검증된 검산기로
LLVM의 최적화 단계 검산하기

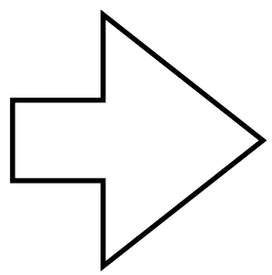
최준원, 조성근, 강지훈, 허충길, 이광근
서울대학교

2014. 7. 29.
ROSAEC workshop

ROSAEC center
Research On Software Analysis for Error-free Computing
소프트웨어 무결점 연구센터 NRF ERC

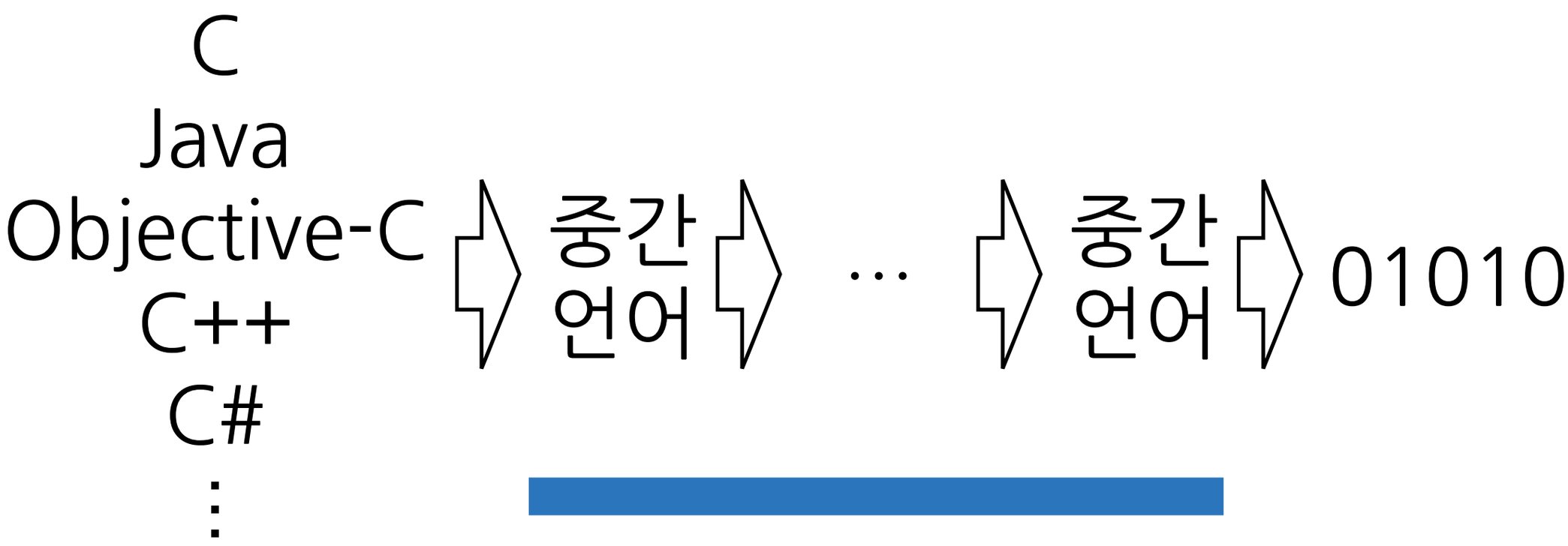
컴파일러가 하는 일

C
Java
Objective-C
C++
C#
⋮



01010

컴파일러가 하는 일



컴파일러가 올바른가?

C
Java
Objective-C
C++
C#
⋮

? ? ? ?

⇨ 중간 언어 ⇨ … ⇨ 중간 언어 ⇨ 01010

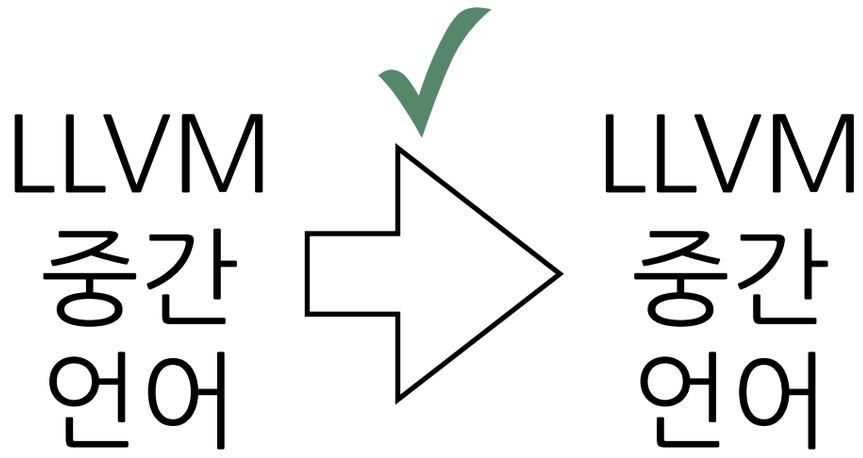
버그박멸단의 목표 (llvm -O1)

C
Java
Objective-C
C++
C#
⋮

✓ ✓ ✓ ✓

⇨ 중간 언어 ⇨ … ⇨ 중간 언어 ⇨ 01010

이번 연구에선,



한 최적화 패스
instruction combining

컴파일러의 올바름을 보이는 방법

- 검증(verified compilation)
  - 증명보조기(proof assistant)로 컴파일러 구현+증명

$$\forall s, t.\ \ C(s) = \mathrm{Some}\ t \Rightarrow [\![s]\!] = [\![t]\!]$$

  - 다른 언어로 추출하여 컴파일에 사용
- 검산(verified validation)
  - 증명보조기로 검산기 구현+증명

$$\forall s, t.\ \ V(s, t) = \mathrm{true} \Rightarrow [\![s]\!] = [\![t]\!]$$

  - 컴파일은 이미 있는 컴파일러 사용
  - 컴파일된 결과가 옳은지 확인하는 데에 검산기 사용

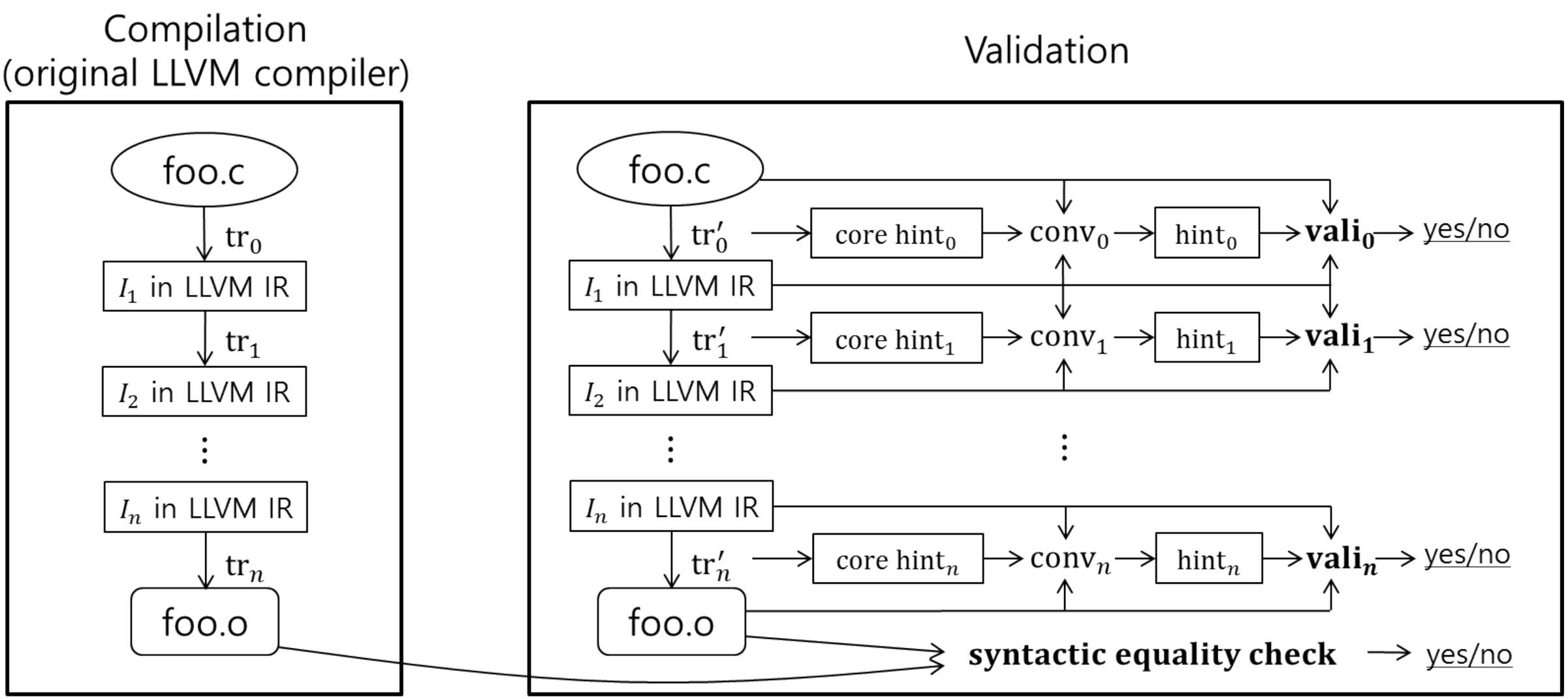
검산(verified validation)의 좋은점

- 이미 있는 컴파일러를 다시 구현할 필요 없음
- 컴파일러 제작팀과 검산기 제작팀의 구분이 가능
- 검산기가 더 간단하여 증명이 더 쉬움
- 컴파일러의 변경에 유연함

검산(verified validation)의 나쁜점

- 올바른 컴파일인데 검산에 실패할 수 있음
- 검산에 많은 시간이 필요할 수 있음

큰 그림



실험 결과

- 검산 가능한 꼬마최적화: 50개 (전체 401개) / 3주
- 추론규칙의 올바름 증명: 32개 (현재까지 63개) / 2주
- 검산기의 올바름 증명:
  핵심이 되는 시뮬레이션 관계 증명
  (Vellvm*에서 정의한 실행의미 사용)

* J. Zhao, S. Nagarakatte, M. M. Martin, and S. Zdancewic. Formal Verification of SSA-Based Optimizations for LLVM. PLDI'13.

꼬마최적화 종류

파일	설명	개수(검산/전체)
InstructionCombining	공통으로 사용되는 최적화	2 / 32
InstCombineAddSub	+, - 관련 최적화	22 / 46
InstCombineMulDivRem	*, /, % 관련 최적화	11 / 39
InstCombineAndOrXor	&, |, ^ 관련 최적화	4 / 55
InstCombineSelect	select 관련 최적화	3 / 13
InstCombineCompares	<, <=, >, >=, == 관련 최적화	3 / 31
InstCombineLoadStoreAlloca	힙 메모리 연산 관련 최적화	2 / 24
InstCombinePHI	phi 함수 관련 최적화	1 / 13
InstCombineCasts	cast 관련 최적화	1 / 35
InstCombineShifts	<<, >> 관련 최적화	1 / 14
InstCombineCalls	intrinsic 함수 관련 최적화	0 / 50
InstCombineSimplifyDemanded	요구되는 비트분석을 이용한 최적화	0 / 42
InstCombineVectorOps	벡터 연산 관련 최적화	0 / 7
전체		50 / 401

실험

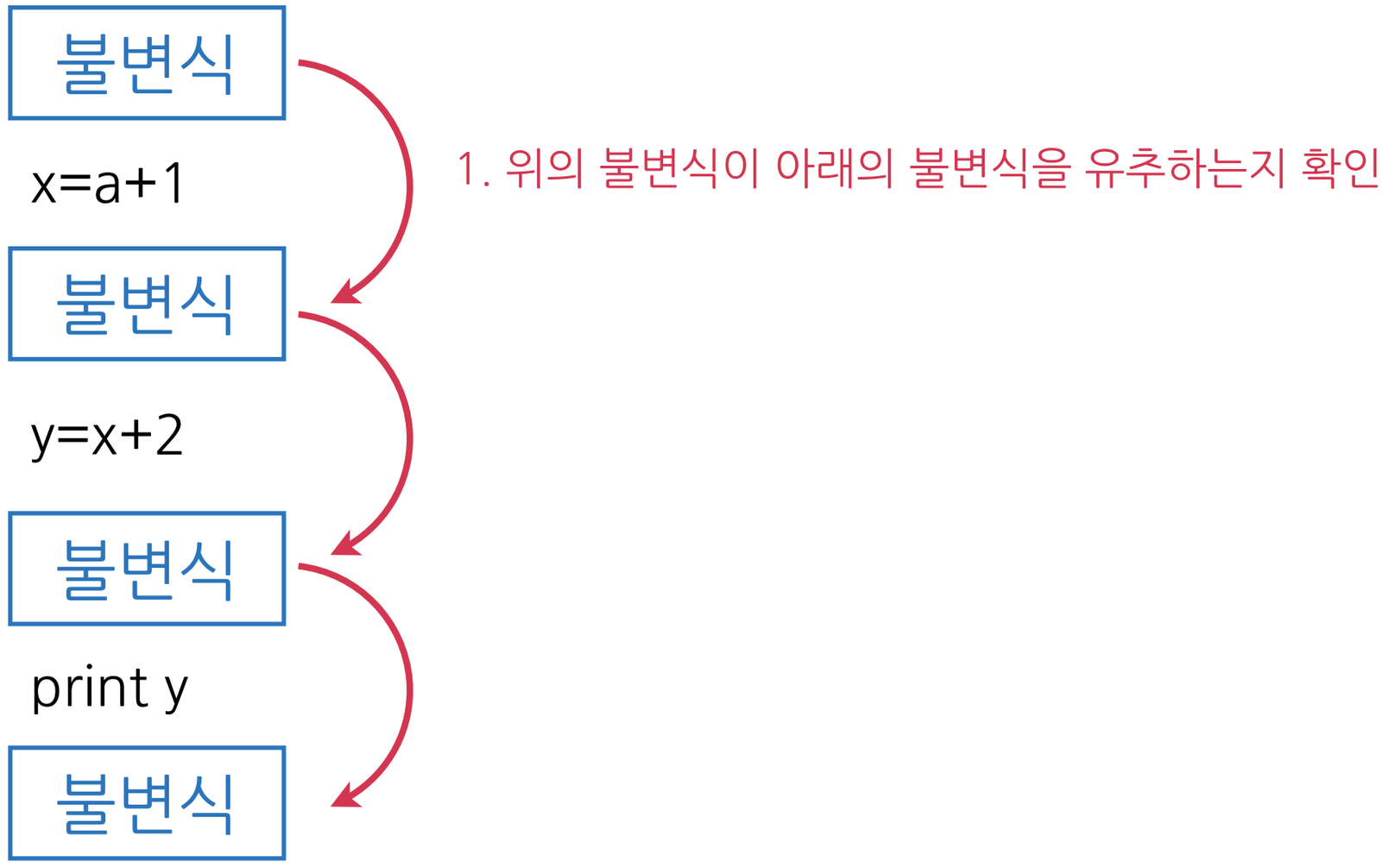
프로그램	LOC	파일수	컴파일시간	힌트생성	검산시간	생성된 힌트	파싱에러	검산 오버해드
bc-1.06	13K	19	2.1s	2.3s	23.7s	860	0	x12
tar-1.27	17K	67	5.8s	2.8s	31.4s	948	0	x6
less-451	24K	35	3.9s	2.4s	22.9s	1119	0	x6
make-4.0	28K	27	4.0s	7.5s	122.0s	2393	0	x32
a2ps-4.14	52K	83	16.3s	11.4s	299.0s	2926	0	x19
python-3.4.1	304K	221	79.7s	633.3s	12779.6s	33946	15	x168

