겹쳐진 자료구조의 모양분석

(A Divide-And-Conquer Approach for Analysing Overlaid Data Structures)

이욱세 @ 한양대학교

26/07/2012

소프트웨어무결점연구센터 워크샵 (2012 여름)

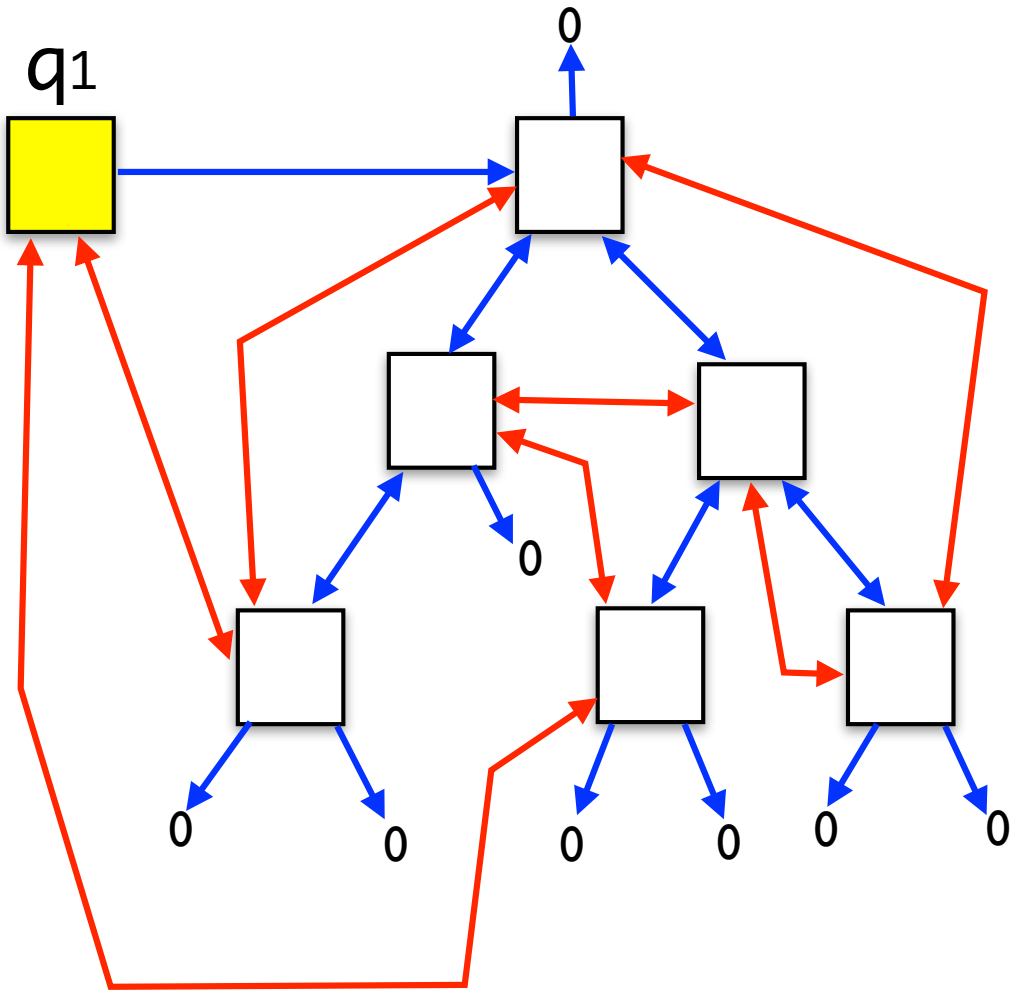
양홍석 (University of Oxford), Rasmus Petersen (Queen Mary University of London)와 협업

Formal Methods in System Design 41(1):4-24, August 2012

목표

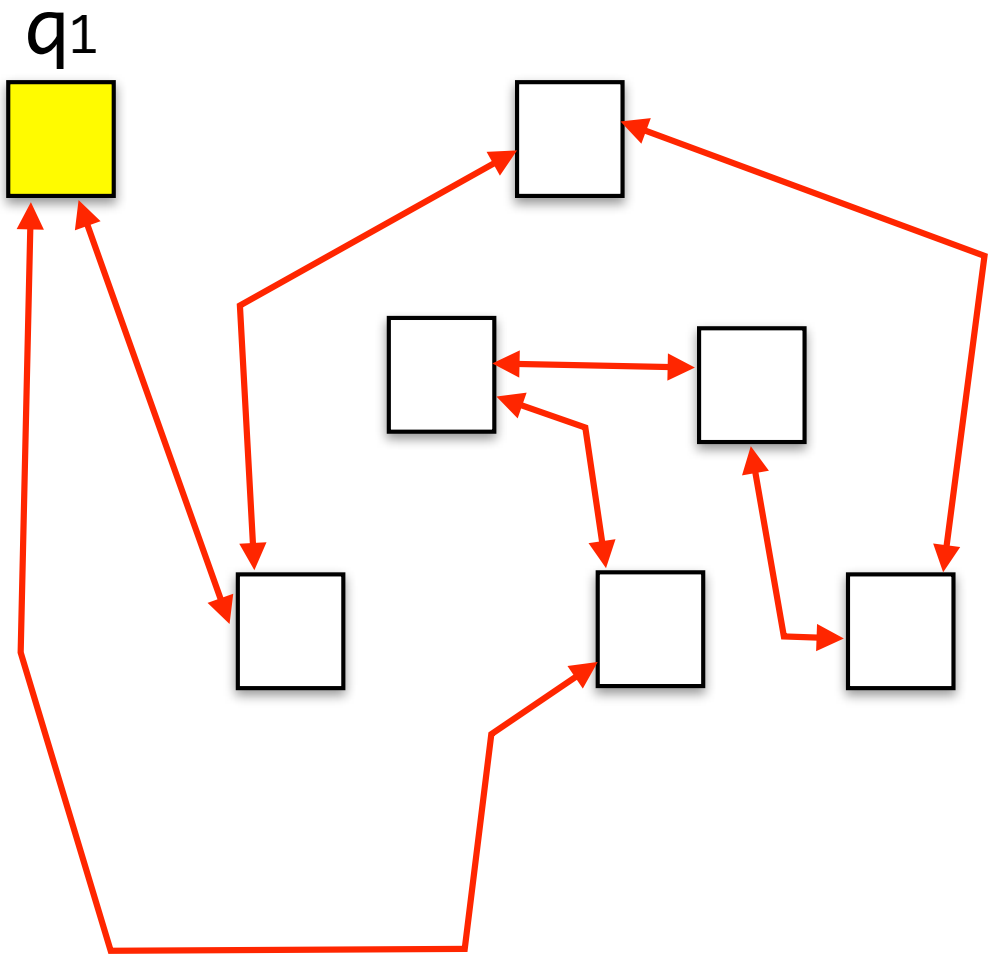
- 실제 시스템 코드의 힙에 관한 성질 자동 증명
  - 널 포인터 오류, 끊어진 포인터 오류, 메모리 누수
  - 자료 구조의 모양 유지
- 지난 6년간 괄목할만한 성과
- 남은 문제들:
  - 복잡한 자료구조 (highly-shared data structures)
  - 동시성
  - ...

