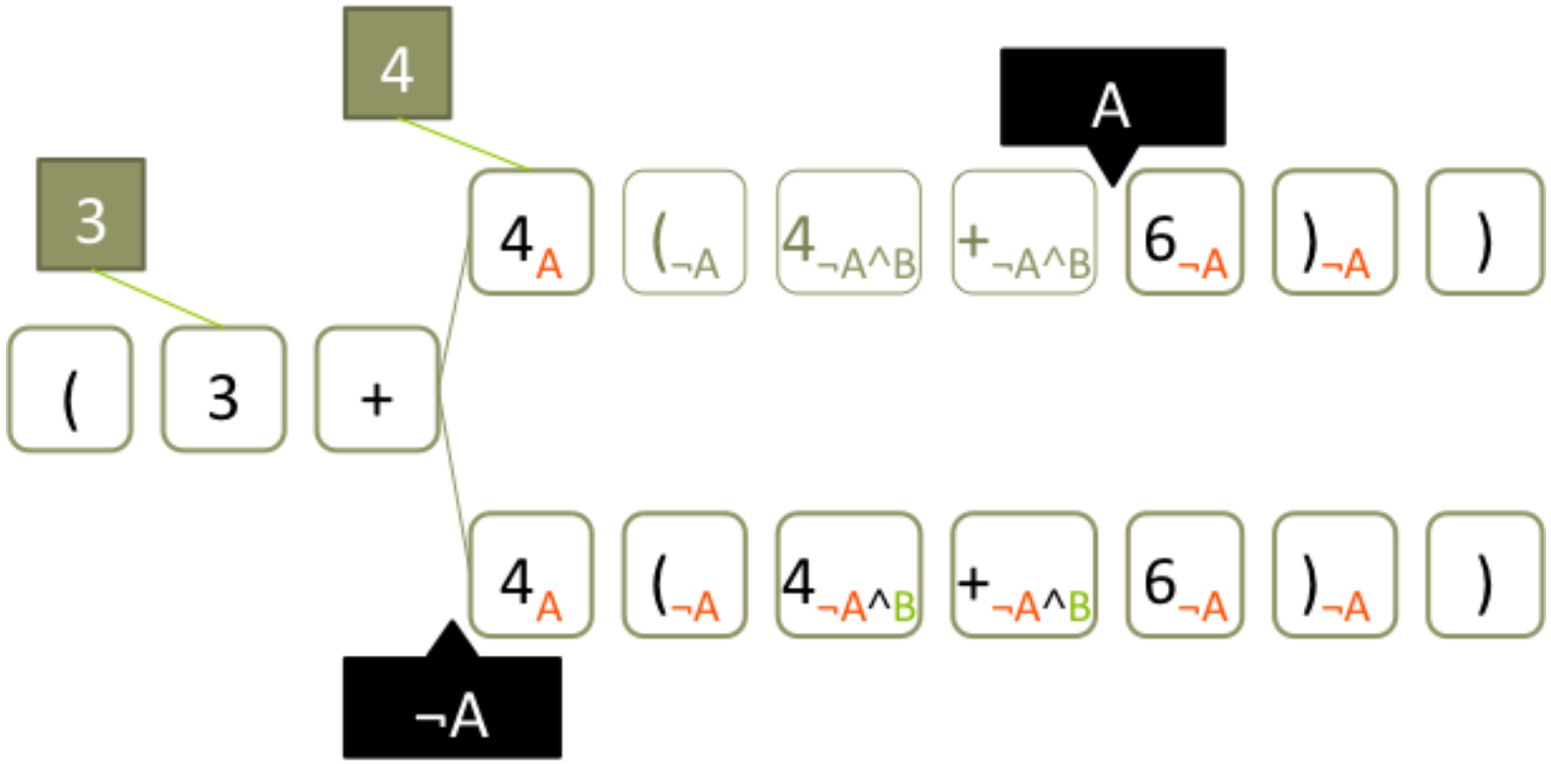

3
true
(
3
+
4A
(¬A
4¬A^B
+¬A^B
6¬A
)¬A
)