오늘 발표

- 검산 방법
- 꼬마최적화 예
  - add assoc
  - store load
  - dead code elim
- 실험결과

검산 방법

- 불변식을 이용하여 프로그램 성질 확인



검산 방법

- 불변식을 이용하여 프로그램 성질 확인

불변식

x=a+1

불변식

y=x+2

불변식

print y

불변식

1. 위의 불변식이 아래의 불변식을 유추하는지 확인

2. 불변식이 원하는 성질을 만족시키는지 확인

검산 방법

- 두 프로그램을 동시에 실행시킬 때의 불변식

불변식

x=a+1 x=a+1

불변식

y=x+2 y=a+3

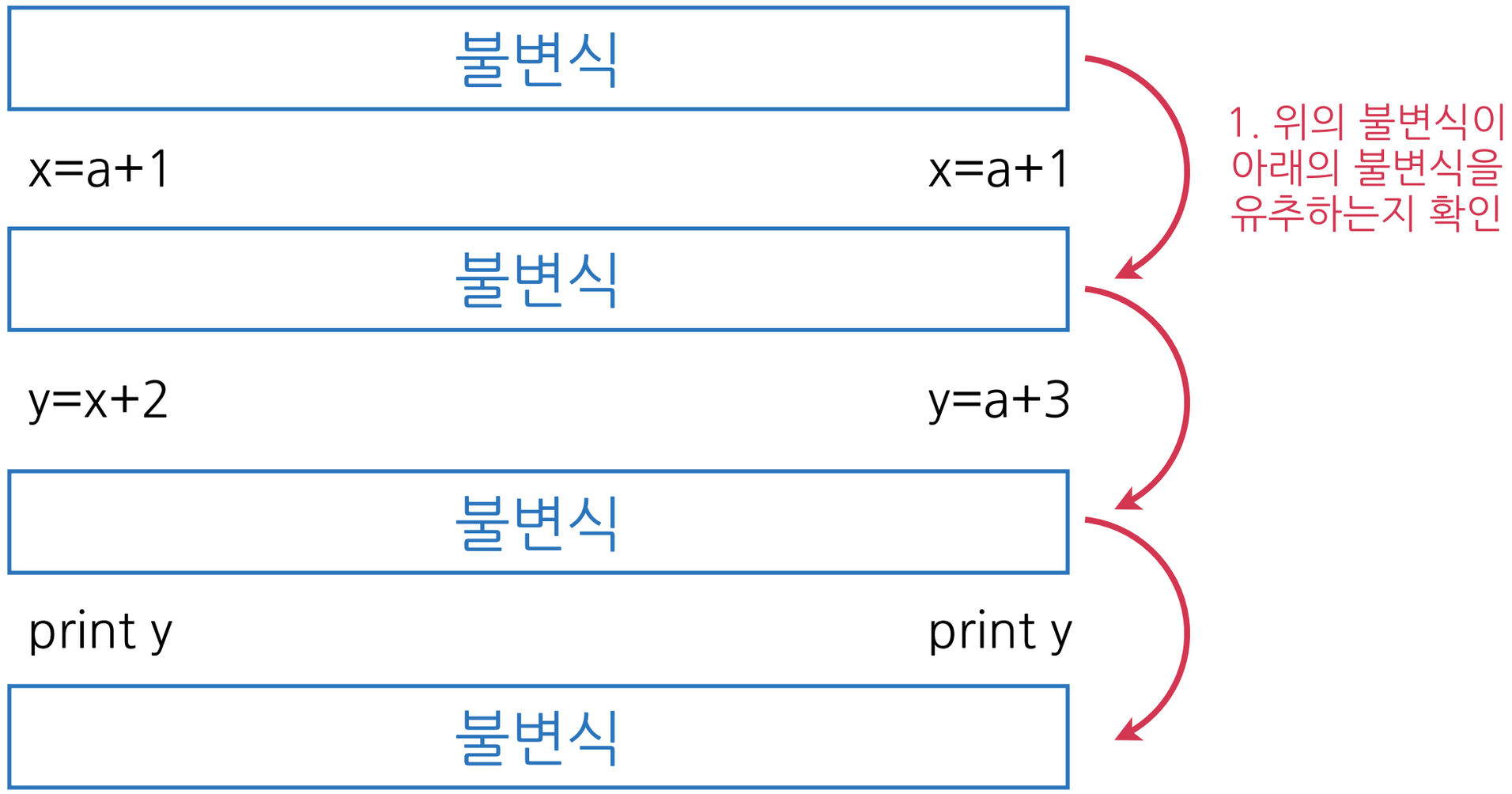
불변식

print y print y

불변식

검산 방법

- 두 프로그램을 동시에 실행시킬 때의 불변식



검산 방법

- 두 프로그램을 동시에 실행시킬 때의 불변식

불변식

x=a+1 | x=a+1

불변식

y=x+2 | y=a+3

불변식

print y | print y

불변식

1. 위의 불변식이 아래의 불변식을 유추하는지 확인

2. 불변식이 원하는 성질을 만족시키는지 확인

힌트(불변식) 구성

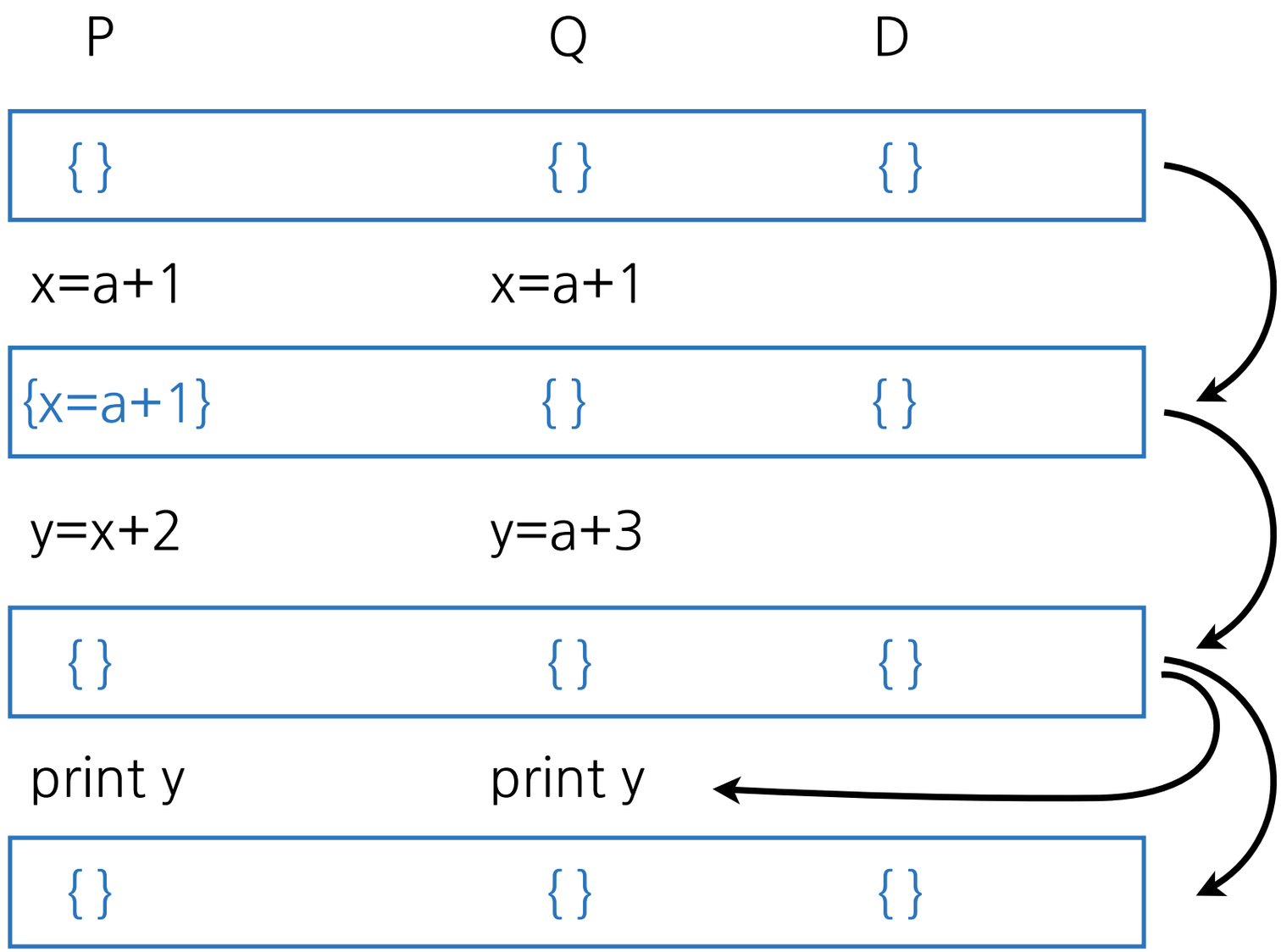
- P: 원본 프로그램에서 만족하는 불변식
- Q: 최적화된 프로그램에서 만족하는 불변식
- D(maydiff): 원본 프로그램과 최적화된 프로그램에서 값이 다를 수도 있는 변수의 집합

최적화 예 1: add assoc

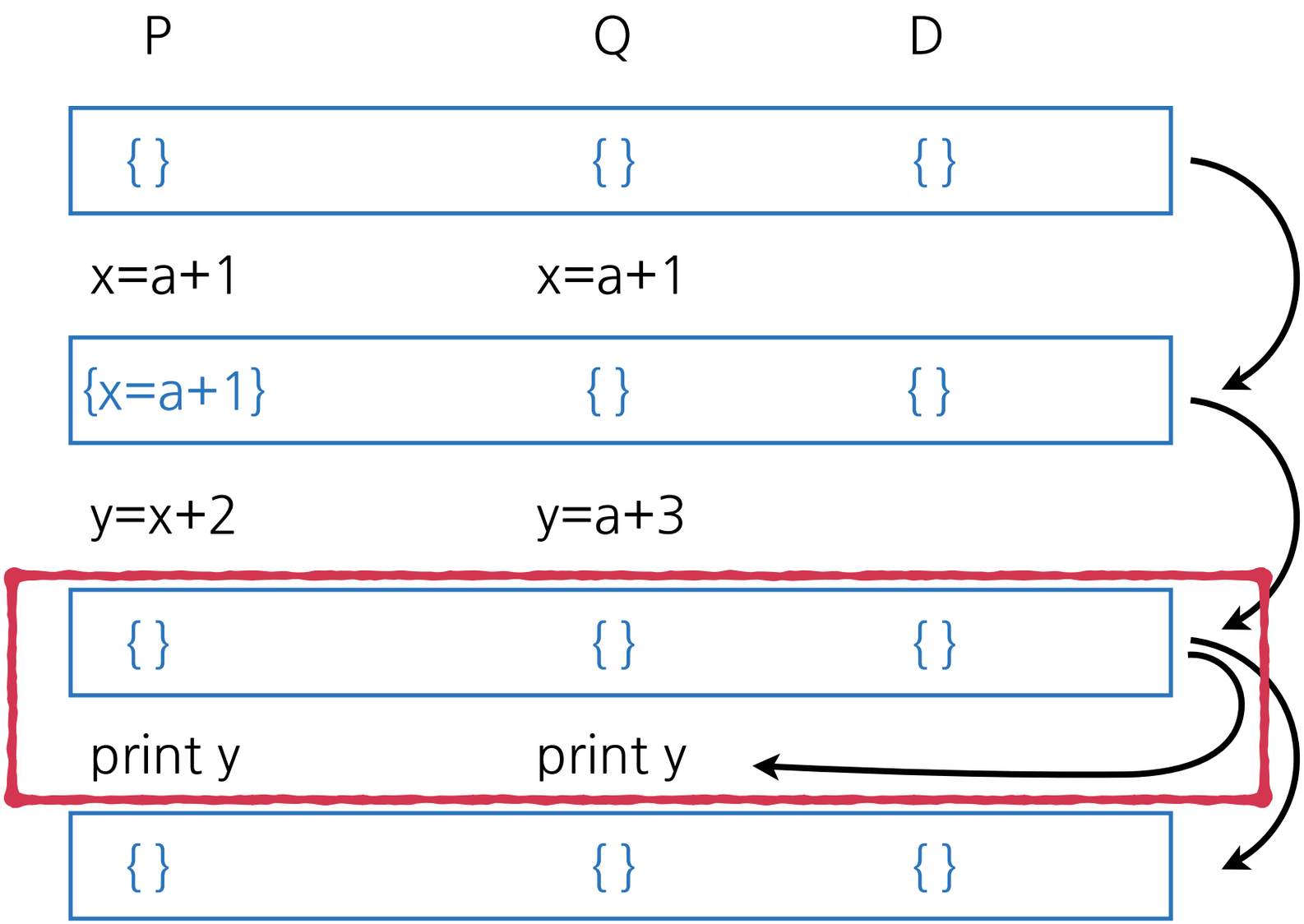
```
x=a+1
y=x+2
print y
```

```
x=a+1
y=a+3
print y
```