문제: 겹쳐진 자료구조

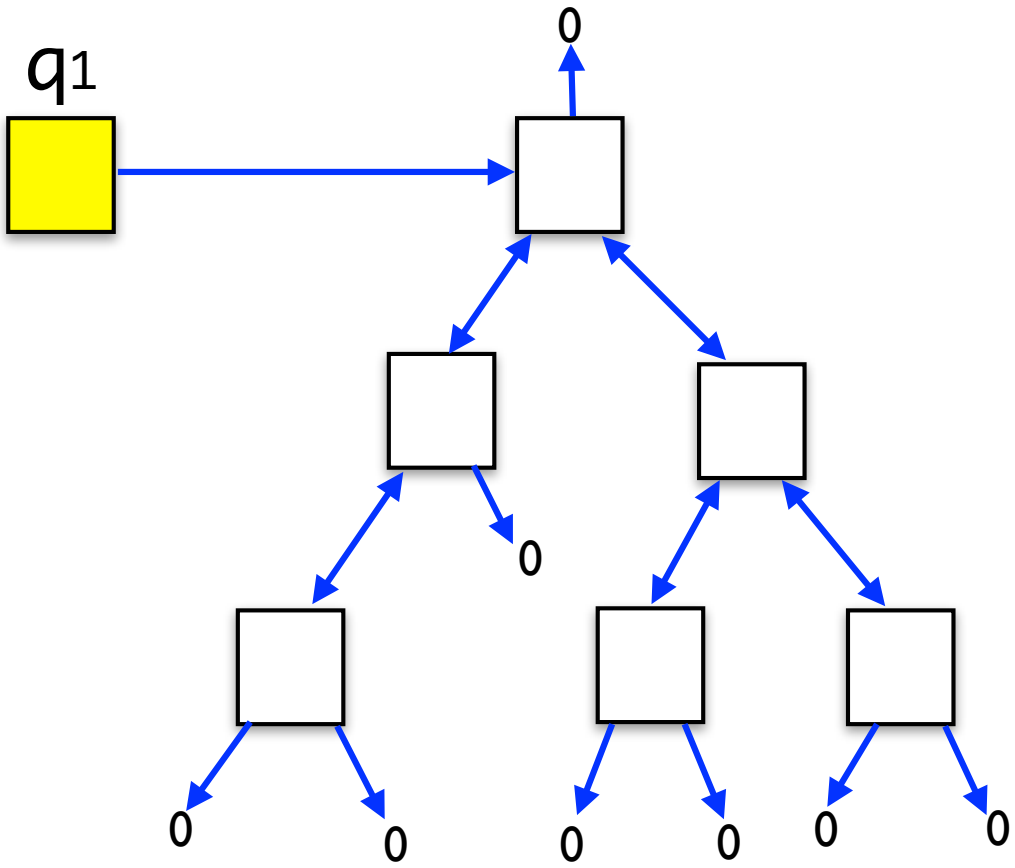


- 두 개의 자료구조가 겹쳐진 형태
- Linux와 같은 운영체제 코드에서 다수 발견

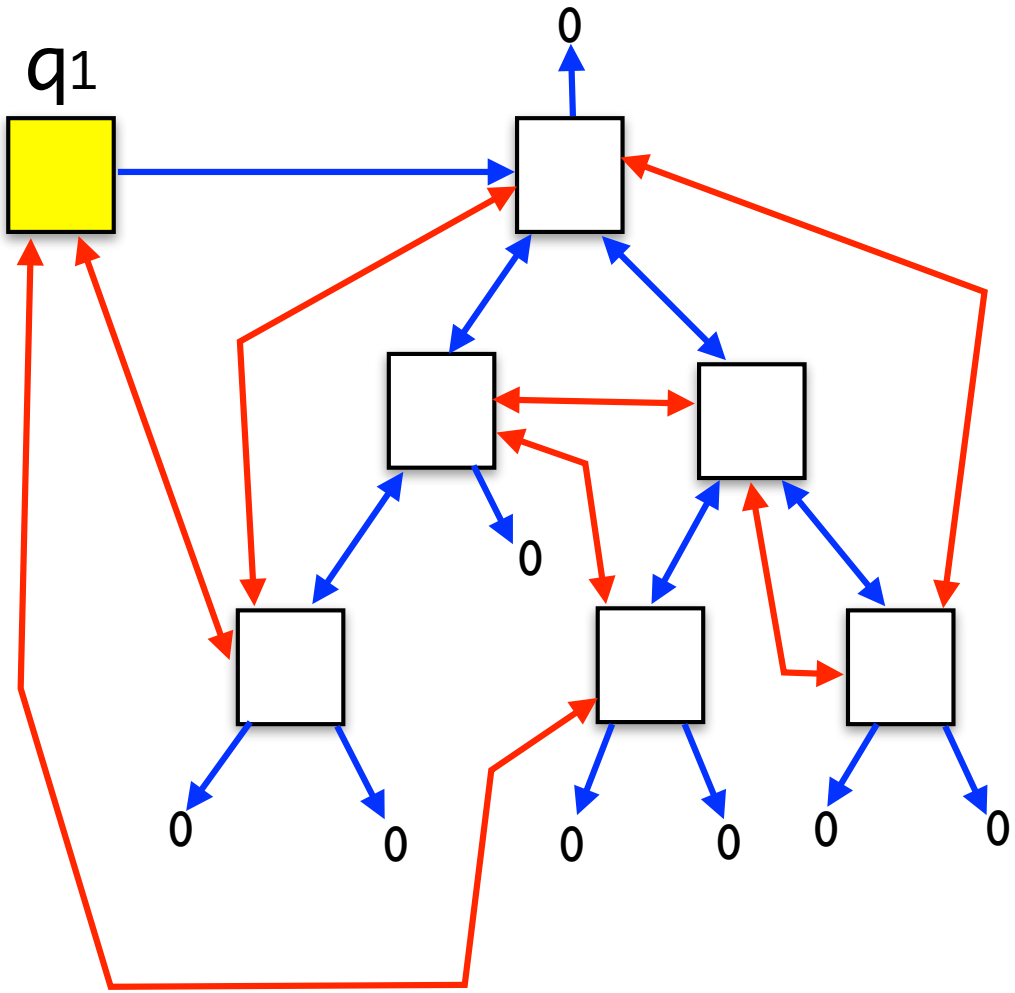
리스트 벅업



트리 벅업



사용 이유: 인덱스 구조 둘 이상



서로 상관없는 여러 개의 인덱스 구조 필요
(예, FIFO + 우선순위 큐)

아이디어

어떻게 증명하지?

- 논리곱 (conjunction) 사용

$$\wedge$$

  - 예, dll(q1) ∧ tree(q1)

- 그렇게 쉽게는 안될껄
  - 논리곱을 어디에 놓지?
  - 논리곱의 양쪽의 관계는 얼만큼 유지?

논리곱의 위치가 문제

- 모양 분석에서는 논리합을 사용
  - 예, $q=0 \vee dll(q)$
  - 도메인으로 표현하면 집합, 2^{SHeap}.
  - 논리합과 논리곱을 어떤 순서로 놓지?

- 자연스러운 (?) 방법: 논리곱을 논리합 안쪽에
  - 예, $(q=0 \wedge q=0) \vee (dll(q) \wedge tree(q))$
  - 도메인으로 표현하면, $2^{SHeap \times SHeap}$

자연스러운 방법은 성능 문제 유발

file	lines	note	analysis time (s)
dll-ip.c	134	a toy program that models the deadline IO scheduler with two DLLs	3.12
deadline-iosched-sim2.c	1,968	a part of real deadline IO scheduler	5,399.73

왜? 경우의 수 폭발

- x, y가 dll(q)∧tree(q)의 어떤 셀을 가리킨다면
  - N: dll(q)에 3 가지 경우
    - x=y, x 먼저, y먼저.
  - M: tree(q)에 5 가지 경우
    - x=y, x가 y의 아래 (좌/우), y가 x의 아래 (좌/우)
- 최종 경우의 수는 N×M: 15
  - 이상해서 제거되는 조합의 수는 많지 않다.

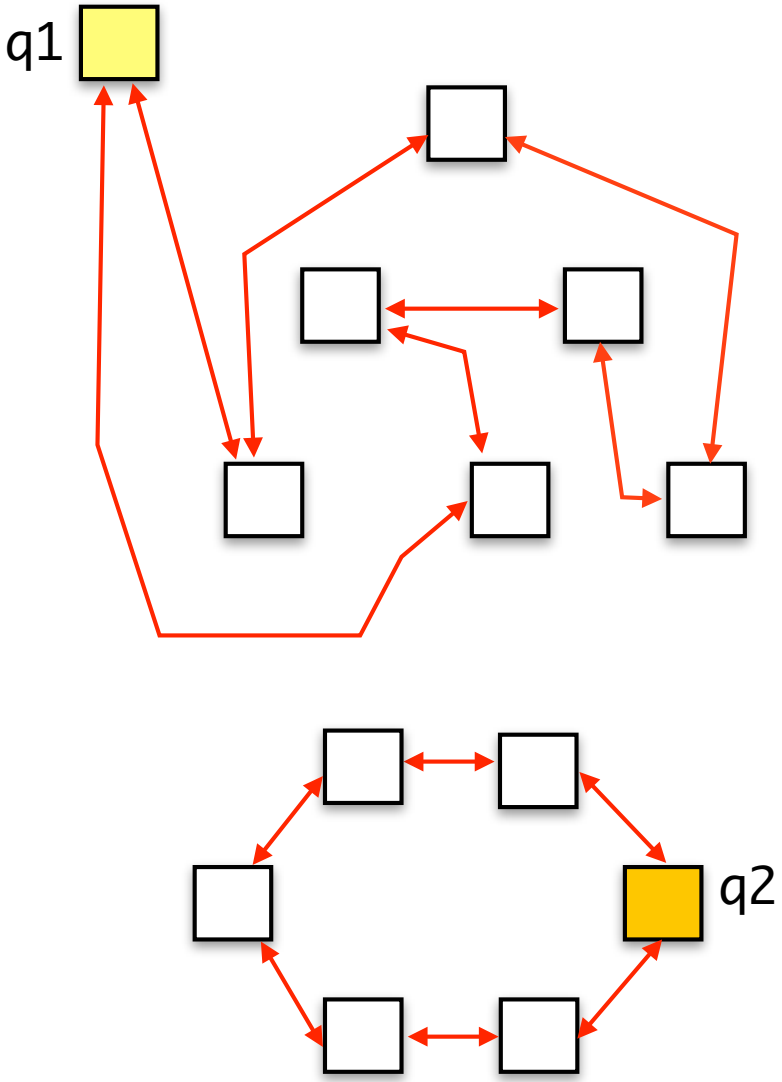
두 자료구조를 독립적으로 보자

- 논리곱을 논리합 바깥으로
  - 예, $(q=0 \vee dll(q)) \wedge (q=0 \vee tree(q))$
  - 도메인으로 표현하면, $2^{SHeap} \times 2^{SHeap}$.
- 경우의 수 폭발 없음
  - 3 for dll(q) + 5 for tree(q) = 8 (N+M)!
- 각각의 부분을 별도로 분석 가능
  - 하지만, 두 자료구조 간의 관계 유지 필요