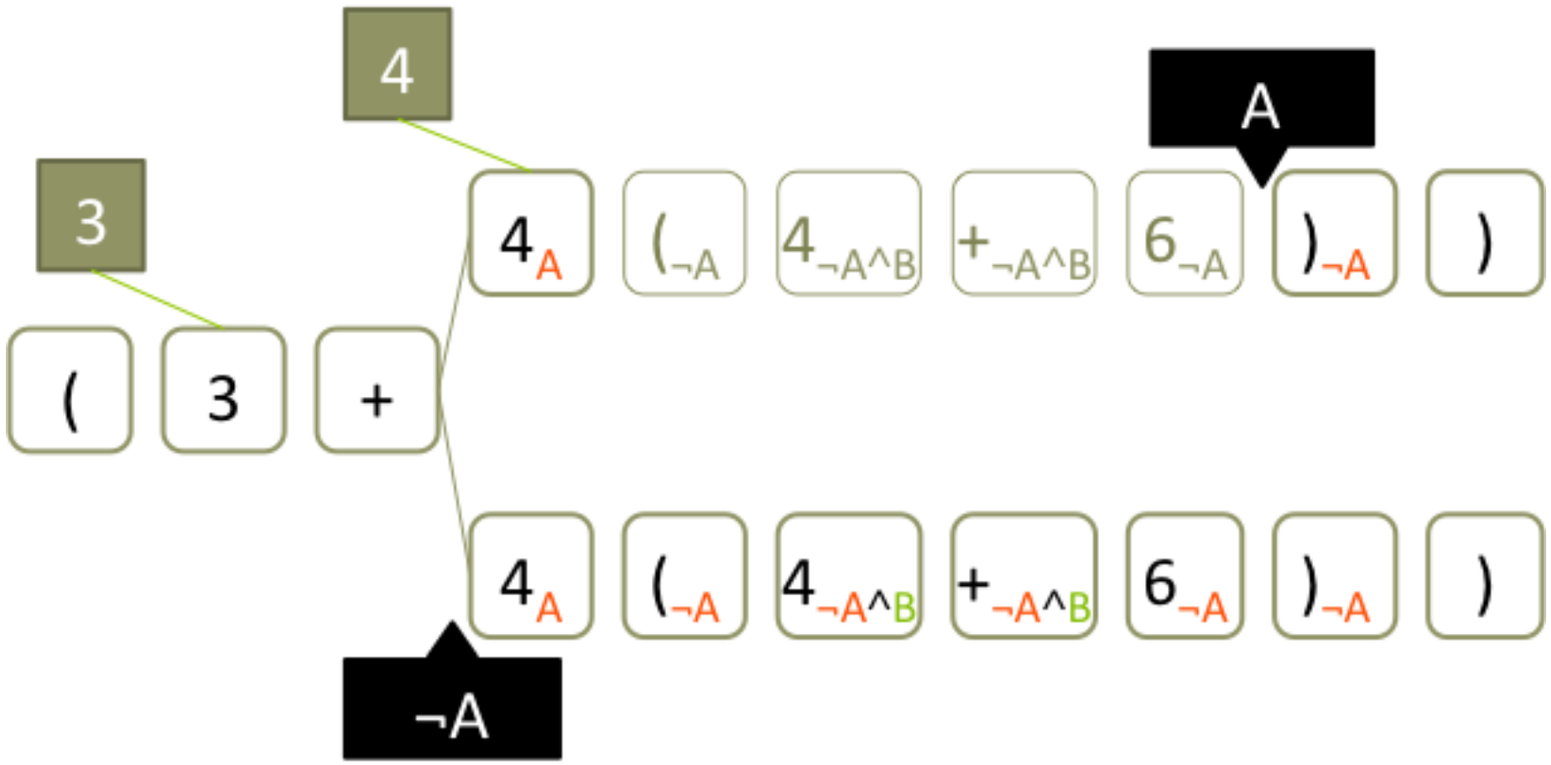

A
3
4A
(¬A
4¬A^B
+¬A^B
6¬A
)¬A
)
(
3
+
4A
(¬A
4¬A^B
+¬A^B
6¬A
)¬A
)
¬A

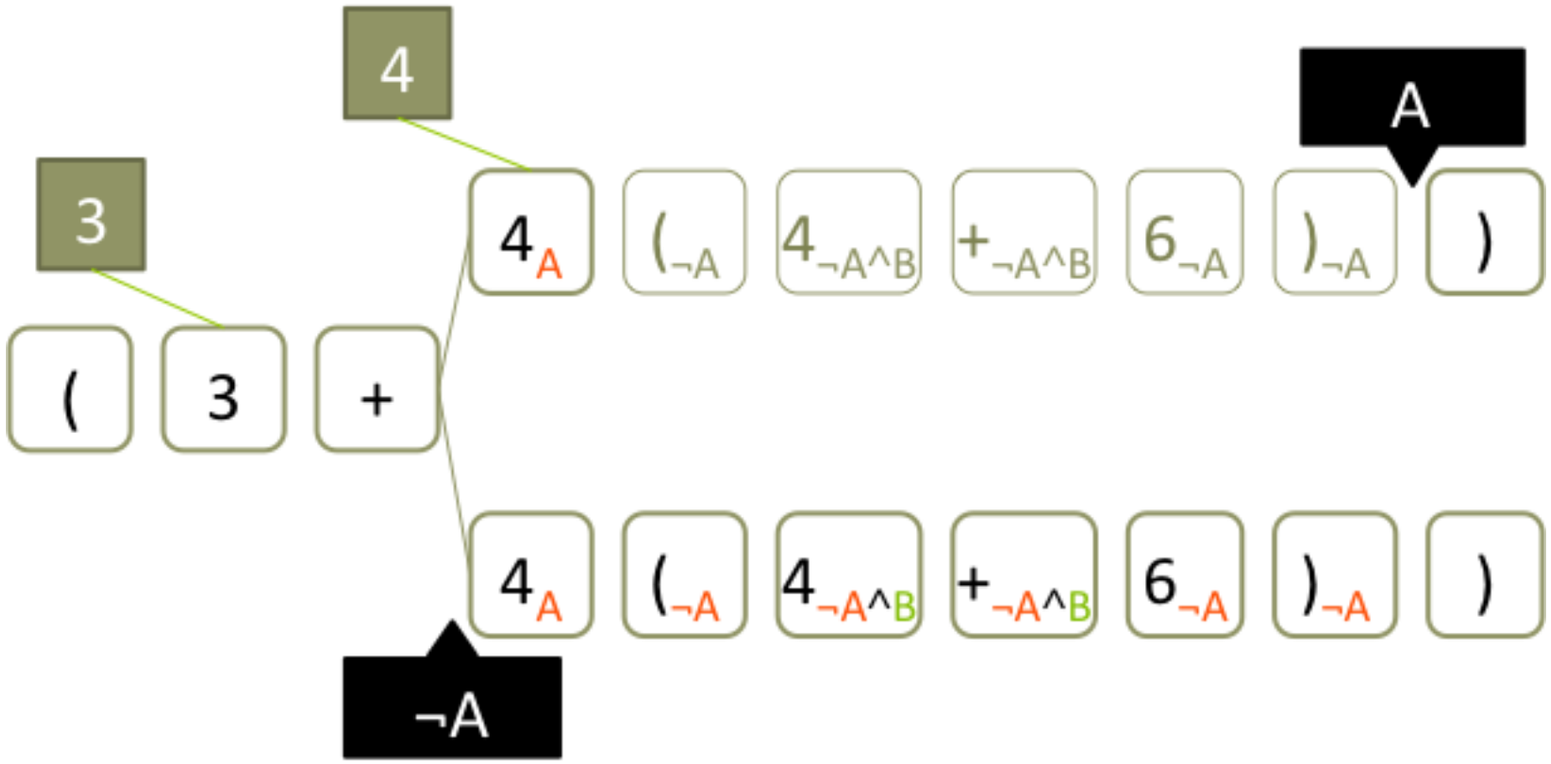

4
A
3
4A
(¬A
4¬A^B
+¬A^B
6¬A
)¬A
)
(
3
+
4A
(¬A
4¬A^B
+¬A^B
6¬A
)¬A
)
¬A

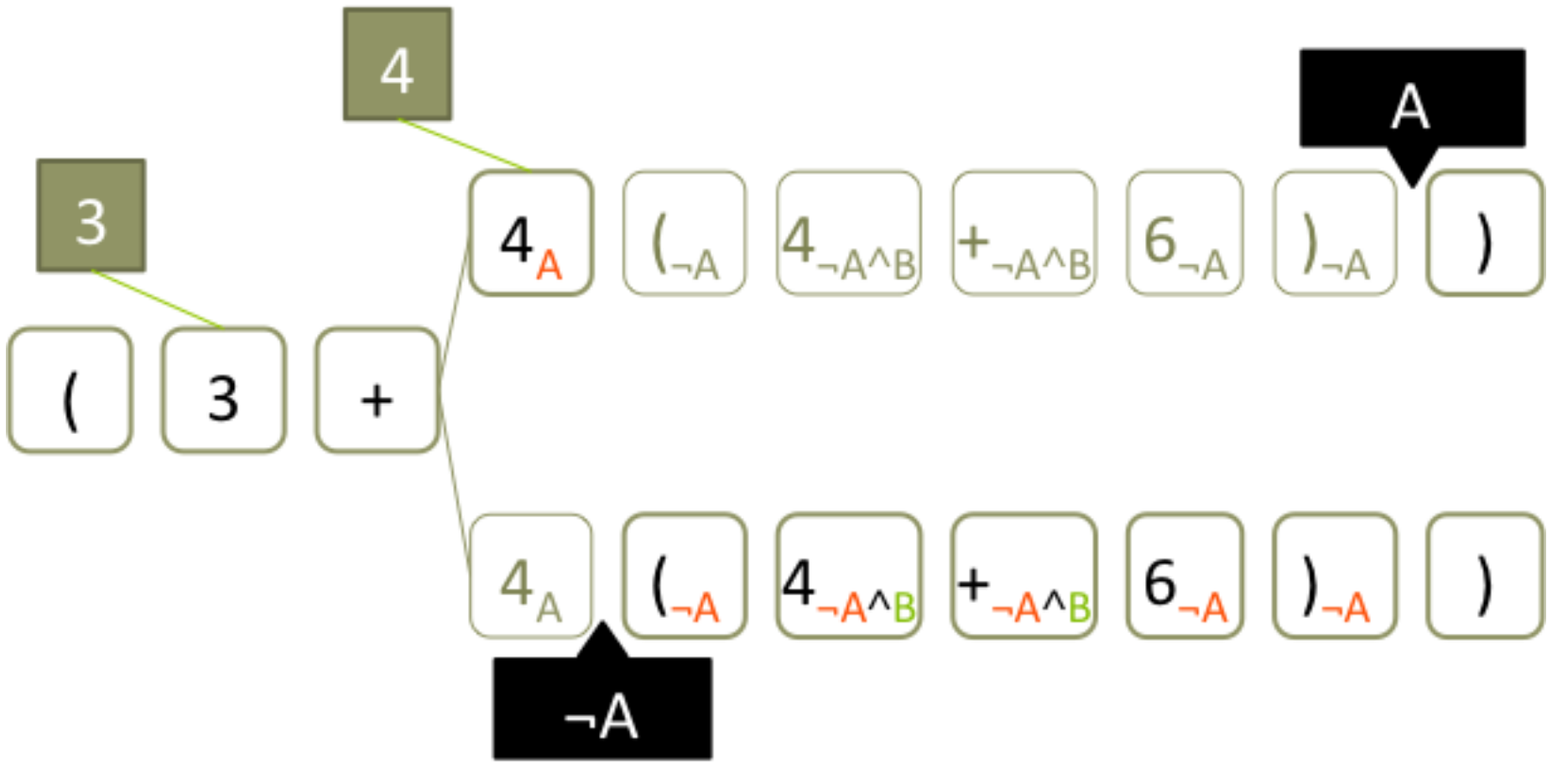

4
A
3
4A
(¬A
4¬A^B
+¬A^B
6¬A
)¬A
)
(
3
+
4A
(¬A
4¬A^B
+¬A^B
6¬A
)¬A