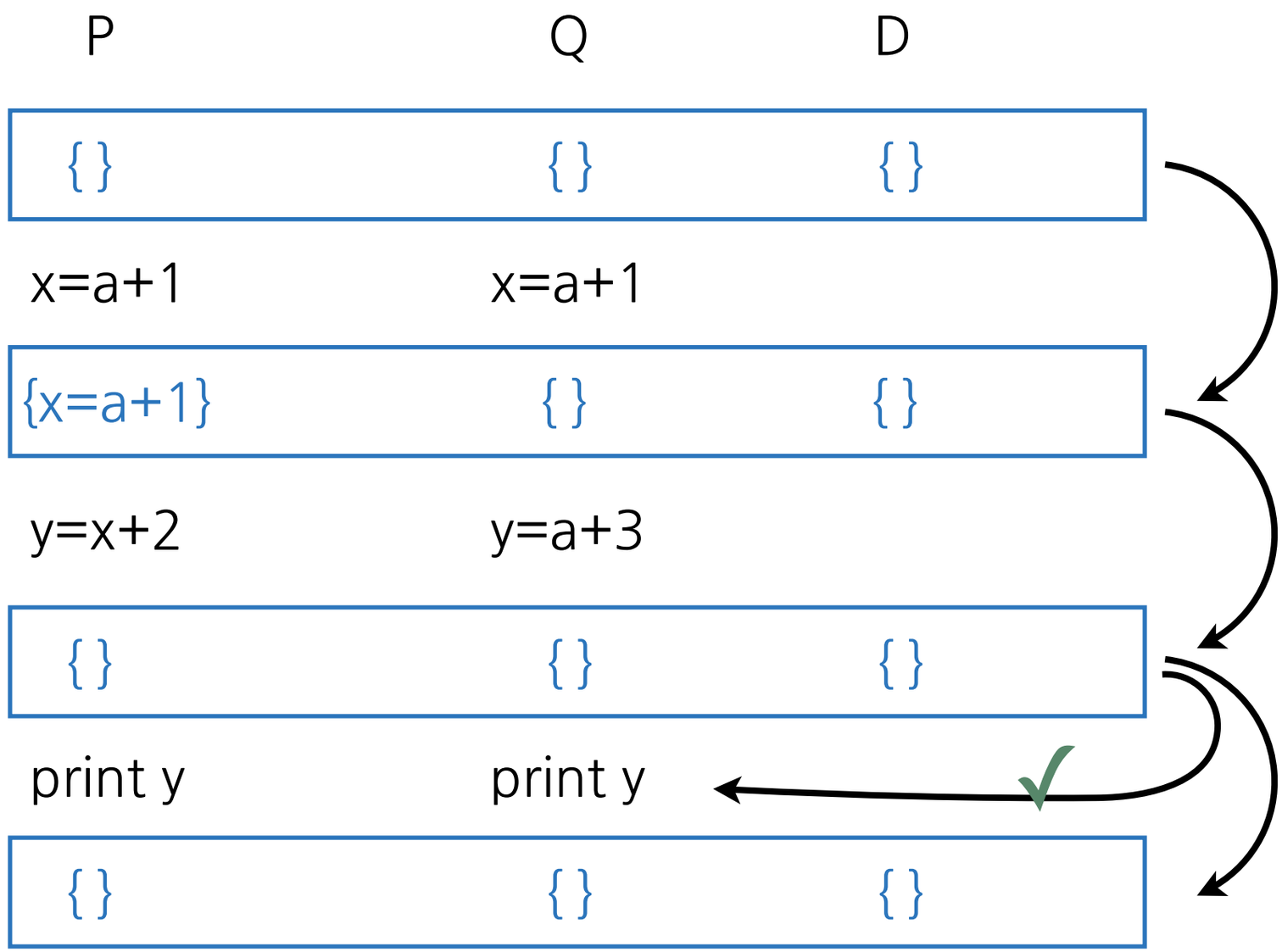
최적화 예 1: add assoc



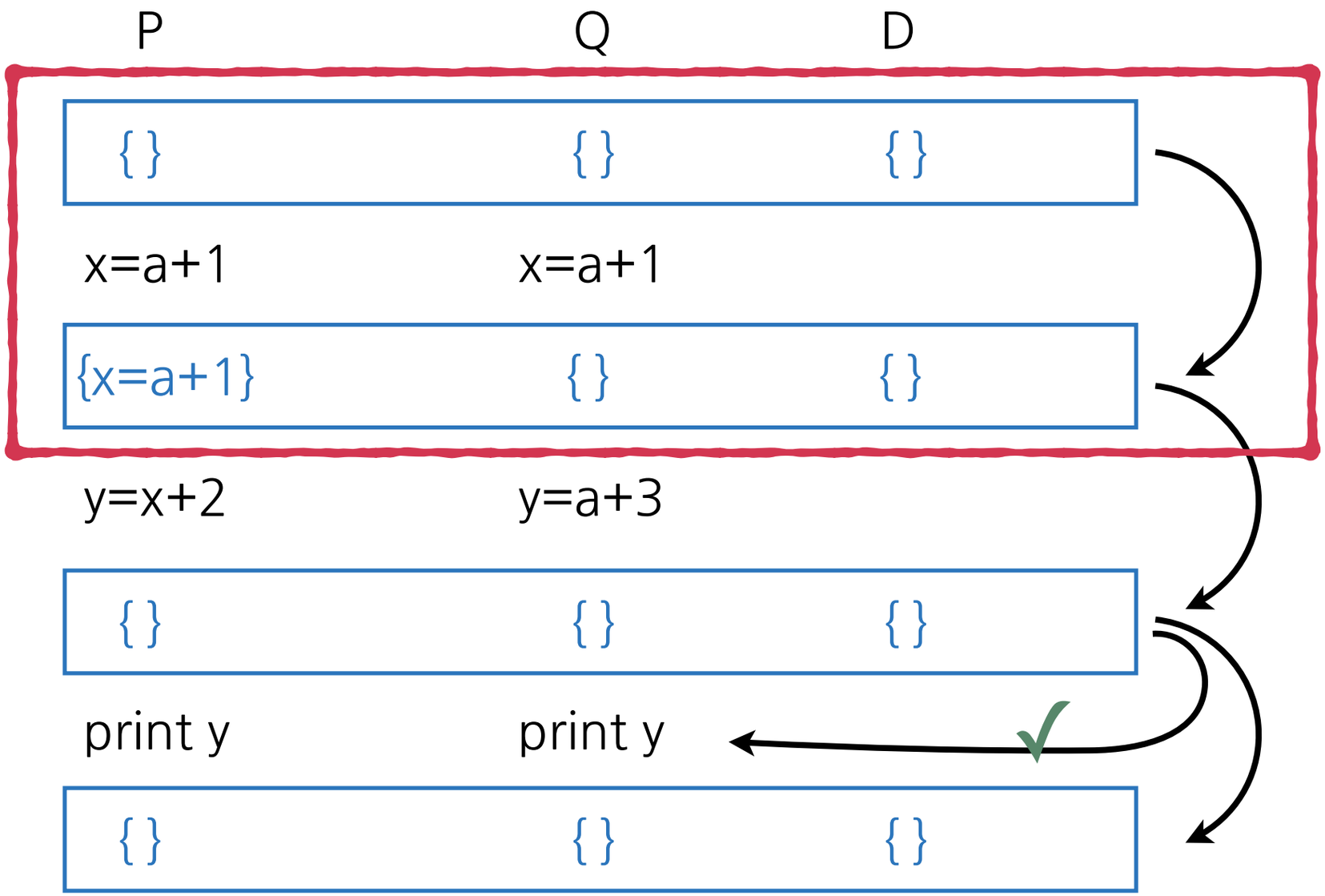
최적화 예 1: add assoc



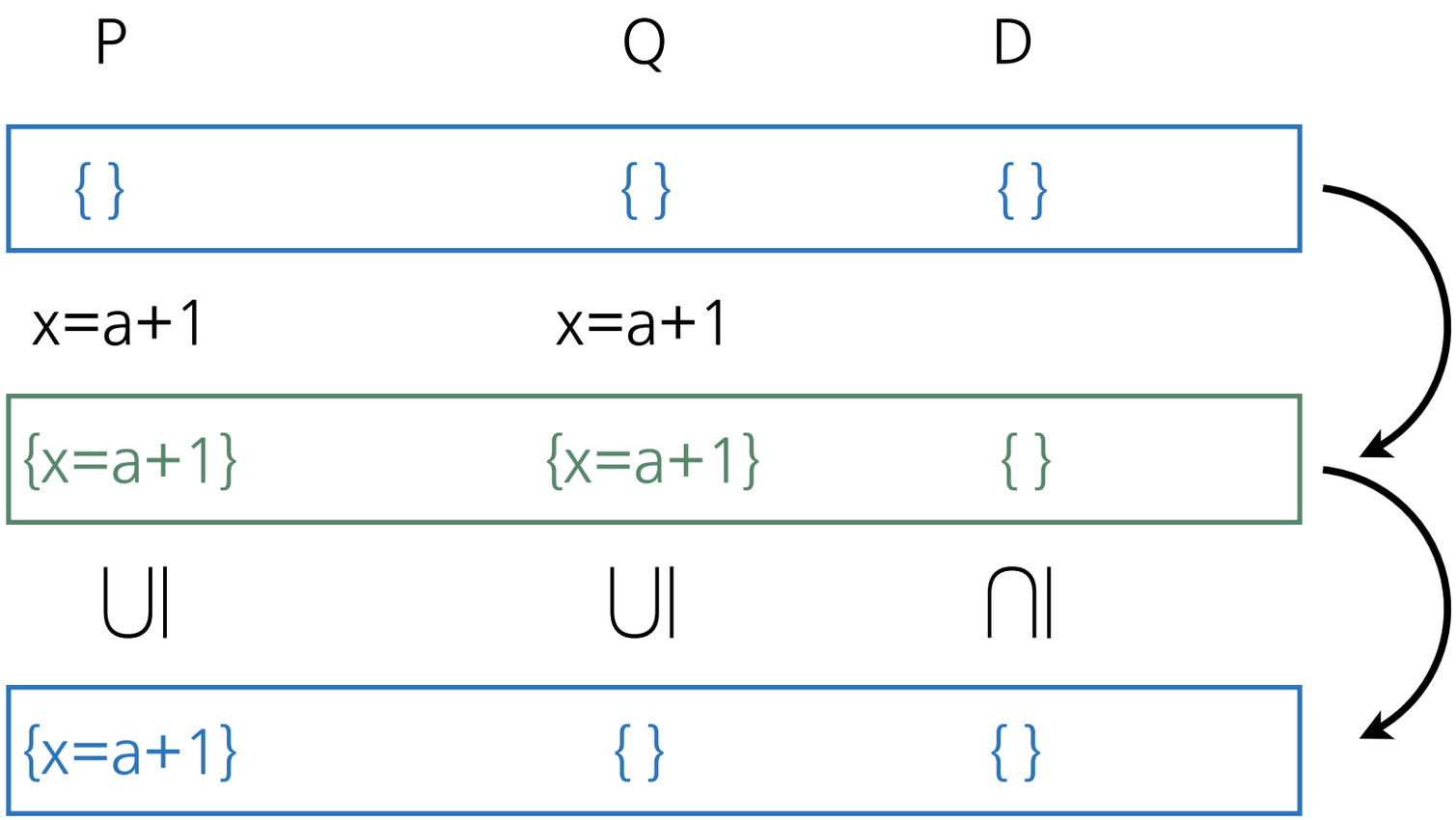
최적화 예 1: add assoc



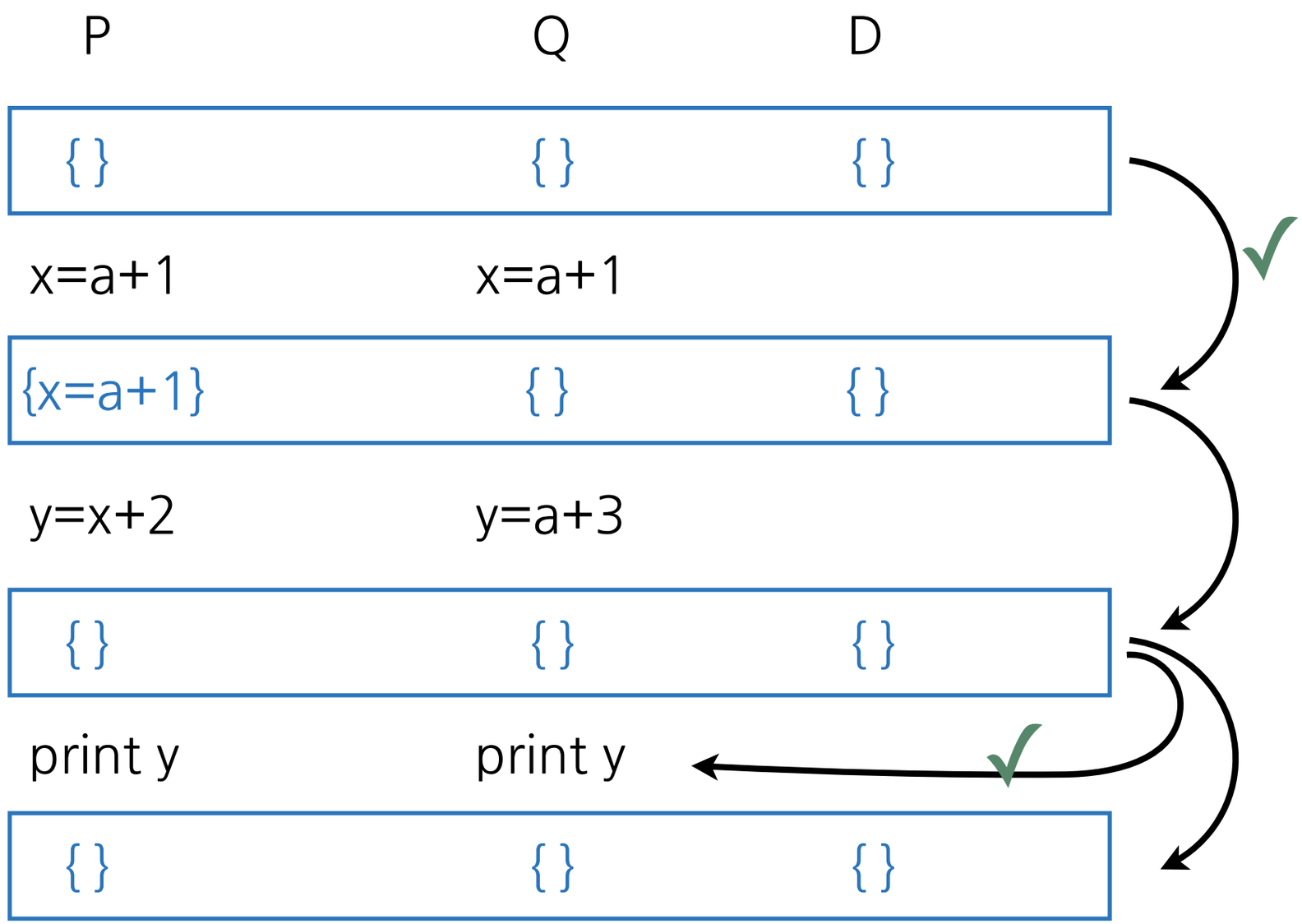
최적화 예 1: add assoc



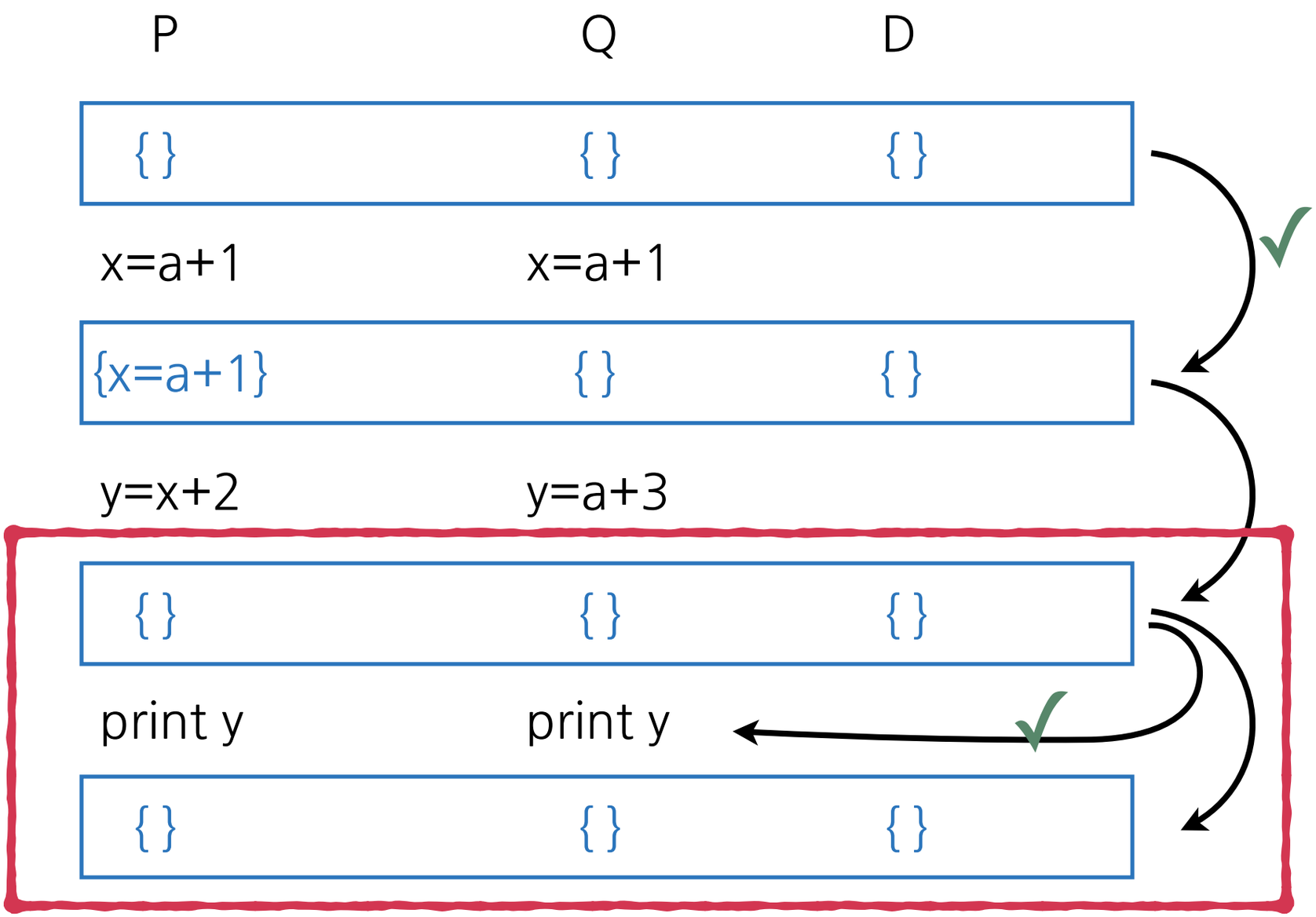
최적화 예 1: add assoc



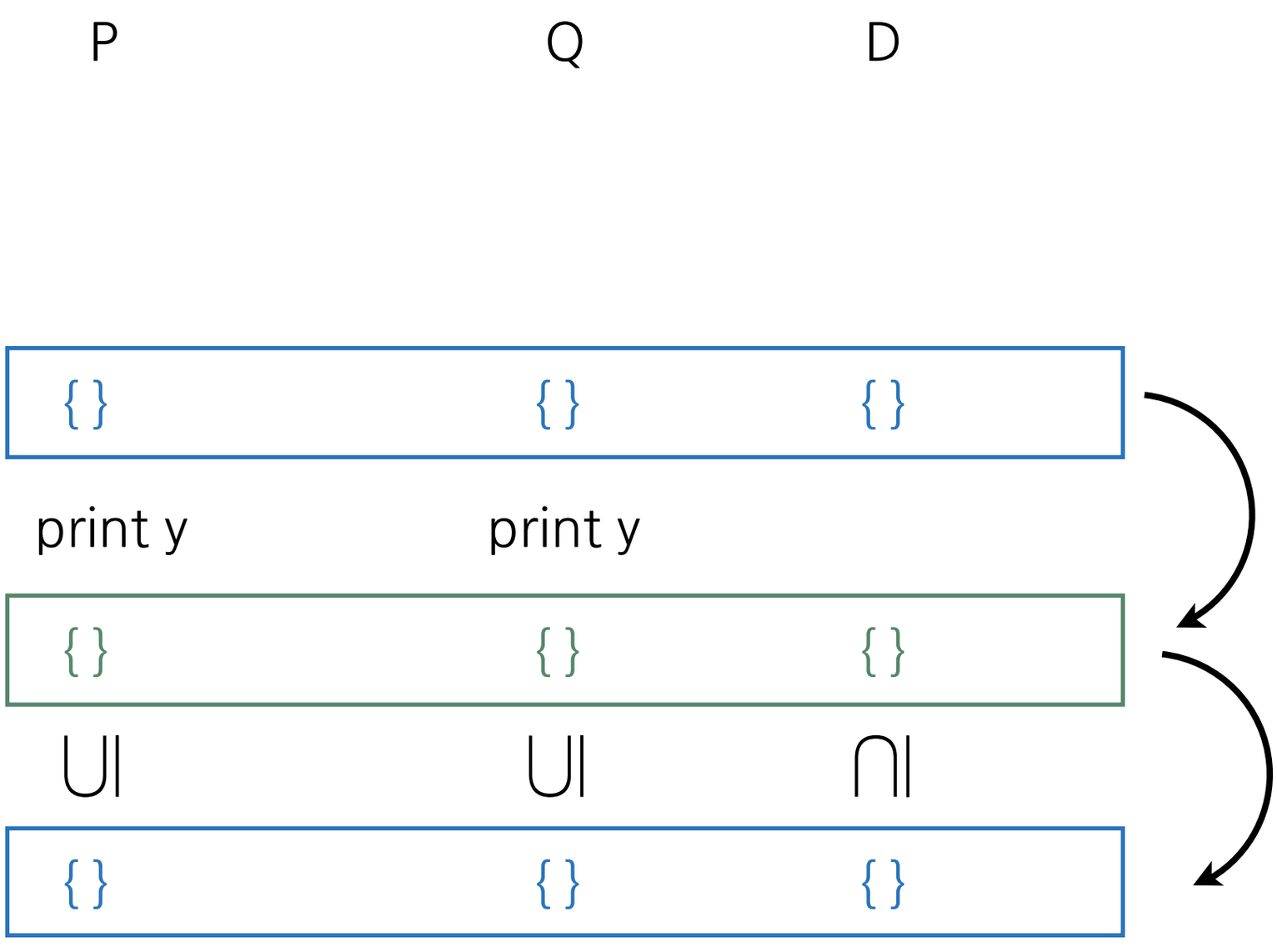
최적화 예 1: add assoc



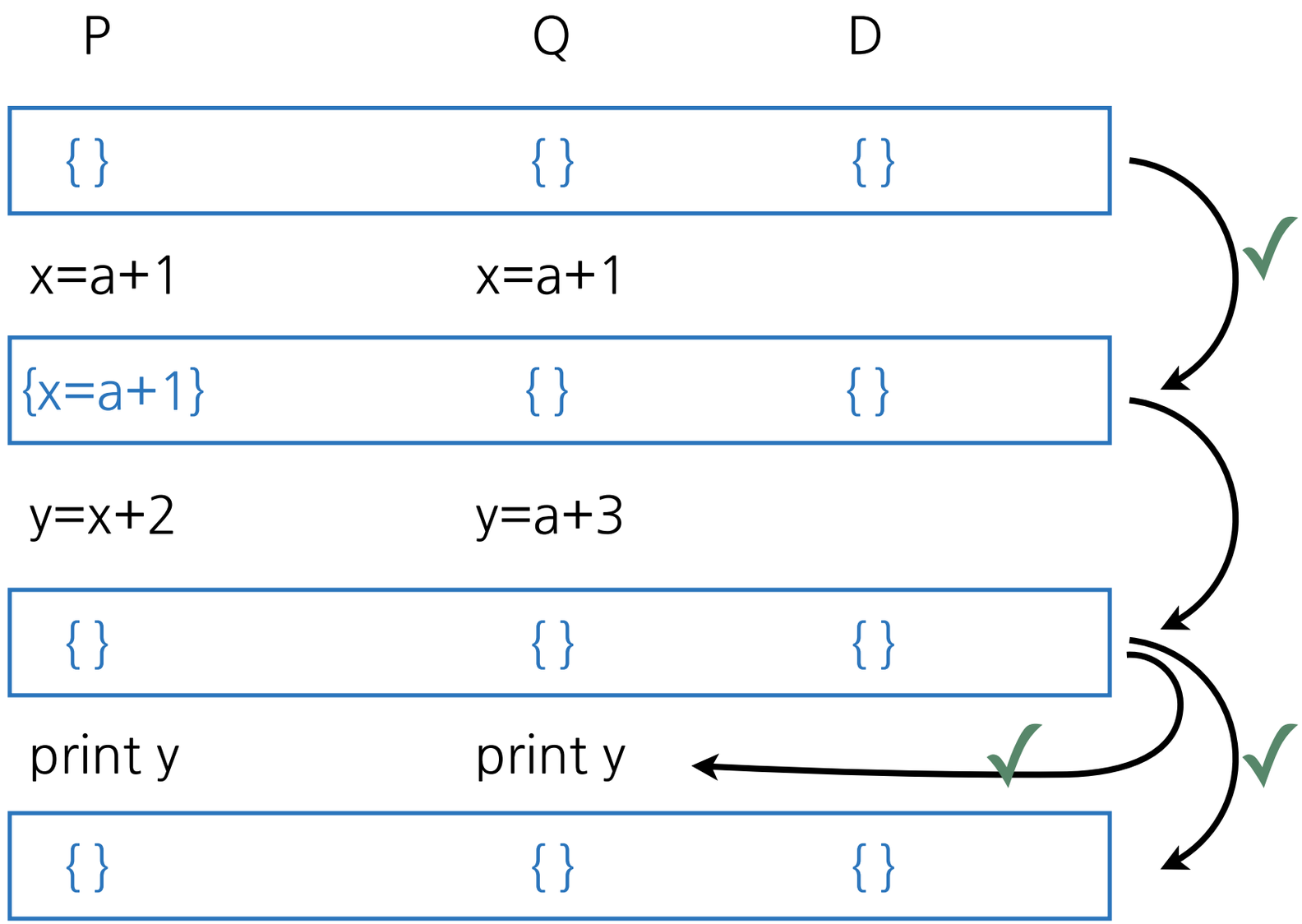
최적화 예 1: add assoc



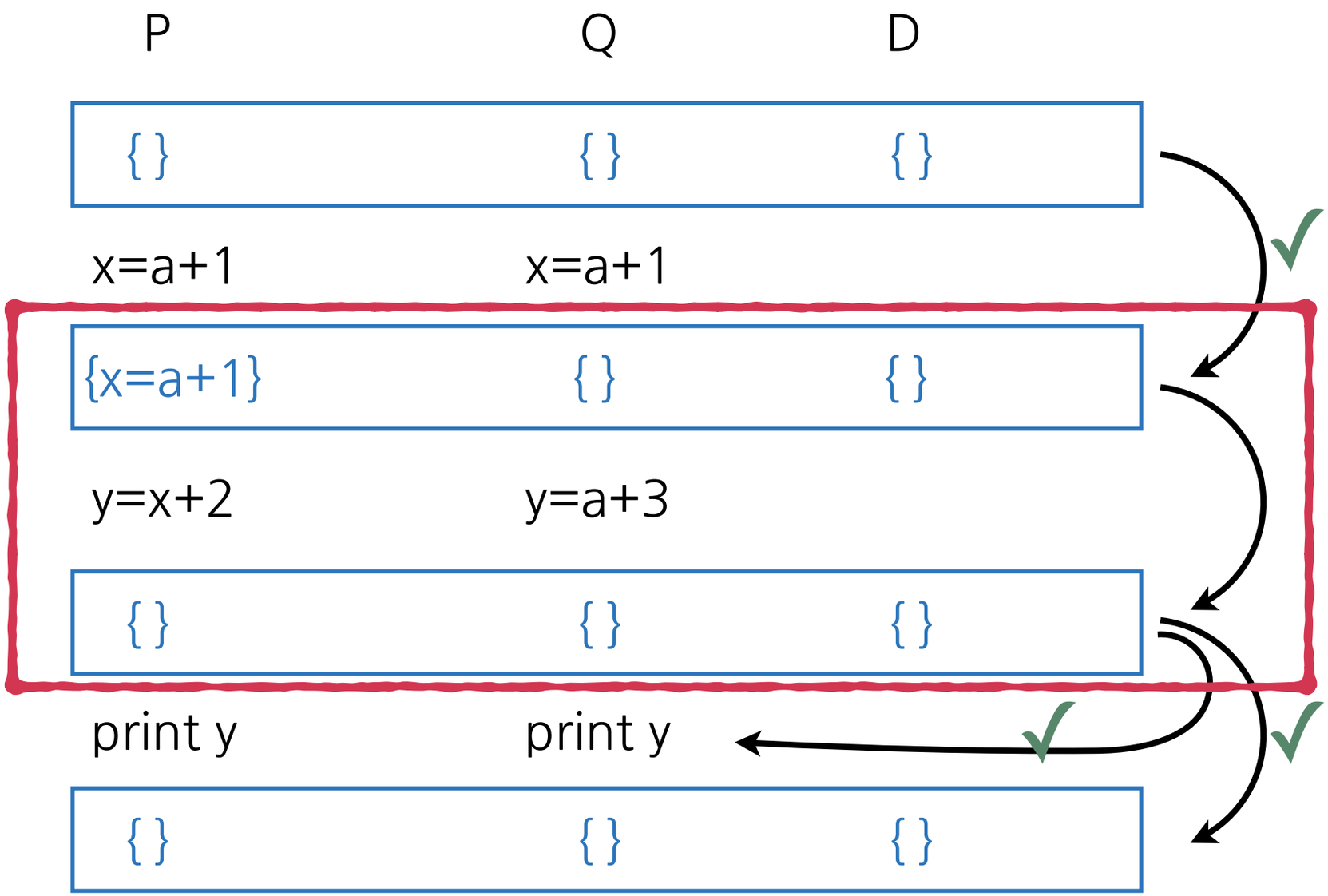
