Perses: Syntax-Guided Program Reduction

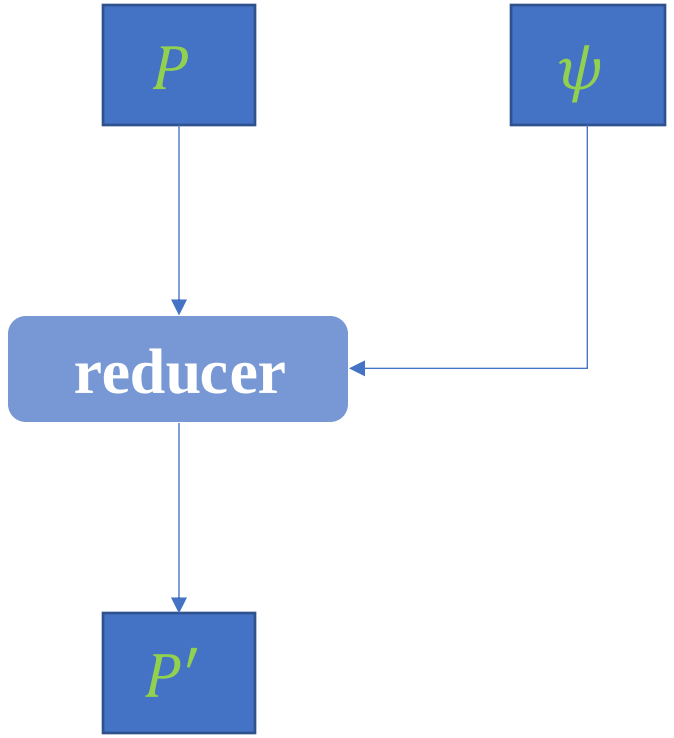
Chengnian Sun, Yuanbo Li, Qirun Zhang, Tianxiao Gu, Zhendong Su

University of California, Davis, USA

program reduction

Input:

- P: a program
- ψ: a property, and $\psi(P)$

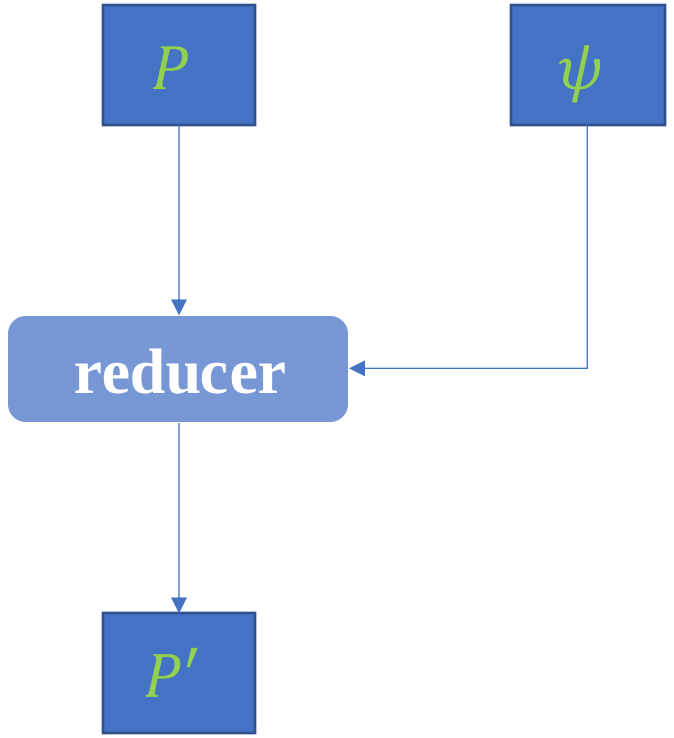


program reduction

Input:

- P: a program
- ψ: a property, and $\psi(P)$

Goal: remove ψ-irrelevant elements from P



program reduction

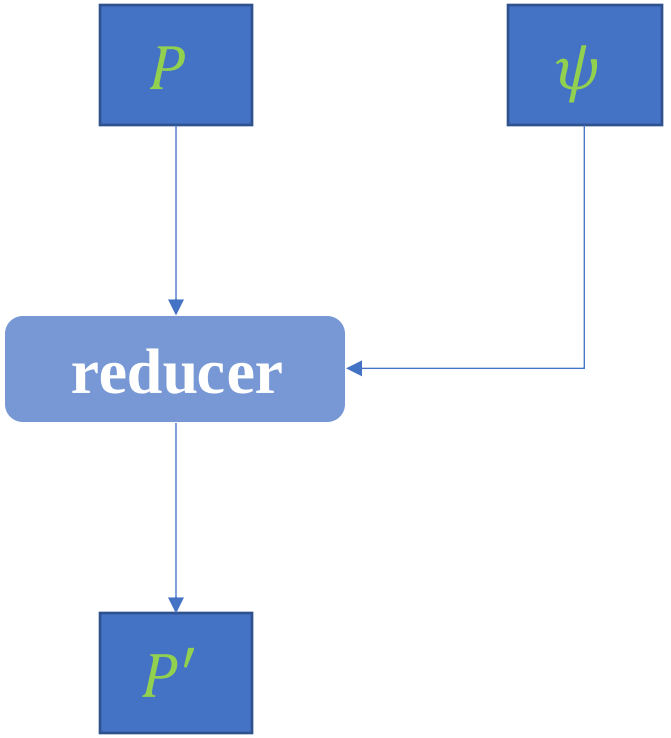
Input:

- P: a program
- ψ: a property, and $\psi(P)$

Goal: remove ψ-irrelevant elements from P

Output:

- P': a minimized program from P, *s.t.* $\psi(P')$



program reduction

specialized test input **minimization**

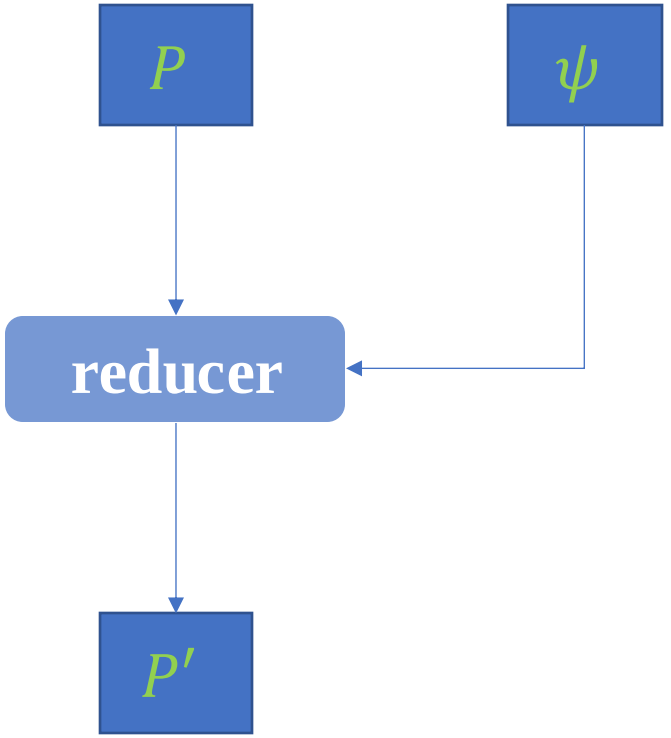
Input:

- P: a program
- ψ: a property, and $\psi(P)$

Goal: remove ψ-irrelevant elements from P

Output:

- P': a minimized program from P, *s.t.* $\psi(P')$



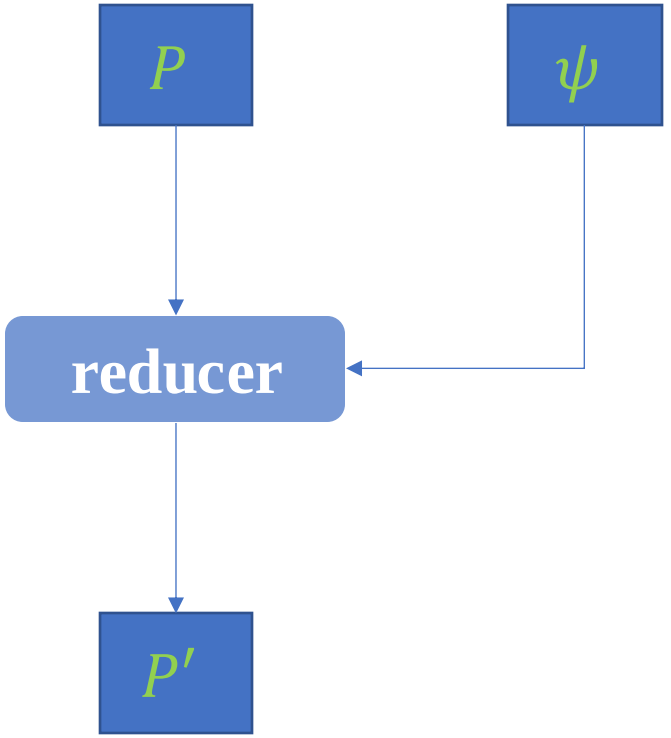
program reduction – an **important** problem

specialized test input **minimization**

Input:

- P: a program
- ψ: a property, and $\psi(P)$

Goal: re... ψ-irrelevant elements from P



ψ can be a bug in:

- static analyses
  - ❖ Frama-C, Soot, Wala

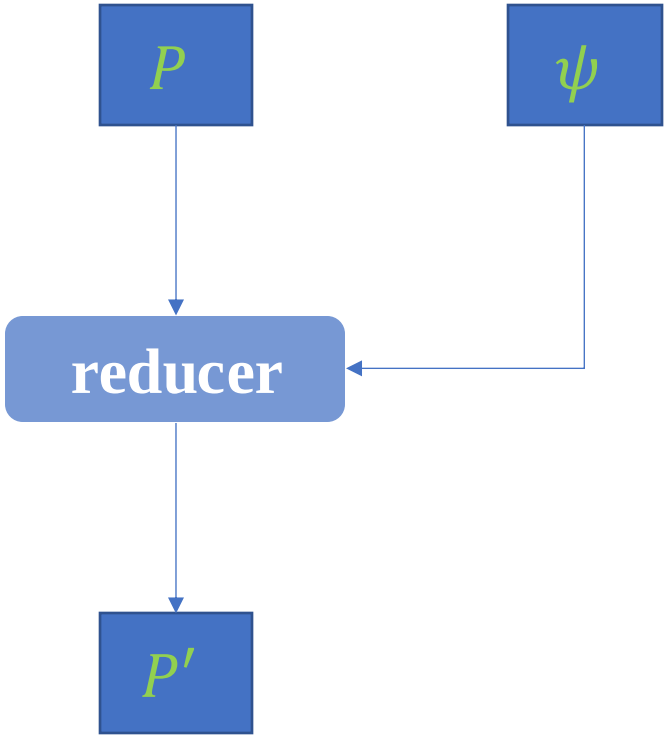
program reduction – an **important** problem

specialized test input **minimization**

Input:

- P: a program
- ψ: a property, and $\psi(P)$

Goal: re... ψ-irrelevant elements from P



ψ can be a bug in:

- static analyses
  - ❖ Frama-C, Soot, Wala
- refactoring engines
  - ❖ Eclipse, IntelliJ, Netbeans

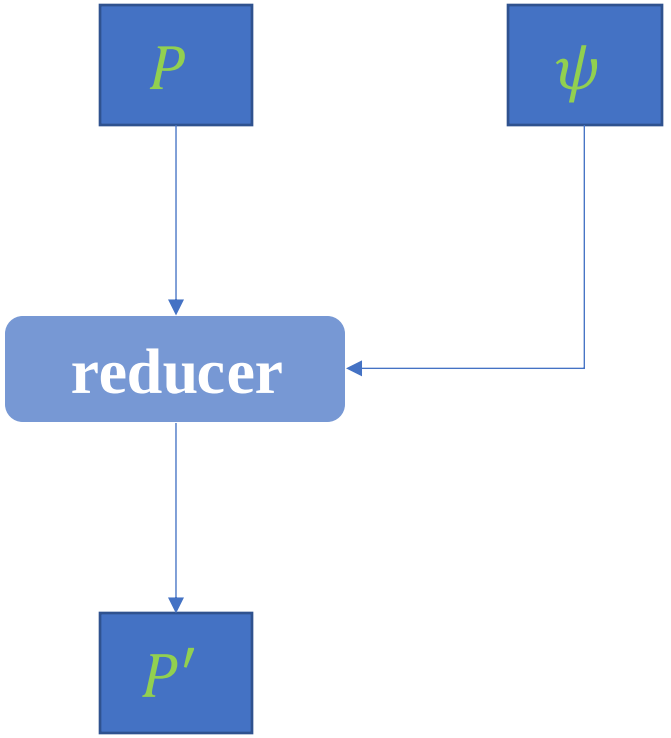
program reduction – an **important** problem

specialized test input **minimization**

Input:

- P: a program
- ψ: a property, and $\psi(P)$

Goal: re... ψ-irrelevant elements from P



ψ can be a bug in:

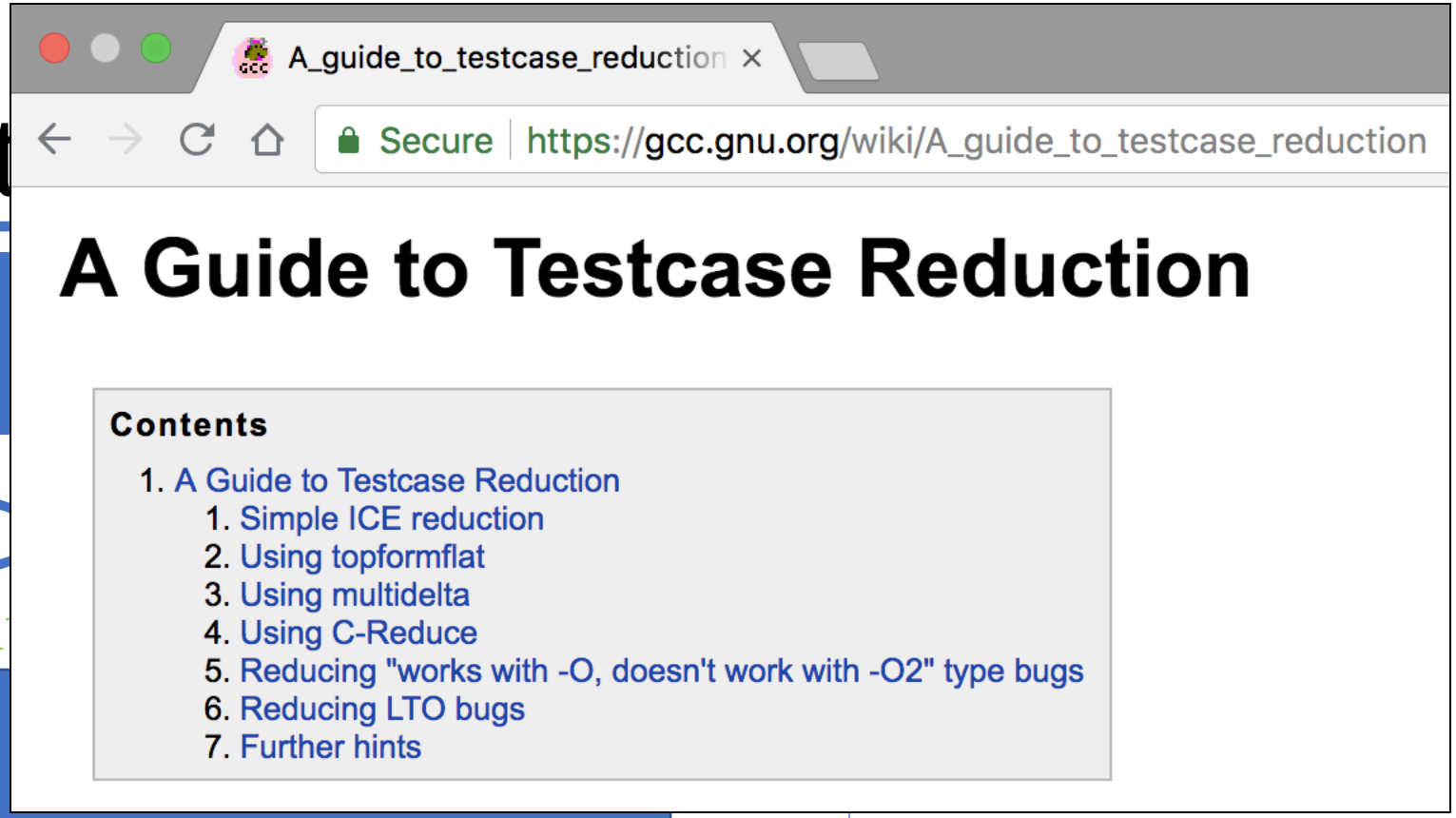
- static analyses
  - ❖ Frama-C, Soot, Wala
- refactoring engines
  - ❖ Eclipse, IntelliJ, Netbeans
- compilers
  - ❖ **GCC** (100k+ bugs), **LLVM** (50k+ bugs), **JVM, V8**

program reduct

Input:

- P: a program
- ψ: a property, and $\psi(P)$

Goal: re... ψ-irrelevan...



P'

ψ can be a bug in:

- static analyses
  - ❖ Frama-C, Soot, Wala
- refactoring engines
  - ❖ Eclipse, IntelliJ, Netbeans
- compilers
  - ❖ **GCC** (100k+ bugs), **LLVM** (50k+ bugs), **JVM, V8**

an example

P

```c
int main() {
 int a = 1;
 if (a) {
   printf("%d\n", a);
   printf("Hello ");
   printf("world!\n");
   printf("End\n");
 }
 return 0;
}
```

```
$ gcc t.c ; ./a.out
1
Hello world!
End
```

an example

P

```c
int main() {
 int a = 1;
 if (a) {
   printf("%d\n", a);
   printf("Hello ");
   printf("world!\n");
   printf("End\n");
 }
 return 0;
}
```

```
$ gcc t.c ; ./a.out
1
Hello world!
End
```

property: ψ

print “**Hello world!**”

an example

P

```c
int main() {
 int a = 1;
 if (a) {
   printf("%d\n", a);
   printf("Hello ");
   printf("world!\n");
   printf("End\n");
 }
 return 0;
}
```

```
$ gcc t.c ; ./a.out
1
Hello world!
End
```

property: ψ

print "**Hello world!**"

ideal result

```c
int main() {
 printf("Hello ");
 printf("world!\n");
 return 0;
}
```

```
$ gcc t.c ; ./a.out
Hello world!
```

state-of-the-art

- DD: Delta Debugging
  - binary search (delete lines each time and check ψ)
  - generate many **syntactically invalid** variants

ideal result

```
int main() {
 printf("Hello ");
 printf("world!\n");
 return 0;
}
```

DD & HDD

```
int main() {
 int a = 1;
 if (a) {
   printf("Hello ");
   printf("world!\n");
 }
 return 0;
}
```

an example

P

```
 1: int main() {
 2:   int a = 1;
 3:   if (a) {
 4:     printf("%d\n", a);
 5:     printf("Hello ");
 6:     printf("world!\n");
 7:     printf("End\n");
 8:   }
 9:   return 0;
10: }
```

```
$ gcc t.c ; ./a.out
1
Hello world!
End
```

property: ψ

print "**Hello world!**"

DD & HDD

```
int main() {
  int a = 1;
  if (a) {
    printf("Hello ");
    printf("world!\n");
  }
  return 0;
}
```

state-of-the-art

- DD: Delta Debugging
  - binary search (delete lines each time and check ψ)
  - generate many **syntactically invalid** variants
- HDD: Hierarchical Delta Debugging
  - parse a program into a tree
  - breadth-first search and apply DD on each level
  - better at program reduction
  - generate many **syntactically invalid** variants

ideal result

```
int main() {
 printf("Hello ");
 printf("world!\n");
 return 0;
}
```

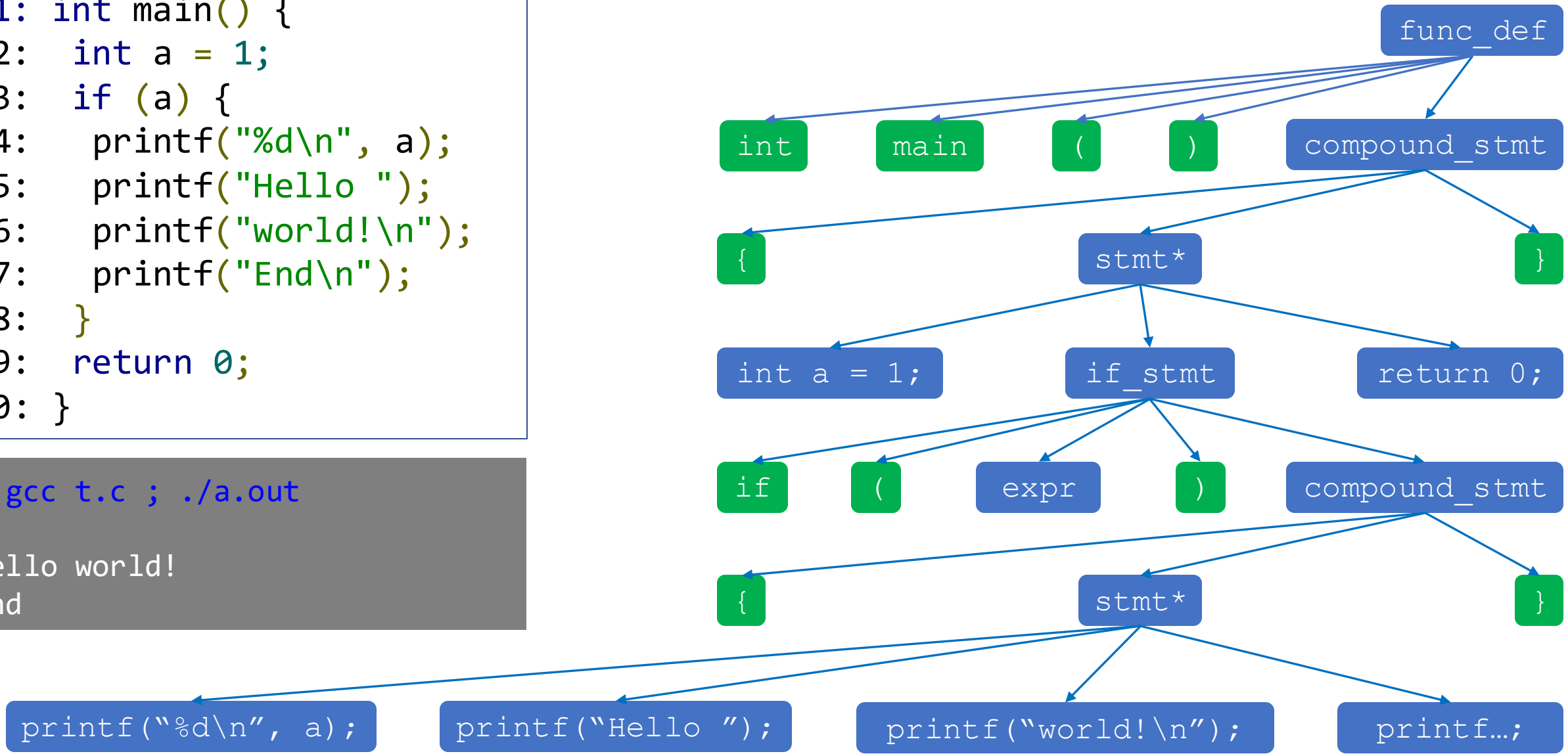
DD & HDD

```
int main() {
 int a = 1;
 if (a) {
   printf("Hello ");
   printf("world!\n");
 }
 return 0;
}
```