)
¬A

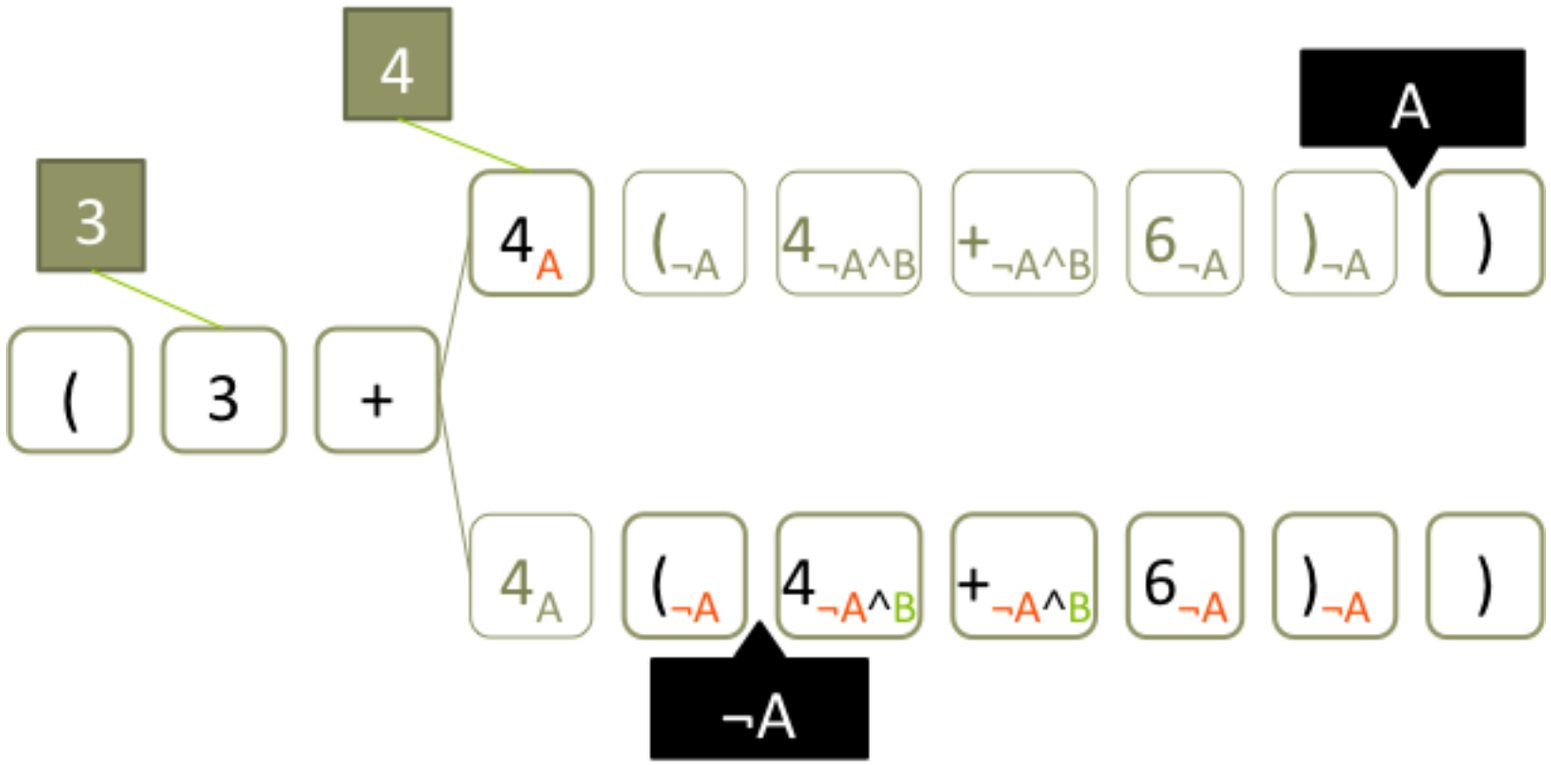

4
A
3
4A
(¬A
4¬A^B
+¬A^B
6¬A
)¬A
)
(
3
+
4A
(¬A
4¬A^B
+¬A^B
6¬A
)¬A
)
¬A

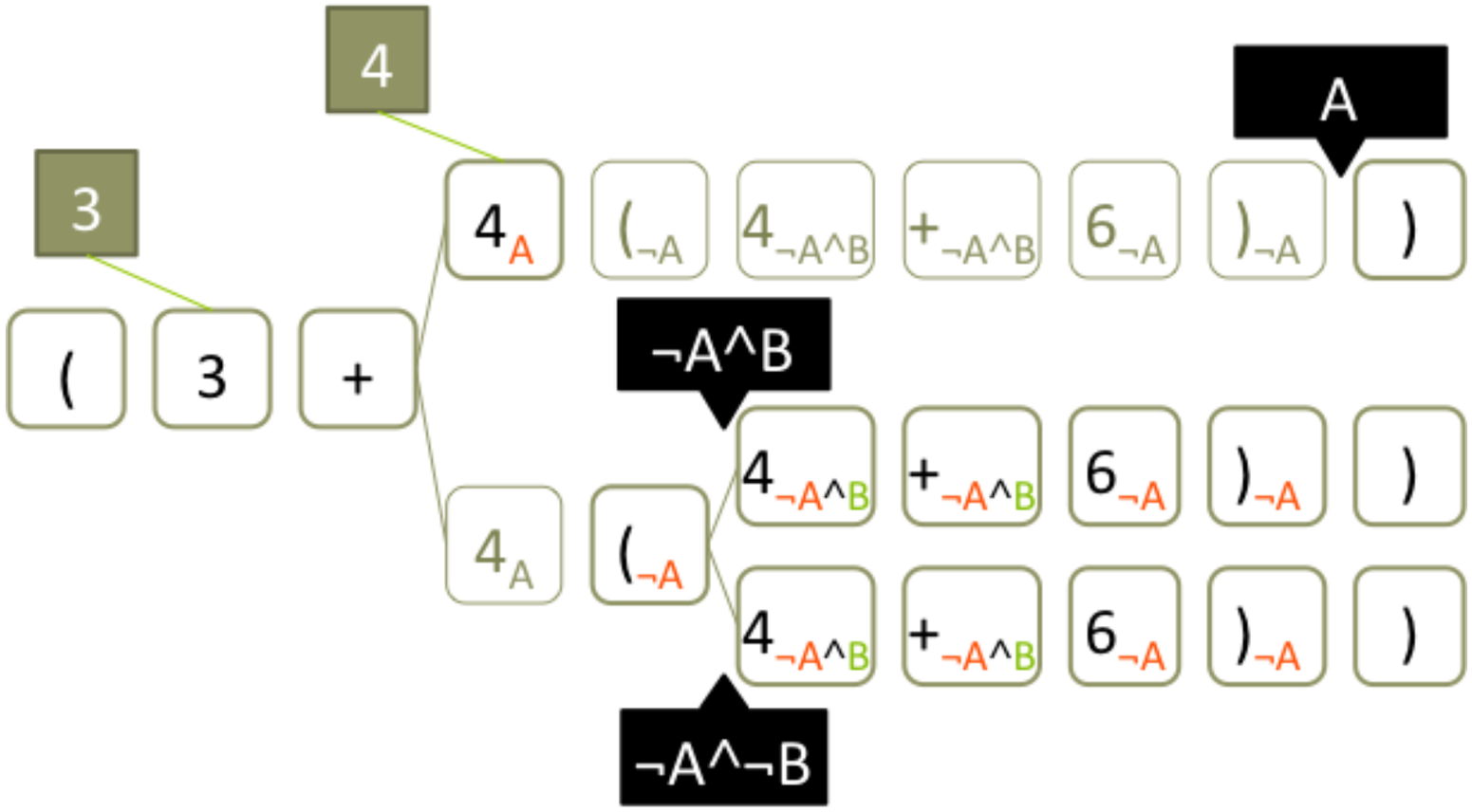

4
A
3
4A
(¬A
4¬A^B
+¬A^B
6¬A
)¬A
)
(
3
+
4A
(¬A
4¬A^B
+¬A^B
6¬A
)¬A
)
¬A

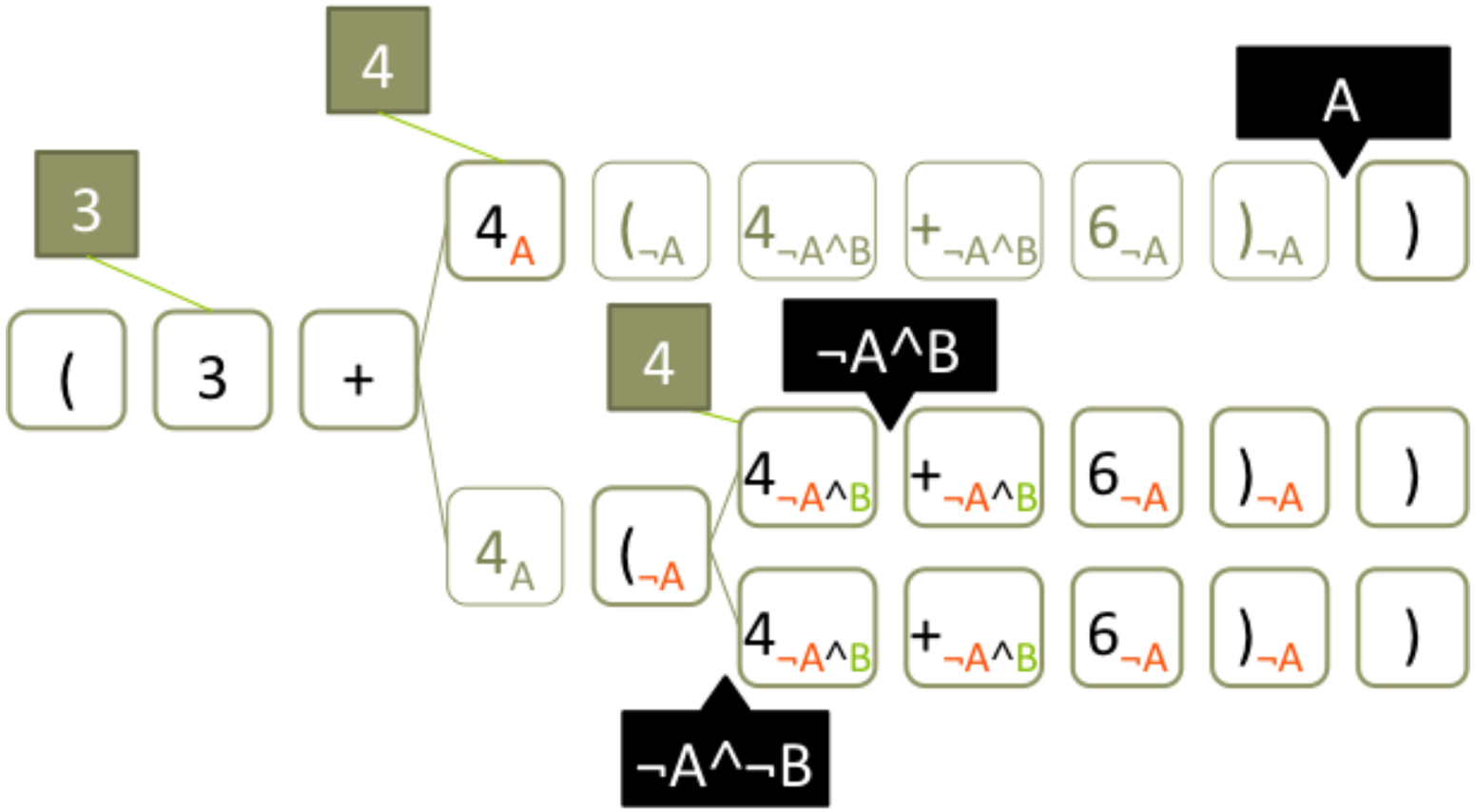

4
A
3
4A
(¬A
4¬A^B
+¬A^B
6¬A
)¬A
)
(
3
+
4A
(¬A
4¬A^B
+¬A^B