어떤 관계가 필요하지?

```
t:=find_tree(q1)
del_list(t)
```

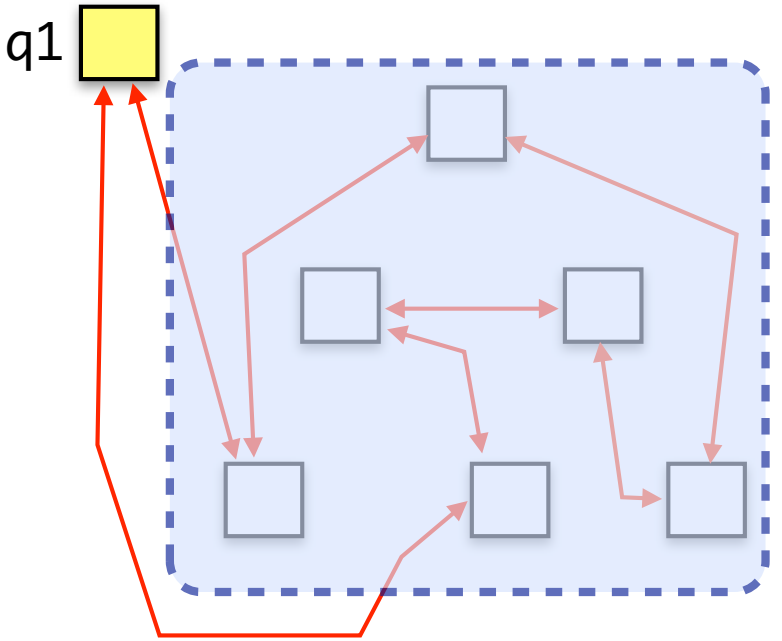
리스트 분석할 때 t 가 무엇을 가르키는지 알아야 한다!

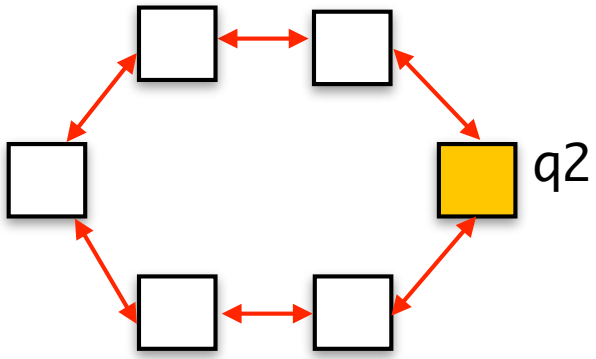


어떤 관계가 필요하지?

```
t:=find_tree(q1)
del_list(t)
```

t 는 점선 안의
임의의 셀을 가르킨다.



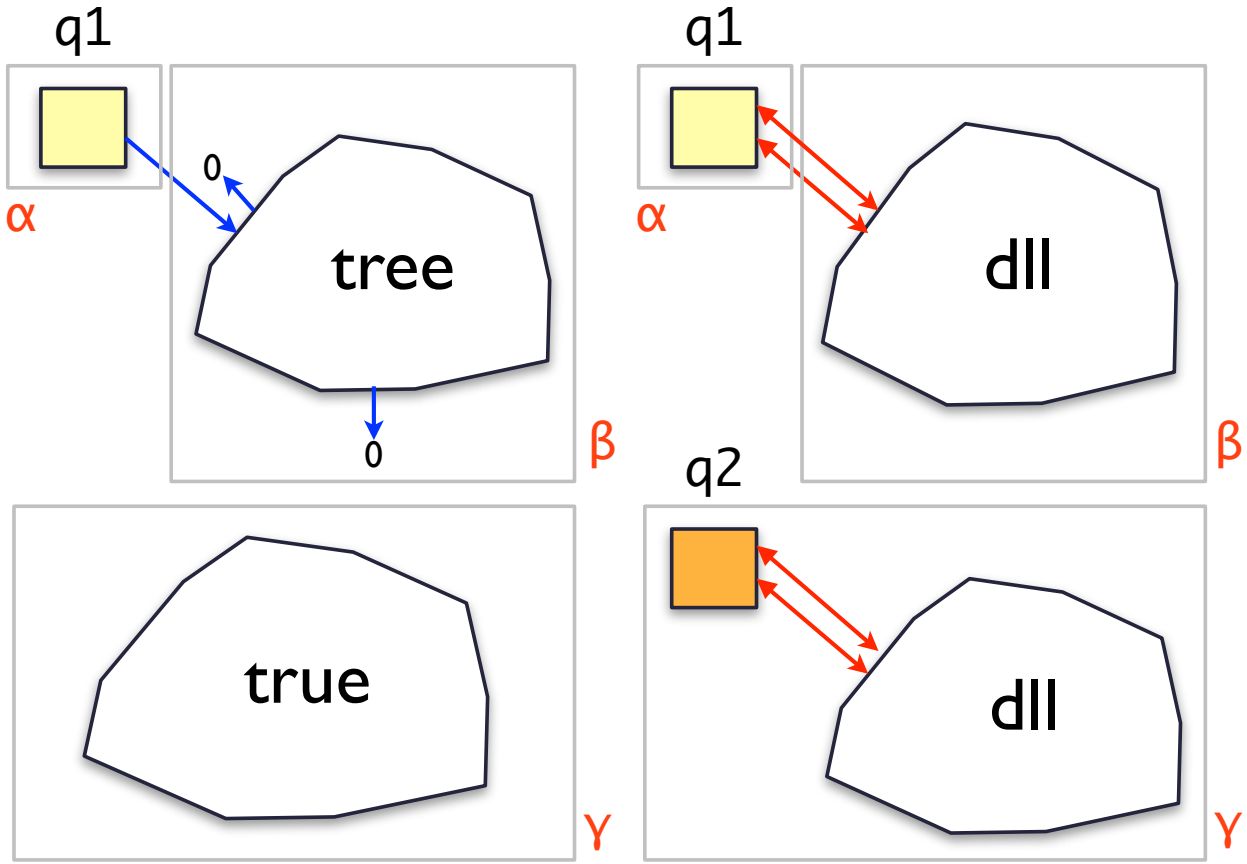


영역 변수 도입

- 영역은 겹치지 않는 메모리 주소의 집합
- 논리곱 간의 관계를 유지
- 실행 도중 영역의 의미 변경 가능

$(q1 \mapsto a\alpha * dll(a,q1)\beta * q2 \mapsto b\gamma * dll(b,q2)\gamma) \wedge$

$(q1 \mapsto a\alpha * tree(a)\beta * true\gamma)$



구현 & 결과

분석을 위한 명령어 삽입

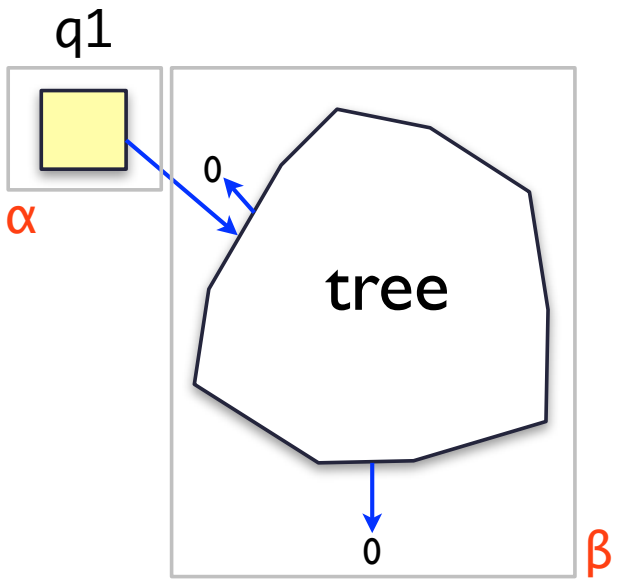
- 분석기 간의 소통을 위해
  - $trans_{A \to B}(x)$ 는 x의 영역정보를 분석기 A에서 B로 전달
- 영역 관리를 위해
  - move(t,α)는 t 셀을 α 영역으로 이동
    - 실제로는 새 영역 α 생성에 사용
  - moveRgn(α,β)는 α 영역의 모든 셀을 β로 이동
    - 실제로는 두 영역을 합치는데 사용
- 모양 분석 이전에 사전 분석을 통해 자동 삽입

