정적분석 진행율 예측하기 (A Progress Bar for Static Analyzer)

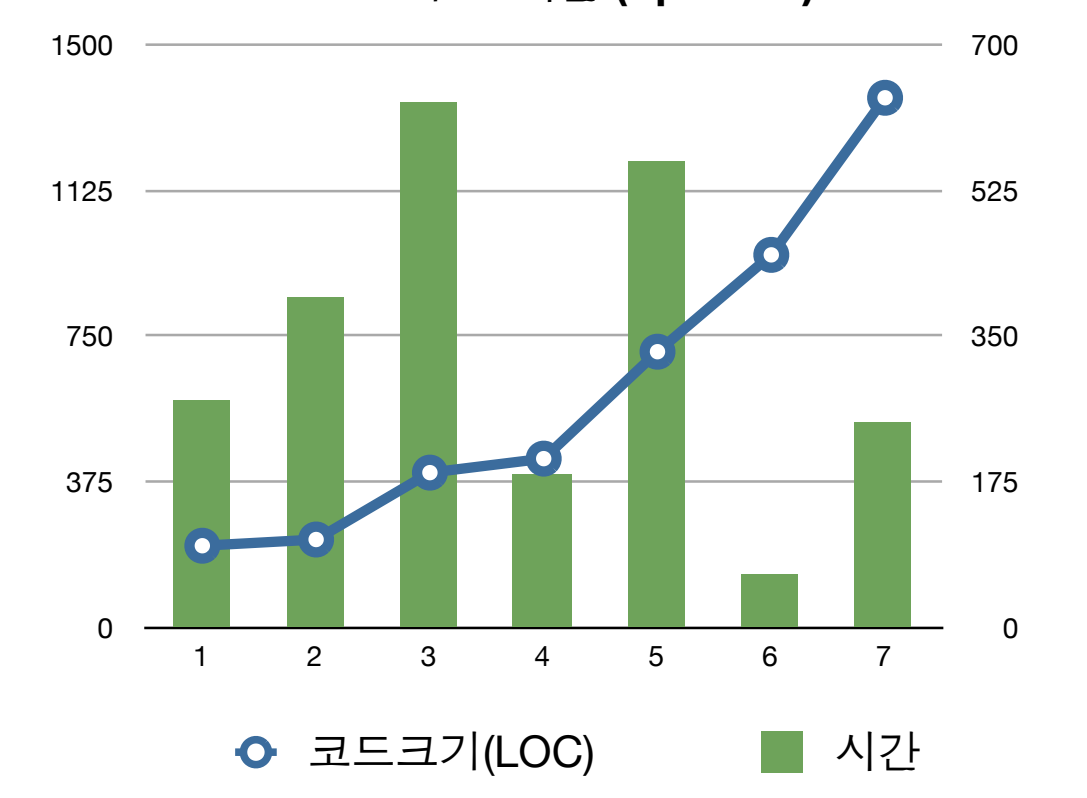
이우석, 오학주, 이광근
서울대학교 프로그래밍 연구실 (ROPAS)

※ 본 연구결과는 Static Analysis Symposium (SAS) 2014 에 게제승인되었습니다.