6¬A
)¬A
)
¬A

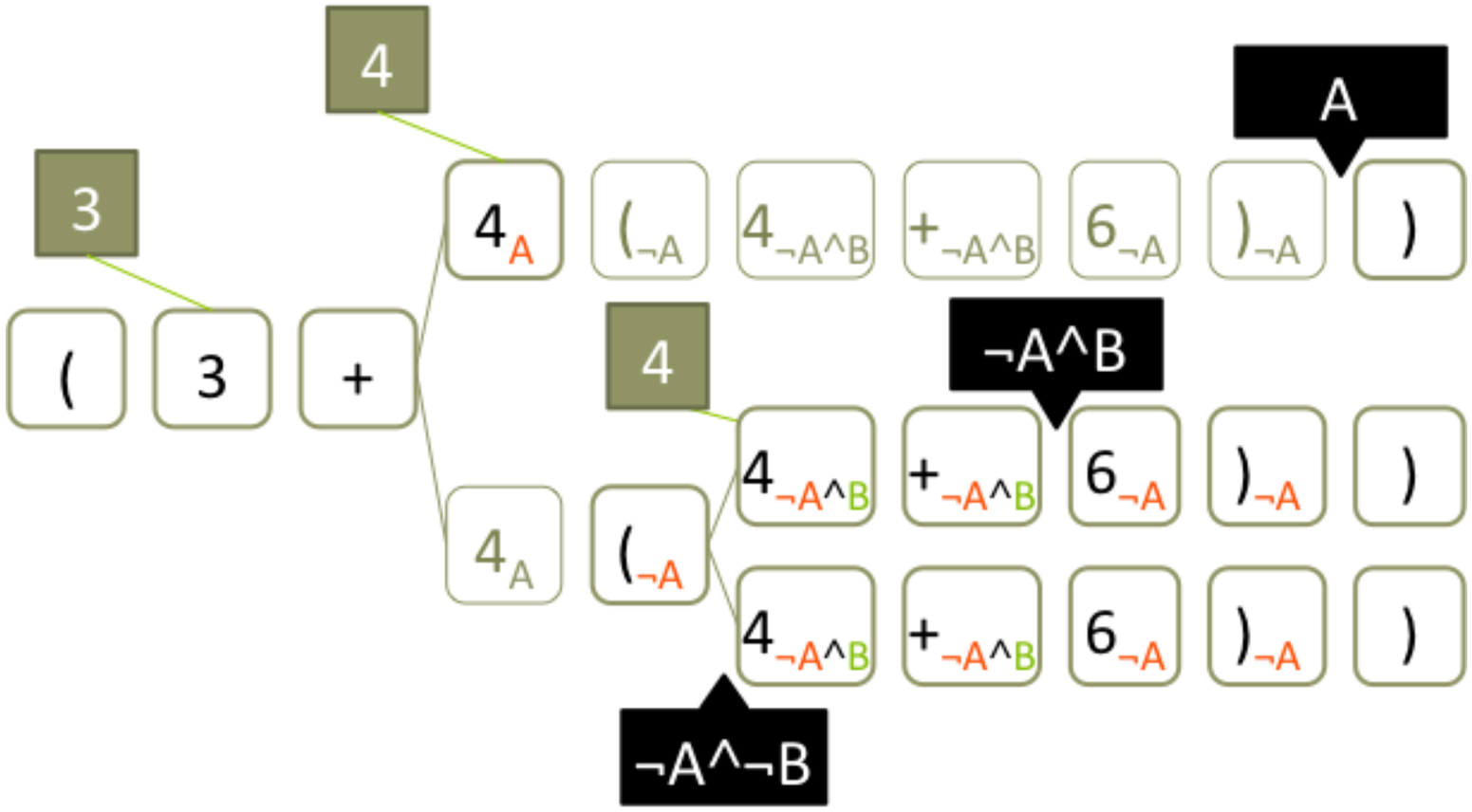

4
A
3
4A
(¬A
4¬A^B
+¬A^B
6¬A
)¬A
)
(
3
+
4A
(¬A
4¬A^B
+¬A^B
6¬A
)¬A
)
¬A

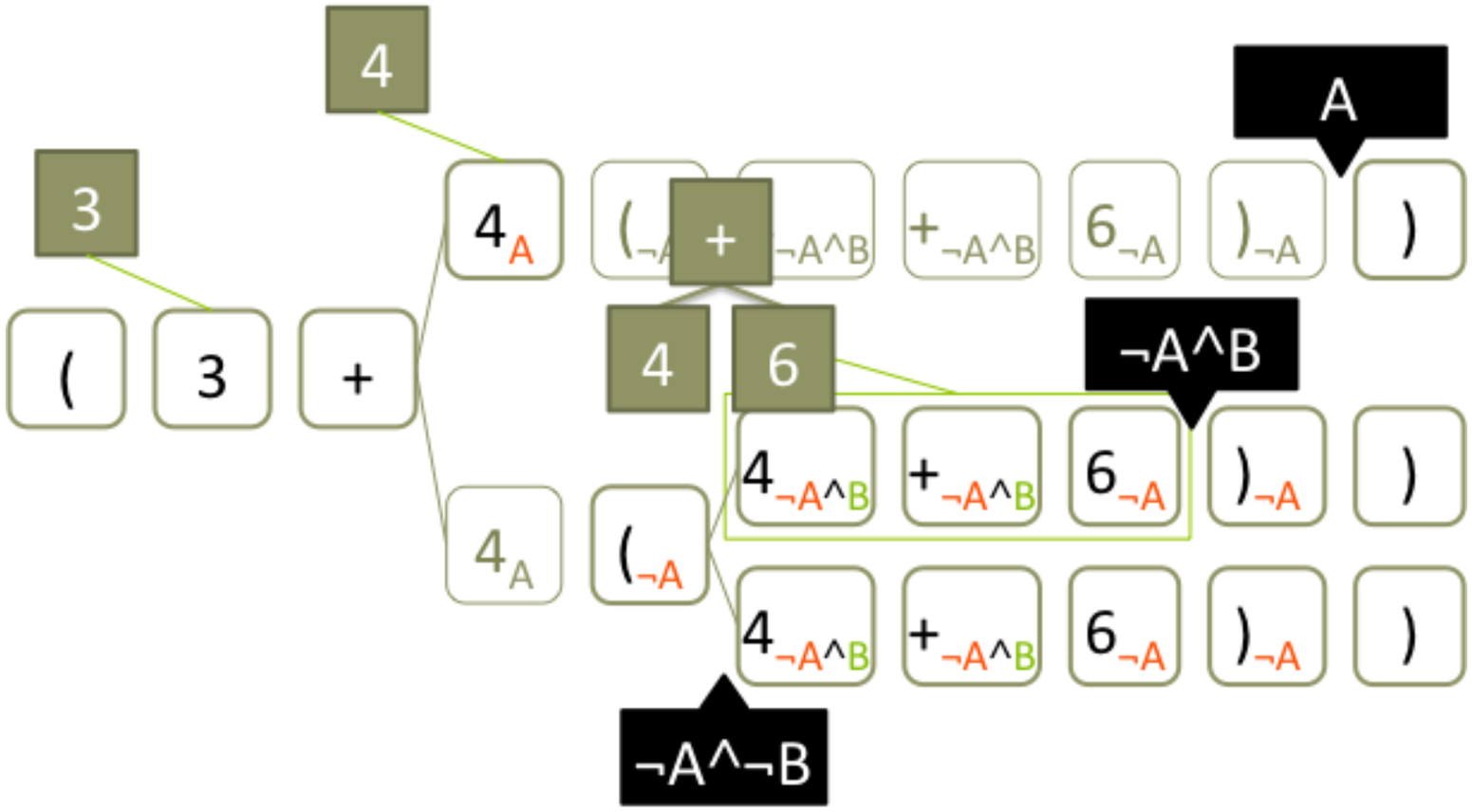

4
3
A
4A
(¬A
4¬A^B
+¬A^B
6¬A
)¬A
)
(
3
+
4A
(¬A
4¬A^B
+¬A^B
6¬A
)¬A
)
¬A

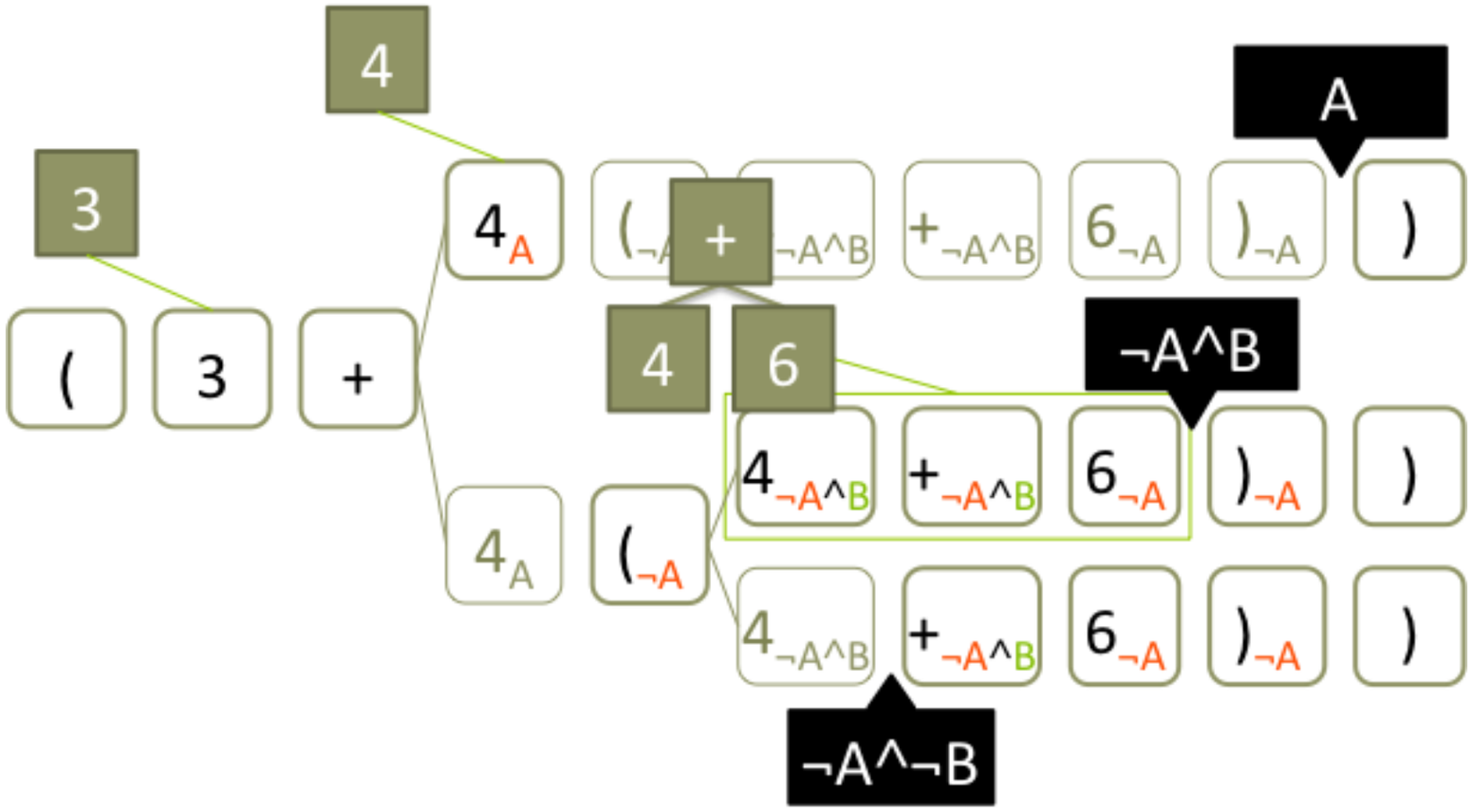

4
3
4A
(¬A
4¬A^B
+¬A^B
6¬A
)¬A
)
A
(
3
+
4A
(¬A
4¬A^B