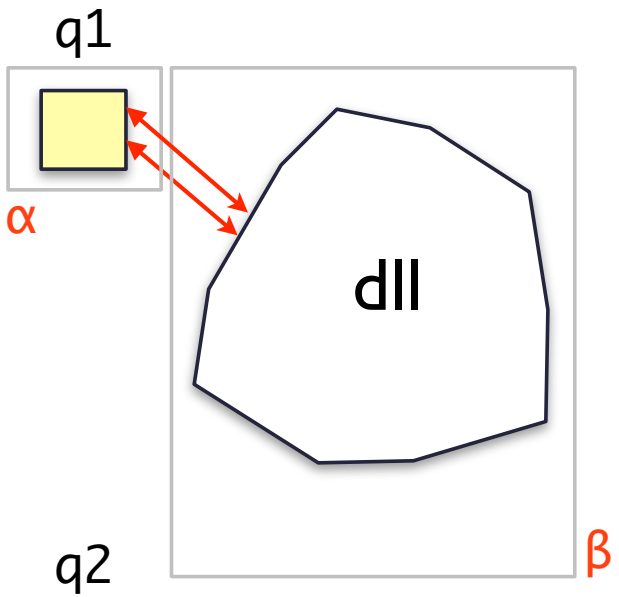
예제1: moveRequest

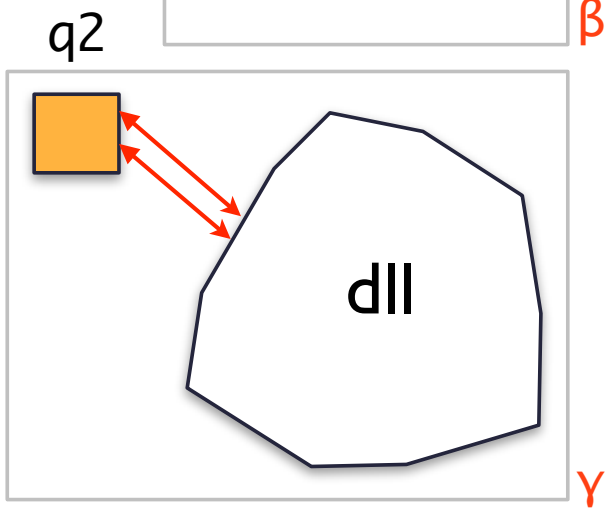
```
t:=find_tree(q1)


del_list(t)
del_tree(t)
add_list(q2,t)

abstract(t)
```

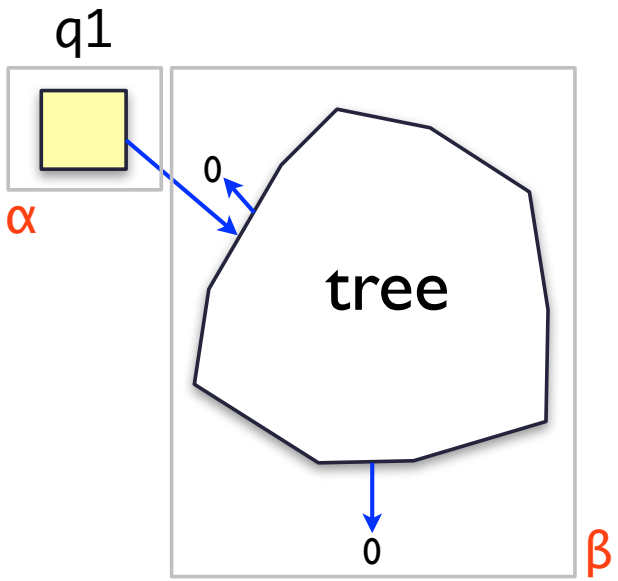


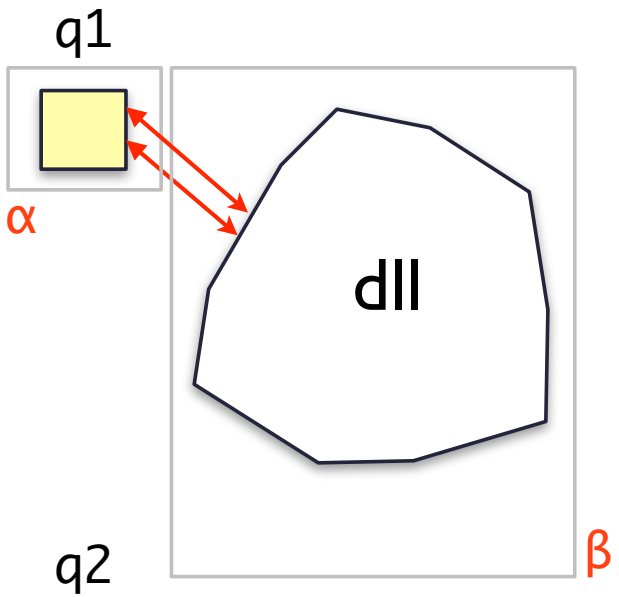


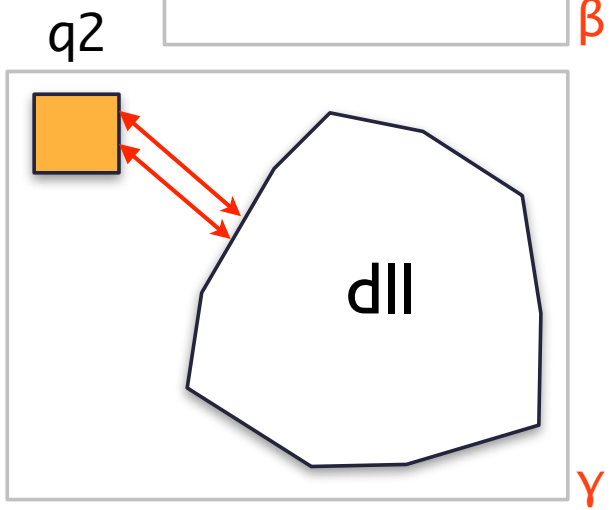


예시

```
t:=find_tree(q1)
trans_tree→list(t)
move(t,δ)
del_list(t)
del_tree(t)
add_list(q2,t)
moveRgn(δ,γ)
abstract(t)
```