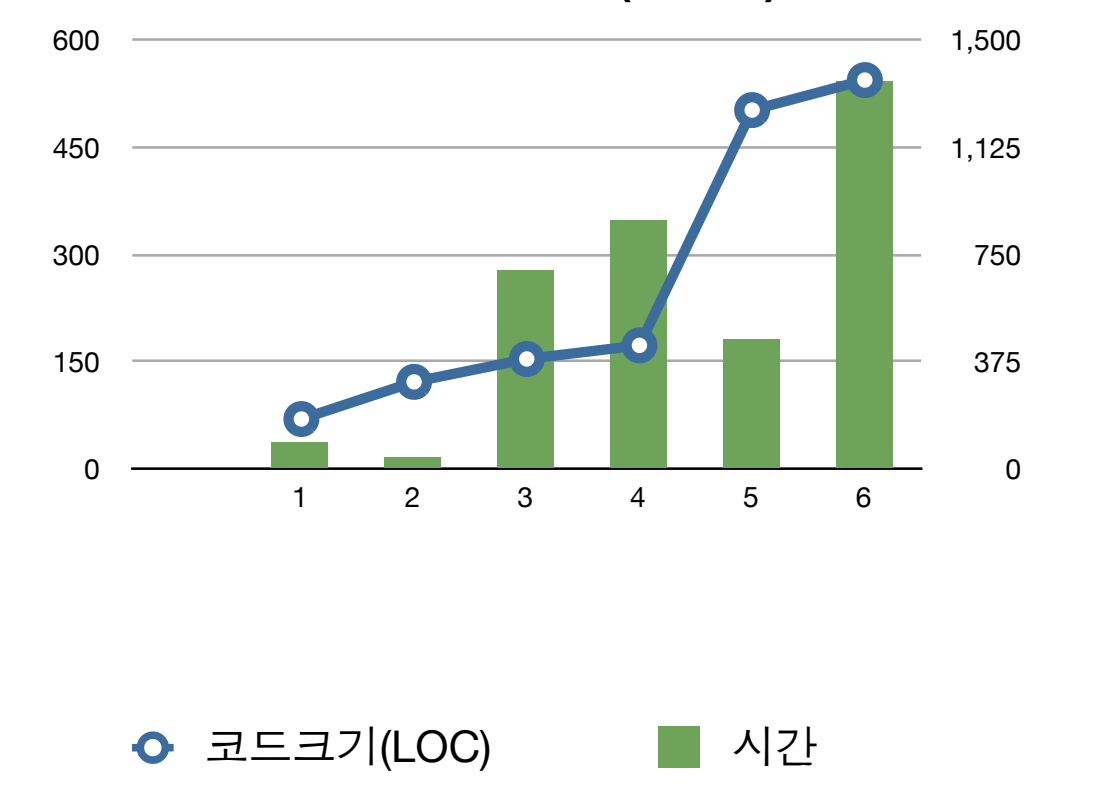
문제

- 크고 복잡해지는 소프트웨어, 오래걸리는 정적 분석 시간
  - Sparrow (SNU) - 40만줄 코드 10시간
  - Astrée (ENS) - 78만줄 코드 32시간
  - CGS (NASA) - 55만줄 코드 20시간
- 사용자가 진행율을 알 수 없는 문제
- 코드 크기보다 의미적 복잡도 - 정확한 진행율을 알기 어려운 이유

코드크기 vs 시간 (Sparrow)



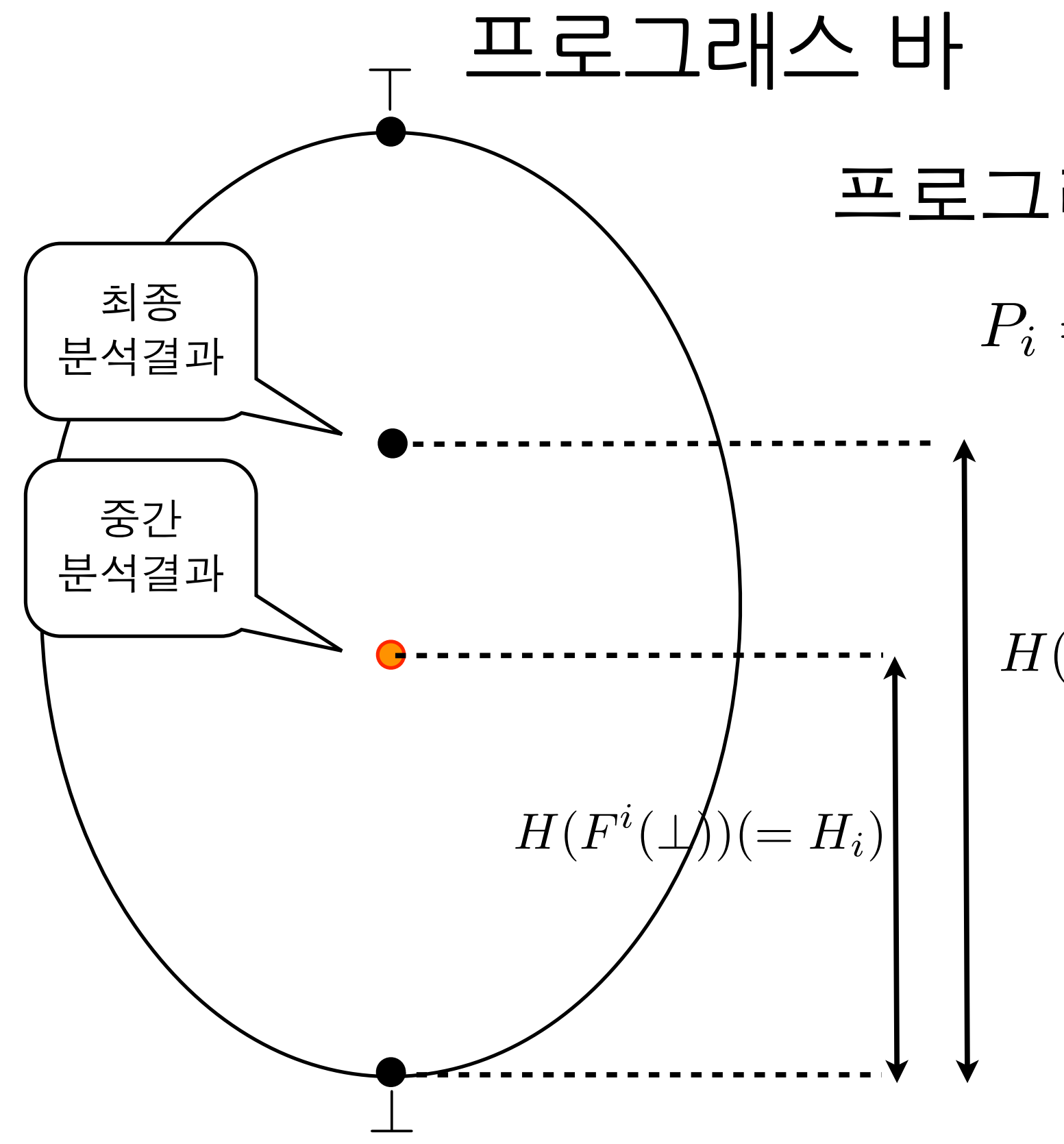
코드크기 vs 시간 (Astree)