+¬A^B
6¬A
)¬A
)
¬A

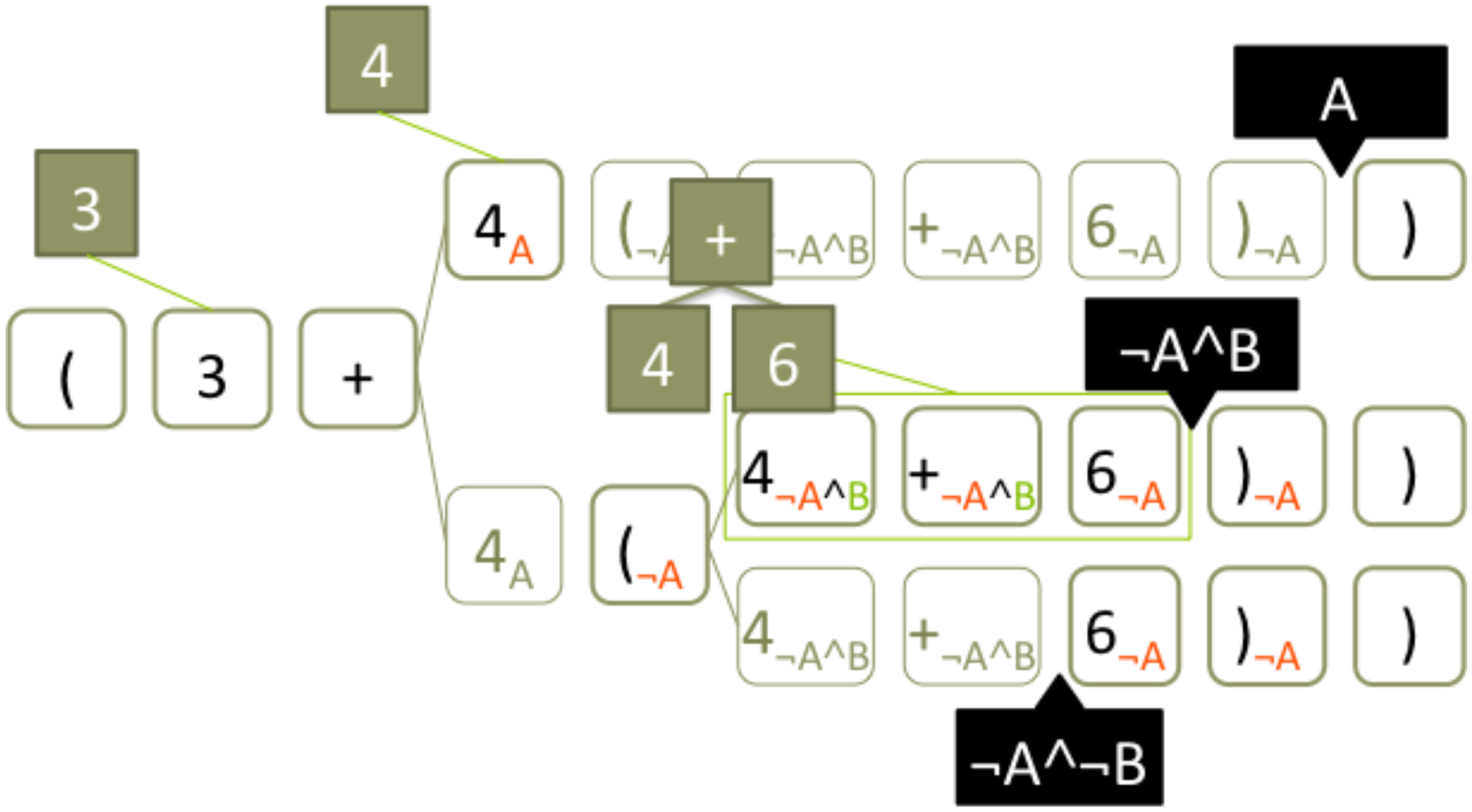

4
3
A
4_{A}
$(_{\neg A}$
$4_{\neg A \wedge B}$
$+_{\neg A \wedge B}$
$6_{\neg A}$
$)_{\neg A}$
)
(
3
+
¬A^B
$4_{\neg A \wedge B}$
$+_{\neg A \wedge B}$
$6_{\neg A}$
$)_{\neg A}$
)
4_{A}
$(_{\neg A}$
$4_{\neg A \wedge B}$
$+_{\neg A \wedge B}$
$6_{\neg A}$
$)_{\neg A}$
)
¬A^¬B

