디지털 컴퓨터의 소수점 표현 방법

서울대학교 프로그래밍 연구실
허 기 홍

동기

- 소수점 분석을 위한 배경지식
  - 예) 로봇 제어 프로그램
- 소수점 안전 불감증
  - 문제는 알고 있지만 잘 될것이라 믿음

예

- 황금비의 거듭제곱 $(\frac{\sqrt{5}-1}{2})^i$

```
main(){
   float x, y, z;
   int i;
   x = 1;
   y = (sqrt(5) - 1)/2;
   for (i = 1; i <= 20; i++){
      z = x;
      x = y;
      y = z - y;
      printf("phi^%d=%f\n",i,x);
   }
}
```

```
main(){
   float t;
   int i;
   for (i = 1; i <= 20; i++){
      t = t*(sqrt(5)-1)/2
      printf("phi^%d=%f\n",i,x);
   }
}
```

* $(\frac{\sqrt{5}-1}{2})^{n+2} = (\frac{\sqrt{5}-1}{2})^n - (\frac{\sqrt{5}-1}{2})^{n+1}$

예제

```
main(){
    float x, y, z;
    int i;
    x = 1;
    y = (sqrt(5) - 1)/2;
    for (i = 1; i <= 20; i++){
        z = x;
        x = y;
        y = z - y;
        printf("phi^%d=%f\n",i,x);
    }
}
```

	왼	오
phi^1	0.618034	0.618034
phi^2	0.381966	0.381966
phi^3	0.236068	0.236068
phi^4	0.145898	0.145898
phi^5	0.090170	0.090170
phi^6	0.055728	0.055728
phi^7	0.034442	0.034442
phi^8	0.021286	0.021286
phi^9	0.013156	0.013156
phi^10	0.008130	0.008131
phi^11	0.005026	0.005025
phi^12	0.003103	0.003106
phi^13	0.001923	0.001919
phi^14	0.001180	0.001186
phi^15	0.000743	0.000733
phi^16	0.000437	0.000453
phi^17	0.000306	0.000280
phi^18	0.000131	0.000173
phi^19	0.000176	0.000107
phi^20	-0.000045	0.000066

```
main(){
    float t;
    int i;
    for (i = 1; i <= 20; i++){
        t = t*(sqrt(5)-1)/2
        printf("phi^%d=%f\n",i,x);
    }
}
```

Eric Goubault. *Static analysis of floating-point operations*. SAS 01

예

- 무심코 지나치는 타입 변환

```
int main(){
    double x, a;
    double y, z, f;
    x = 1125899973951488.0;
    a = 1.0;
    y = (x+a);
    z = (x-a);
    f = y - z;
    printf("%f\n", f);
}
```

```
int main(){
    double x, a;
    float y, z, f;
    x = 1125899973951488.0;
    a = 1.0;
    y = (x+a);
    z = (x-a);
    f = y - z;
    printf("%f\n", f);
}
```

```
int main(){
    double x, a;
    float y, z, f;
    x = 1125899973951487.0;
    a = 1.0;
    y = (x+a);
    z = (x-a);
    f = y - z;
    printf("%f\n", f);
}
```

*P. Cousot, ROSAEC seminar slide

예

- 무심코 지나치는 타입 변환

```
int main(){
    double x, a;
    double y, z, f;
    x = 1125899973951488.0;
    a = 1.0;
    y = (x+a);
    z = (x-a);
    f = y - z;
    printf("%f\n", f);
}
```

2.0

```
int main(){
    double x, a;
    float y, z, f;
    x = 1125899973951488.0;
    a = 1.0;
    y = (x+a);
    z = (x-a);
    f = y - z;
    printf("%f\n", f);
}
```

```
int main(){
    double x, a;
    float y, z, f;
    x = 1125899973951487.0;
    a = 1.0;
    y = (x+a);
    z = (x-a);
    f = y - z;
    printf("%f\n", f);
}
```


*P. Cousot, ROSAEC seminar slide

예

- 무심코 지나치는 타입 변환

```
int main(){
    double x, a;
    double y, z, f;
    x = 1125899973951488.0;
    a = 1.0;
    y = (x+a);
    z = (x-a);
    f = y - z;
    printf("%f\n", f);
}
```

2.0

```
int main(){
    double x, a;
    float y, z, f;
    x = 1125899973951488.0;
    a = 1.0;
    y = (x+a);
    z = (x-a);
    f = y - z;
    printf("%f\n", f);
}
```

134217728.0

```
int main(){
    double x, a;
    float y, z, f;
    x = 1125899973951487.0;
    a = 1.0;
    y = (x+a);
    z = (x-a);
    f = y - z;
    printf("%f\n", f);
}
```

*P. Cousot, ROSAEC seminar slide

예

- 무심코 지나치는 타입 변환

```
int main(){
    double x, a;
    double y, z, f;
    x = 1125899973951488.0;
    a = 1.0;
    y = (x+a);
    z = (x-a);
    f = y - z;
    printf("%f\n", f);
}
```