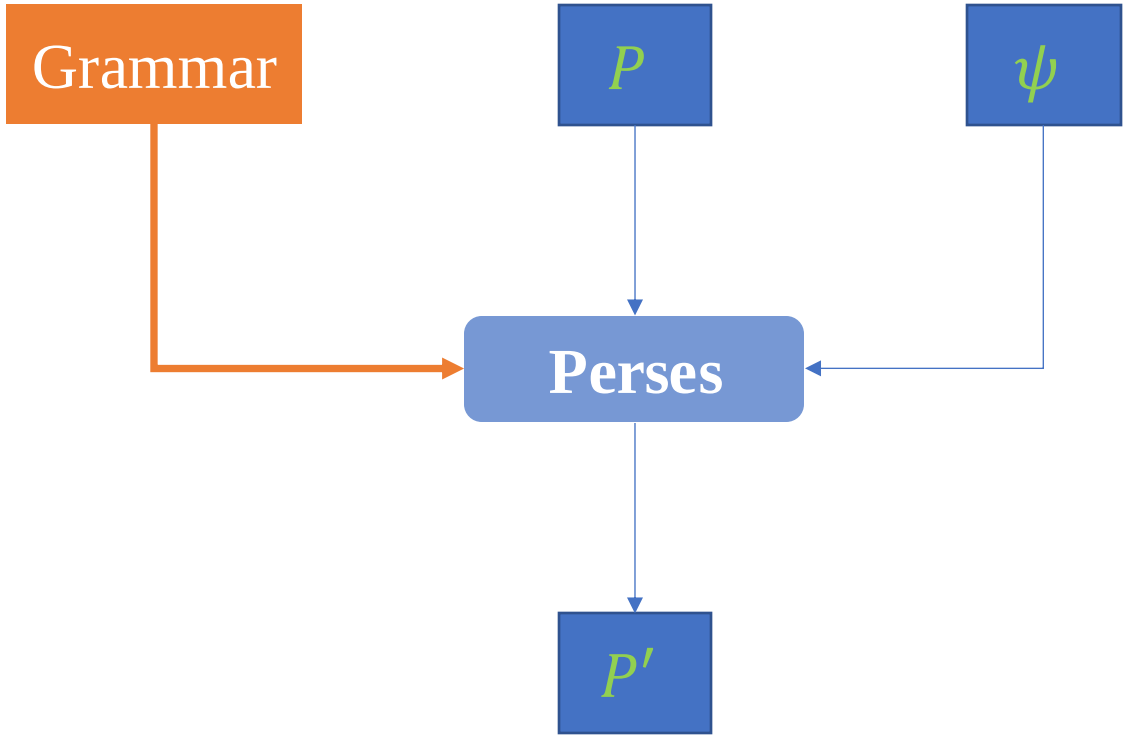
P

```
 1: int main() {
 2:   int a = 1;
 3:   if (a) {
 4:     printf("%d\n", a);
 5:     printf("Hello ");
 6:     printf("world!\n");
 7:     printf("End\n");
 8:   }
 9:   return 0;
10: }
```

```
$ gcc t.c ; ./a.out
1
Hello world!
End
```



Perses: fully syntax-guided



Perses: fully syntax-guided

- Avoid generating **syntactically invalid** variants

```c
int main() {
 int a = 1;
 if (a) {
   printf("%d\n", a);
   printf("Hello ");
   printf("world!\n");
   printf("End\n");
 }
 return 0;
}
```

```
<func_def>      ::= <type> <identifier> ‘(’ ‘)’ <compound_stmt>
<compound_stmt> ::= ‘{’ <stmt_star> ‘}’
<stmt_star>     ::= <stmt>*      // A list of zero or more statements
<stmt>          ::= <expr_stmt>
                  | <decl_stmt>
                  | <if_stmt>
                  | <compound_stmt>
<if_stmt>  ::= ‘if’ ‘(‘ <expr> ‘)’ <stmt>
```

Perses: fully syntax-guided

- Avoid generating **syntactically invalid** variants
  - Most symbols are NOT removable
    - e.g., all symbols in `<func_def>`

```c
int main() {
  int a = 1;
  if (a) {
    printf("%d\n", a);
    printf("Hello ");
    printf("world!\n");
    printf("End\n");
  }
  return 0;
}
```

```
<func_def>      ::= <type> <identifier> '(' ')' <compound_stmt>
<compound_stmt> ::= '{' <stmt_star> '}'
<stmt_star>     ::= <stmt>*      // A list of zero or more statements
<stmt>          ::= <expr_stmt>
                  | <decl_stmt>
                  | <if_stmt>
                  | <compound_stmt>
<if_stmt>  ::= 'if' '(' <expr> ')' <stmt>
```

Perses: fully syntax-guided

- Avoid generating **syntactically invalid** variants
  - Most symbols are NOT removable
    - e.g., all symbols in `<func_def>`
  - Except symbols that are quantified by `*`, `+` and `?`
    - e.g., `printf` statements in body of `if` are removable

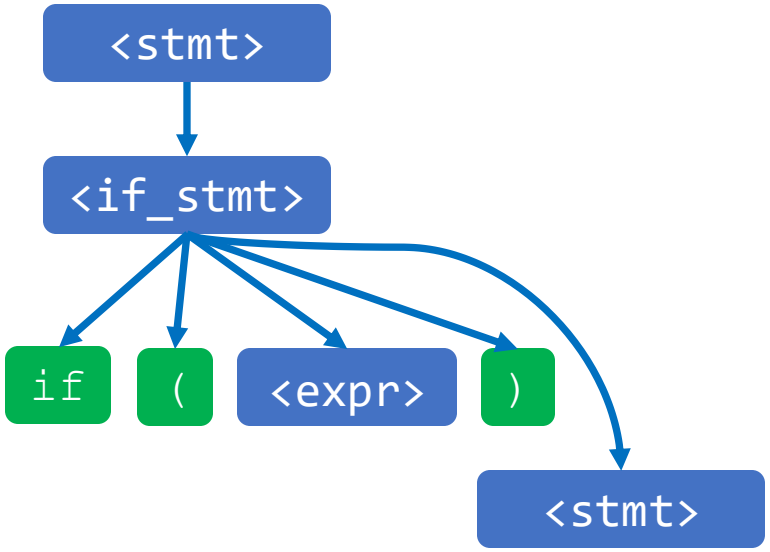
```
int main() {
 int a = 1;
 if (a) {
   printf("%d\n", a);
   printf("Hello ");
   printf("world!\n");
   printf("End\n");
 }
 return 0;
}
```

```
<func_def>      ::= <type> <identifier> '(' ')' <compound_stmt>
<compound_stmt> ::= '{' <stmt_star> '}'
<stmt_star>     ::= <stmt>*      // A list of zero or more statements
<stmt>          ::= <expr_stmt>
                  | <decl_stmt>
                  | <if_stmt>
                  | <compound_stmt>
<if_stmt>  ::= 'if' '(' <expr> ')' <stmt>
```

Perses: fully syntax-guided

- Avoid generating **syntactically invalid** variants
- Enable more program transformations
  - e.g., replace with a **syntax-compatible** descendant

```c
int main() {
 int a = 1;
 if (a) {
   printf("%d\n", a);
   printf("Hello ");
   printf("world!\n");
   printf("End\n");
 }
 return 0;
}
```

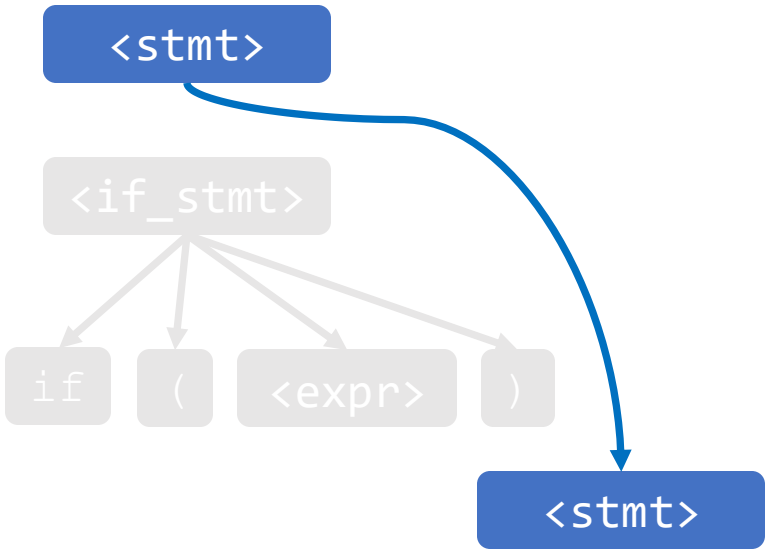



```
<func_def>      ::= <type> <identifier> '(' ')' <compound_stmt>
<compound_stmt> ::= '{' <stmt_star> '}'
<stmt_star>     ::= <stmt>*      // A list of zero or more statements
<stmt>          ::= <expr_stmt>
                  | <decl_stmt>
                  | <if_stmt>
                  | <compound_stmt>
<if_stmt>  ::= 'if' '(' <expr> ')' <stmt>
```

Perses: fully syntax-guided

- Avoid generating **syntactically invalid** variants
- Enable more program transformations
  - e.g., replace with a **syntax-compatible** descendant

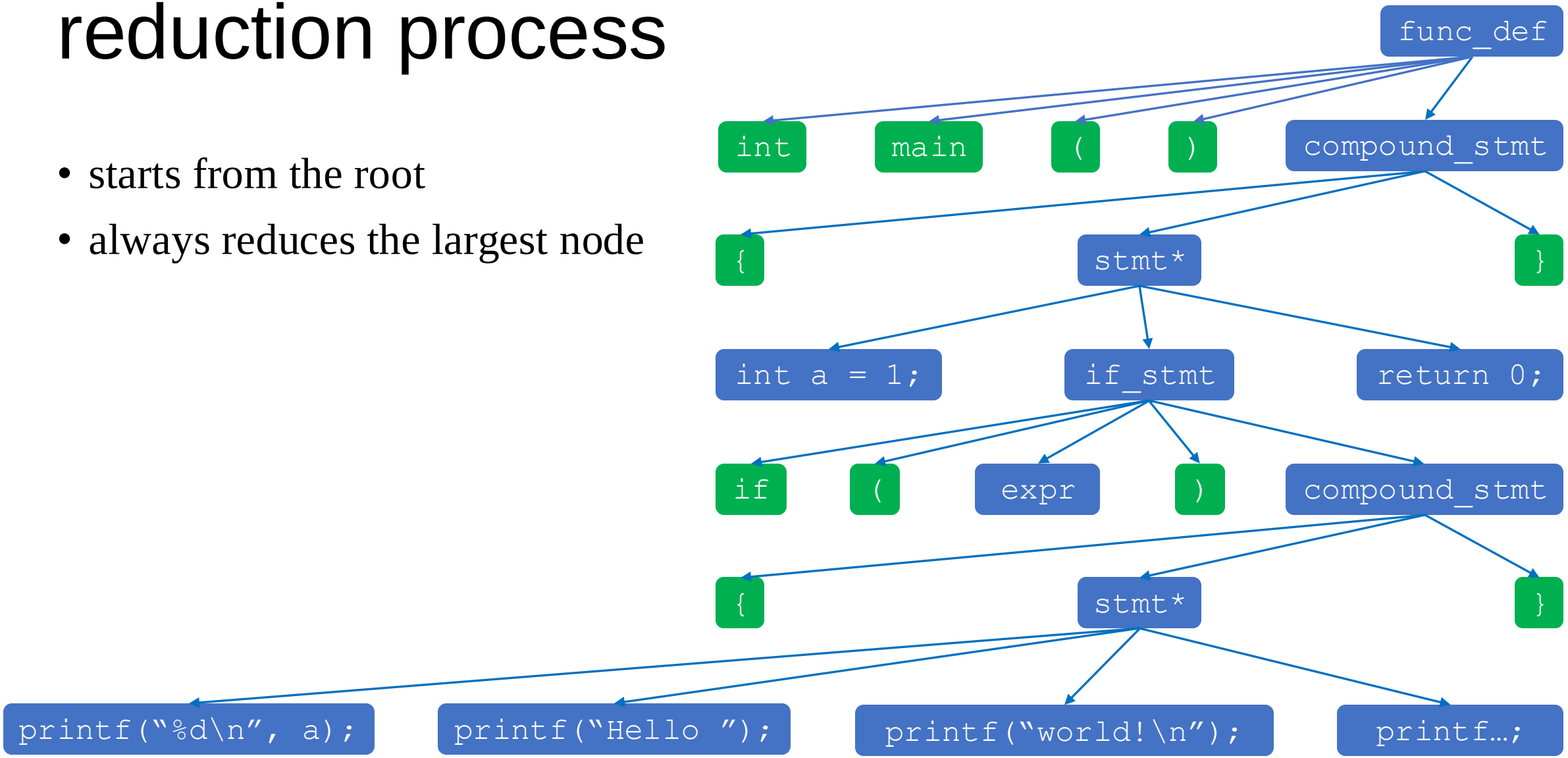
```
int main() {
 int a = 1;
 if (a) {
   printf("%d\n", a);
   printf("Hello ");
   printf("world!\n");
   printf("End\n");
 }
 return 0;
}
```




```
<func_def>      ::= <type> <identifier> '(' ')' <compound_stmt>
<compound_stmt> ::= '{' <stmt_star> '}'
<stmt_star>     ::= <stmt>*      // A list of zero or more statements
<stmt>          ::= <expr_stmt>
                  | <decl_stmt>
                  | <if_stmt>
                  | <compound_stmt>
<if_stmt>  ::= 'if' '(' <expr> ')' <stmt>
```