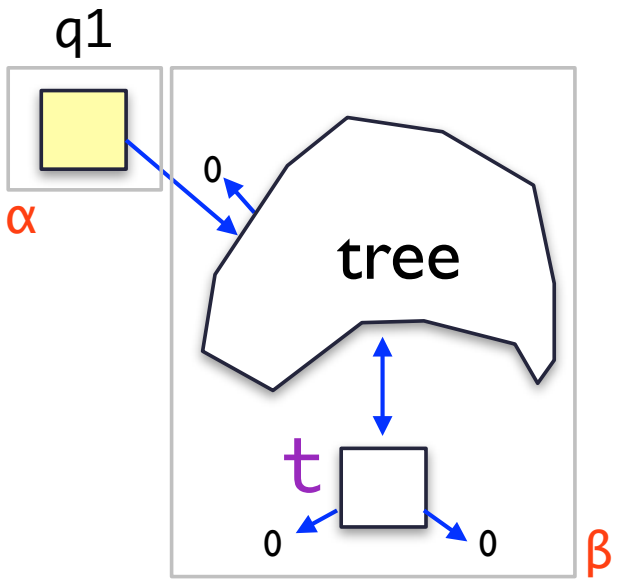


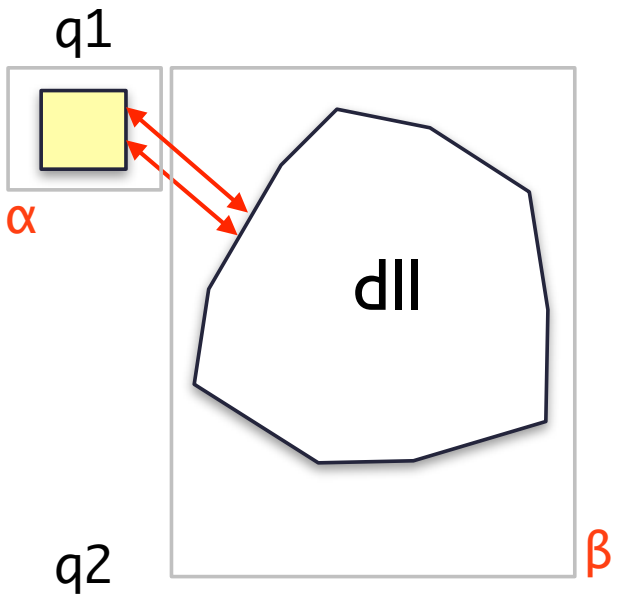


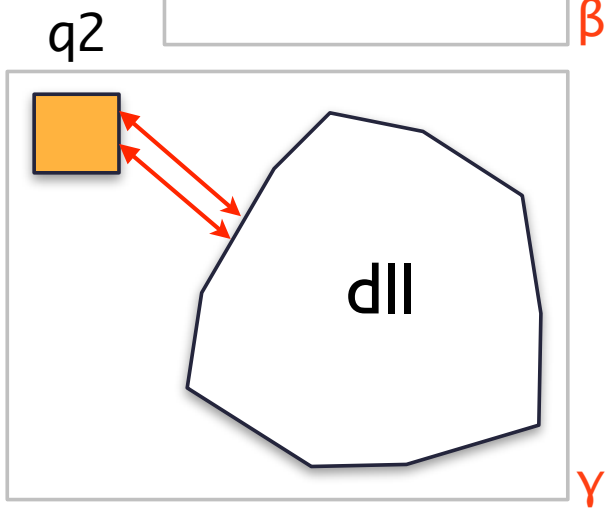


예시

```
t:=find_tree(q1)
trans_tree→list(t)
move(t,δ)
del_list(t)
del_tree(t)
add_list(q2,t)
moveRgn(δ,γ)
abstract(t)
```

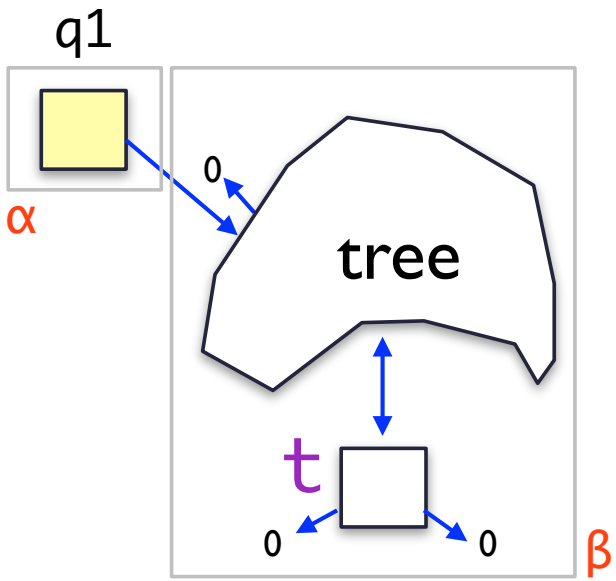


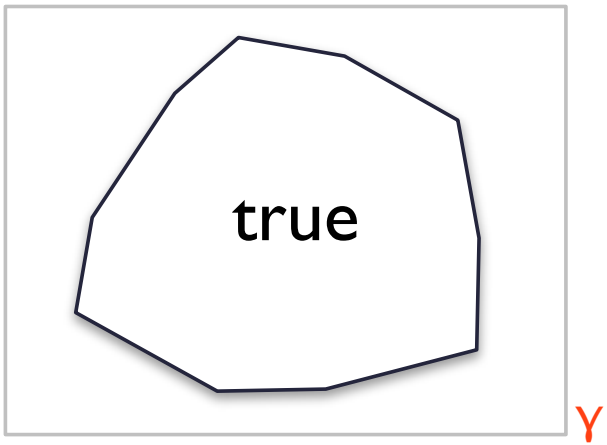


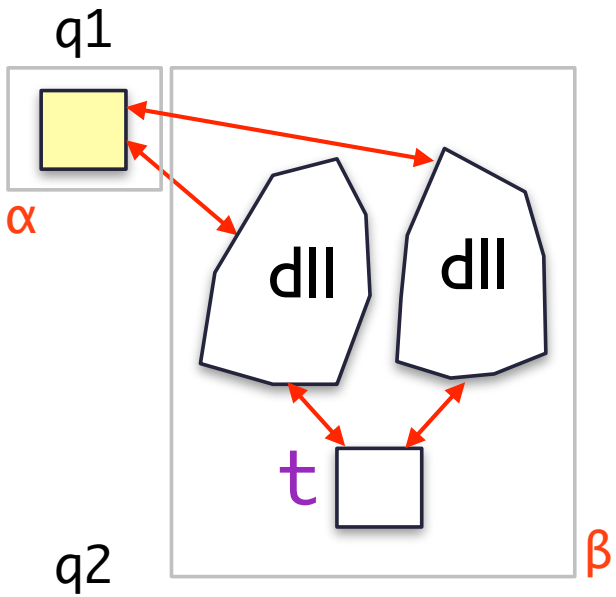


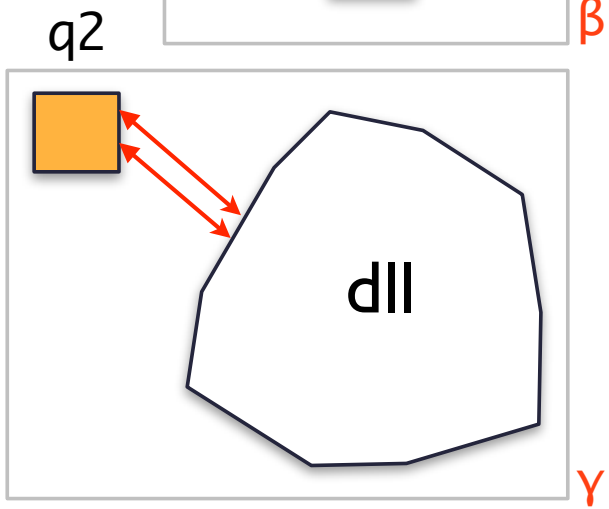
예시

```
t:=find_tree(q1)
trans_tree→list(t)
move(t,δ)
del_list(t)
del_tree(t)
add_list(q2,t)
moveRgn(δ,γ)
abstract(t)
```