최적화 예 1: add assoc



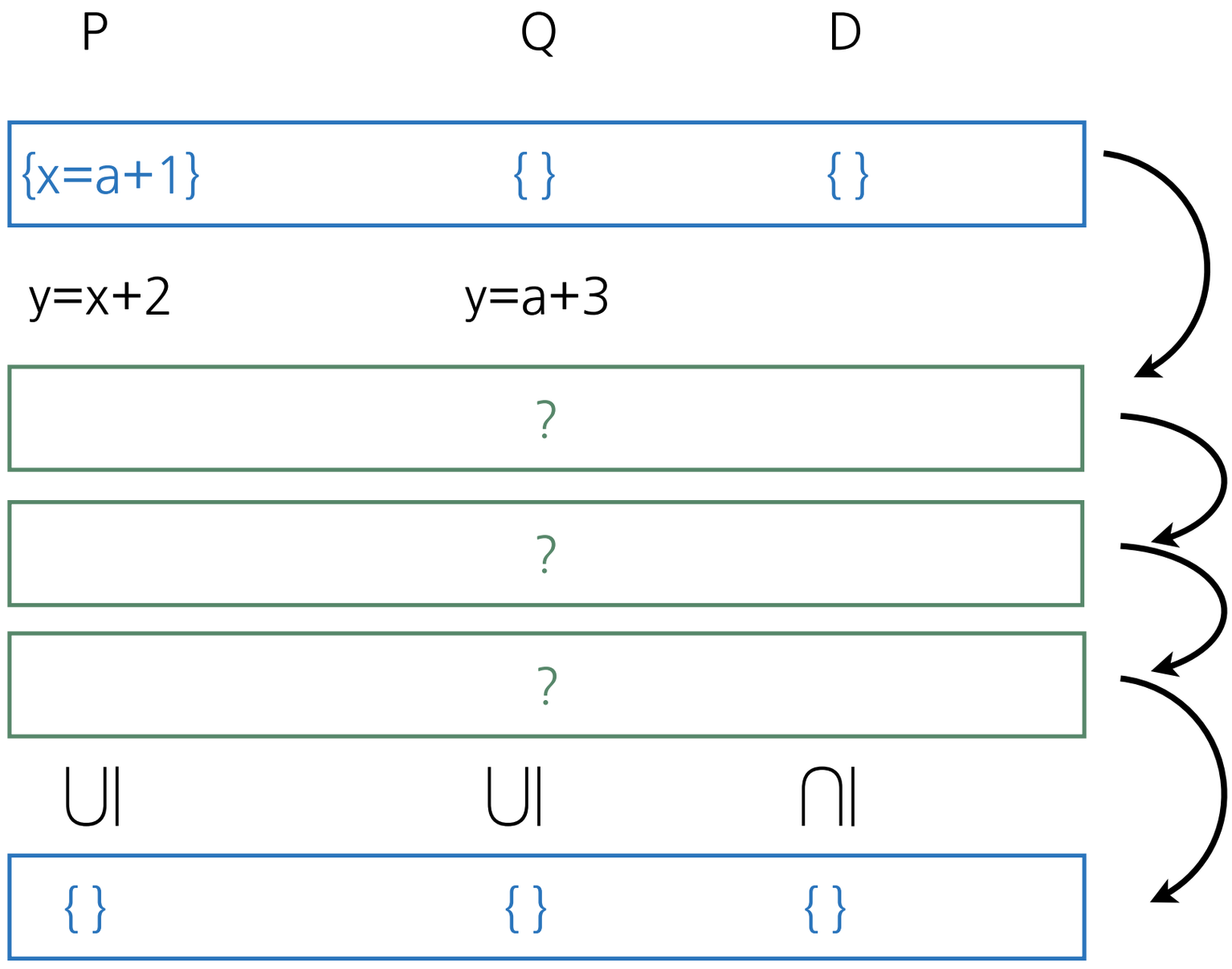
최적화 예 1: add assoc



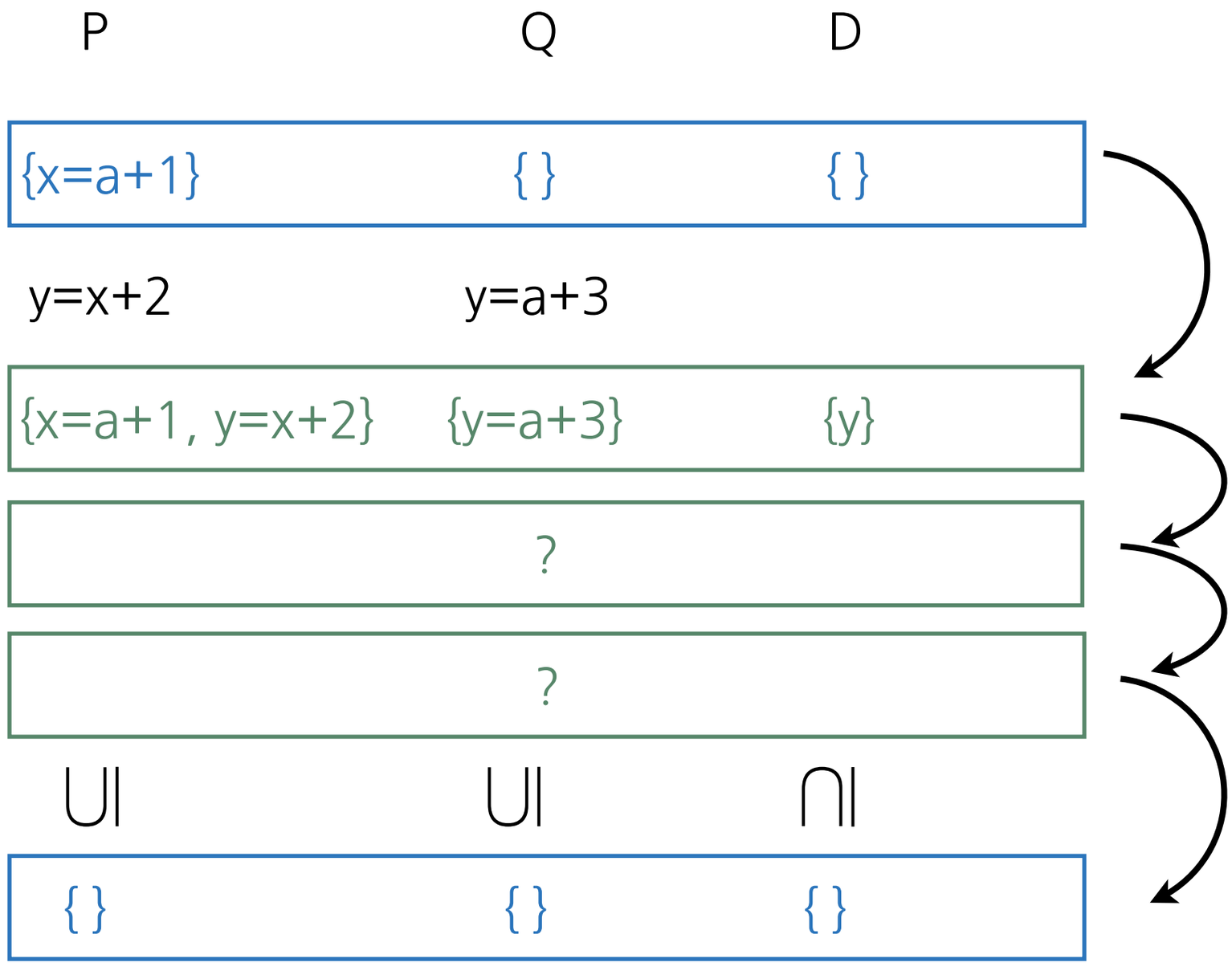
최적화 예 1: add assoc



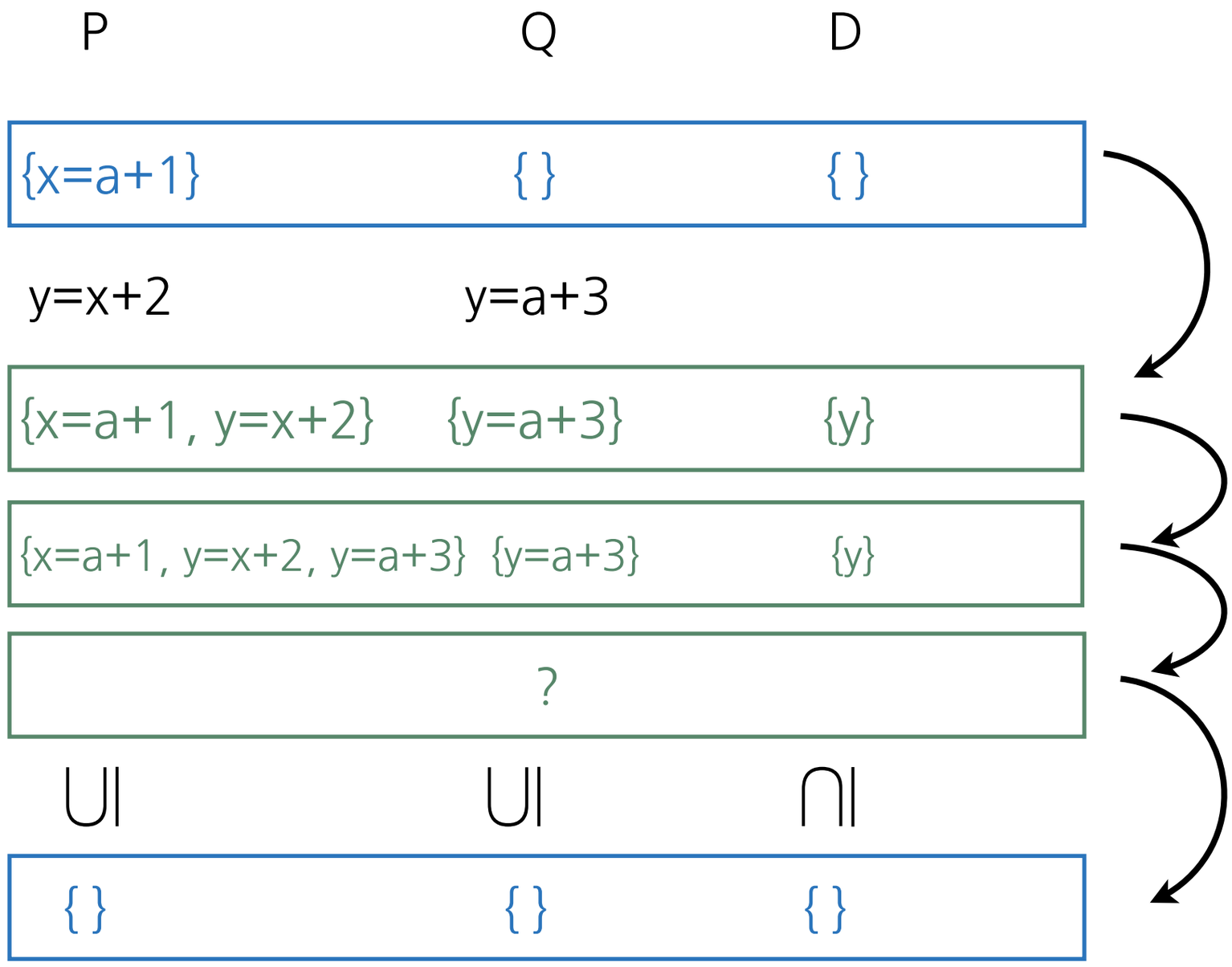
최적화 예 1: add assoc



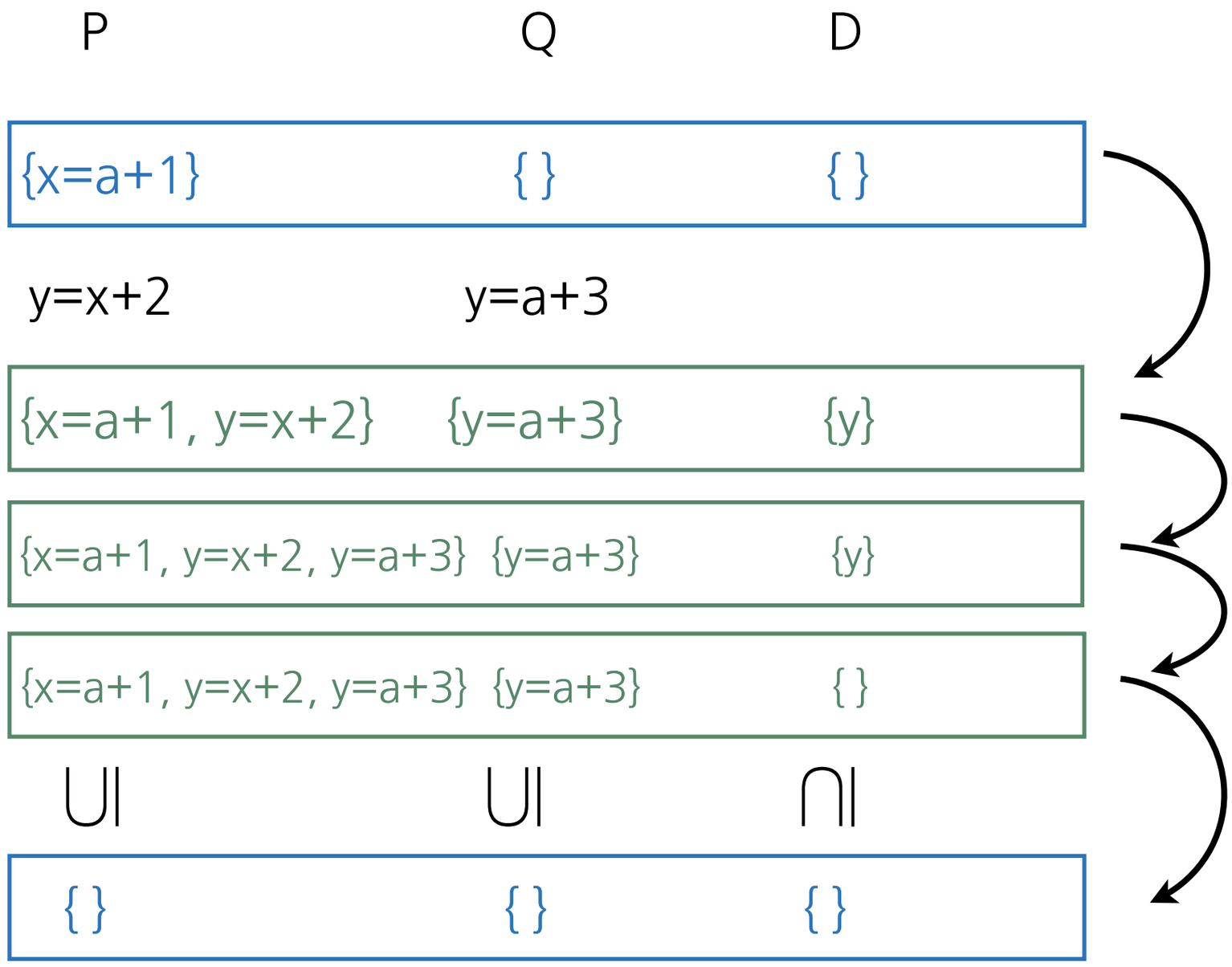
최적화 예 1: add assoc



최적화 예 1: add assoc



최적화 예 1: add assoc



최적화 예 1: add assoc

P	Q	D
{x=a+1}	{ }	{ }
y=x+2	y=a+3	
{x=a+1, y=x+2}	{y=a+3}	{y}
{x=a+1, y=x+2, y=a+3}	{y=a+3}	{y}
{x=a+1, y=x+2, y=a+3}	{y=a+3}	{ }
∪I	∪I	∩I
{ }	{ }	{ }