우리의 방법

- 본 분석과 비슷한 사전 분석 + 기계학습
- 분석 진행율 예측에 대한 첫번째 연구
- 요약해석 기반 분석기에 일반적으로 적용가능

래티스 높이에 기반한 프로그래스 바



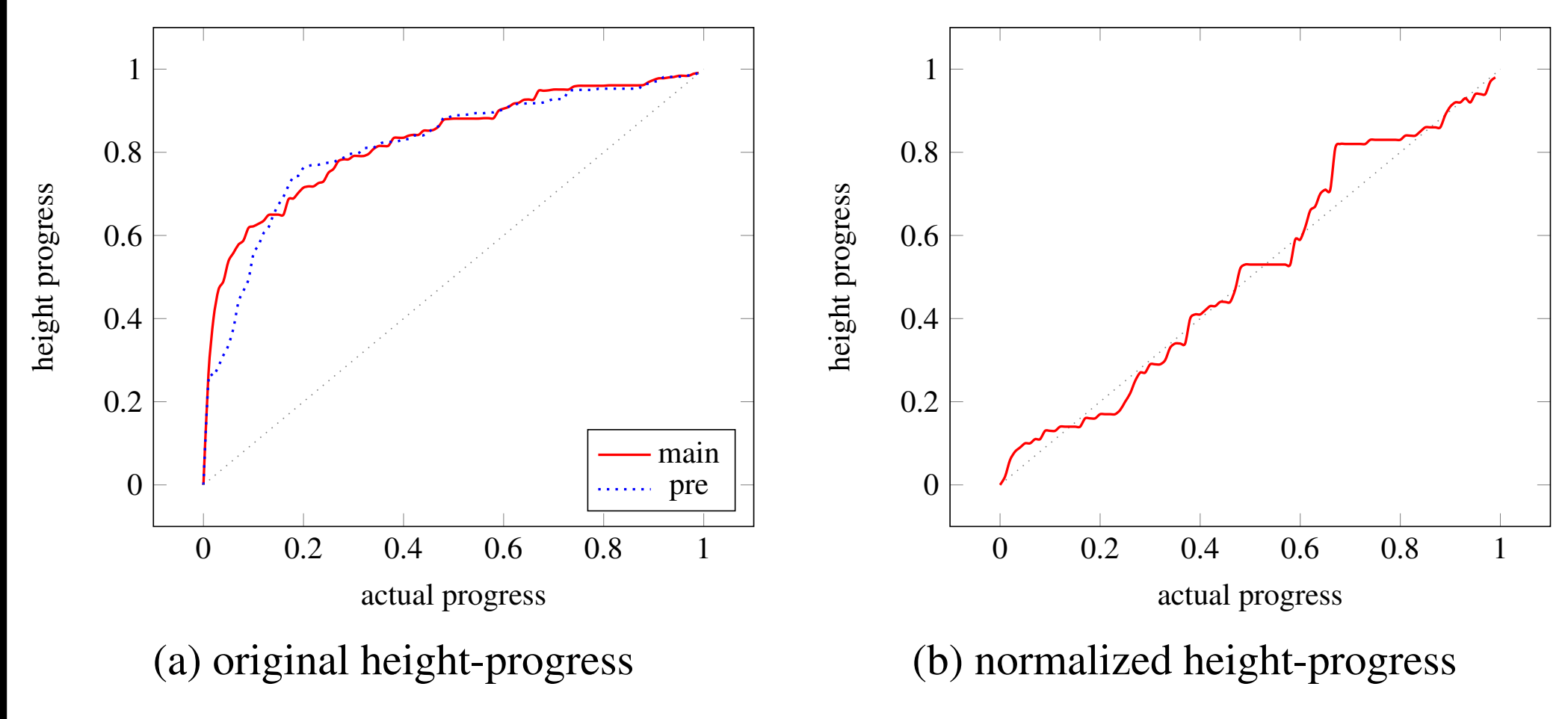
프로그래스 추정치 :

$$P_i = \frac{H_i}{H_{final}} \in [0,1]$$

- **문제 1** ; 프로그래스 바가 시간이 지남에 따라 일정하게 증가할까?
- **문제 2** : 최종 분석결과의 높이는 어떻게 구할까?