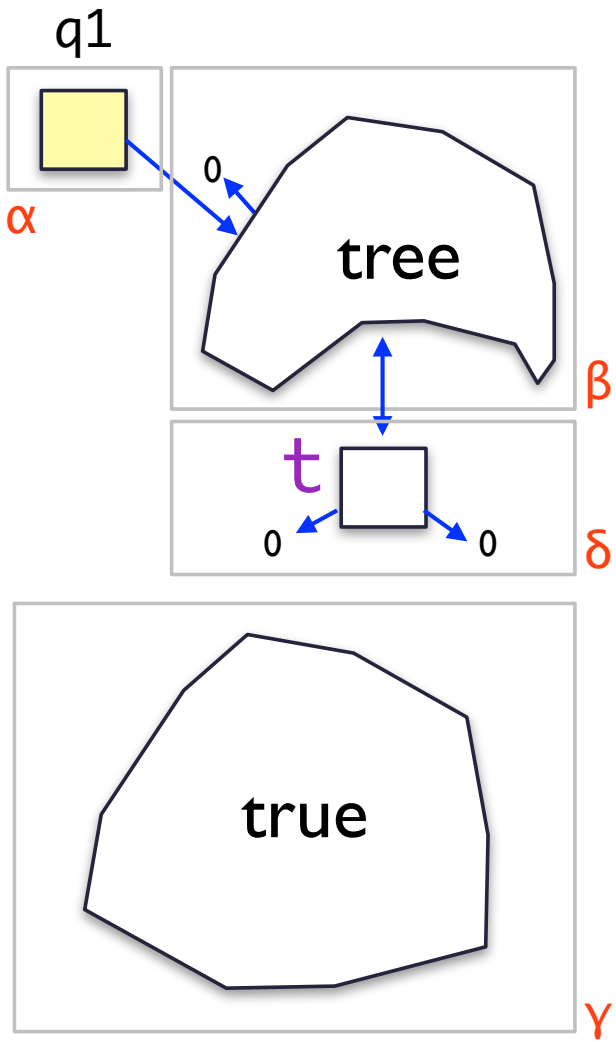


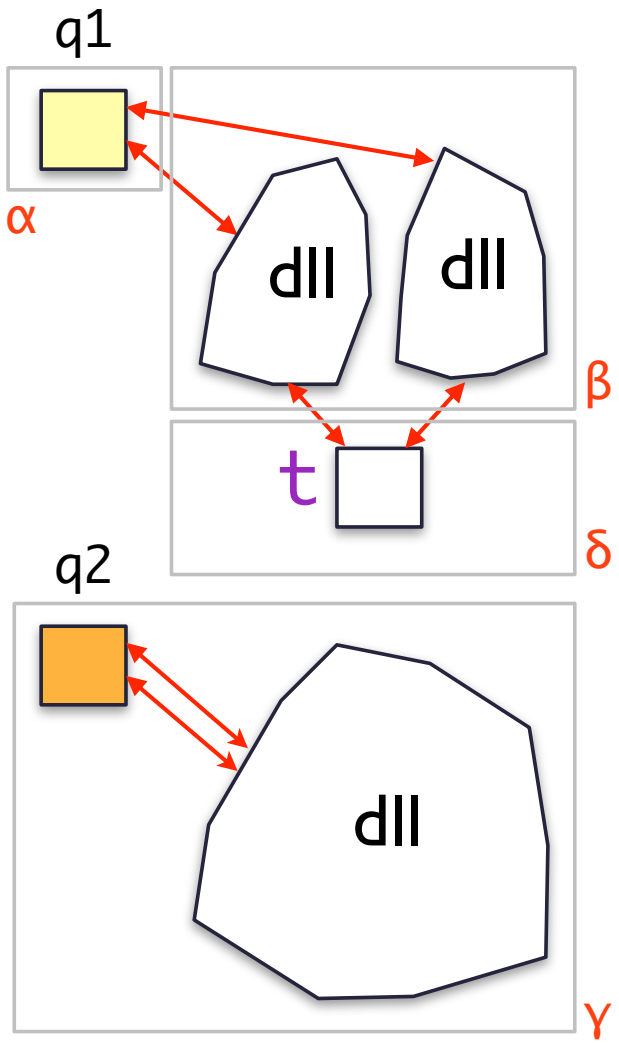




예시

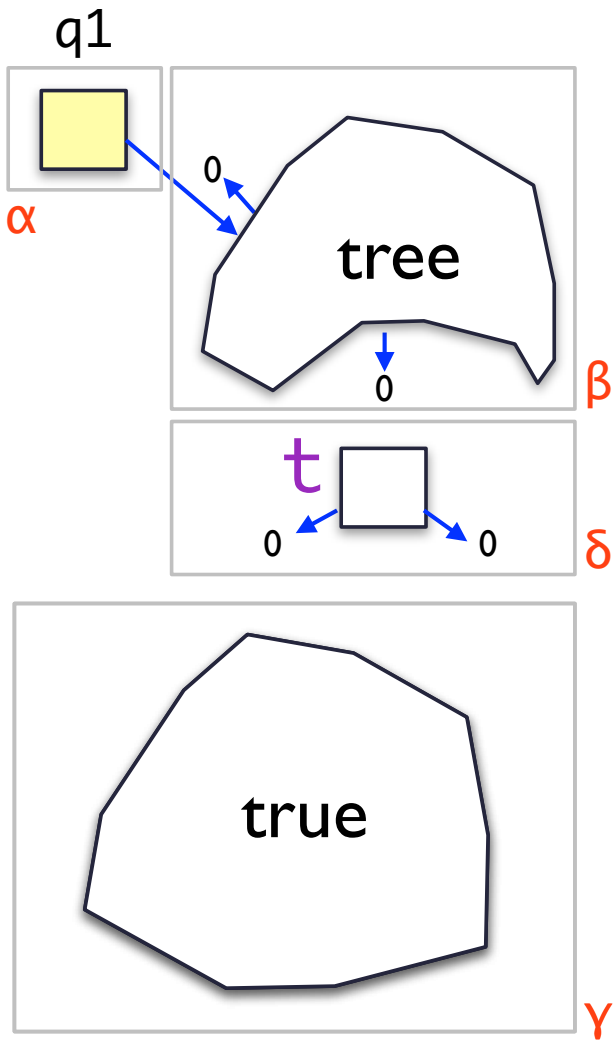
```
t:=find_tree(q1)
trans_tree→list(t)
move(t,δ)
del_list(t)
del_tree(t)
add_list(q2,t)
moveRgn(δ,γ)
abstract(t)
```

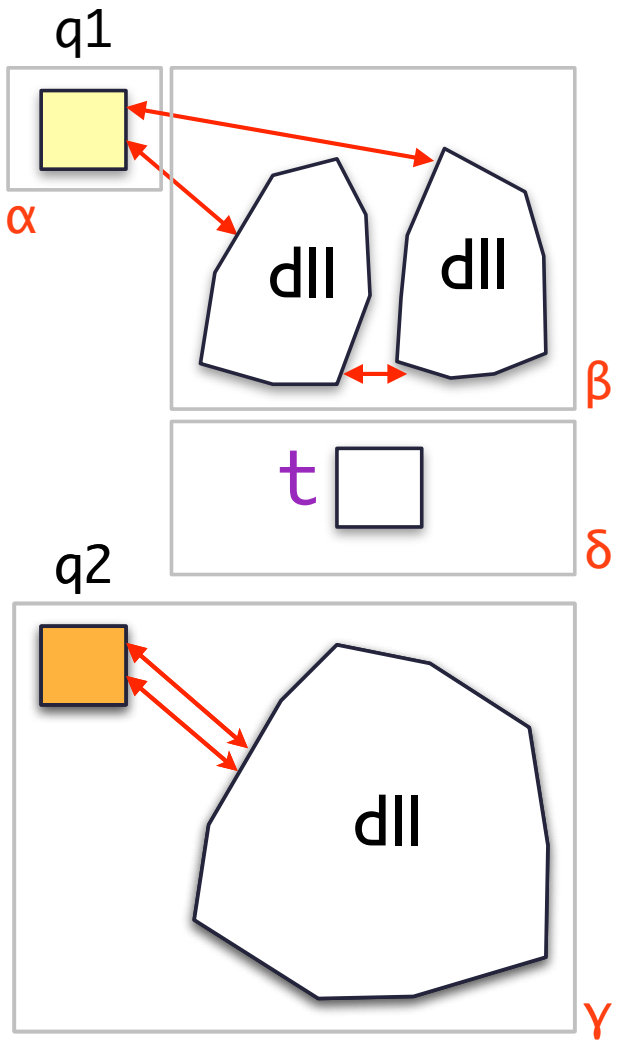




예시

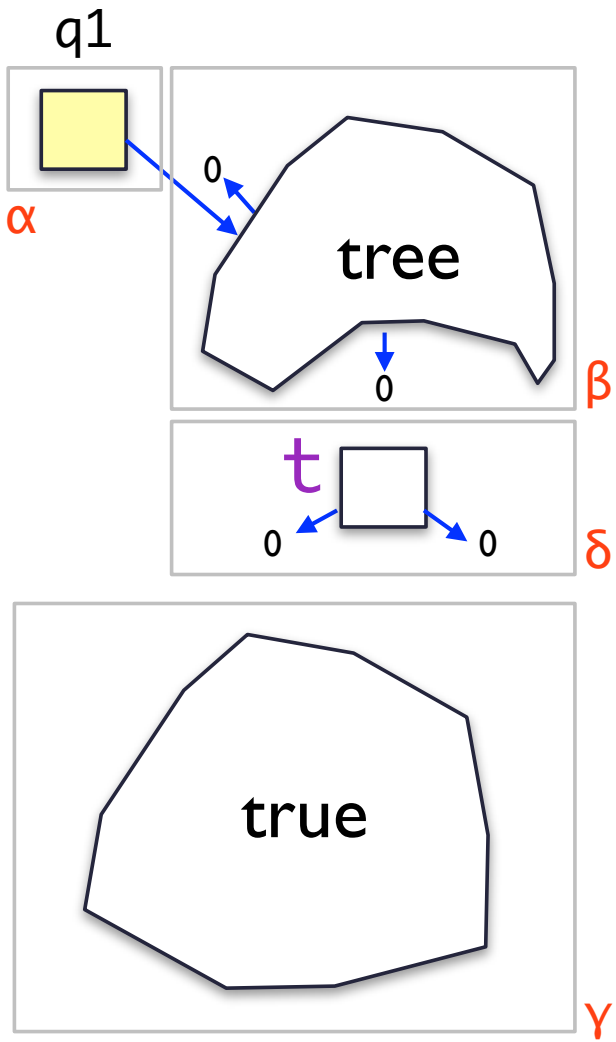
```
t:=find_tree(q1)
trans_tree→list(t)
move(t,δ)
del_list(t)
del_tree(t)
add_list(q2,t)
moveRgn(δ,γ)
abstract(t)
```