reduction process

- starts from the root
- always reduces the largest node

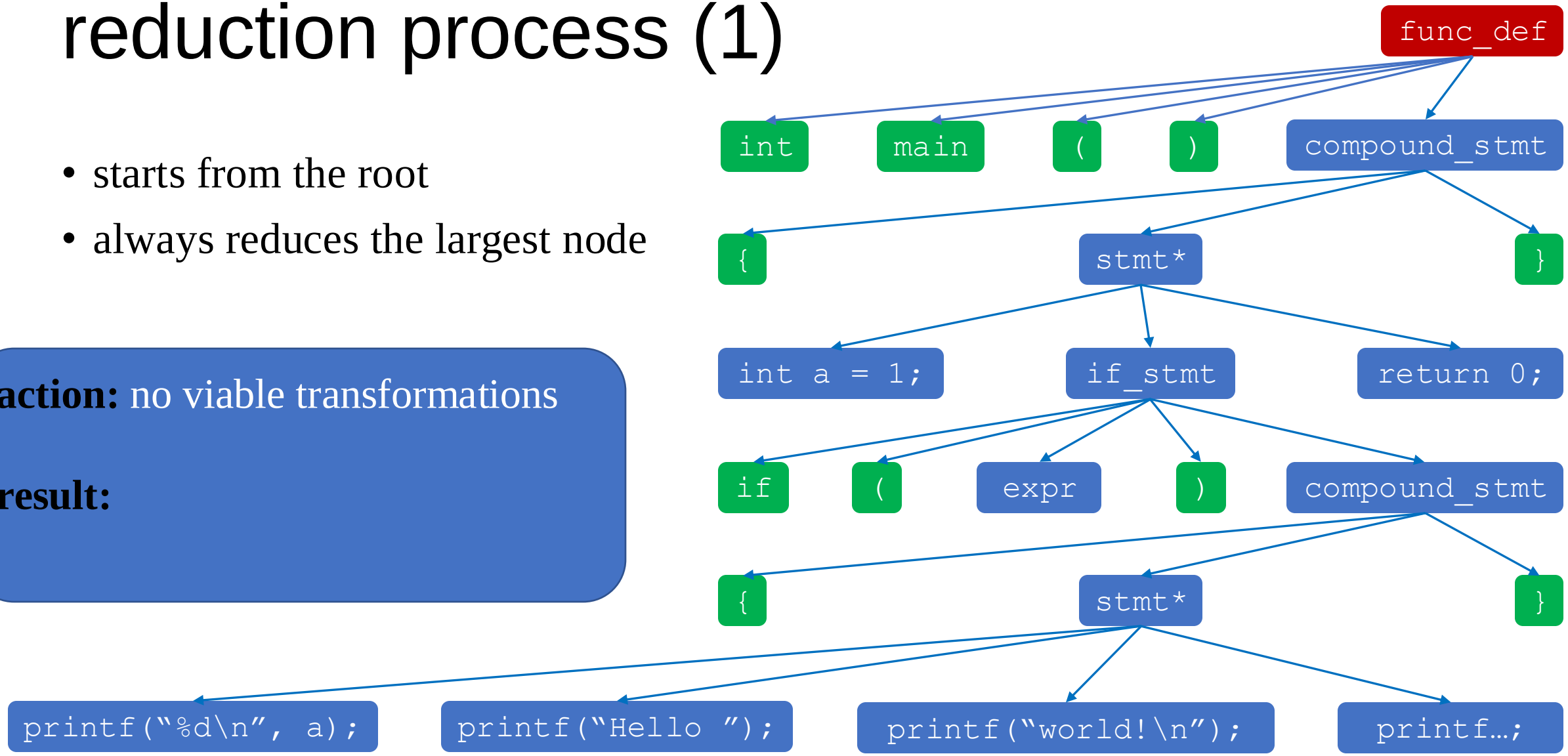


reduction process (1)

- starts from the root
- always reduces the largest node

action: no viable transformations

result:

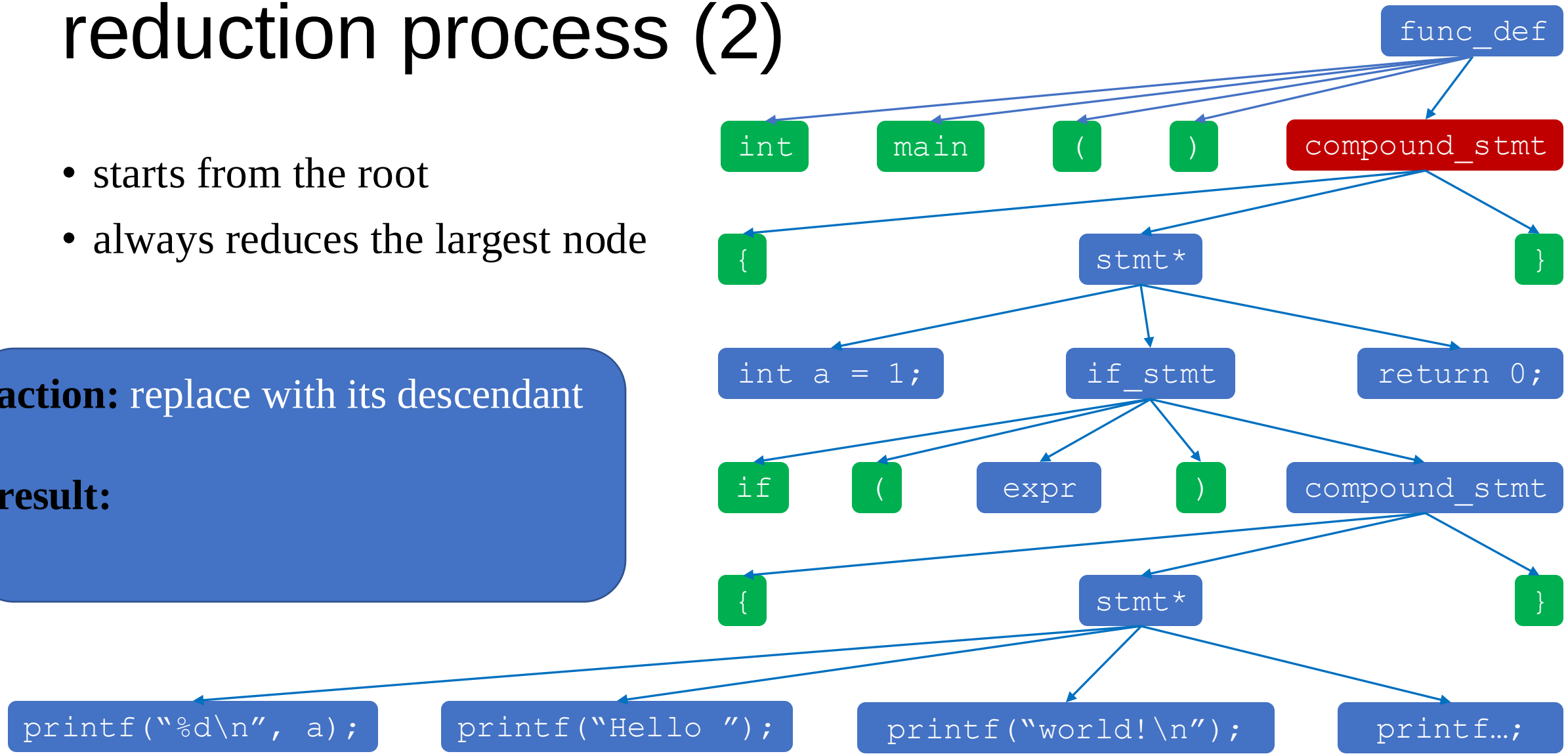


reduction process (2)

- starts from the root
- always reduces the largest node

action: replace with its descendant

result:

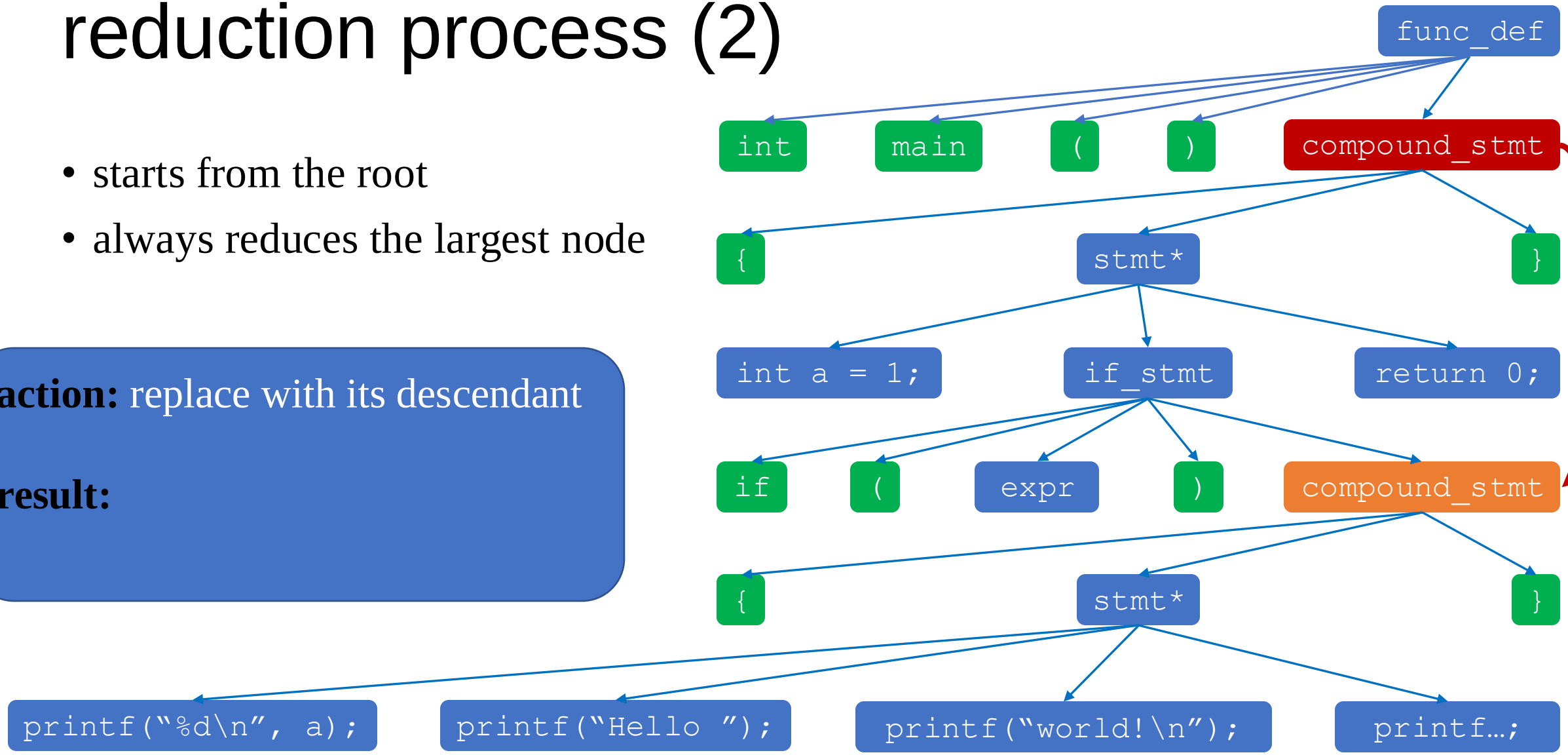


reduction process (2)

- starts from the root
- always reduces the largest node

action: replace with its descendant

result:

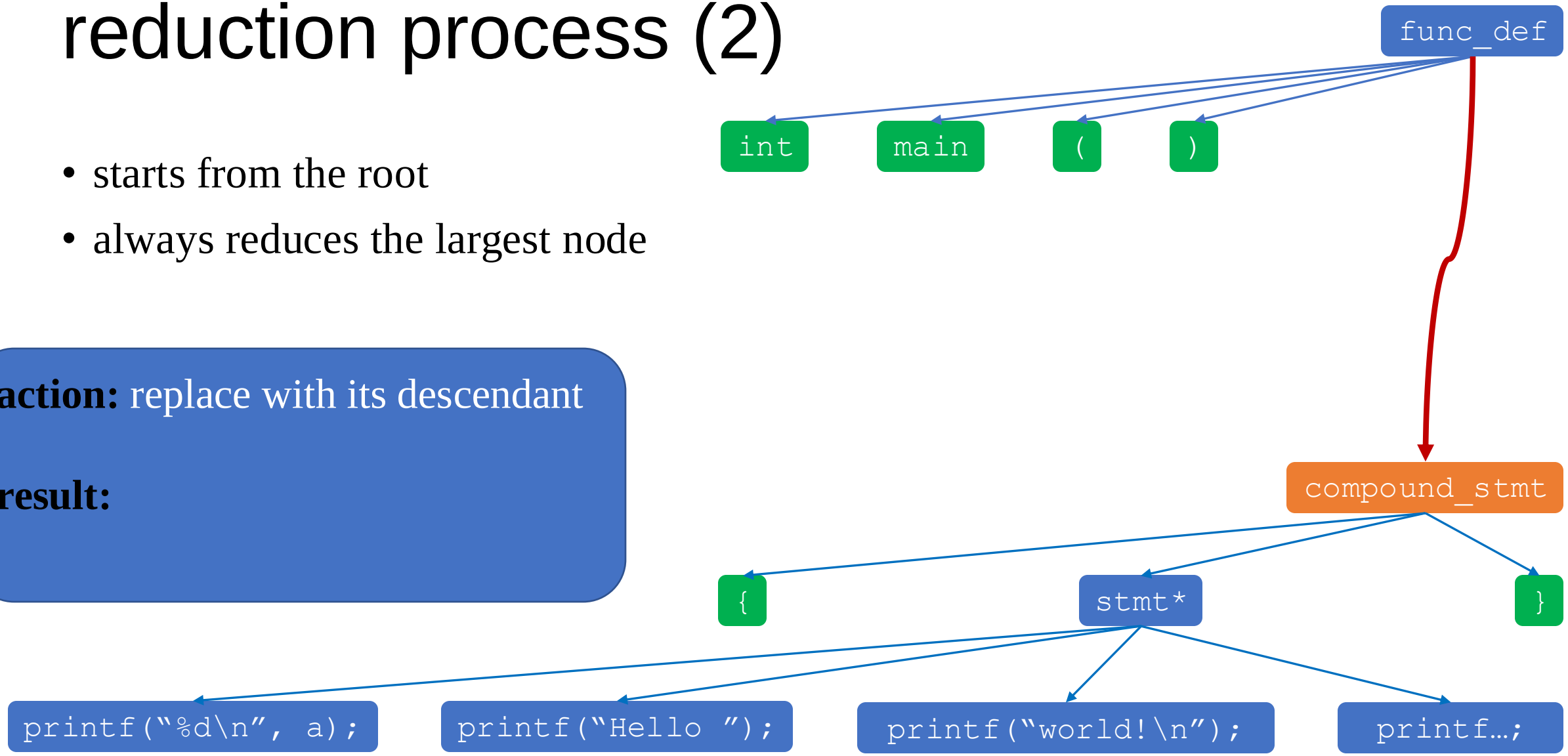


reduction process (2)

- starts from the root
- always reduces the largest node

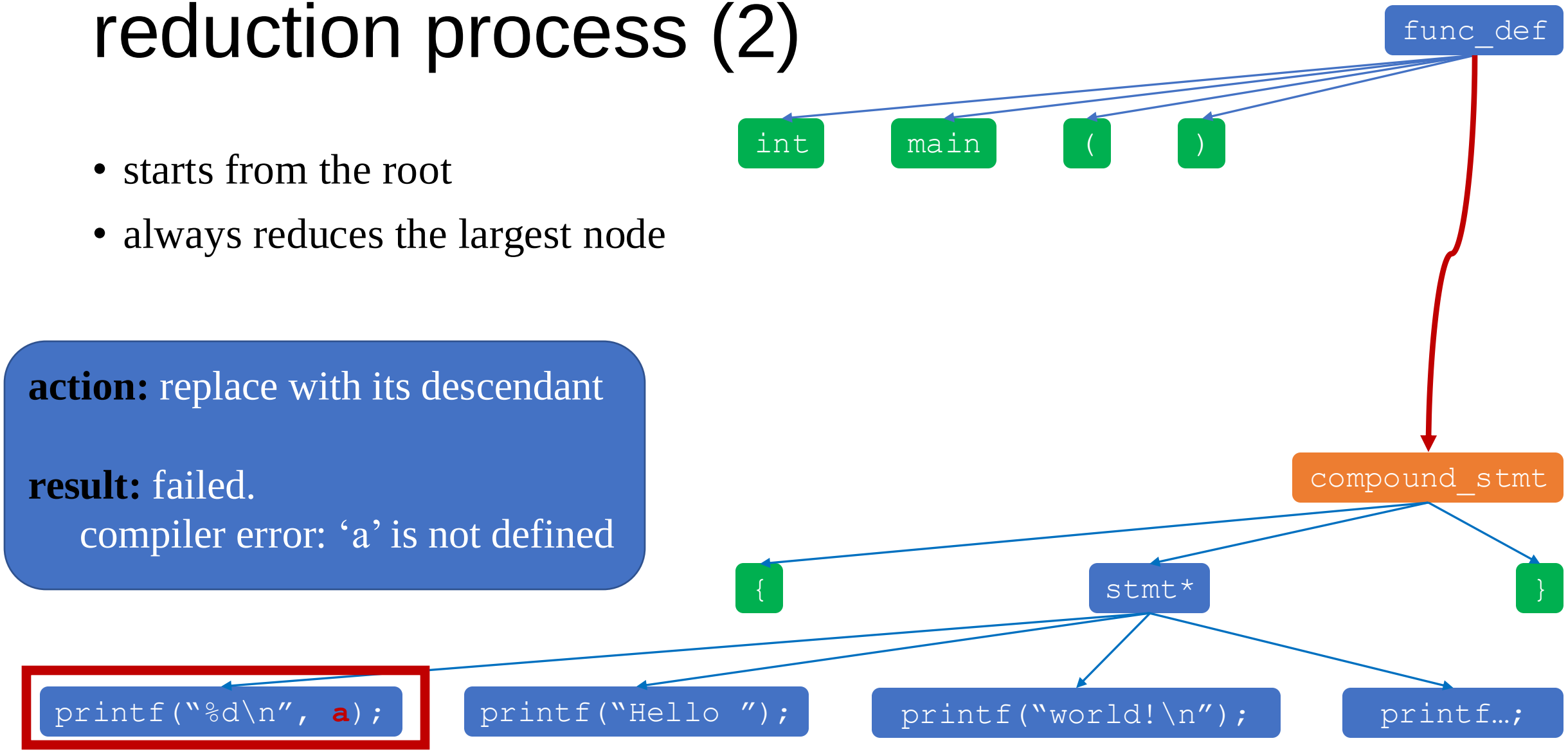
action: replace with its descendant

result:



reduction process (2)

- starts from the root
- always reduces the largest node



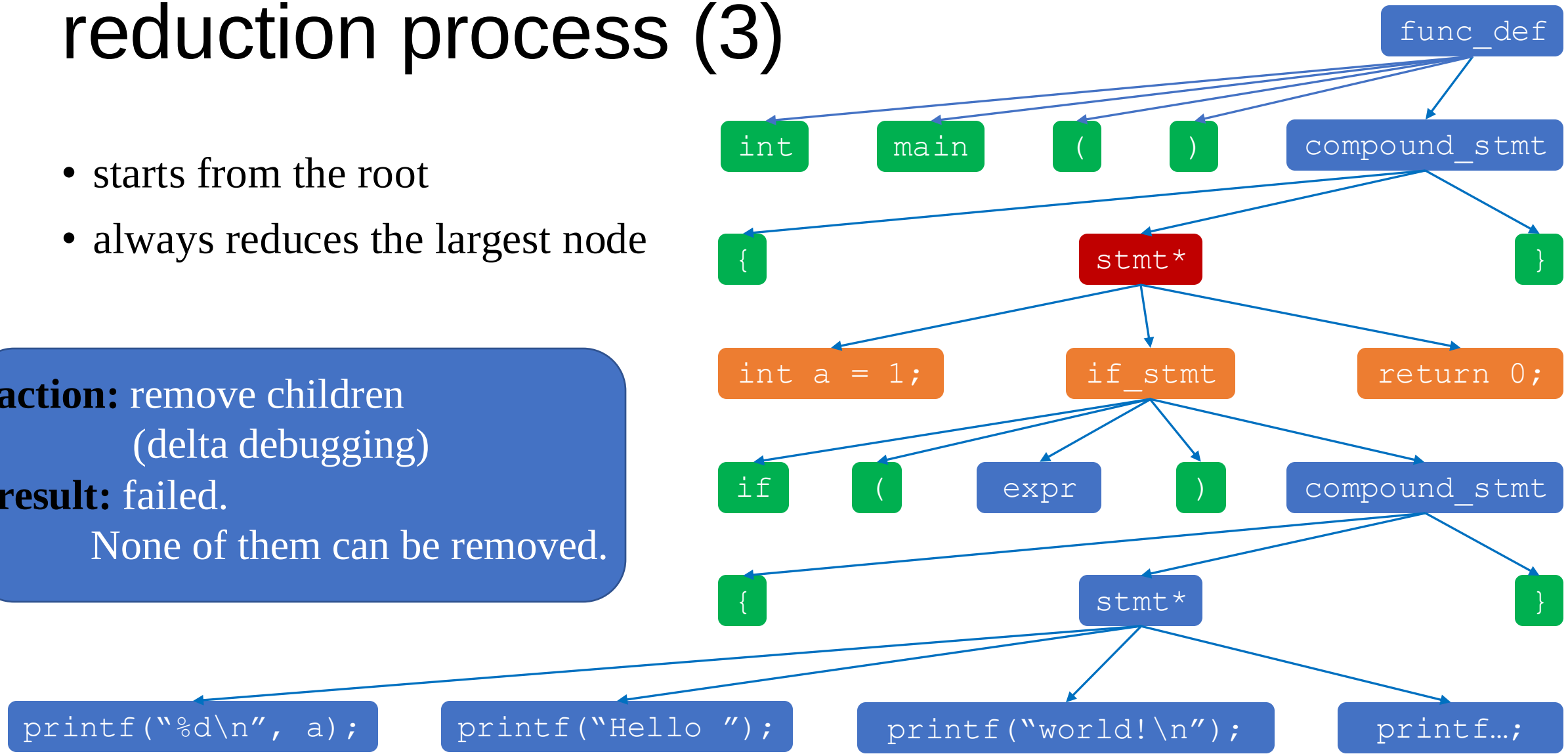
action: replace with its descendant

result: failed.
compiler error: 'a' is not defined

reduction process (3)

- starts from the root
- always reduces the largest node

action: remove children
(delta debugging)
result: failed.
None of them can be removed.