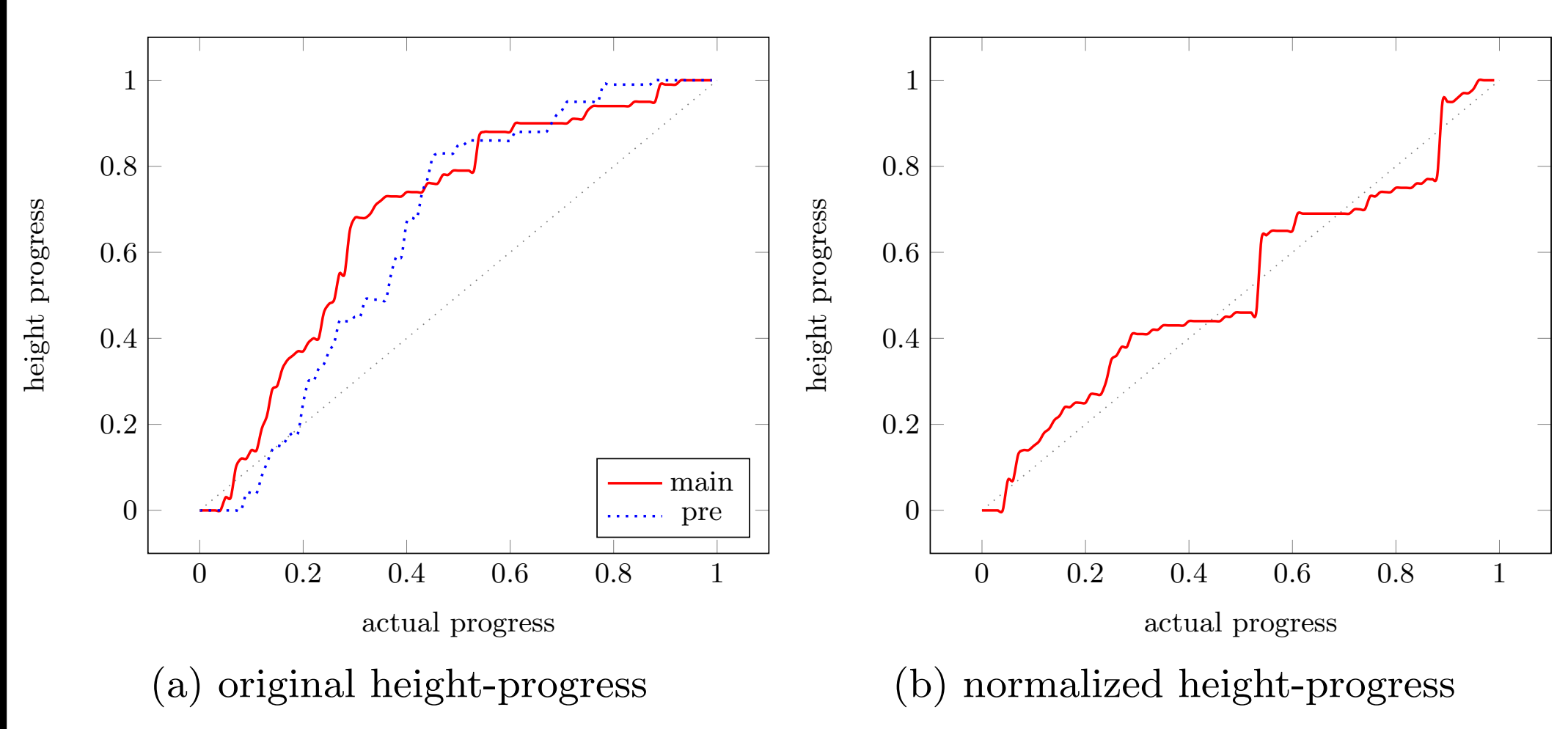
방법

본 분석보다 부정확하지만 비용이 더 저렴한 사전분석 이용



(a) original height-progress (b) normalized height-progress

sendmail-8.14.6 (interval analysis)



(a) original height-progress (b) normalized height-progress

wget-1.9 (octagon analysis)

- 1단계 : 본 분석보다 더 요약된 (부정확한) 사전분석을 설계

$$\mathbb{D} \underset{\alpha}{\overset{\gamma}{\leftrightarrows}} \mathbb{D}^\sharp$$

$$F^\sharp : \mathbb{D}^\sharp \to \mathbb{D}^\sharp \qquad \alpha \circ F \sqsubseteq F^\sharp \circ \alpha$$

$$\bigsqcup_{i\in\mathbb{N}} {F^\sharp}^i(\perp^\sharp) = {F^\sharp}^0(\perp^\sharp) \sqcup {F^\sharp}^1(\perp^\sharp) \sqcup {F^\sharp}^2(\perp^\sharp) \sqcup \cdots$$