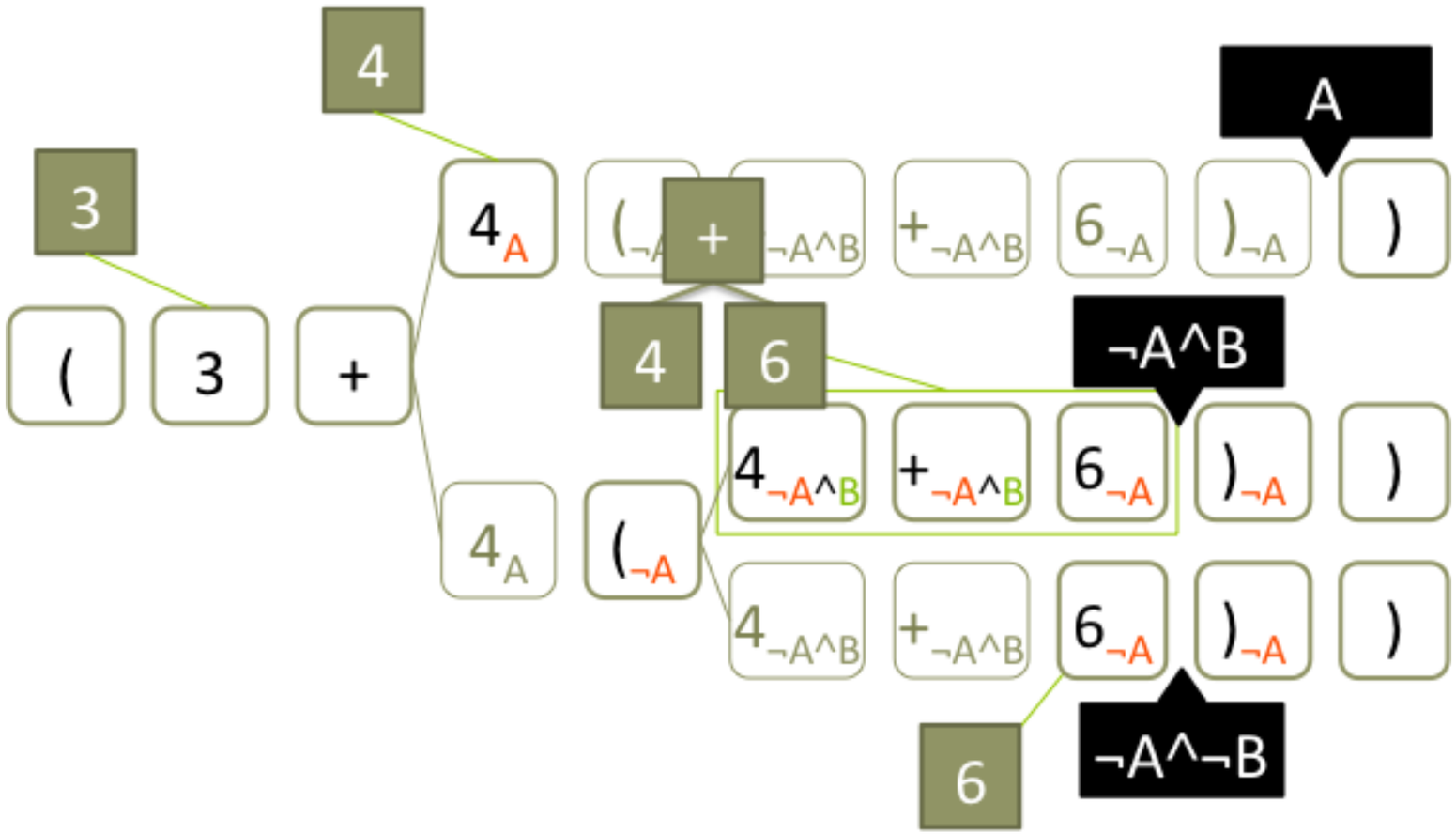

4
3
A
4A
(¬A
4¬A^B
+¬A^B
6¬A
)¬A
)
(
3
+
4
¬A^B
4A
(¬A
4¬A^B
+¬A^B
6¬A
)¬A
)
4¬A^B
+¬A^B
6¬A
)¬A
)
¬A^¬B

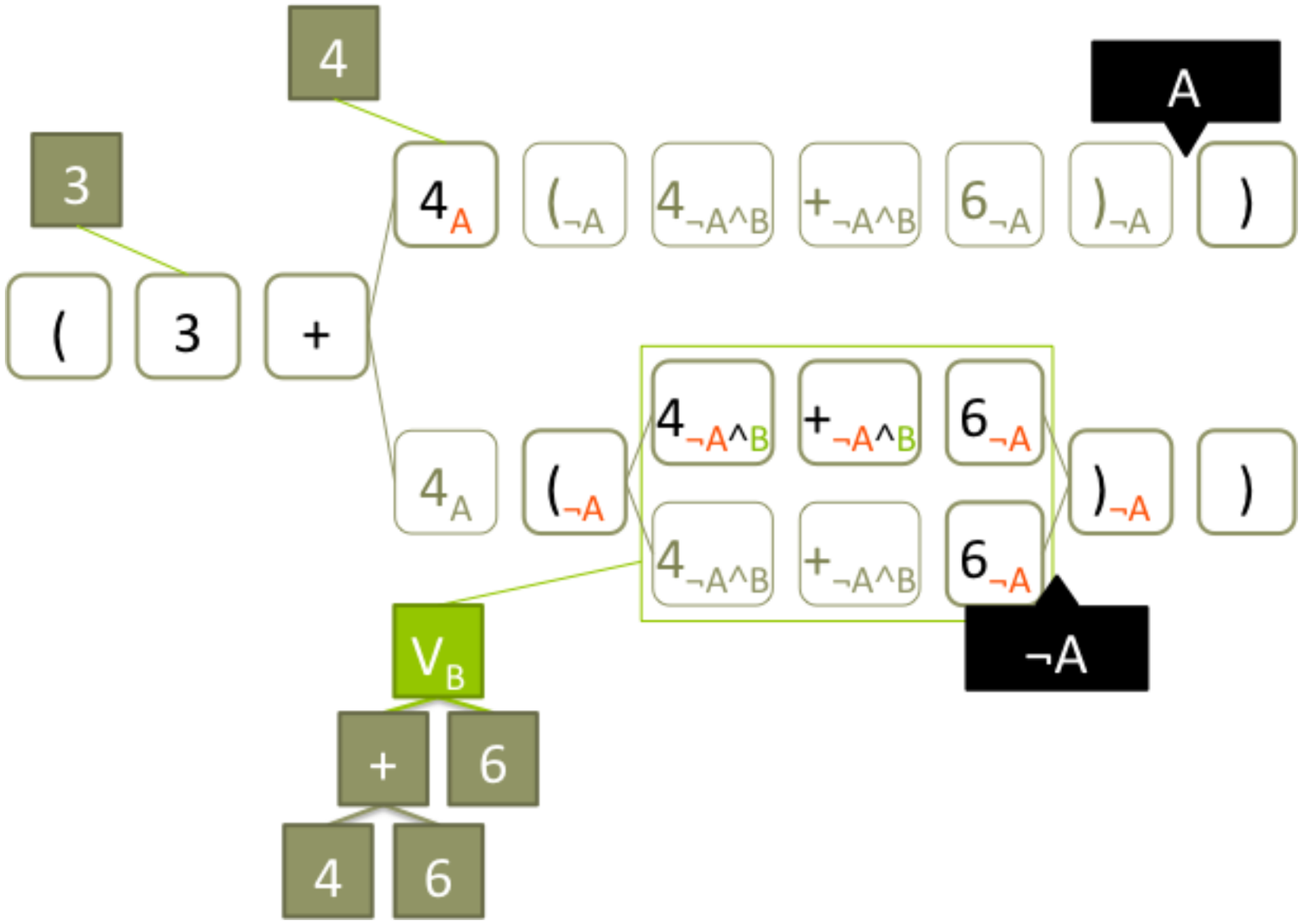

4
3
4A
(¬A
4¬A^B
+¬A^B
6¬A
)¬A
)
A
(
3
+
4
¬A^B
4A
(¬A
4¬A^B
+¬A^B
6¬A
)¬A
)
4¬A^B
+¬A^B
6¬A
)¬A
)
¬A^¬B

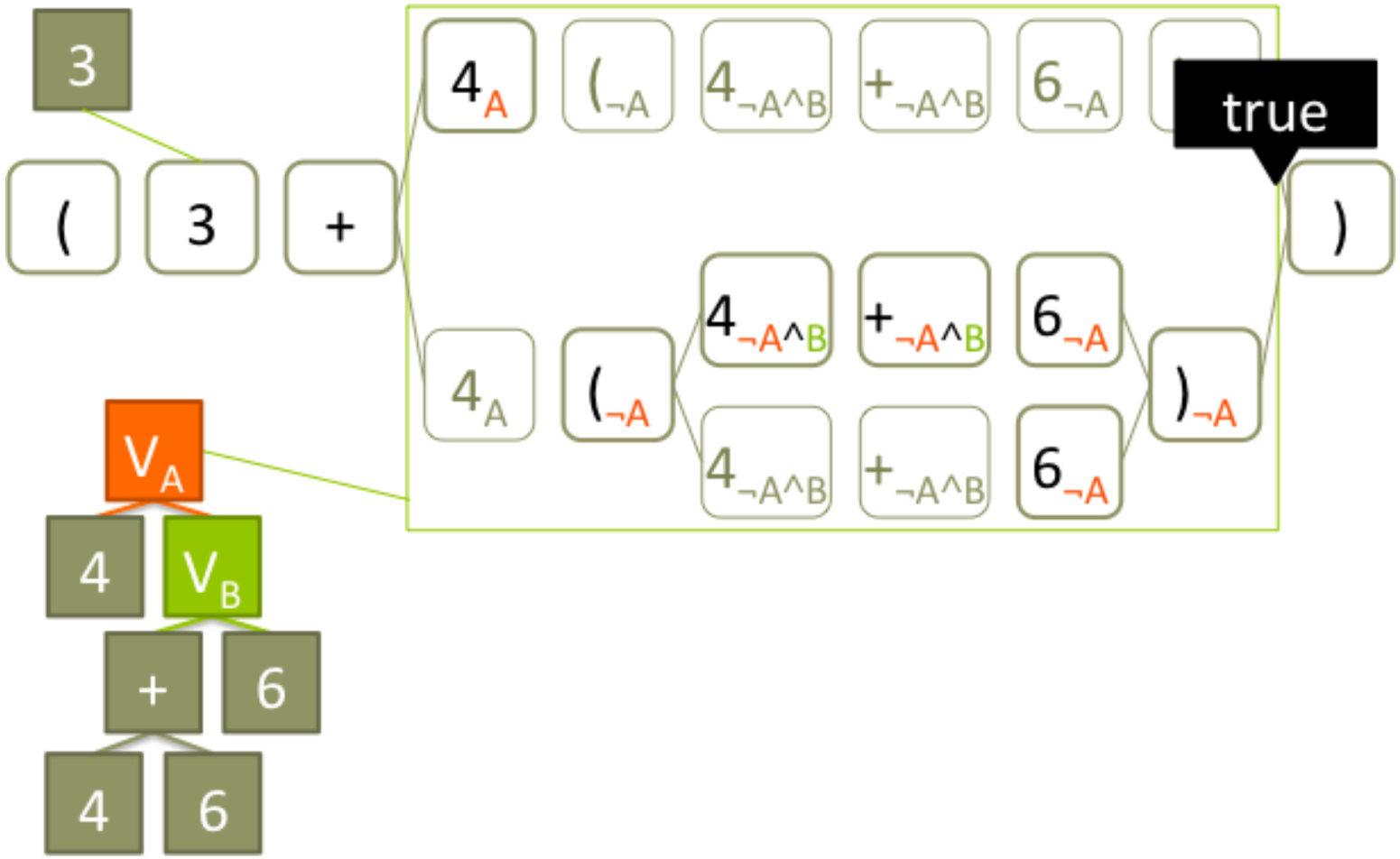

4
3
4A
(¬A
+¬A^B
+¬A^B
6¬A
)¬A
)
A
(
3
+
4
6
¬A^B
4¬A^B
+¬A^B
6¬A
)¬A
)
4A
(¬A
4¬A^B
+¬A^B
6¬A
)¬A
)
¬A^¬B

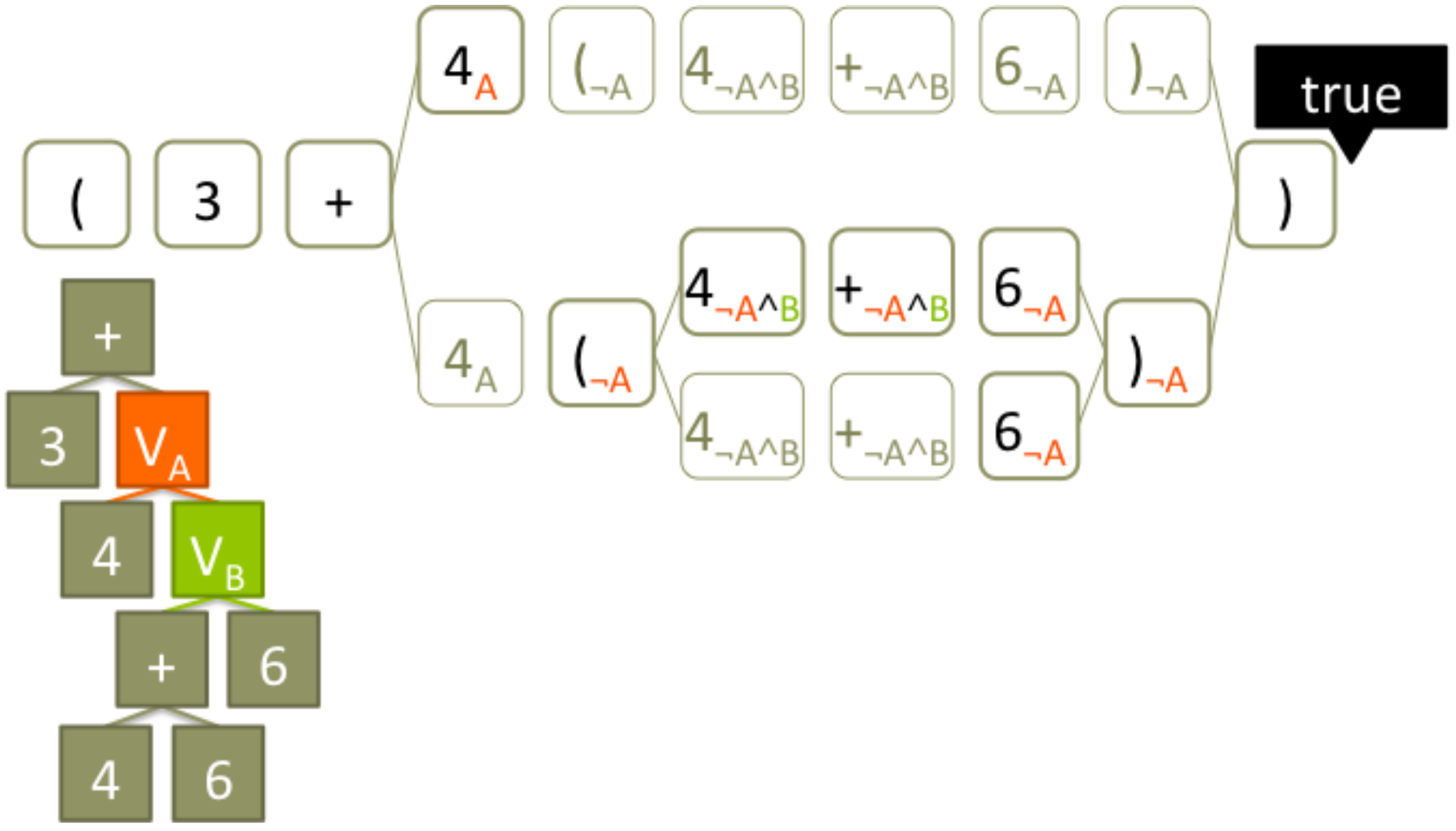

4
3
A
4A
(¬A
+¬A^B
+¬A^B
6¬A
)¬A
)
(
3
+
4
6
¬A^B
4¬A^B
+¬A^B
6¬A
)¬A
)
4A