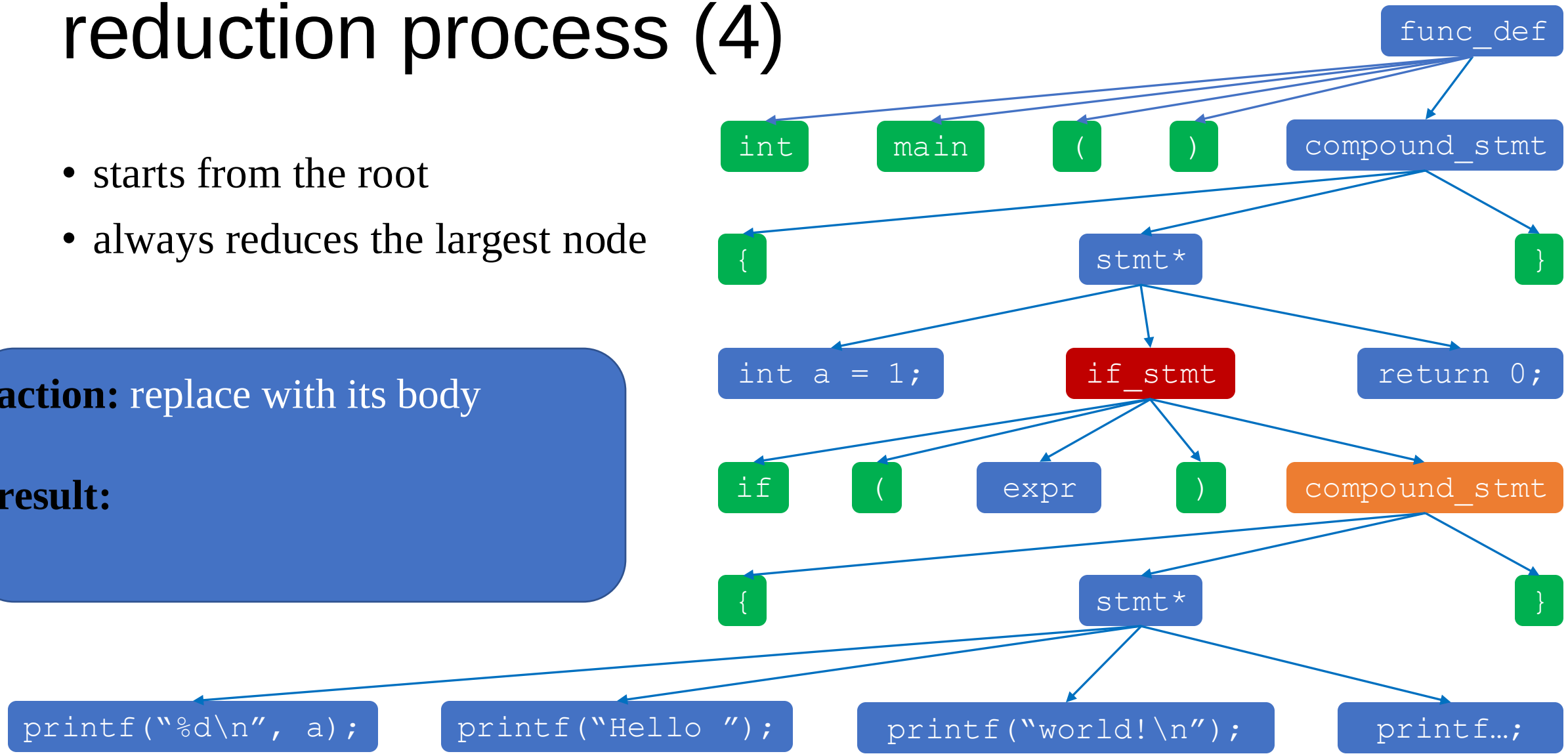


reduction process (4)

- starts from the root
- always reduces the largest node

action: replace with its body

result:

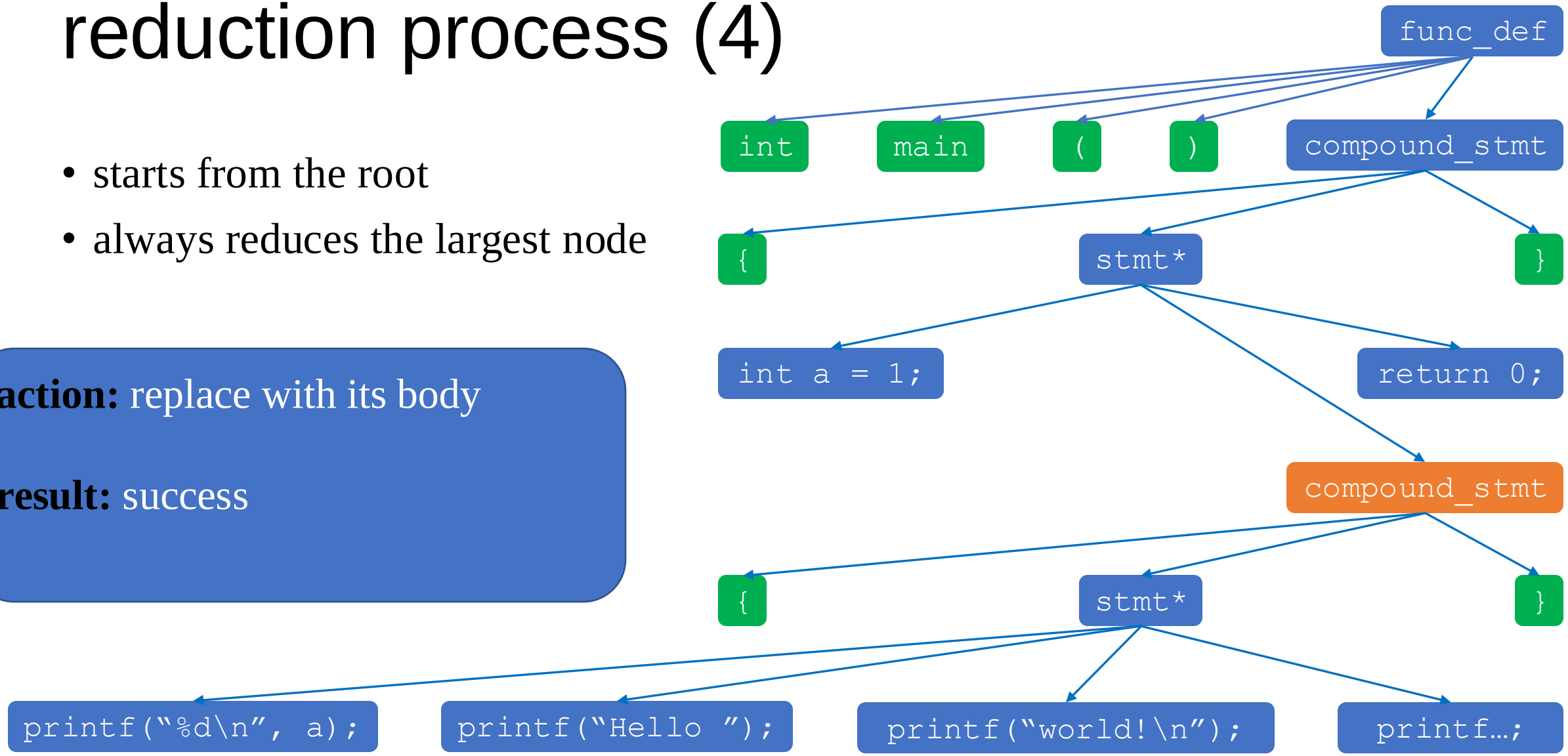


reduction process (4)

- starts from the root
- always reduces the largest node

action: replace with its body

result: success

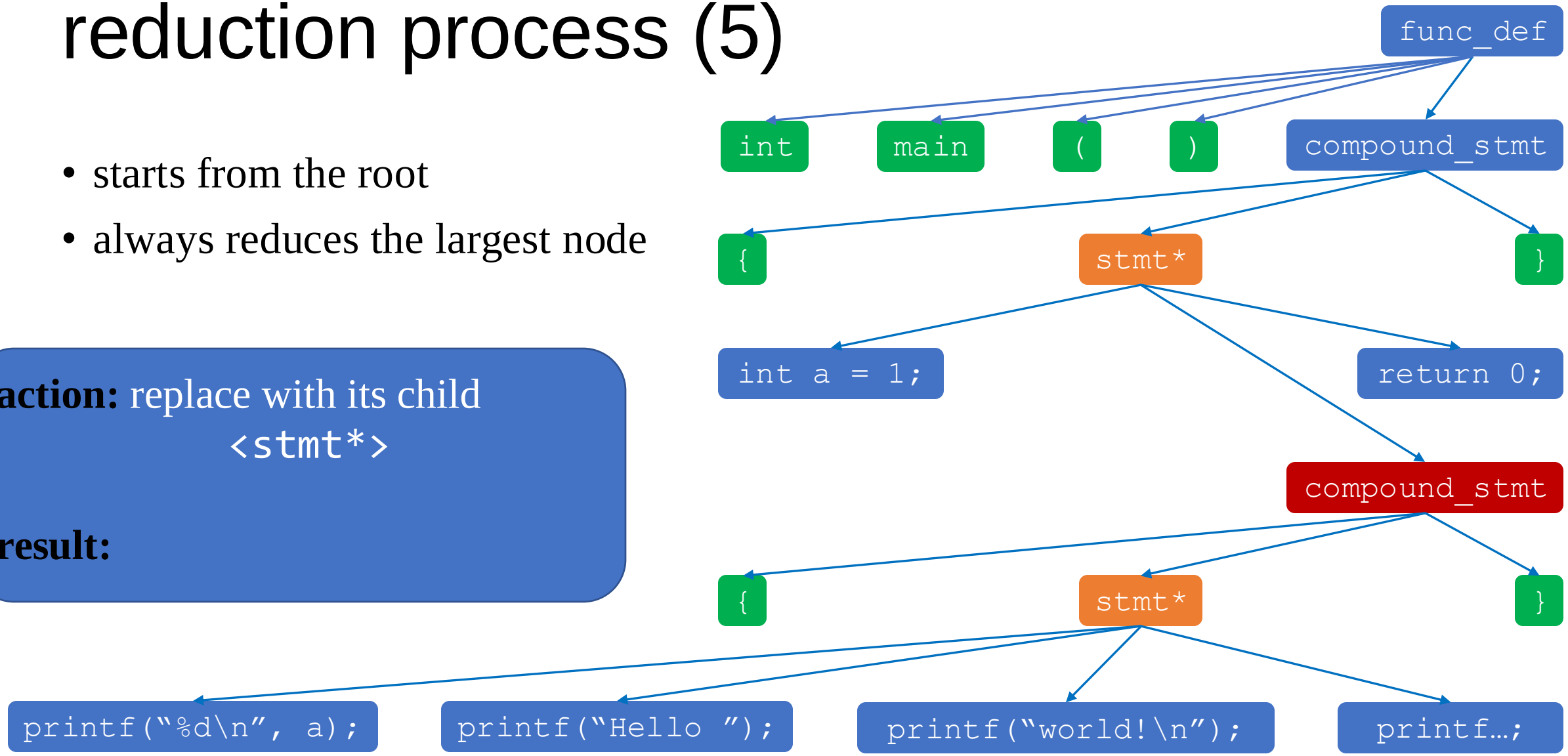


reduction process (5)

- starts from the root
- always reduces the largest node

action: replace with its child
<stmt*>

result:

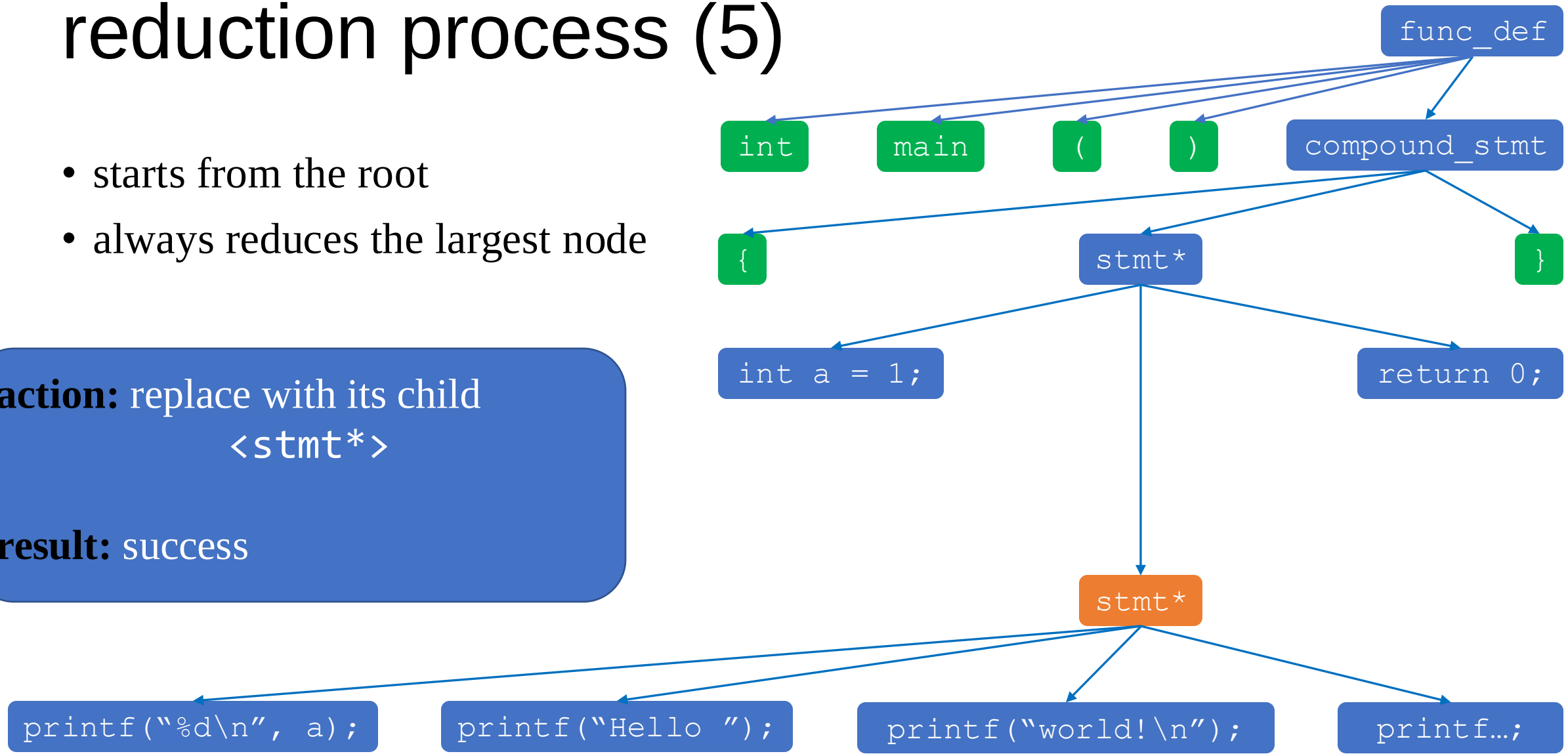


reduction process (5)