추론규칙 사용

add_assoc

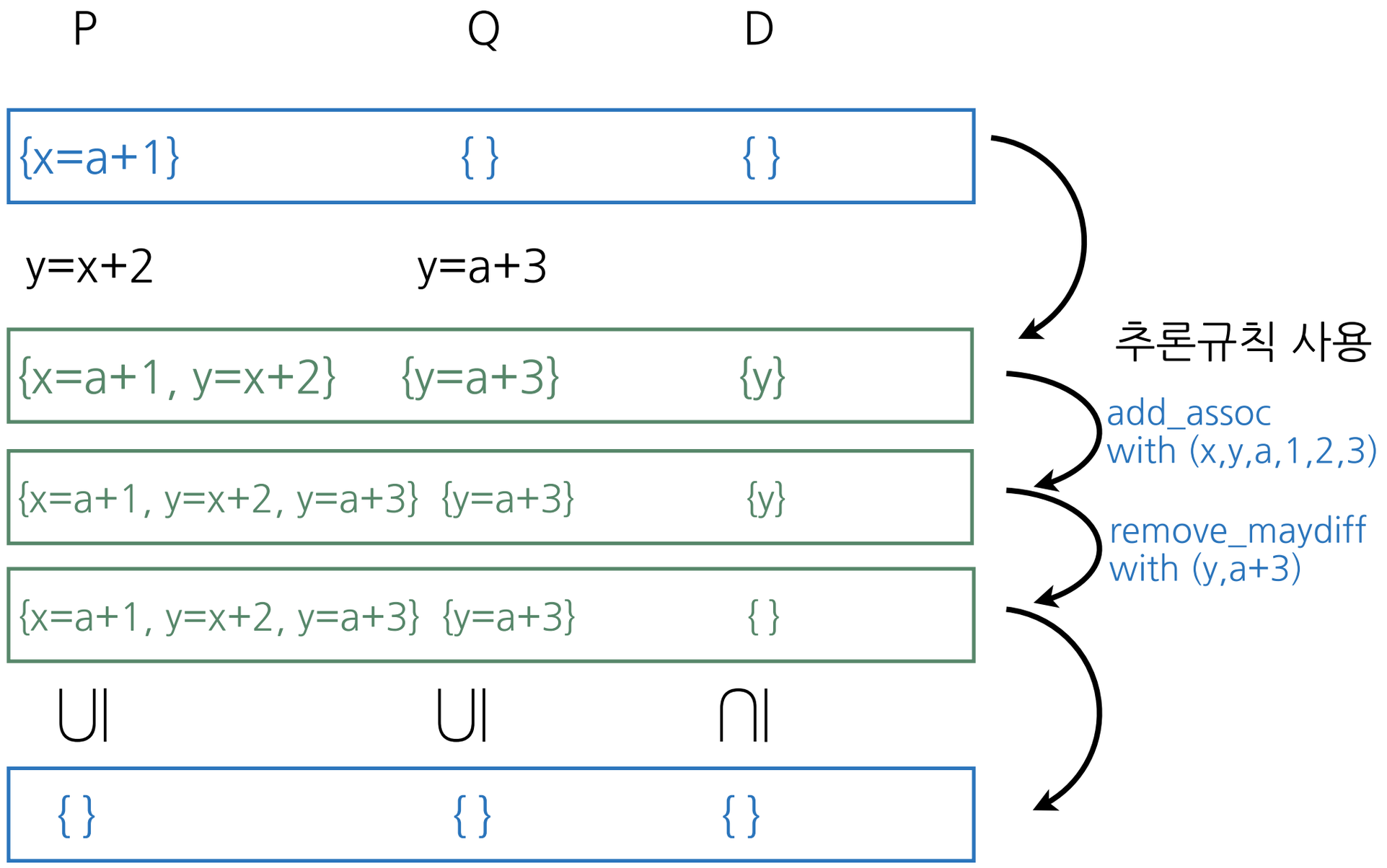
remove_maydiff

추론규칙

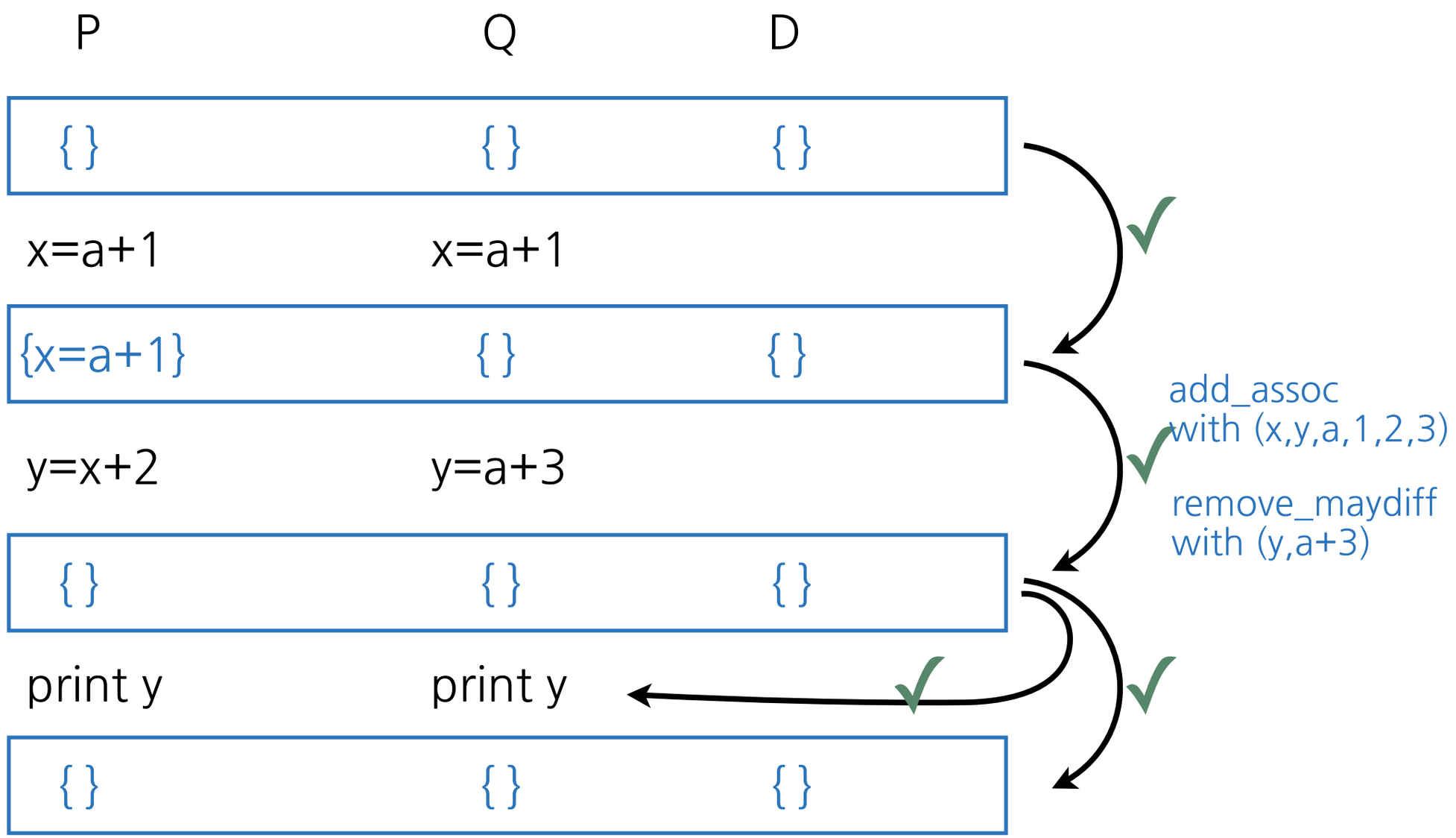
$$\texttt{add\_assoc}\ x\ y\ a\ c_1\ c_2\ c_3\ \texttt{in src} := \lambda(P, Q, D).$$
$$\quad \texttt{if}\ (x = a + c_1 \in P) \wedge (y = x + c_2 \in P) \wedge c_3 = c_1 + c_2$$
$$\quad \texttt{then}\ (P \cup \{\, y = a + c_3 \,\}, Q, D)\ \texttt{else}\ (P, Q, D)$$

$$\texttt{remove\_maydiff}\ x\ e := \lambda(P, Q, D).$$
$$\quad \texttt{if}\ x = e \in P \wedge x = e \in Q \wedge \mathrm{Fv}(e) \cap D = \emptyset$$
$$\quad \texttt{then}\ (P, Q, D \backslash \{\, x \,\})\ \texttt{else}\ (P, Q, D)$$