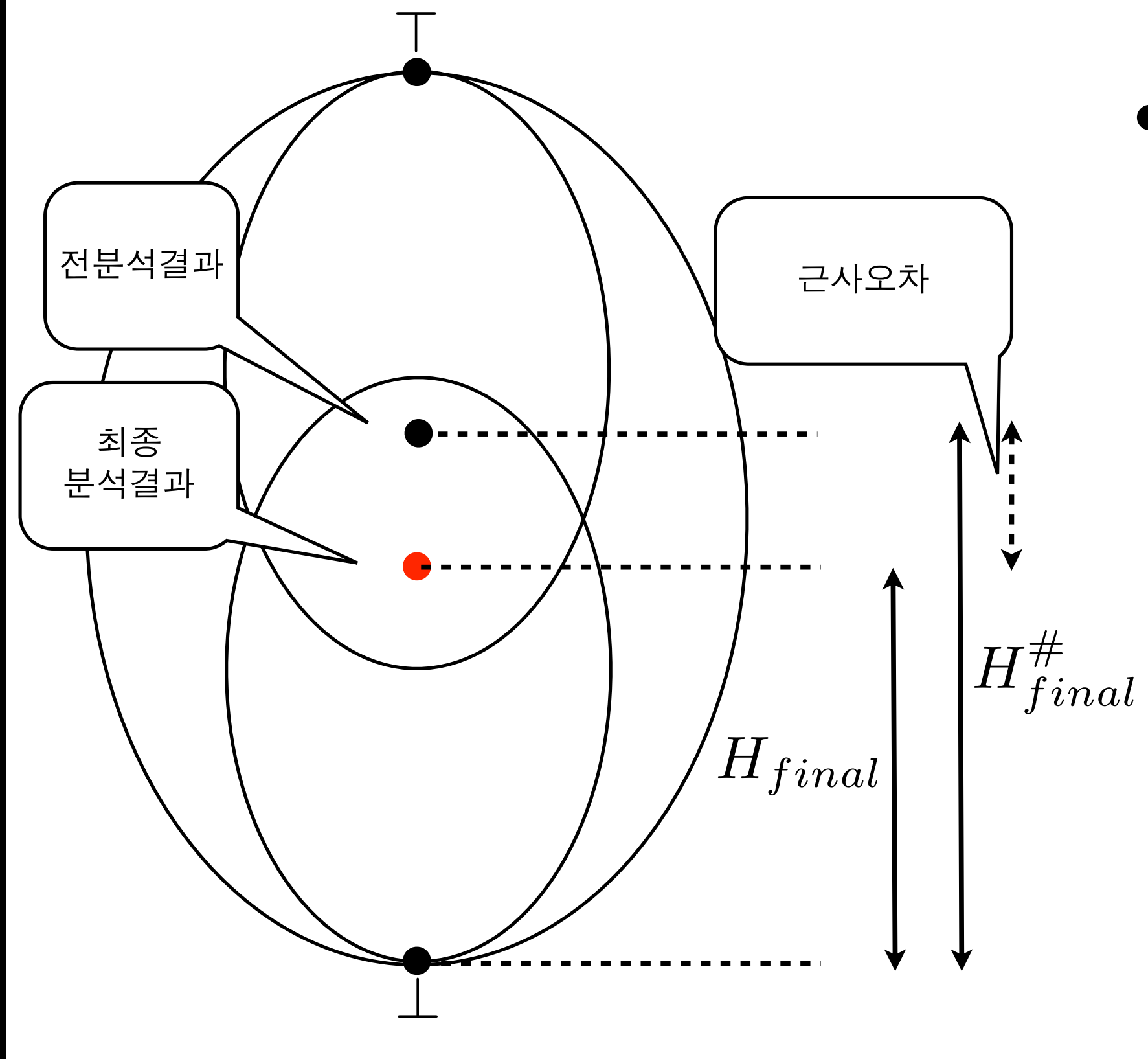
- 2단계 : 다음 데이터를 사전분석 실행 중 기록
  (사전분석은 m번의 iteration으로 고정점에 도달한다고 가정)

$$\left(\frac{H^\sharp_0}{H^\sharp_m}, \frac{0}{m}\right), \quad \left(\frac{H^\sharp_1}{H^\sharp_m}, \frac{1}{m}\right), \quad \cdots, \quad \left(\frac{H^\sharp_i}{H^\sharp_m}, \frac{i}{m}\right), \quad \cdots, \quad \left(\frac{H^\sharp_m}{H^\sharp_m}, \frac{m}{m}\right)$$

where $H^\sharp_i = \mathsf{H}(\gamma({F^\sharp}^i(\perp^\sharp)))$.

- 3단계 : 기록된 값을 선형보간하여 다음 함수를 얻음. 본 분석 실행 중 아래함수를 이용하여 프로그래스 보정

normalize : $[0,1] \to [0,1]$



- 다음 α 를 학습하여 정확한 최종 높이 계산

$$H_{final} = \alpha \cdot H^{\#}_{final} (0 \le \alpha \le 1)$$

- 프로그램 구조, 전분석 결과와 관련된 값들을 이용하여 Ridge 선형 회귀 분석

Table 1. The feature vector used by linear regression to construct prediction models

Category	Feature
Inter-procedural (syntactic)	# function calls in the program
	# functions in recursive call cycles
	# undefined library function calls
Loop-related (syntactic)	the maximum loop size
	the average loop sizes
	the standard deviation of loop sizes
	the standard deviation of depths of loops
	# loopheads
Numerical analysis (semantic)	# bounded intervals in the pre-analysis result
	# unbounded intervals in the pre-analysis result
Pointer analysis (semantic)	# points-to sets of cardinality over 4 in the pre-analysis result
	# points-to sets of cardinality under 4 in the pre-analysis result
Post-fixpoint (semantic)	# program points where applying the transfer function once improves the precision
	height decrease when transfer function is applied once

- 254개의 프로그램을 이용하여 학습

정확도 (3-fold validation) :

인터벌 : 평균오차 0.063, 표준편차 0.007 포인터 : 평균오차 0.053, 표준편차 0.001

사전 분석 : 부분적 흐름 민감 분석 (cf. 본 분석 : 흐름 민감 분석)

(본분석과 달리 특정 프로그램 지점들을 제외한 나머지 지점들은 요약 값을 뭉뚱그림)

- 요약도메인

$$\mathbb{C} \to \mathbb{S} \underset{\alpha}{\overset{\gamma}{\leftrightarrows}} \Delta \to \mathbb{S} \qquad \Delta = \Phi \cup \{\bullet\} \qquad \gamma(X) = \lambda c.\ X(\delta(c))$$