- starts from the root
- always reduces the largest node

action: replace with its child
<stmt*>

result: success

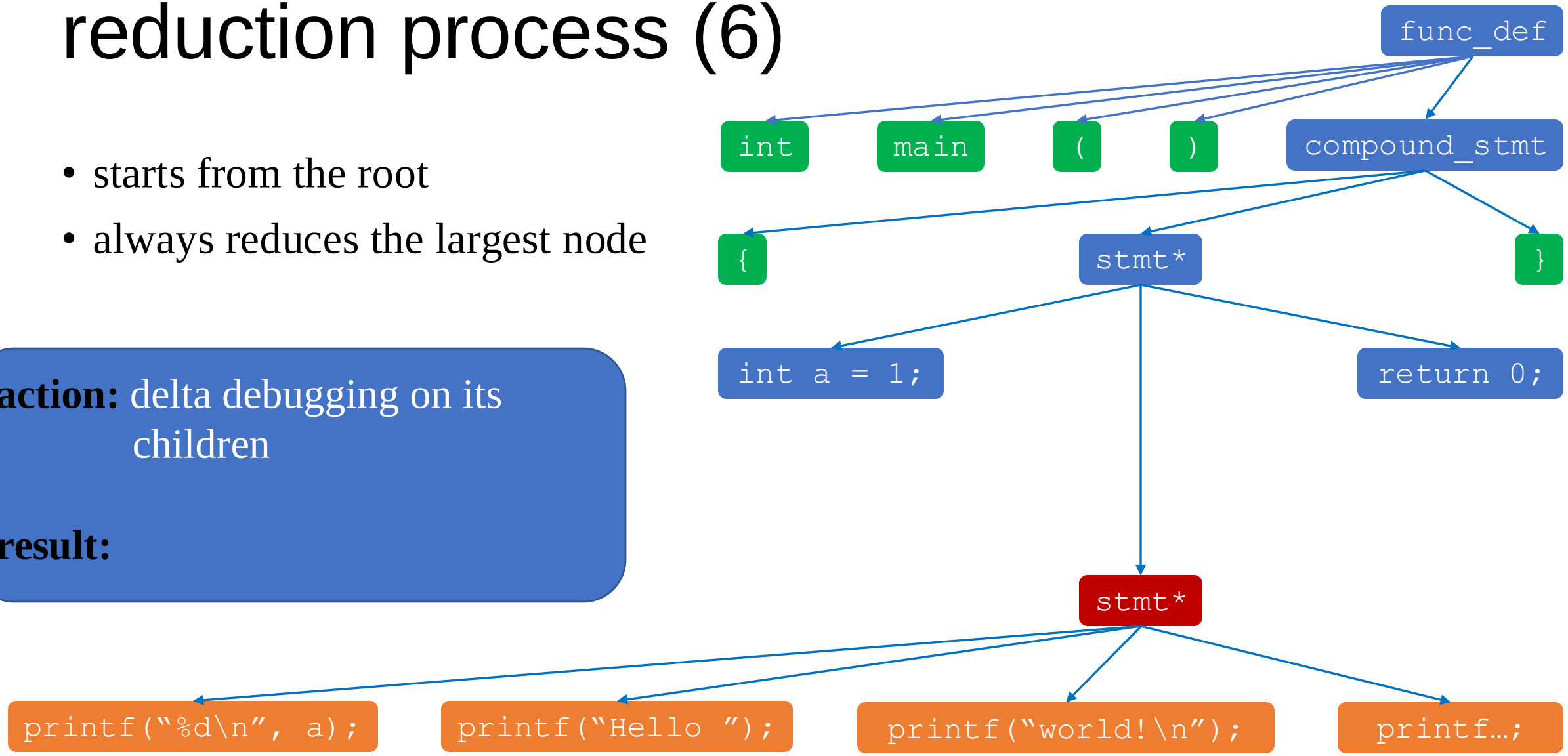


reduction process (6)

- starts from the root
- always reduces the largest node

action: delta debugging on its children

result:

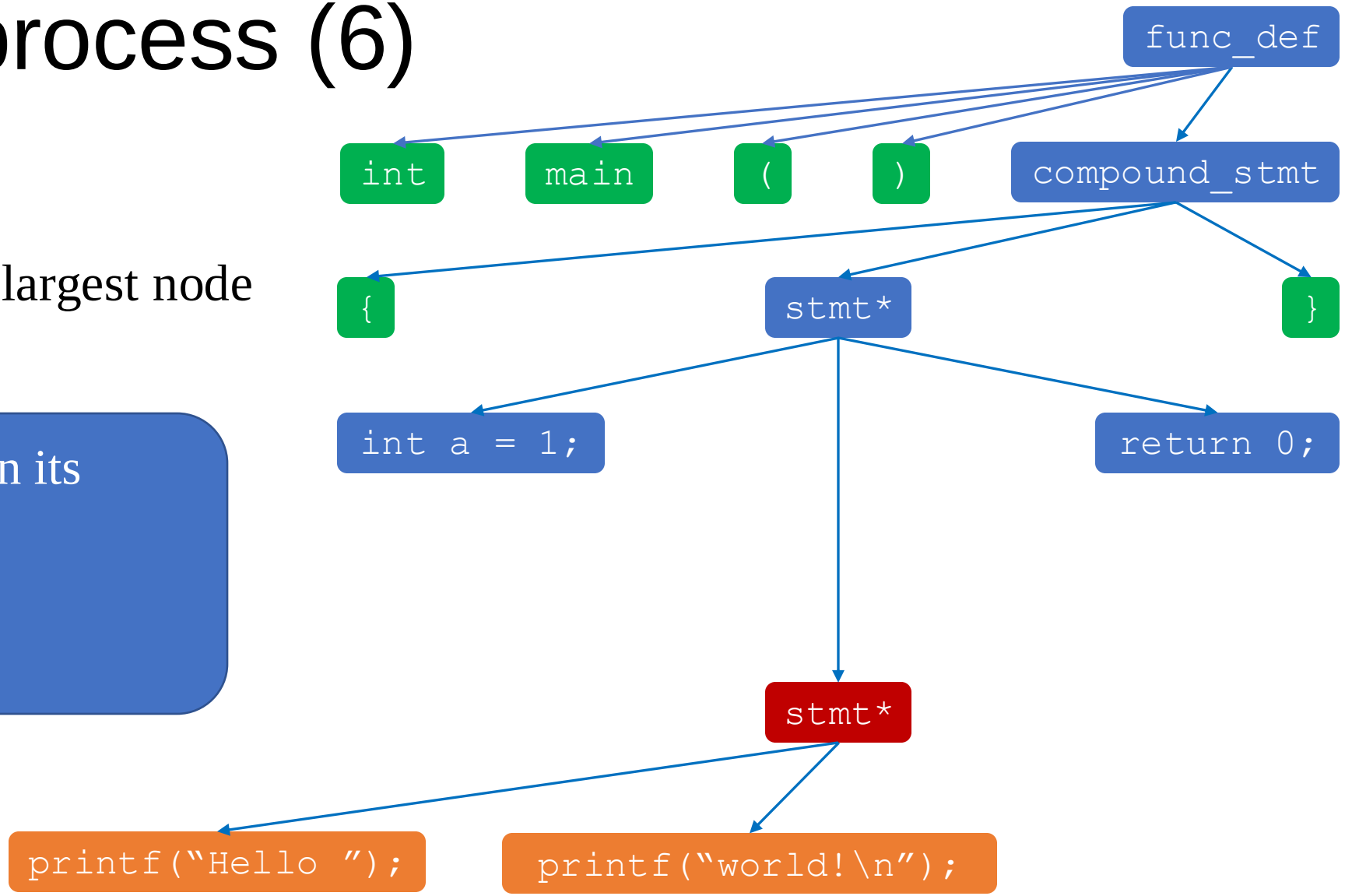


reduction process (6)