최적화 예 1: add assoc



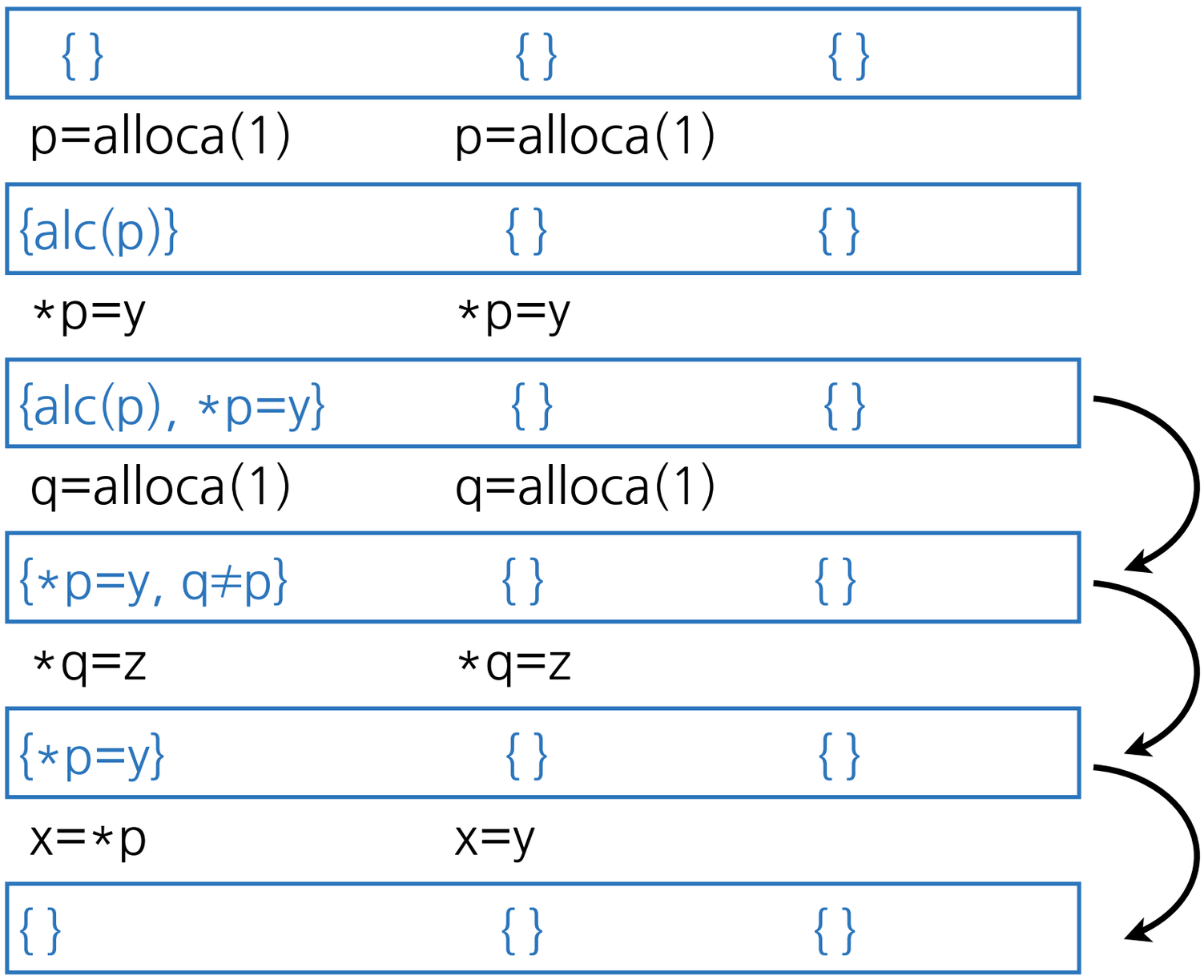
최적화 예 1: add assoc



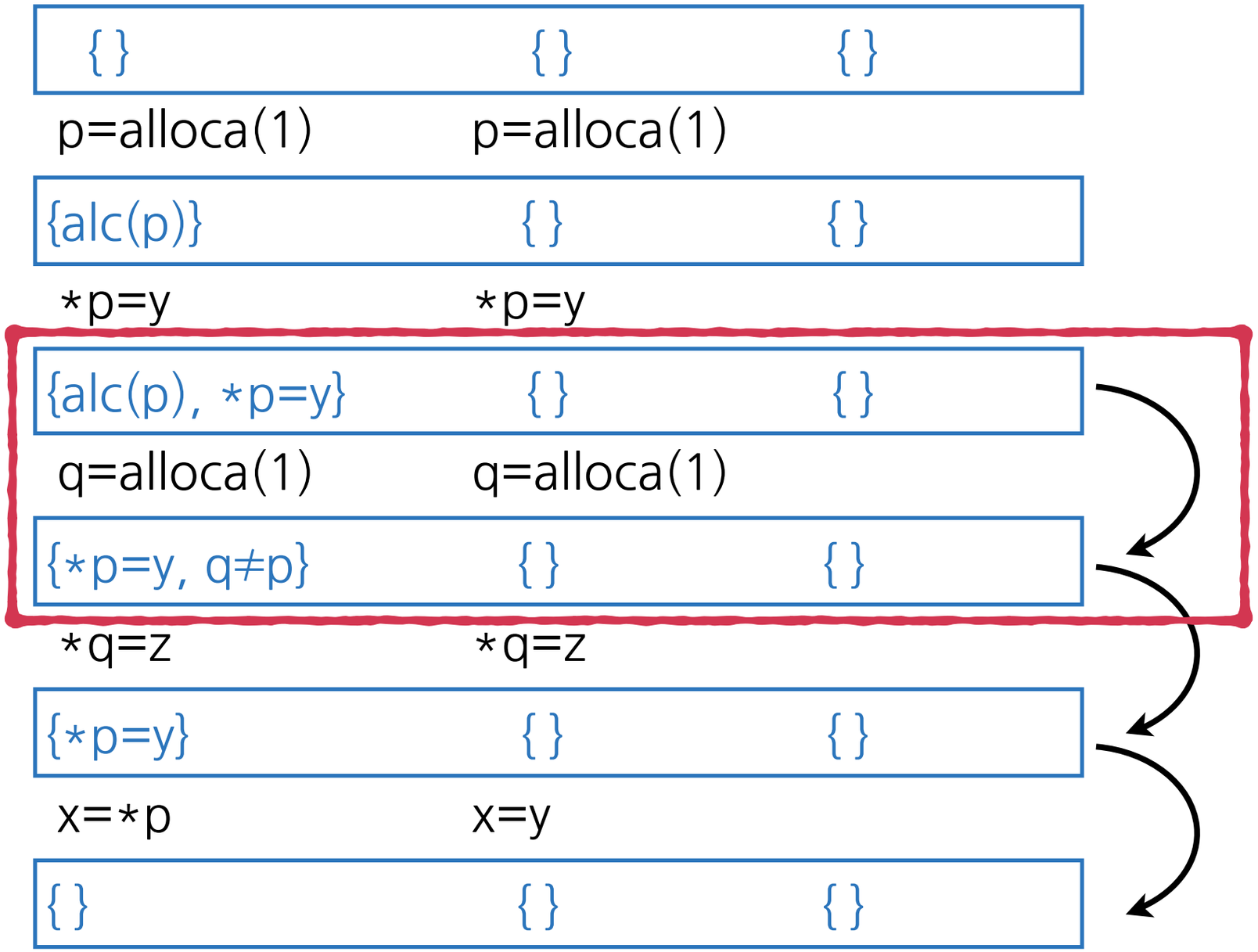
최적화 예 2: store load

p=alloca(1)	p=alloca(1)
*p=y	*p=y
q=alloca(1)	q=alloca(1)
*q=z	*q=z
x=*p	x=y

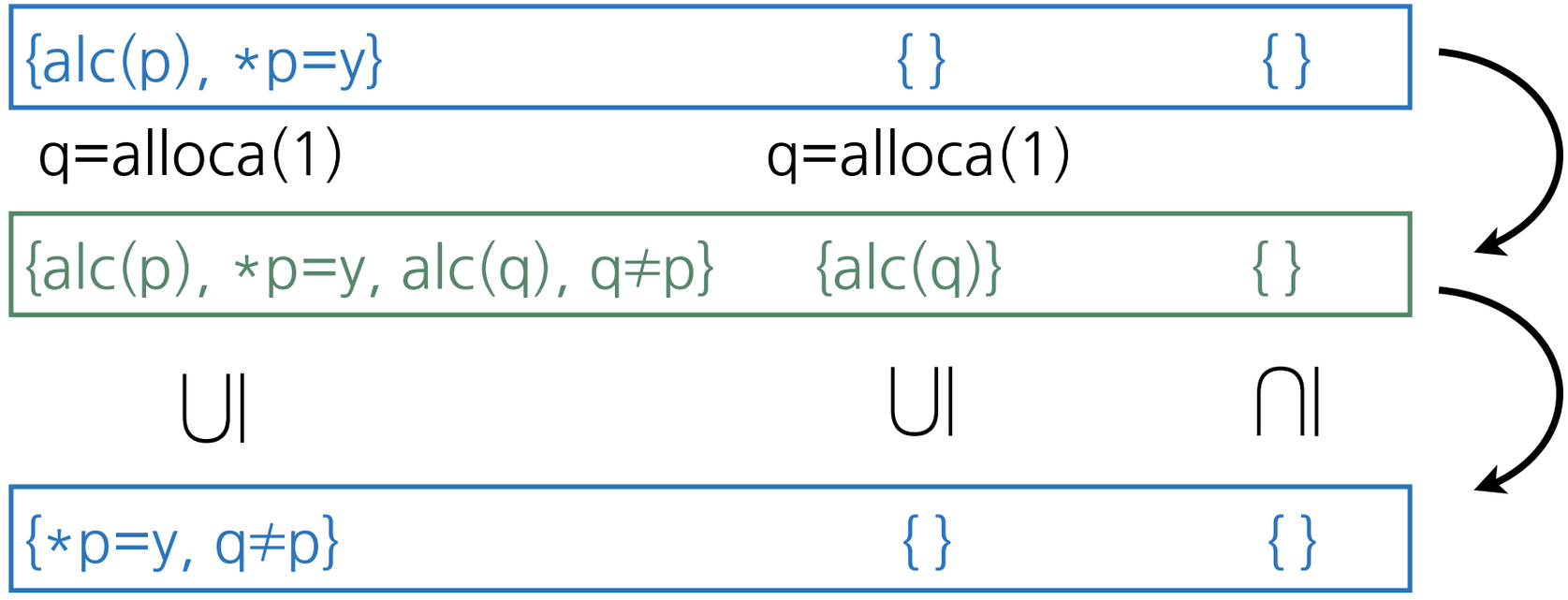
최적화 예 2: store load



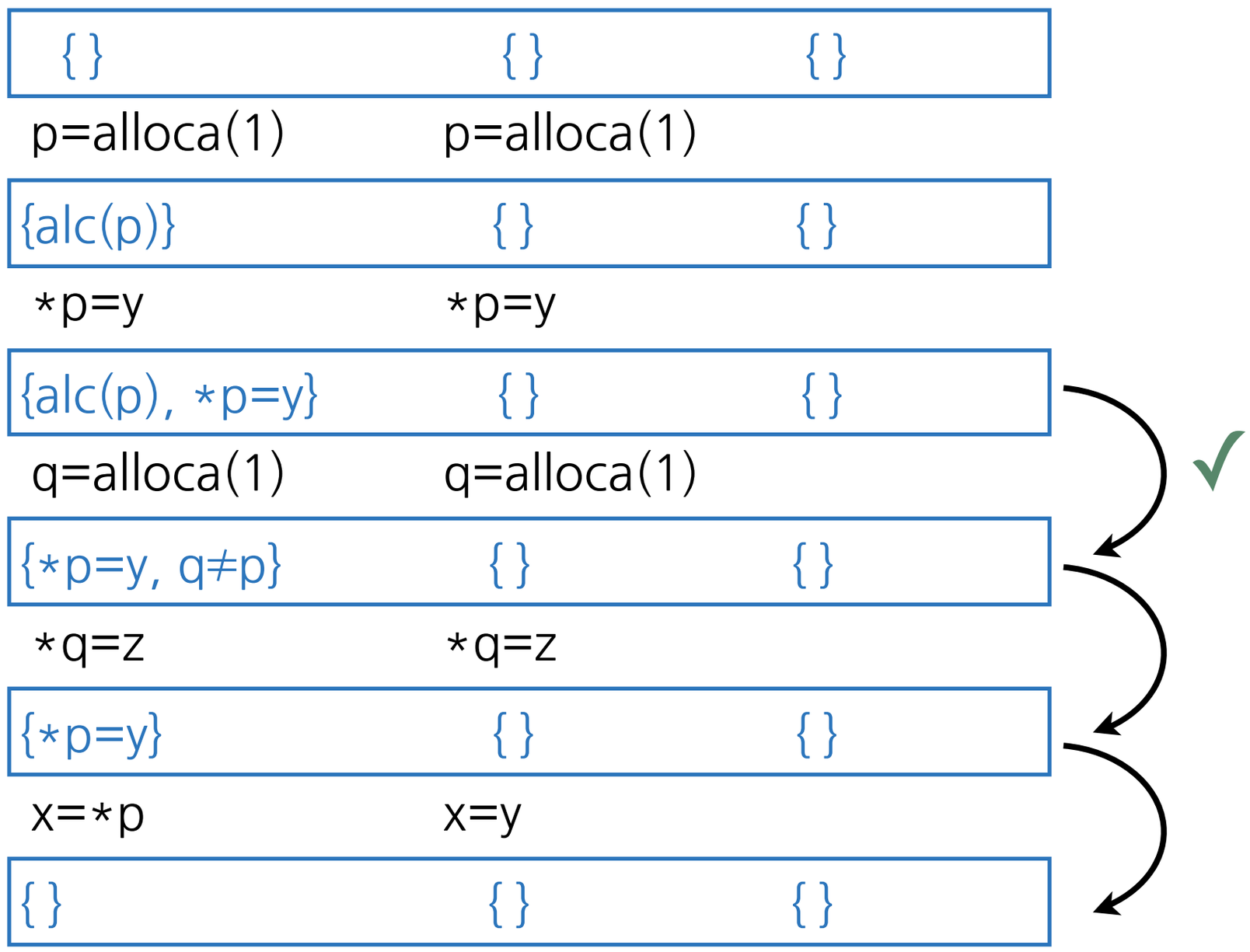
최적화 예 2: store load



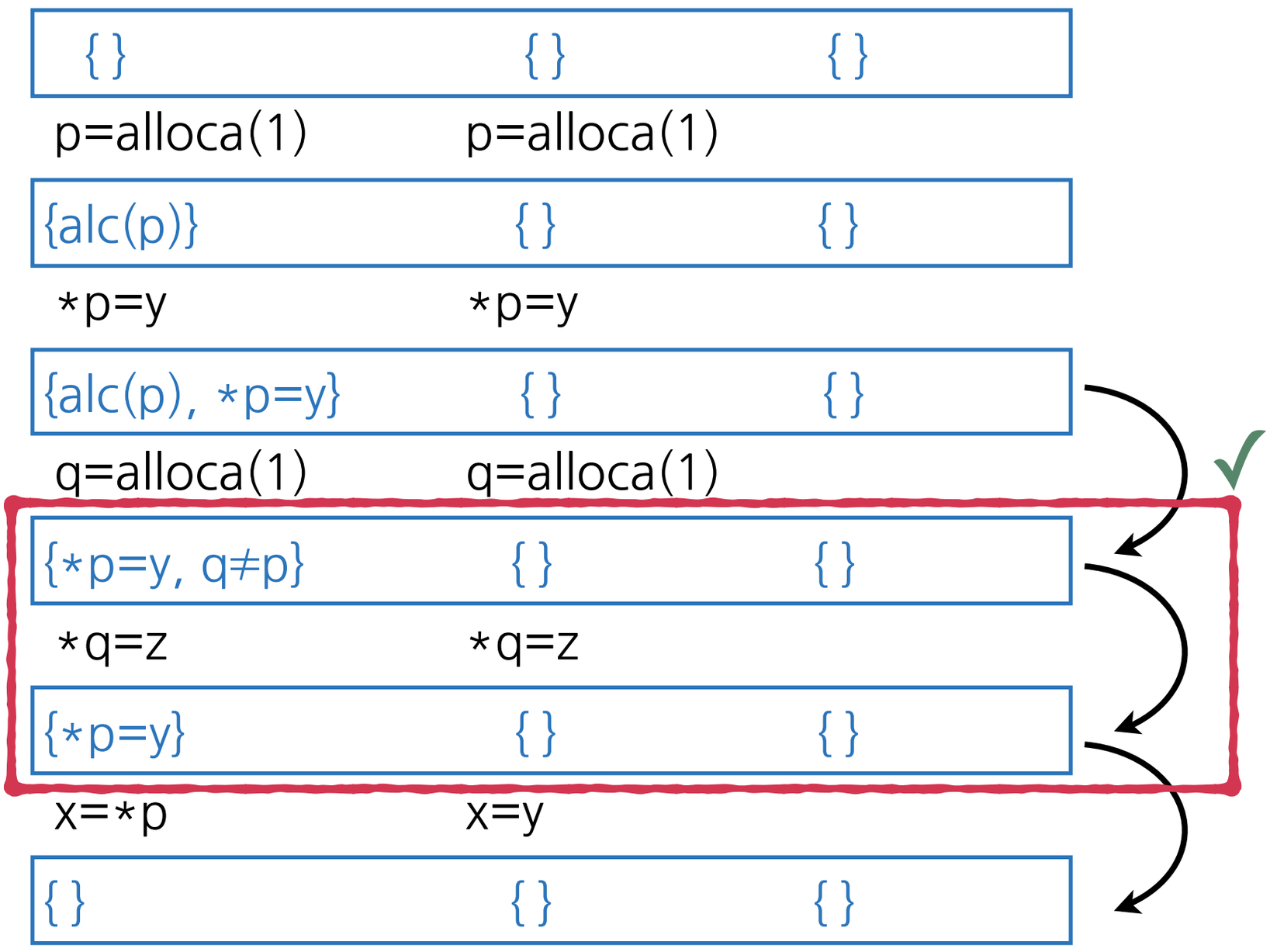
최적화 예 2: store load



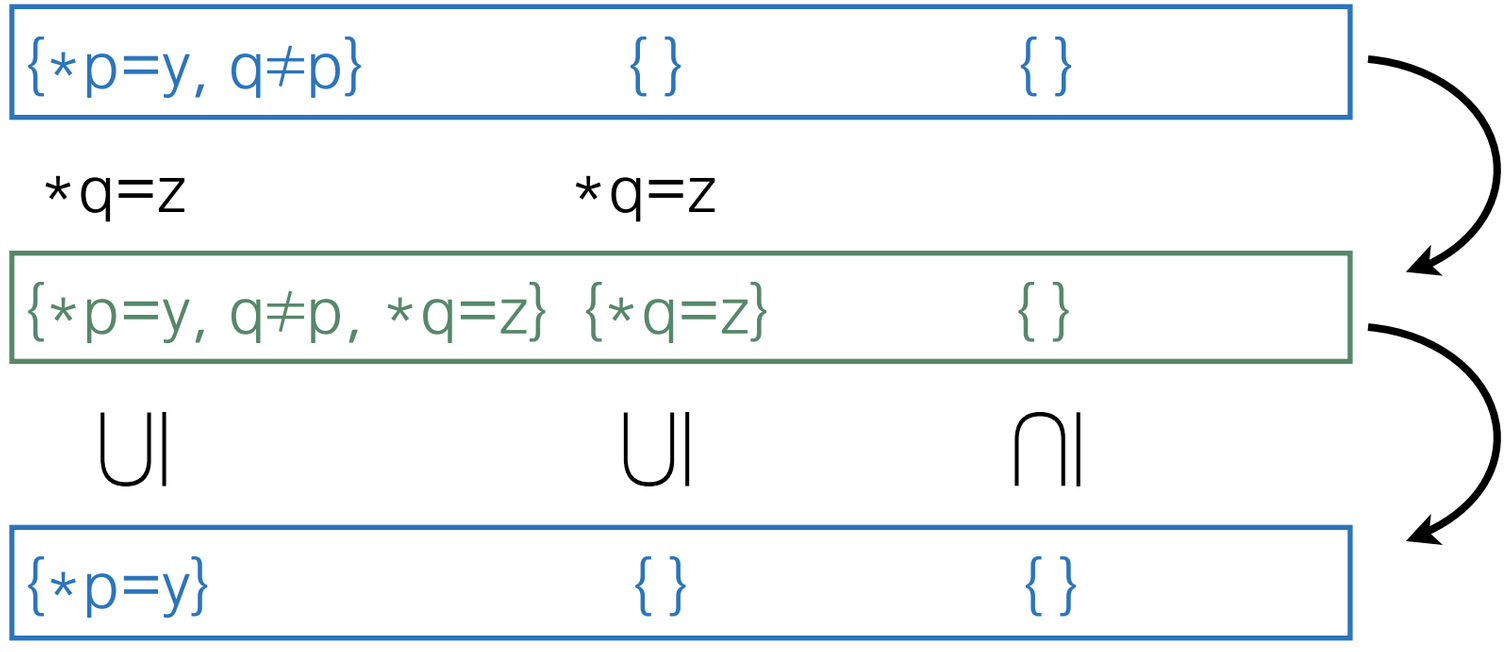
최적화 예 2: store load