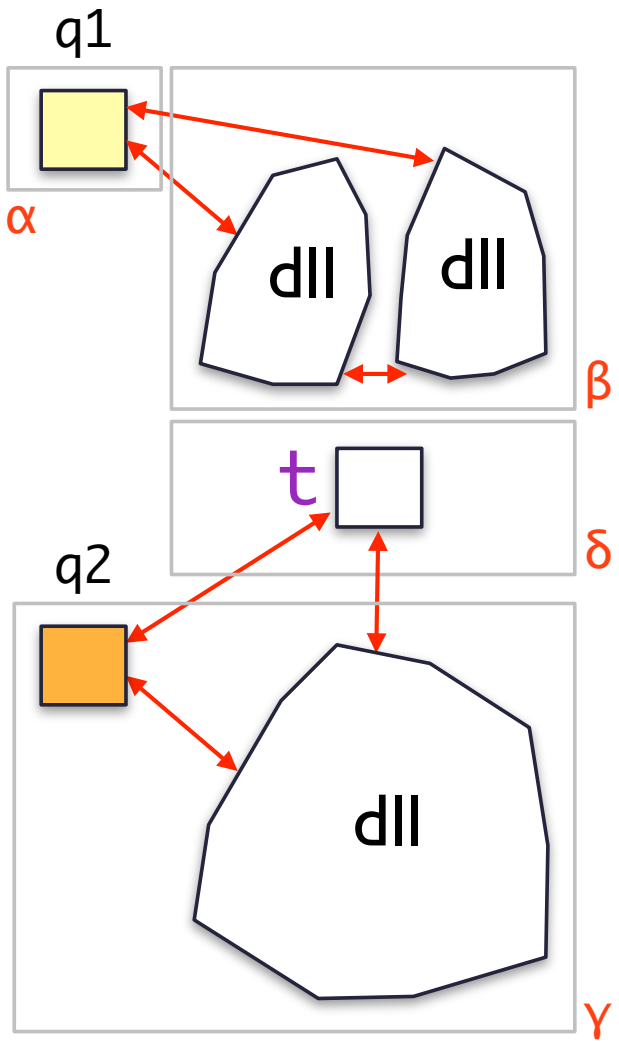




예시

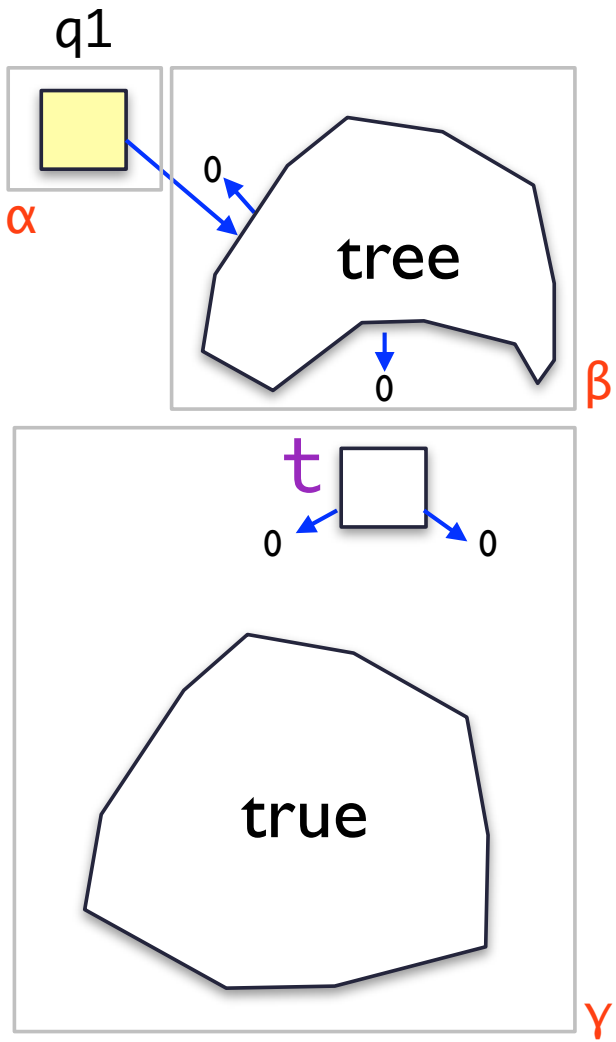
```
t:=find_tree(q1)
trans_tree→list(t)
move(t,δ)
del_list(t)
del_tree(t)
add_list(q2,t)
moveRgn(δ,γ)
abstract(t)
```

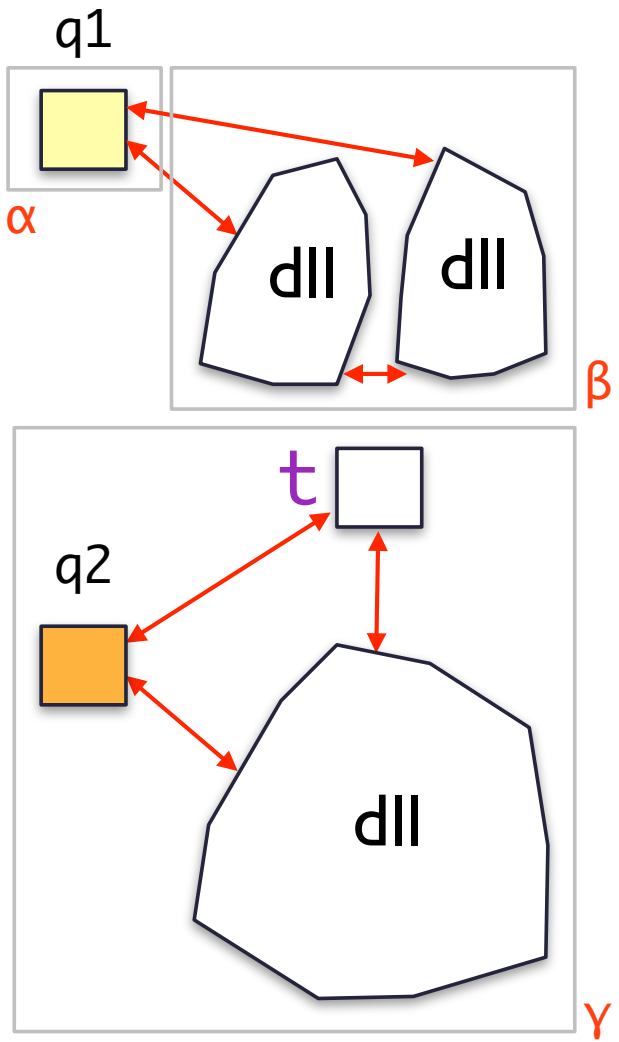




예시

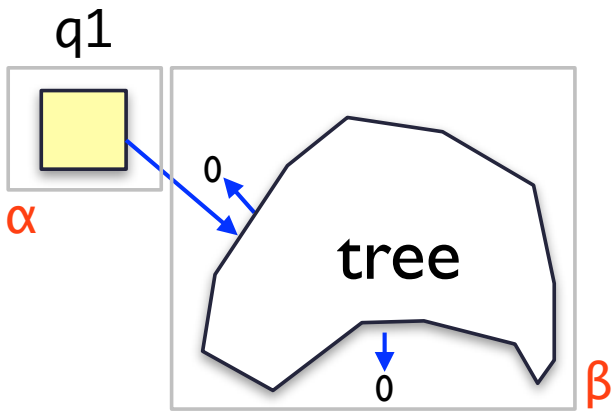
```
t:=find_tree(q1)
trans_tree→list(t)
move(t,δ)
del_list(t)
del_tree(t)
add_list(q2,t)
moveRgn(δ,γ)
abstract(t)
```