- starts from the root
- always reduces the largest node

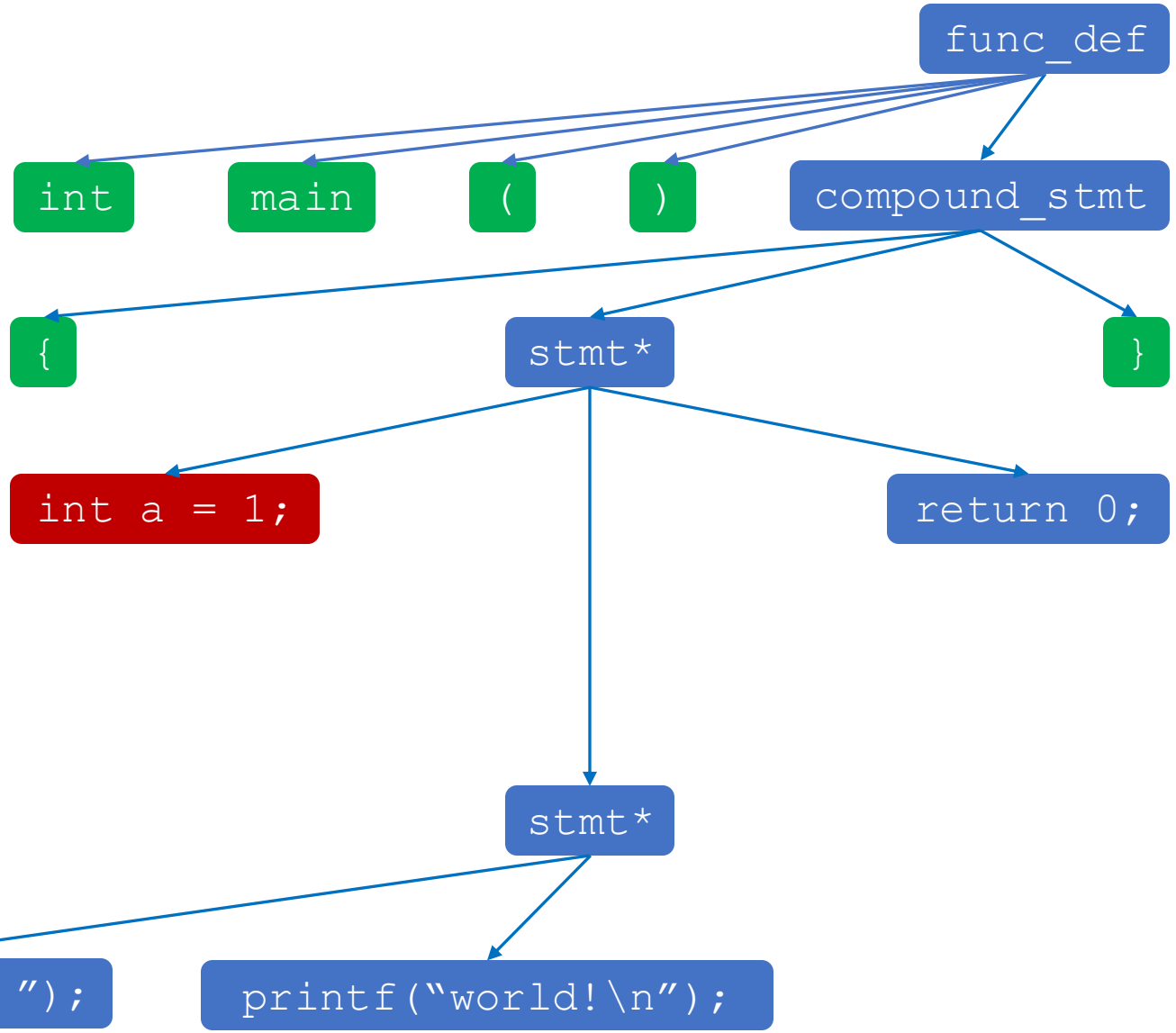
action: delta debugging on its children

result: success



reduction process

- starts from the root
- always reduces the largest node



later

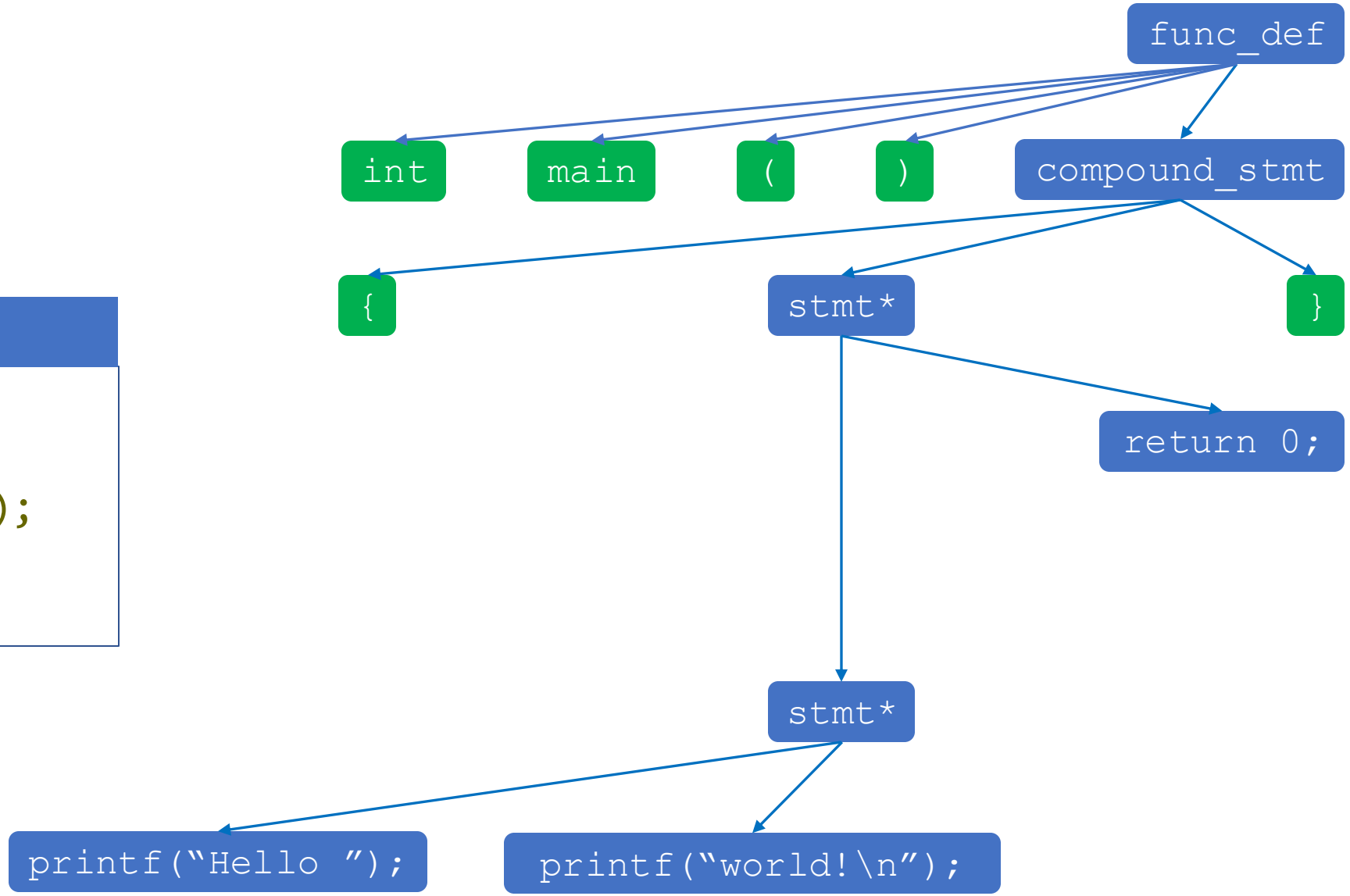
```
int a = 1;
```

will be deleted.

final result

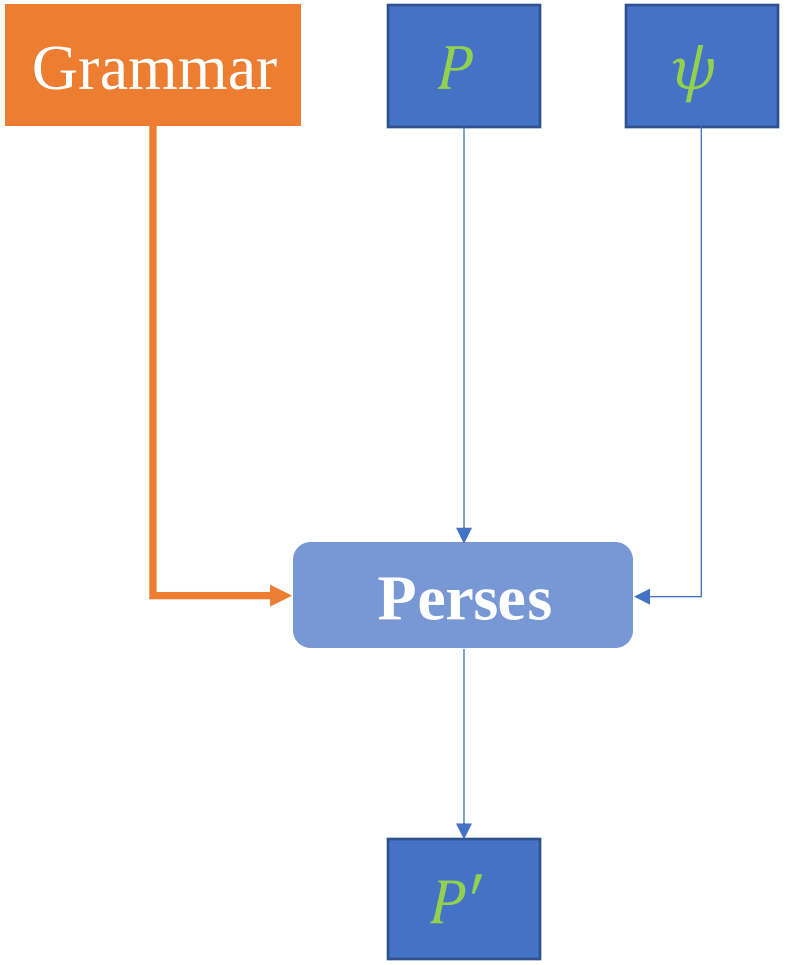
ideal result

```
int main() {
 printf("Hello ");
 printf("world!\n");
 return 0;
}
```

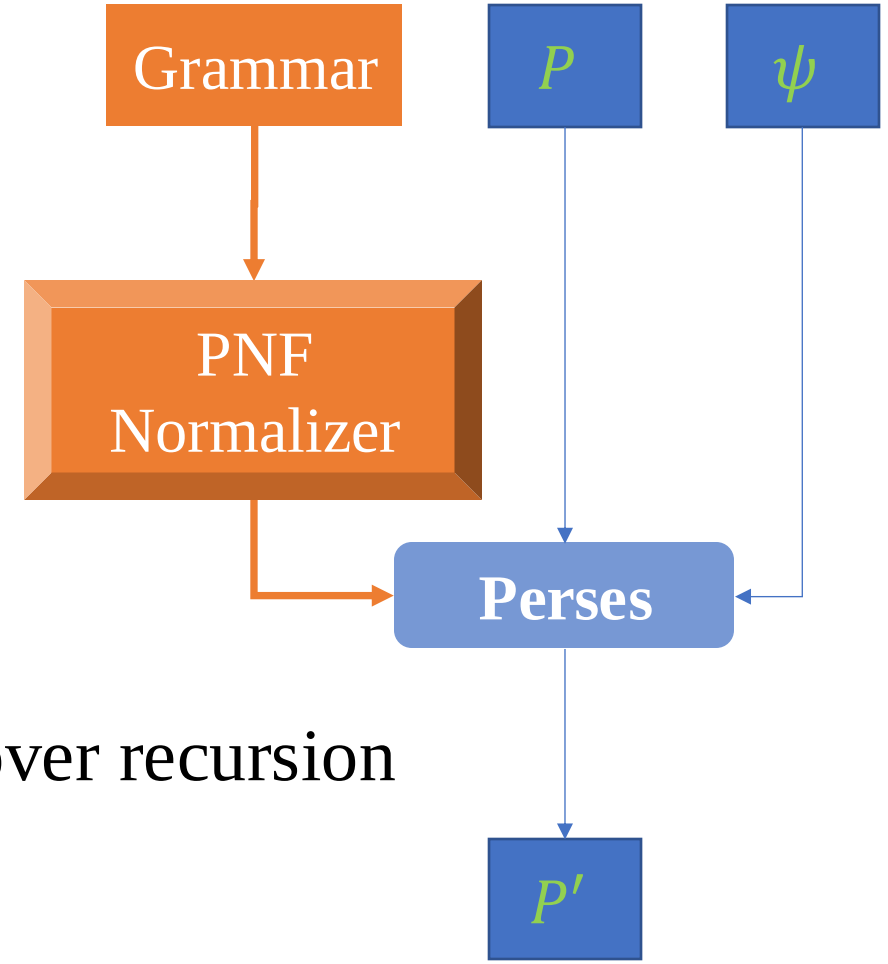


Perses normal form

- Perses relies on quantifiers
  - `*`, `+`, `?`
- A grammar can be written recursively
  - `<stmt_list> ::= <stmt> <stmt_list>`
    `| ϵ`



Perses normal form



- Perses relies on quantifiers
  - `*`, `+`, `?`
- A grammar can be written recursively
  - `<stmt_list> ::= <stmt> <stmt_list>`
    `| ϵ`
- Perses Normal Form: in favor of `*`, `+`, `?` over recursion
  - `<stmt_list> ::= <stmt>*`
- Propose an automatic conversion algorithm (section 4.1 in icse'18)

evaluation

- benchmarks
  - 20 C programs: each triggered a bug in a stable compiler release
    - GCC, Clang
  - size: 6K ~ 212K tokens
- comparison
  - DD: delta debugging
  - HDD: hierarchical delta debugging
  - C-Reduce:
    - specialized reducer for C/C++
    - based on Clang front-end

evaluation

- effectiveness
  - number of tokens in the reduced result
  - number of characters in the reduced result (not meaningful for programs)
- efficiency
  - time (wall time of the reduction process)
  - number of queries/tests/variants
  - reduction speed (#tokens deleted per second)

evaluation – effectiveness

size of reduced programs (mean)					
	original	DD	HDD	C-Reduce	**Perses**
size (#)	94,486	4,573	575	90	**257**
ratio (Perses/other)		6%	45%	285%	**100%**

evaluation – efficiency (1)

A property check ψ takes constant time

→ then number of ψ checks becomes a good measure of efficiency.

→ more checks mean more reduction time

number of property checks				
	DD	HDD	C-Reduce	**Perses**
#	78,305	16,886	27,359	**5,095**
ratio (Perses/other)	7%	30%	19%	**100%**

evaluation – efficiency (2)

reduction time				
	DD	HDD	C-Reduce	**Perses**
seconds	9,710	4,658	3,675	**2,198**
ratio (Perses/other)	23%	47%	60%	**100%**

reduction speed				
	DD	HDD	C-Reduce	**Perses**
tokens/seconds	20	37	33	**63**
ratio (Perses/other)	3.2x	1.7x	1.9x	**100%**

conclusion

- general, syntax-guided program reduction
- outperform DD, HDD, complement C-Reduce on C/C++
  - smaller reduction results in less time
- language-agnostic and fully automated support for any? new language