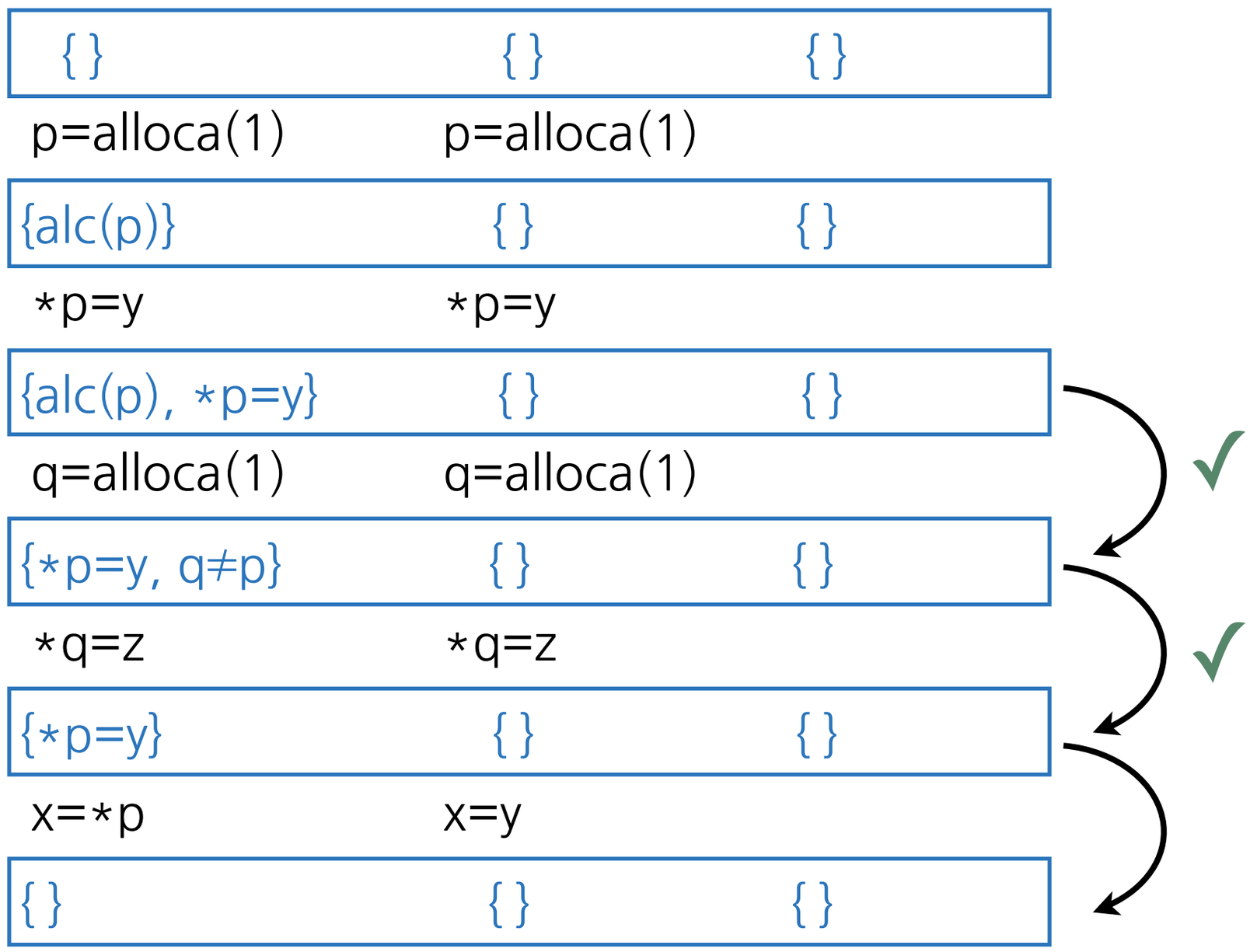
최적화 예 2: store load



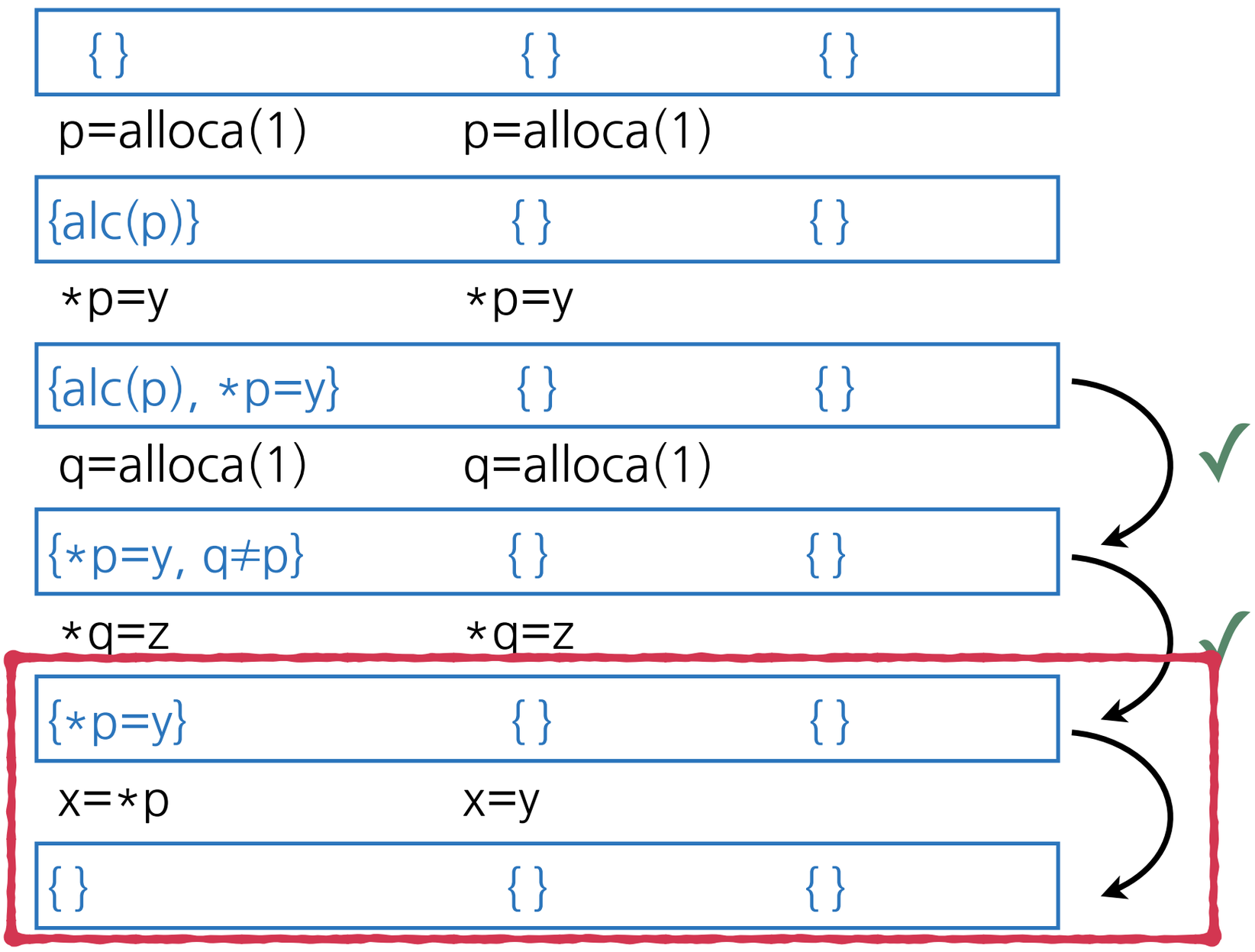
최적화 예 2: store load



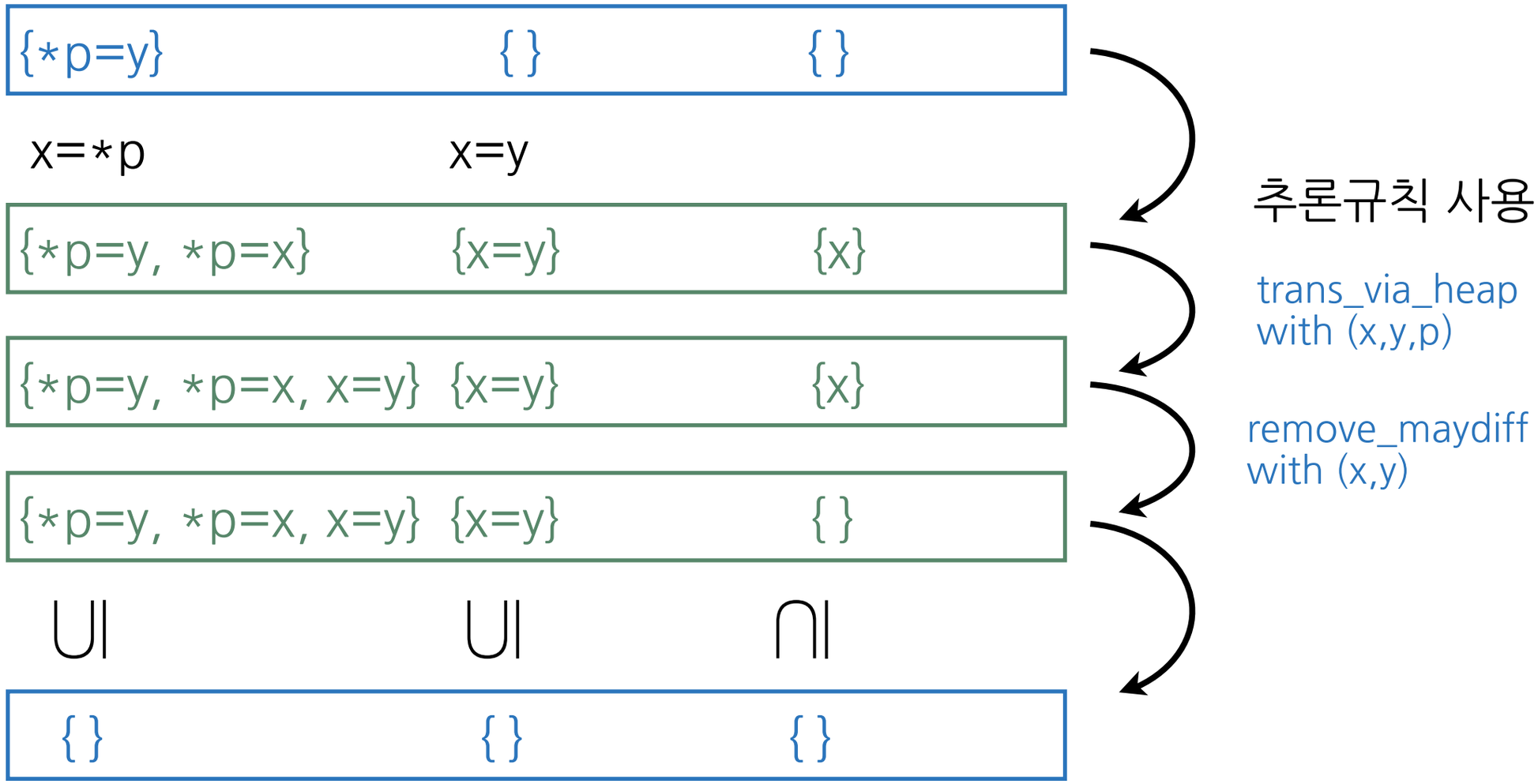
최적화 예 2: store load



최적화 예 2: store load



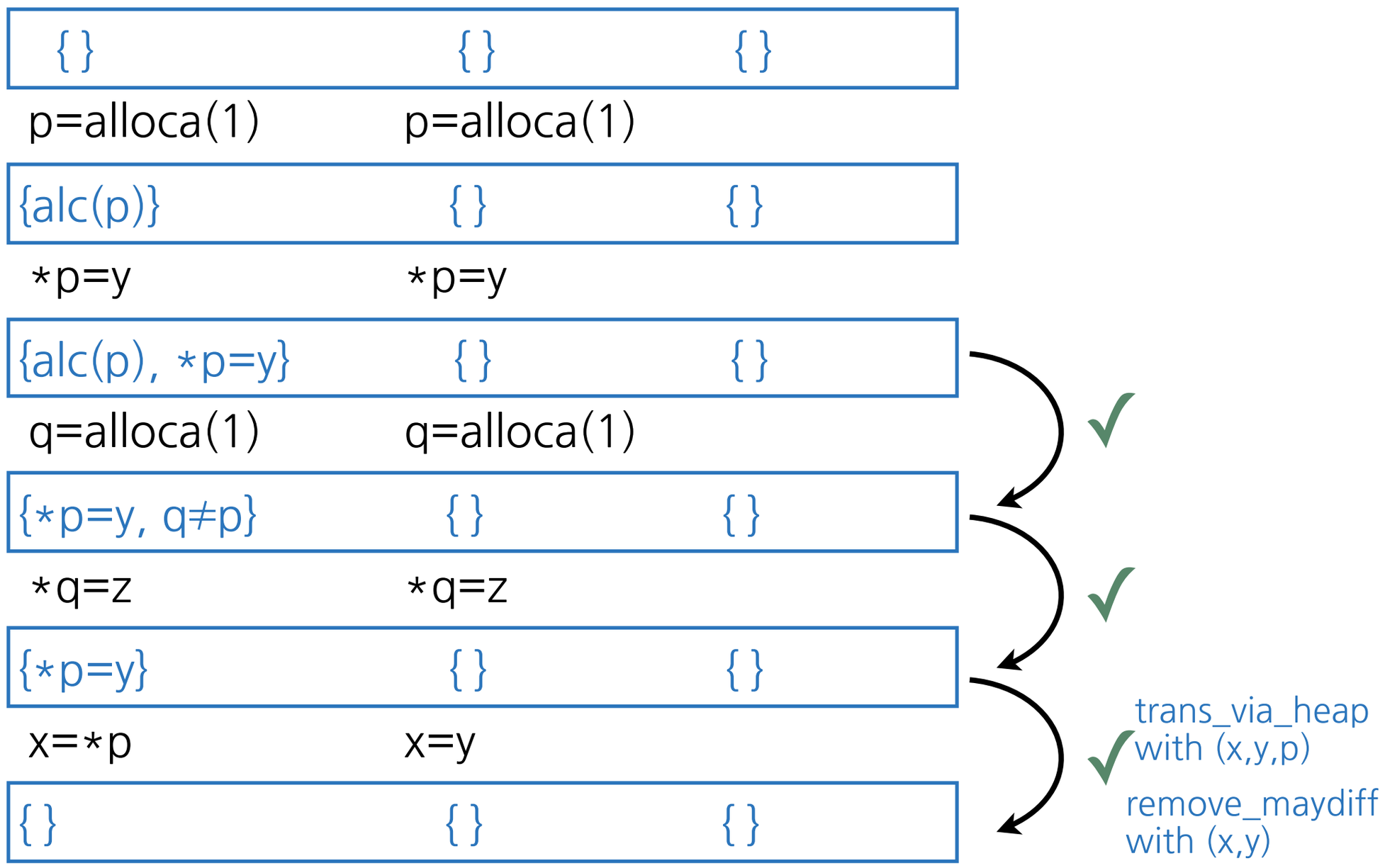
최적화 예 2: store load



추론규칙

$$\texttt{trans\_via\_heap}\ x\ y\ p\ \texttt{in}\ \texttt{src} := \lambda(P, Q, D).$$
$$\quad \texttt{if}\ *p = x \in P \wedge *p = y \in P$$
$$\quad \texttt{then}\ (P \cup \{\, x = y \,\}, Q, D)\ \texttt{else}\ (P, Q, D)$$