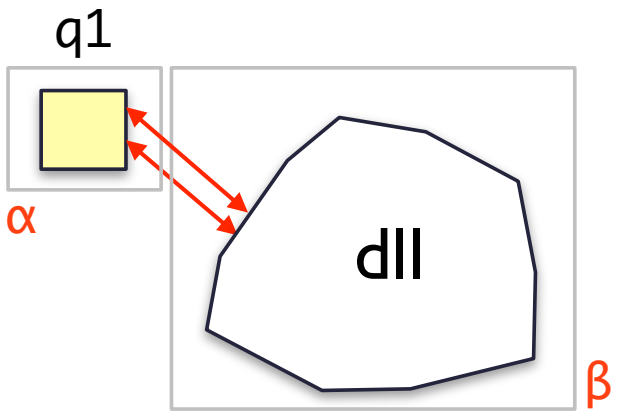


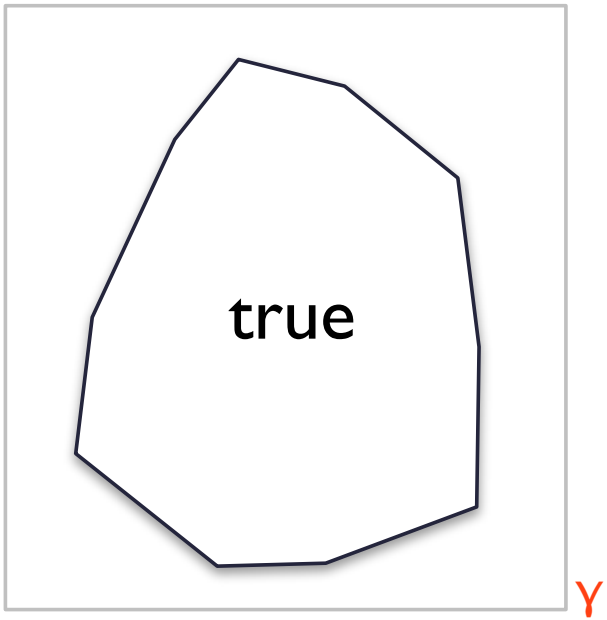


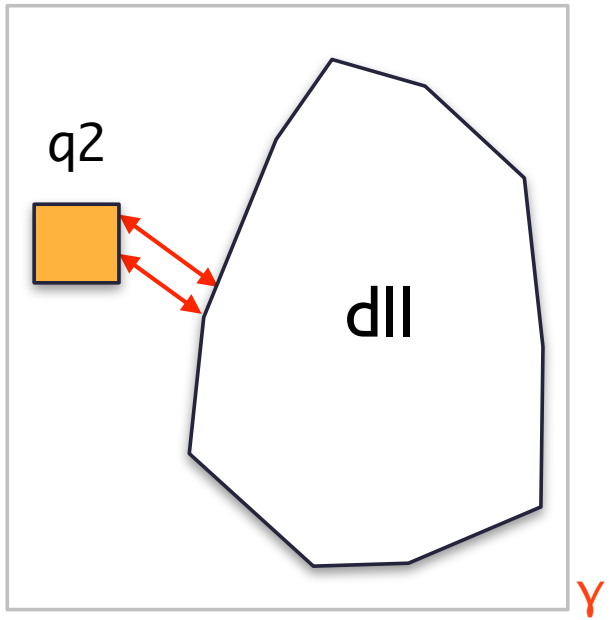
예시

```
t:=find_tree(q1)
trans_tree→list(t)
move(t,δ)
del_list(t)
del_tree(t)
add_list(q2,t)
moveRgn(δ,γ)
abstract(t)
```









실험 결과

file	lines	note	analysis time (s)	
			naive	ours
dll-ip.c	134	a toy program that models the deadline IO scheduler with two DLLs	3.12	1.56
deadline-iosched-sim2.c	1,968	a part of real deadline IO scheduler	5,399.73	100.06

자세한 구현

trans 명령어 삽입

- 각 변수를 각 분석기가 알고 있는지 추적
  - 순방향 프로시저 간 자료흐름 분석을 통해
  - 예, x:=y->left 직후, 트리 분석기는 x를 알고 리스트 분석기는 모른다.

- $trans_{A \to B}(x)$ 삽입
  - 분석기 B는 x를 모르고 A는 알 때
  - 가능한 뒤에다 삽입

- 안다는 것, 모른다는 것이, 정확하지 않다!

따로 실행 의미구조

- 프로그램이 각 부분 별로 기본적으로 독립적으로 수행

$$[\![c]\!]_{\mathsf{CE}} : (\mathsf{Stack} \times \mathsf{Heap})^k \to (\mathsf{Stack} \times \mathsf{Heap})^k$$

- 변수가 미정의 되어 있을 수 있고, 힙에는 관련 필드만 유지

$$[\![x := y\,.\,\mathtt{f}]\!]_i(s,h) = \begin{cases} [\![x := y\,.\,\mathtt{f}]\!](s,h) & \text{if } \mathtt{f} \in \mathcal{F}_i \\ (\exists x.s, h) & \text{if } \mathtt{f} \notin \mathcal{F}_i \end{cases}$$

- 원래 의미구조와는 무슨 관계?

스택 간 값 전달이 있으면 원래와 같은 실행

- 스택 간 필요한 값이 전달되면 원래 의미구조와 동일하게 수행가능

$$[\![c]\!]_{\mathsf{CE}}((s_1,h_1),\ldots,(s_k,h_k)) =$$
$$\quad \text{let } s'_i = s_i \wedge \mathsf{reveal}(i,c,\textstyle\bigwedge_{1\le j\le n} s_j) \text{ for all } 1 \le i \le n$$
$$\quad [\![c]\!]_1(s'_1,h_1) \times \cdots \times [\![c]\!]_n(s'_n,h_n)$$
$$\text{where}$$
$$\quad \mathsf{reveal}(i,c,s) = \textstyle\bigwedge\{x = s(x) \mid (i,x) \in \mathsf{need}(c)\}$$

- 스택 간 값 전달이 필요한 곳에 명령어를 삽입하면 목적 달성!

trans 삽입 위한 자료흐름 분석

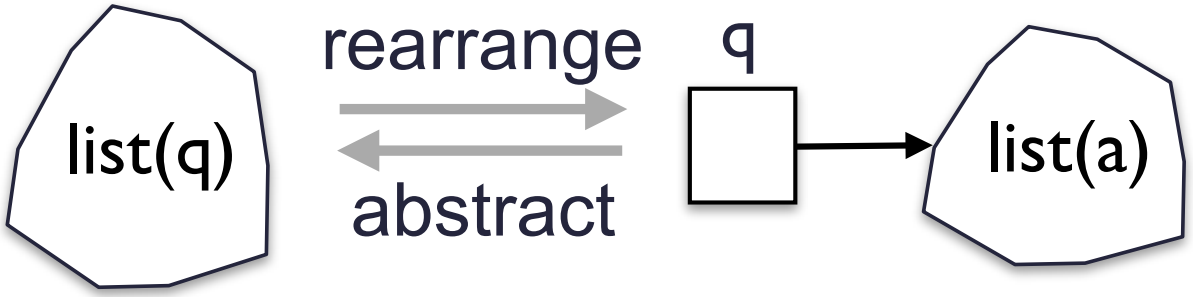
- $(i, x) \in \mathsf{Avail}(v)$
  - 프로그램 지점 v에서
  - i번 째 스택에
  - x가 **반드시** 정의되어 있다.

- $\mathtt{trans}_{i \to j}(x)$는 다음과 같은 v 직전에 삽입

  $(i, x) \in \mathsf{need}(v)$
  $(i, x) \notin \mathsf{Avail}(v)$
  $(j, x) \in \mathsf{Avail}(v)$

영역 관리 명령어 삽입

- 영역 관리는 어떻게 해도 안전, 하지만 필요한 만큼 나누어 주어야 정확도 향상
- 모양 분석의 꺼내기 (rearrange), 집어넣기 (abstract)에서 영역관리 필요



- move(x,α) 삽입: x가 셀을 가르키는 것이 확실할 때
- moveRgn(α,β) 삽입: 모양 분석에서 집어넣기가 수행될 때

move 삽입 위한 자료흐름 분석

- $(i, x) \in \mathsf{Alloc}(v)$
  - 프로그램 지점 v에서
  - i번 째 스택에
  - x가 반드시 메모리 주소이다.

- move(x, a)는 다음과 같은 (v, v')직후에 삽입

$(i, x) \notin \mathsf{Alloc}(v)$ for some i
$(j, x) \in \mathsf{Alloc}(v')$ for all $1 \leq j \leq n$

예, 명령어들이 삽입된 이유

```
t:=find_tree(q1)
trans_tree→list(t)
move(t,δ)
del_list(t)
del_tree(t)
add_list(q2,t)
moveRgn(δ,γ)
```

t가 트리 쪽에서는 정의되어 있는데 리스트 쪽에서는 미정의

t가 트리와 리스트 쪽에서 셀을 가리킨다

리스트 쪽 분석에서 t의 셀이 q2 리스트와 합체

결론

- 복잡한 자료구조를 나누어서 분석 가능
- 영역과 분석을 위한 추가 명령어 삽입을 통한 분석이 성공적이었고 원하는 분석 성능을 보여 주었음