- 요약 값을 구분할 지점들 Φ

$$\Phi = \{c \in \mathbb{C} \mid w \in \mathbb{W} \ \wedge\ c \hookrightarrow^{depth} w\}$$

조정할 수 있는 매개변수. 클수록 본분석에 가까워짐

- 요약 의미 함수

$$F^\sharp(X) = \lambda i \in \Delta.\ (\bigsqcup_{c\in\delta^{-1}(i)} f_c(\bigsqcup_{c'\hookrightarrow c} X(\delta(c'))) \qquad \delta^{-1}(i) = \{c \in \mathbb{C} \mid \delta(c) = i\}$$

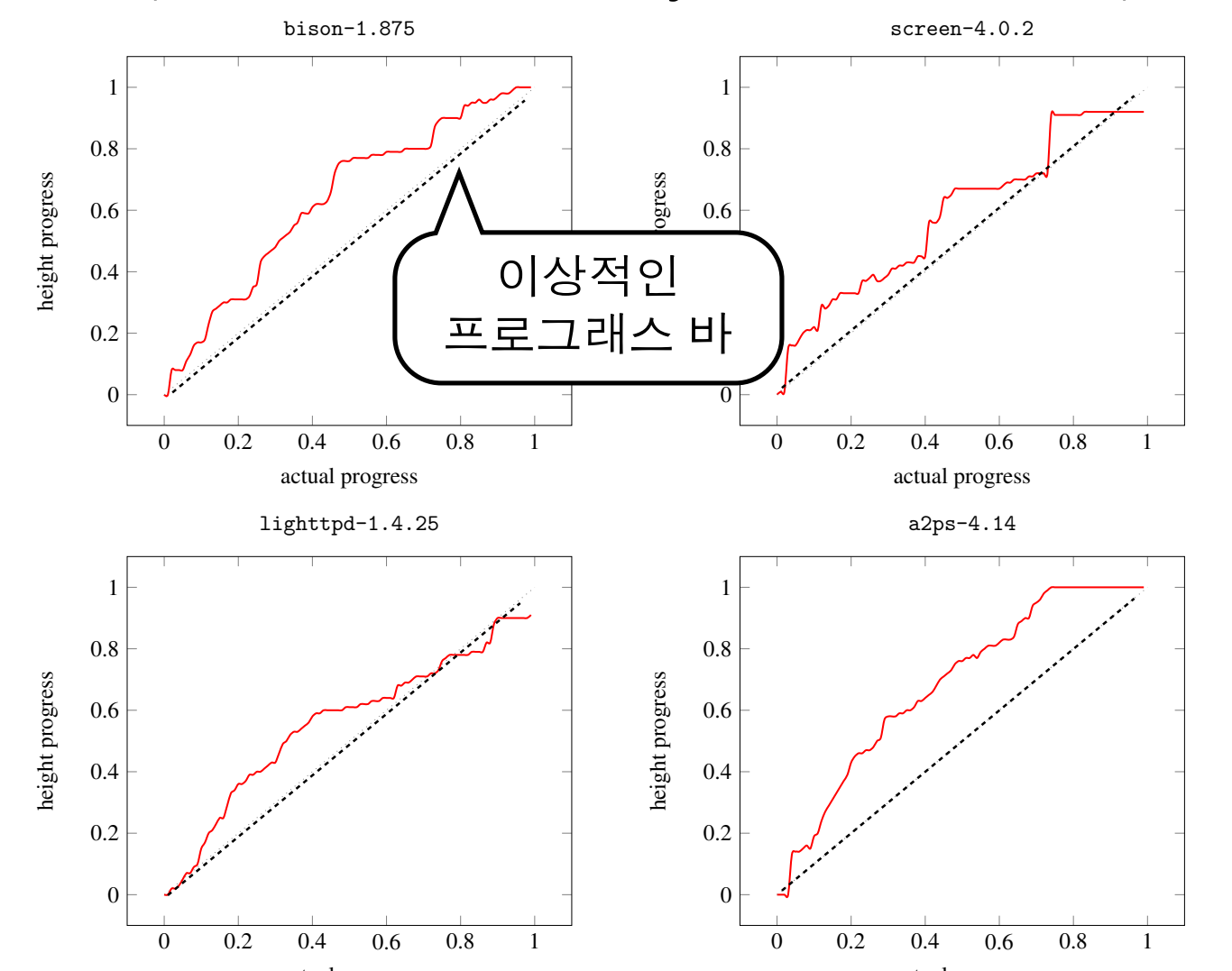
$$\delta(c) = \begin{cases} c & c \in \Phi \\ \bullet & c \notin \Phi \end{cases}$$

본 분석의 요약의미함수 :

$$F(X) = \lambda c \in \mathbb{C}. f_c(\bigsqcup_{c'\hookrightarrow c} X(c'))$$

결과

- 실험 결과 : 8개 GNU 프로그램에 대해서 실험
- 인터벌 분석 (평균 3.8%의 추가시간으로 예측)
  (x축 : 실제 진행율 y축 : 추정 진행율)