(¬A
4¬A^B
+¬A^B
6¬A
)¬A
)
¬A^¬B

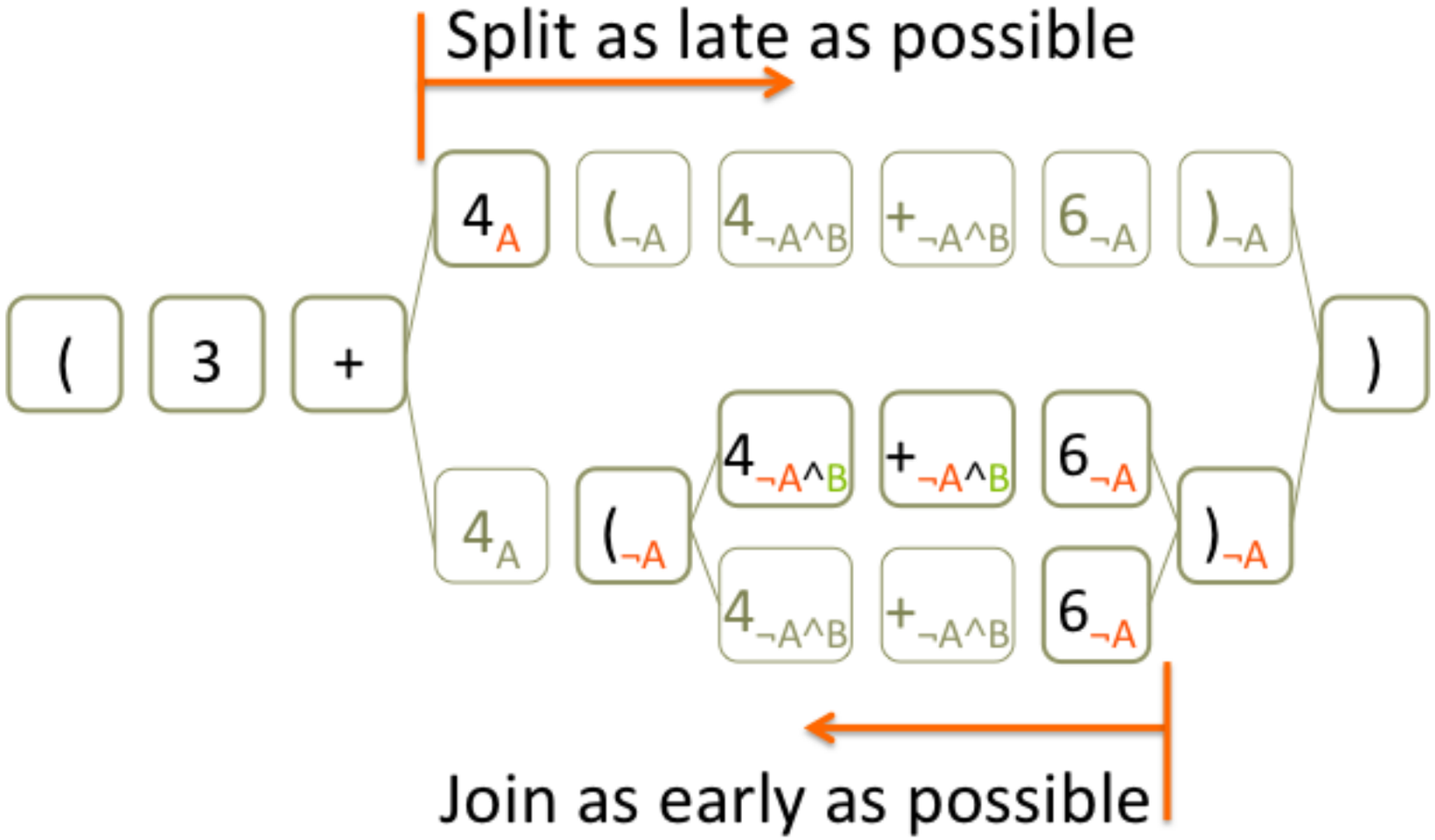

4
3
4A
(¬A
+¬A^B
+¬A^B
6¬A
)¬A
)
A
(
3
+
4
6
¬A^B
4¬A^B
+¬A^B
6¬A
)¬A
)
4A
(¬A
4¬A^B
+¬A^B
6¬A
)¬A
)
¬A^¬B


4
3
4A
(¬A
+¬A^B
+¬A^B
6¬A
)¬A
)
A
(
3
+
+
4
6
¬A^B
4¬A^B
+¬A^B
6¬A
)¬A
)
4A
(¬A
4¬A^B
+¬A^B
6¬A
)¬A
)
6
¬A^¬B


4
3
A
4_A
$(_{\neg A}$
$4_{\neg A \wedge B}$
$+_{\neg A \wedge B}$
$6_{\neg A}$
$)_{\neg A}$
)
(
3
+
4_A
$(_{\neg A}$
$4_{\neg A \wedge B}$
$+_{\neg A \wedge B}$
$6_{\neg A}$
$)_{\neg A}$
)
$4_{\neg A \wedge B}$
$+_{\neg A \wedge B}$
$6_{\neg A}$
V_B
¬A
+
6
4
6