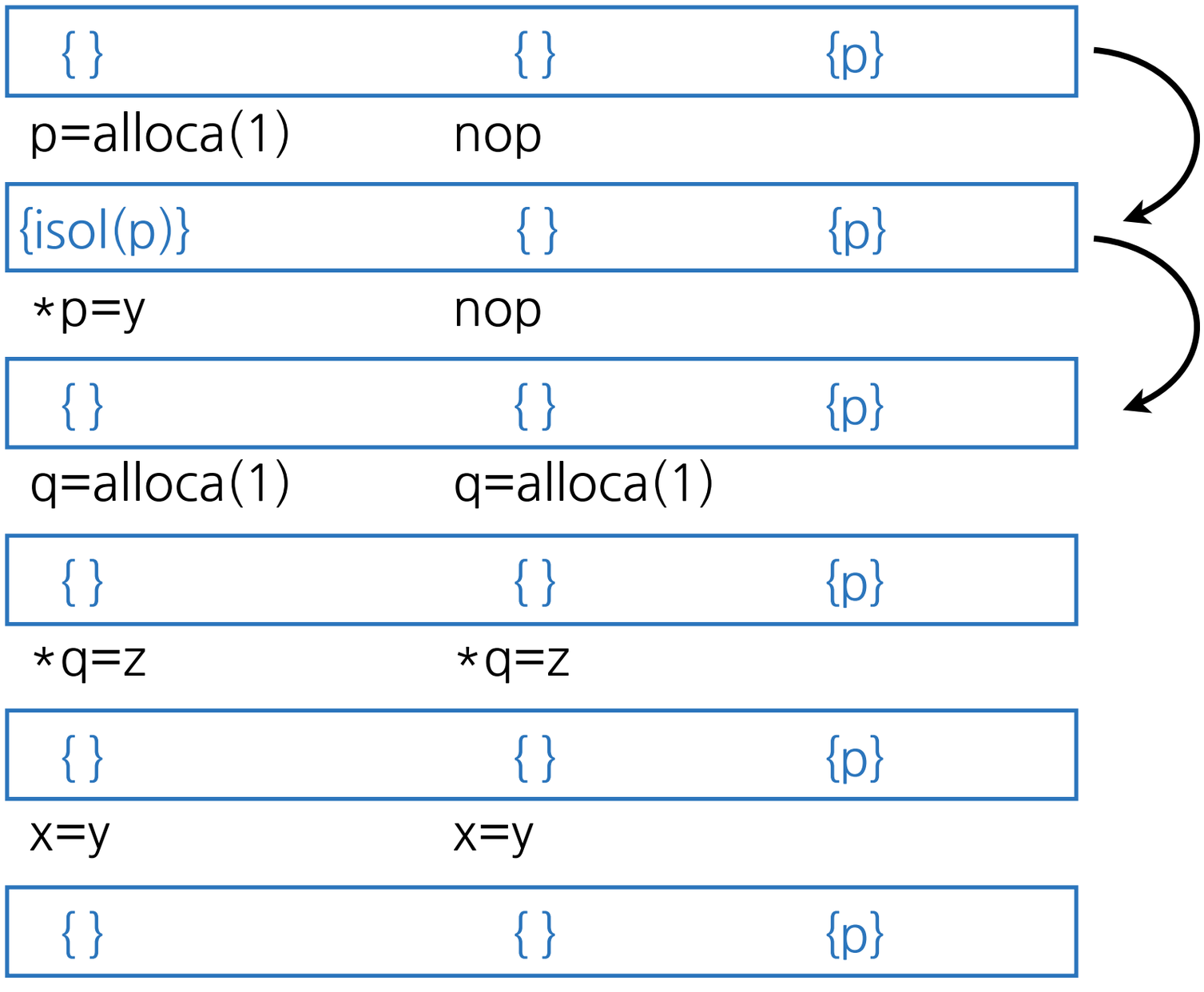
최적화 예 2: store load



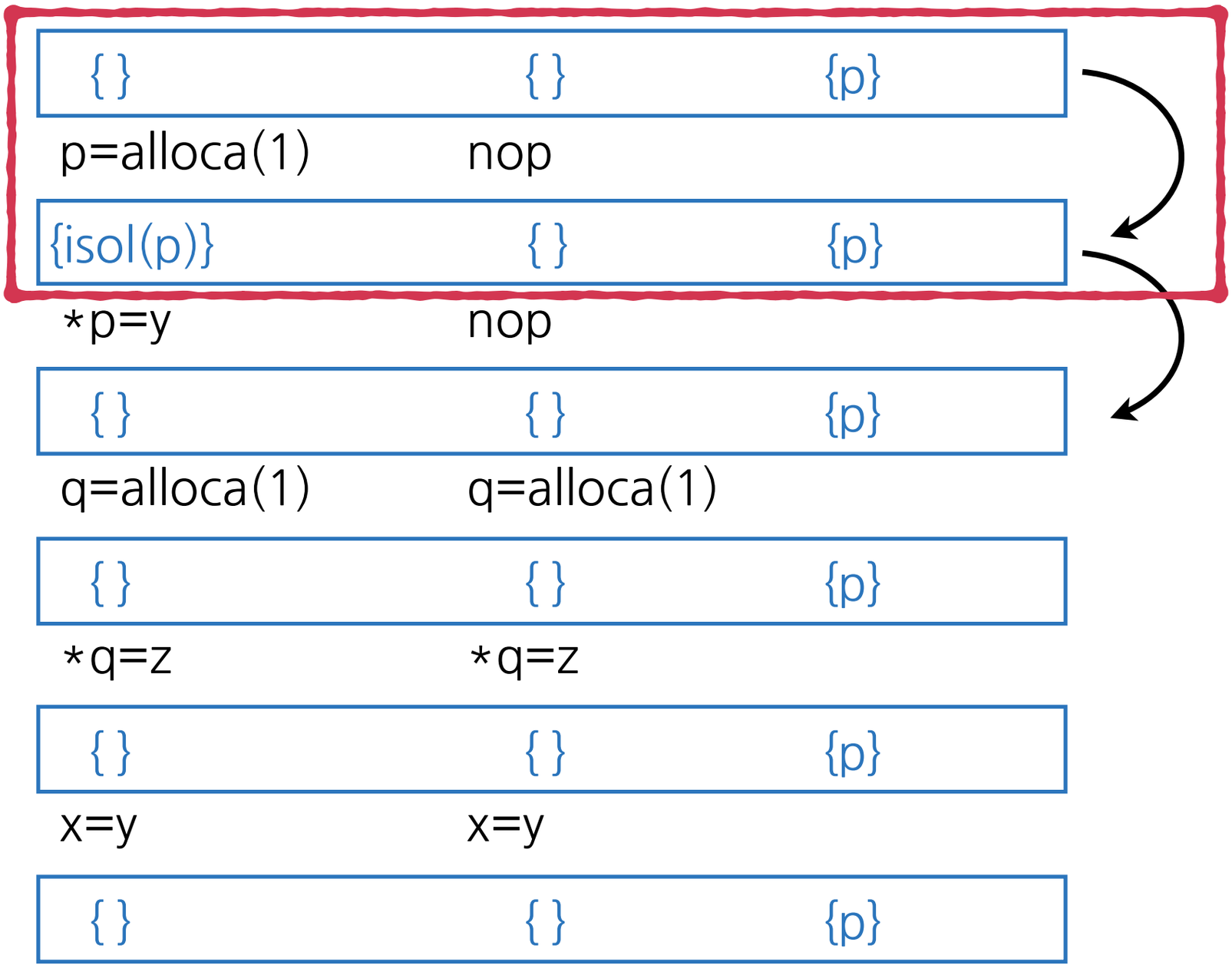
최적화 예 3: dead code elim

p=alloca(1)	nop
*p=y	nop
q=alloca(1)	q=alloca(1)
*q=z	*q=z
x=y	x=y

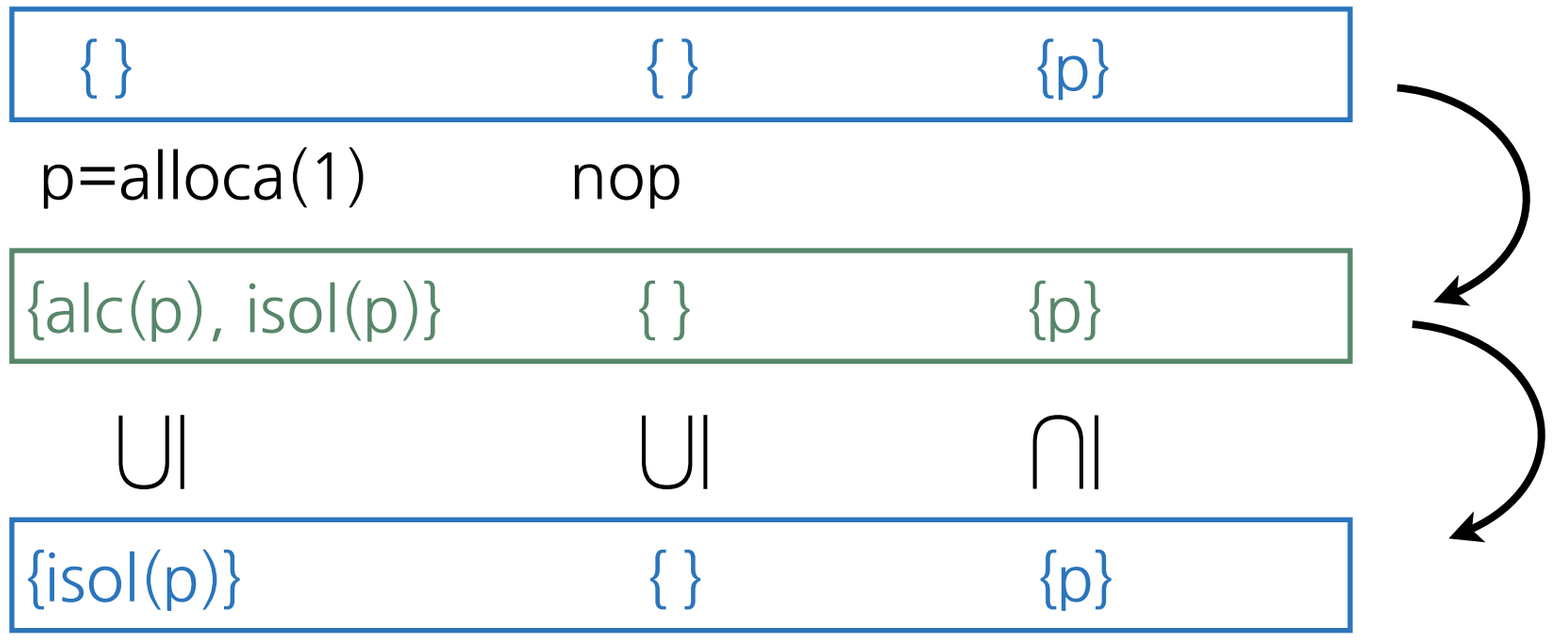
최적화 예 3: dead code elim



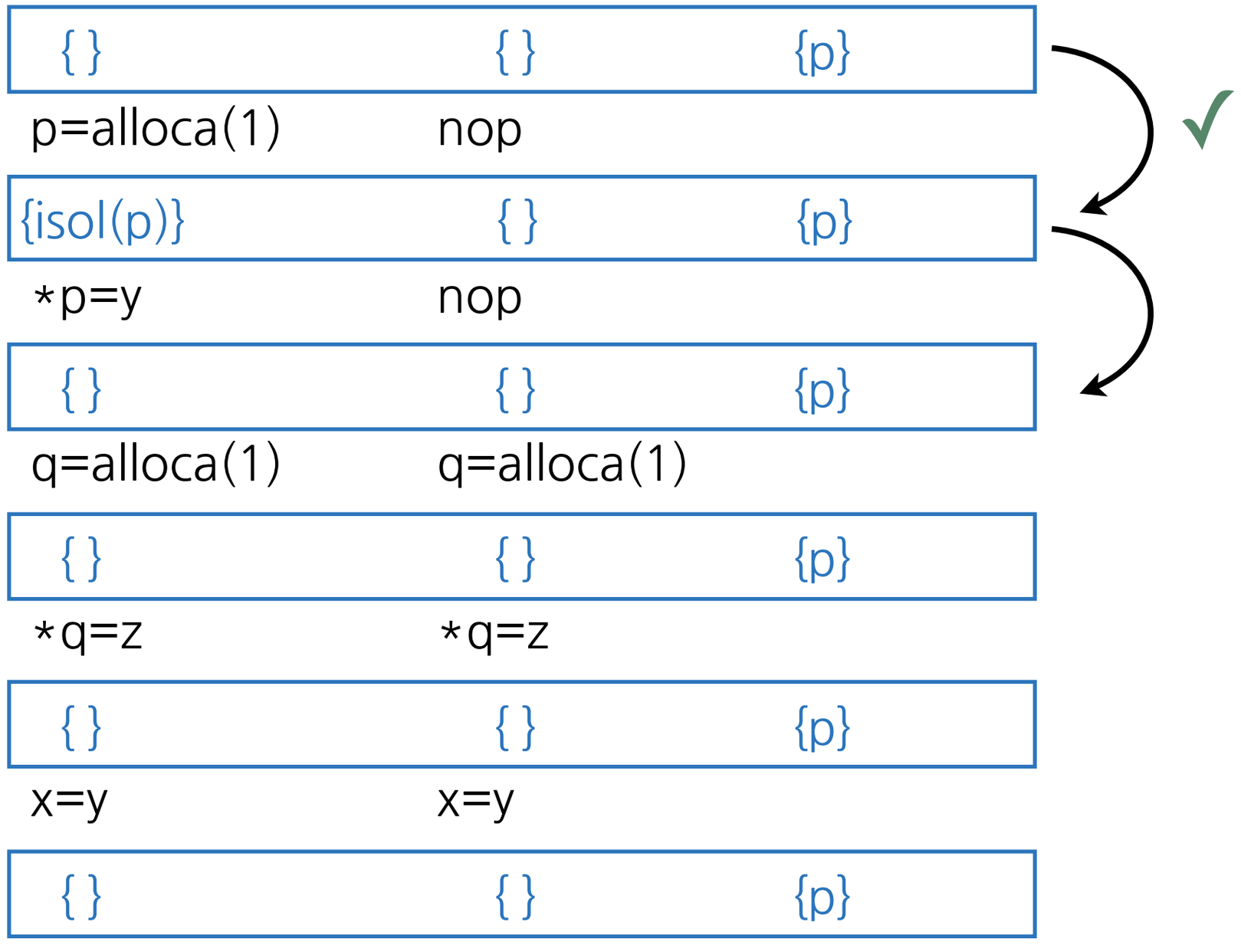
최적화 예 3: dead code elim



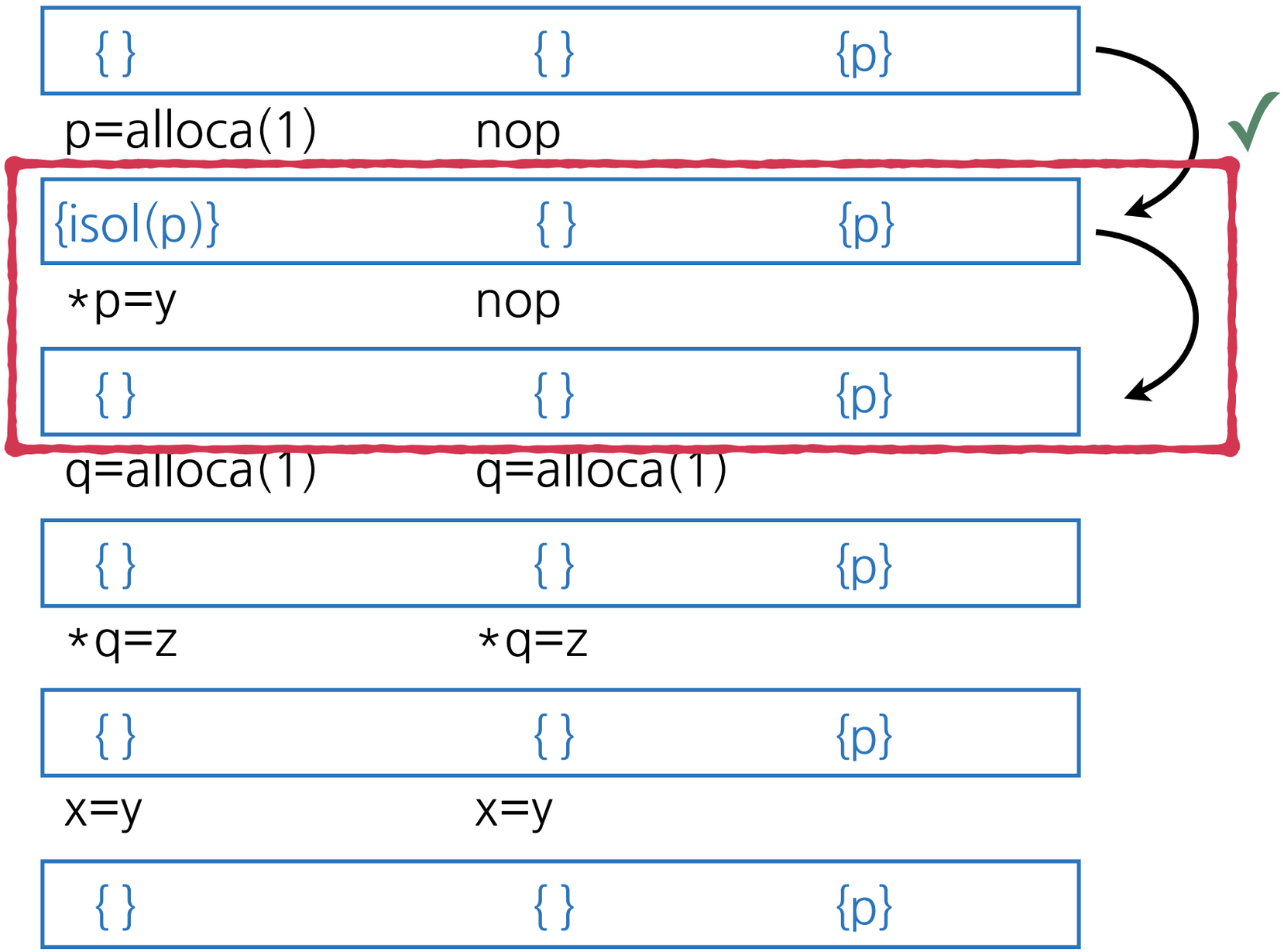
최적화 예 3: dead code elim



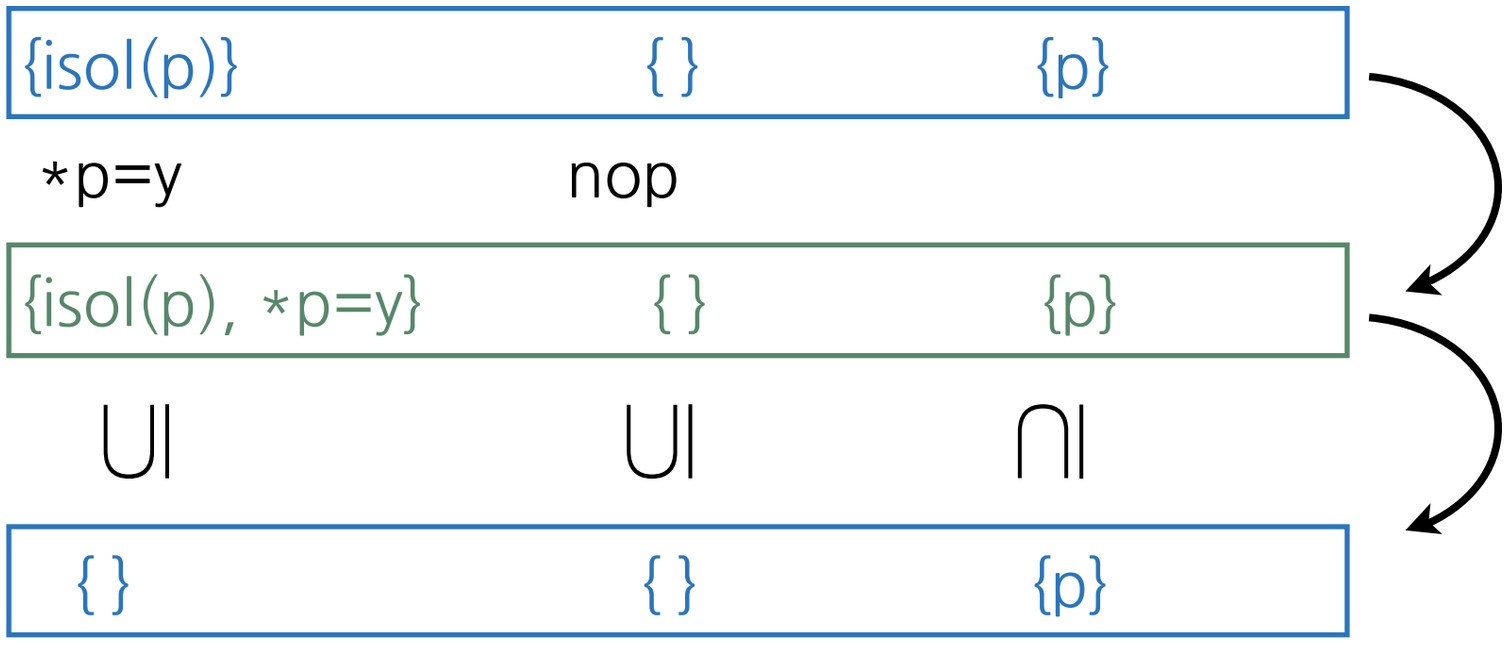
최적화 예 3: dead code elim



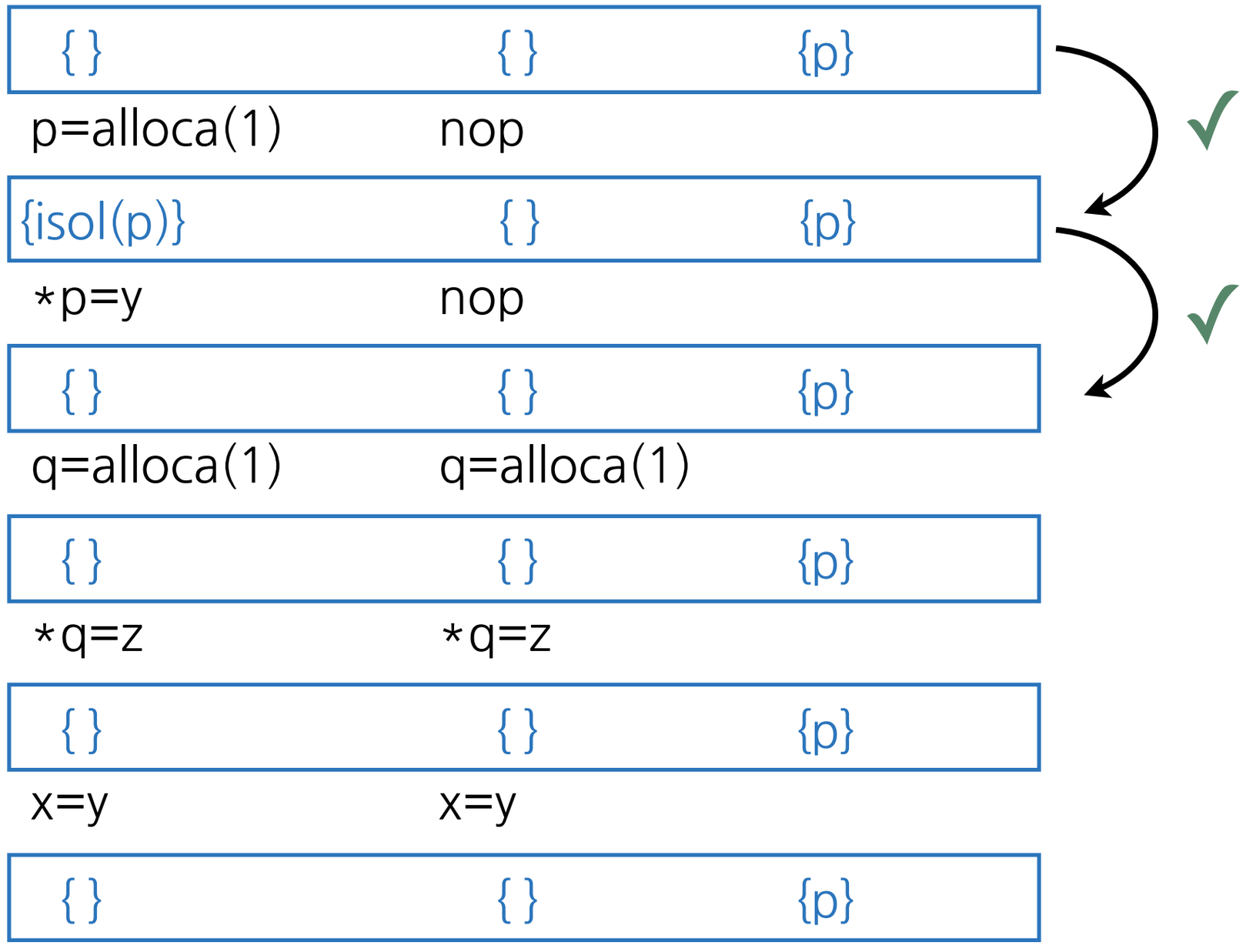
최적화 예 3: dead code elim



최적화 예 3: dead code elim



최적화 예 3: dead code elim



실험 결과

- 검산 가능한 꼬마최적화: 50개 (전체 401개) / 3주
- 추론규칙의 올바름 증명: 32개 (현재까지 63개) / 2주
- 검산기의 올바름 증명:
  핵심이 되는 시뮬레이션 관계 증명
  (Vellvm*에서 정의한 실행의미 사용)

* J. Zhao, S. Nagarakatte, M. M. Martin, and S. Zdancewic. Formal Verification of SSA-Based Optimizations for LLVM. PLDI'13.

결론

LLVM컴파일러에
검증된 검산기를 달았는데
흡족하더라.

감사합니다.