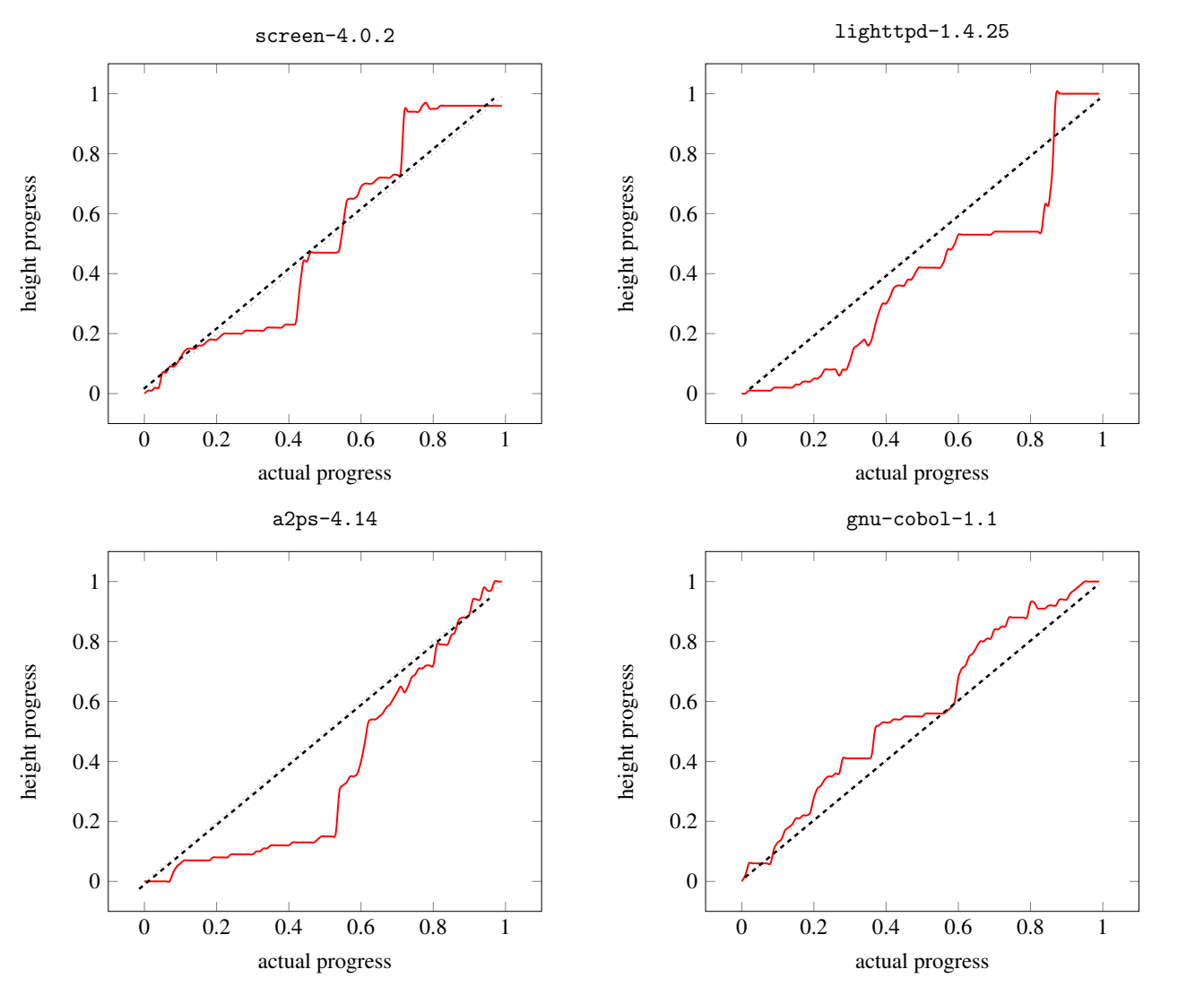


얼마나 이상적인 프로그래스 바에 가까운가? (Best : 1)

Program	LOC	Time(s) Main	Time(s) Pre	Linearity	Overhead	Height-Approx.
bison-1.875	38841	3.66	0.91	0.73	24.86%	1.03
screen-4.0.2	44745	40.04	2.37	0.86	5.92%	0.96
lighttpd-1.4.25	56518	27.30	1.21	0.89	4.43%	0.92
a2ps-4.14	64590	32.05	11.26	0.51	35.13%	1.06
gnu-cobol-1.1	67404	413.54	99.33	0.54	24.02%	0.91
gnugo	87575	1541.35	7.35	0.89	0.48%	1.12
bash-2.05	102406	16.55	2.26	0.80	13.66%	0.93
sendmail-8.14.6	136146	1348.97	5.81	0.69	0.43%	0.93
TOTAL	686380	3423.46	130.5	**0.74**	**3.81%**	**Err : 0.07**