3
(
3
+
4_A
$(_{\neg A}$
$4_{\neg A \wedge B}$
$+_{\neg A \wedge B}$
$6_{\neg A}$
true
)
4_A
$(_{\neg A}$
$4_{\neg A \wedge B}$
$+_{\neg A \wedge B}$
$6_{\neg A}$
$)_{\neg A}$
$4_{\neg A \wedge B}$
$+_{\neg A \wedge B}$
$6_{\neg A}$
V_A
4
V_B
+
6
4
6


(
3
+
4A
(¬A
4¬A^B
+¬A^B
6¬A
)¬A
true
)
4A
(¬A
4¬A^B
+¬A^B
6¬A
)¬A
4¬A^B
+¬A^B
6¬A
+
3
VA
4
VB
+
6
4
6

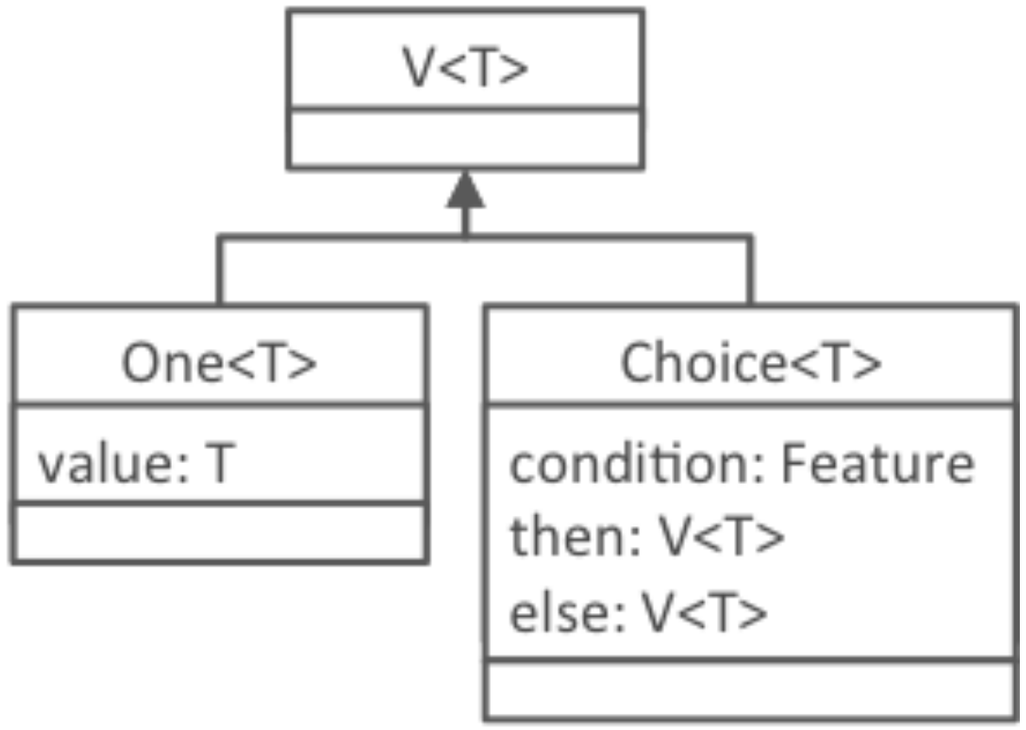


Can avoid splitting for invalid combinations, if variability model exists

Short TypeChef Demo

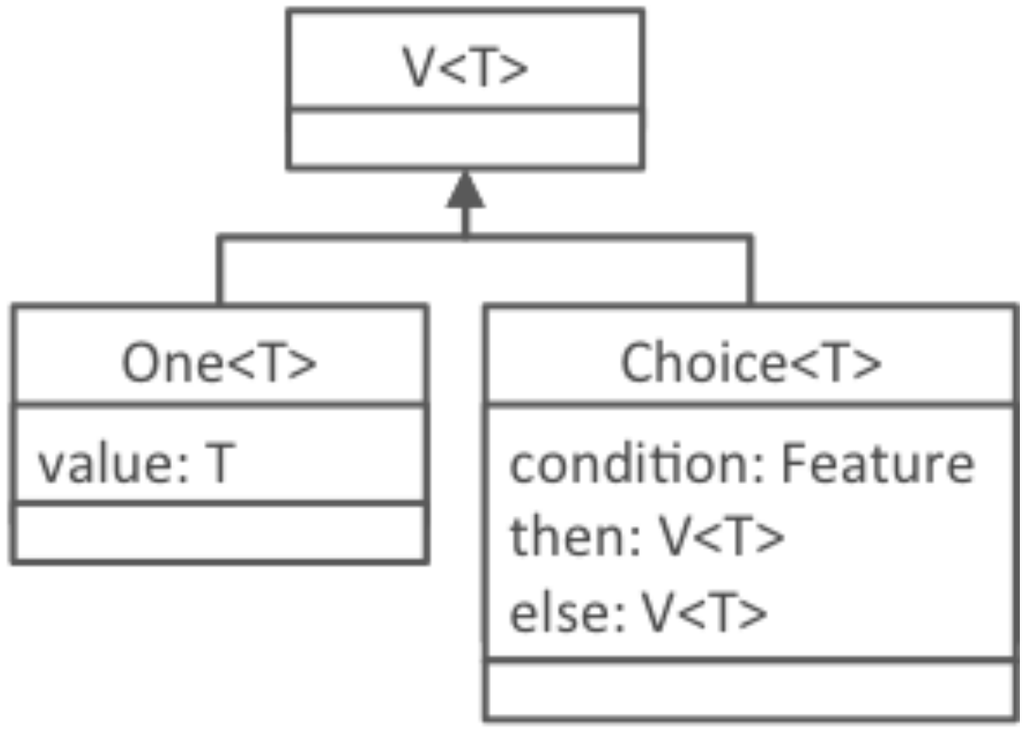
Underlying Variational Data Structures

```
abstract class V[T]

case class One[T](
    value: T
) extends V[T]

case class Choice[T](
    condition: Feature,
    then: V[T],
    else: V[T]
) extends V[T]
```



Underlying Variational Data Structures

```
abstract class V[T]

case class One[T](
    value: T
) extends V[T]

case class Choice[T](
    condition: Feature,
    then: V[T],
    else: V[T]
) extends V[T]
```



Choice(A, One(2), One(3))
Choice(A, One(2), Choice(B, One(4),One(5))

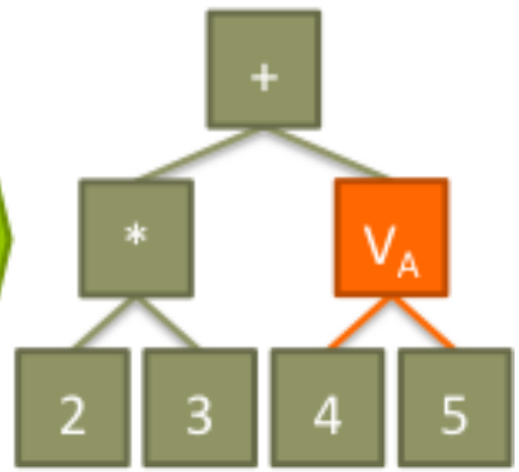

#ifdef A
#define X 4
#else
#define X 5
#endif
(2*3)+X
Variability-Aware Lexer
4
5¬A
Variability-Aware Parser
+
*
VA
2
3
4
5

Variability-Aware Bug Finding

Variability-Aware Metrics

Variability-Aware Control/ Data-Flow

Variability-Aware Parser



Variability-Aware Type Syst.

module …
export .. if A
import .. if ¬A

Variability-Aware Linker

Variability-Aware Interpreter

https://github.com/ckaestne/TypeChef

Further Readings

- Kastner et al., Variability-Aware Parsing in the Presence of Lexical Macros and Conditional Compilation, OOPSLA ’11

- Walkingshaw et al., Variational Data Structures: Exploring Tradeoffs in Computing with Variability, Onward! ’14

- Also check out the references cited in those papers