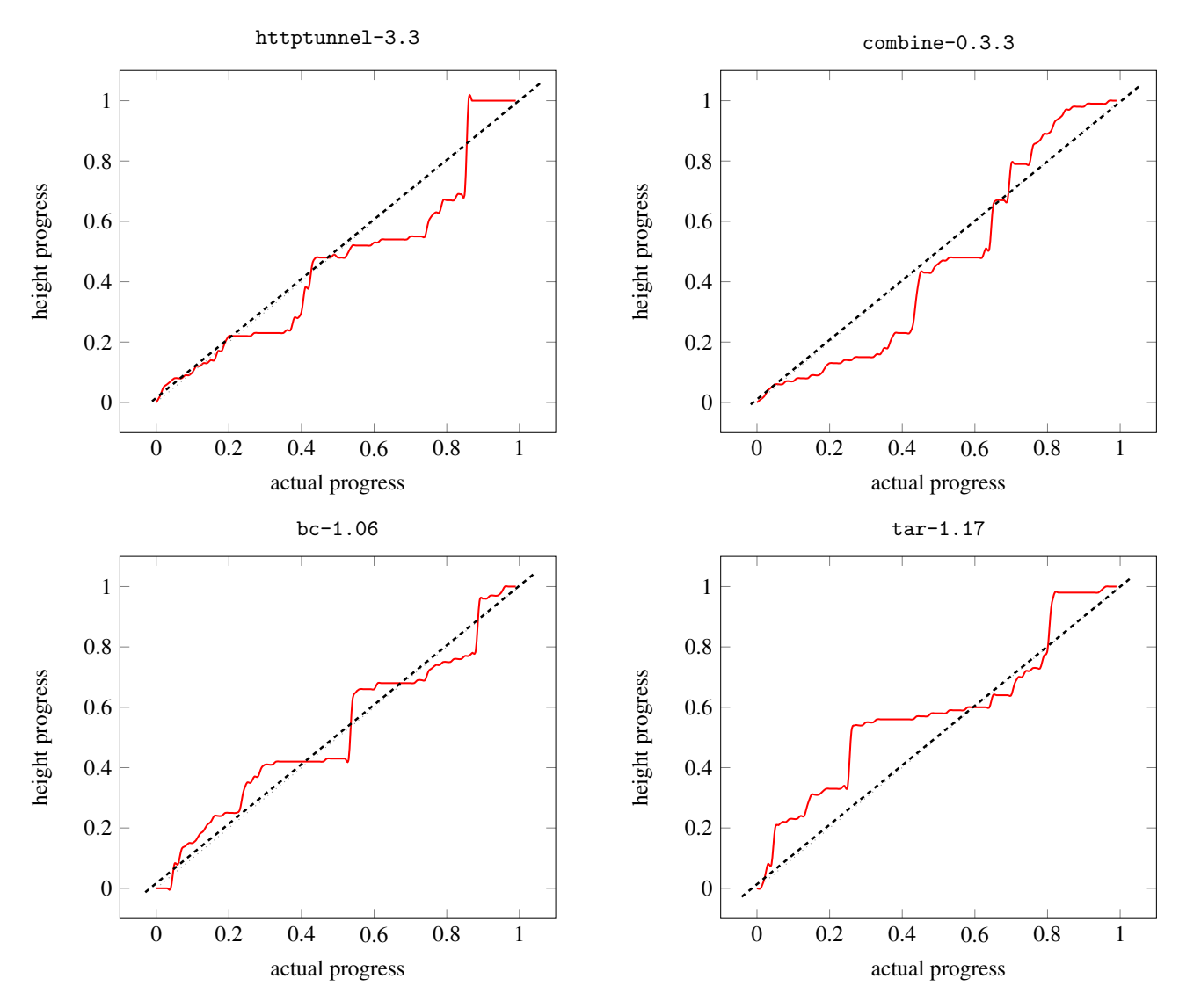
얼마나 최종결과의 래티스 높이를 정확히 어림잡았는가? (Best : 1)

- 포인터 분석 (평균 7.3%의 추가시간으로 예측)



Program	LOC	Time(s) Main	Time(s) Pre	Linearity	Overhead	Height-Approx.
screen-4.0.2	44745	15.89	1.56	0.90	9.82%	0.98
lighttpd	56518	11.54	0.87	0.76	7.54%	1.03
a2ps-4.14	64590	10.06	3.48	0.65	34.59%	1.04
gnu-cobol-1.1	67404	32.27	12.22	0.91	37.87%	1.03
gnugo	87575	217.77	3.88	0.64	1.78%	0.97
bash-2.05	102406	3.68	0.78	0.56	21.20%	1.04
proftpd-1.3.2	126996	74.64	11.14	0.82	14.92%	1.03
sendmail-8.14.6	136146	145.62	3.15	0.58	2.16%	0.98
TOTAL	686380	511.47	37.08	**0.73**	**7.25%**	**Err : 0.03**

- 옥타곤 분석 (평균 36.6%의 추가시간으로 예측)



Program	LOC	Time(s) Main	Time(s) Pre	Linearity	Overhead
httptunnel-3.3	6174	49.5	8.2	0.91	16.6%
combine-0.3.3	11472	478.2	16	0.89	3.4%
bc-1.06	14288	63.9	43.8	0.96	68.6%
tar-1.17	18336	977.0	73.1	0.82	7.5%
parser	18923	190.1	104.8	0.97	55.1%
wget-1.9	35018	3895.36	1823.15	0.92	46.8%
TOTAL	69193	5654.0	2069.49	**0.91**	**36.6%**

결론 : 분석기 프로그래스 바를 만드는 일반적인 방법 제시

설계

- 실제 분석보다 부정확, 그러나 빠른 전분석 설계
- 정확할수록 작고 부정확할 수록 큰 수치값 부여하는 법 결정

진행

- 전분석 후 본분석 진행
- 전분석 수치값 및 수치값이 변화하는 양상이용, 본분석 프로그래스 바 구현