2.0

```
int main(){
    double x, a;
    float y, z, f;
    x = 1125899973951488.0;
    a = 1.0;
    y = (x+a);
    z = (x-a);
    f = y - z;
    printf("%f\n", f);
}
```

134217728.0

```
int main(){
    double x, a;
    float y, z, f;
    x = 1125899973951487.0;
    a = 1.0;
    y = (x+a);
    z = (x-a);
    f = y - z;
    printf("%f\n", f);
}
```

0.0

*P. Cousot, ROSAEC seminar slide

IEEE 754 표준

부동 소수점

- 실수를 표현하는 방식 중 하나
  - 참고) floating slash, signed logarithm
- 표준 실수 표기법으로 지정
  - IEEE 754 (1985년)

형식

- β : 기수(*base*), *p* : 유효숫자 (*precision*)
- *d.d...d* : 가수(*significand*), e : 지수(*exponent*)
- $\pm d_0.d_1 d_2 \cdots d_{p-1} \times \beta^e \qquad (0 \le d_i < \beta, e_{min} \le e \le e_{max})$
- 예) 0.1
  - 1.00×10^{-1} (β = 10, *p* = 3)
  - 1.100110×2^{-4} (β = 2, *p* = 7)

형식

- $\beta = 2$

Parameter	Format			
	Single	Single-Extended	Double	Double-Extended
p	24	32	53	64
max exp	+127	+1023	+1023	>16383
min exp	-126	≤-1022	-1022	≤-16382
exp width	8	≤11	11	15
format width	32	43	64	79

Normalized Number

- 가수의 첫 숫자가 0이 아닌 수
  - 같은 숫자 - 다른 표현 중 대표
  - 예) 0.1 : $0.01 \times 10^1, 1.00 \times 10^{-1}$

Normalized Number

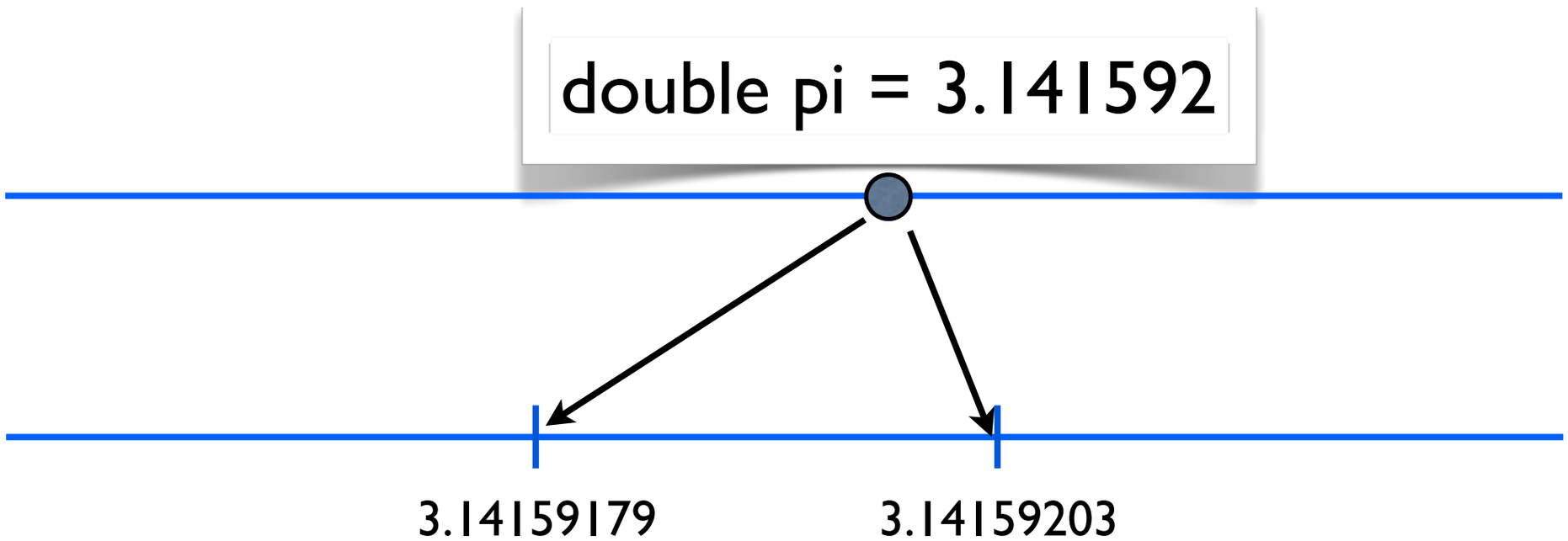
- 가수의 첫 숫자가 0이 아닌 수
  - 같은 숫자 - 다른 표현 중 대표
  - 예) 0.1 : $0.01 \times 10^{1}, 1.00 \times 10^{-1}$

Normalized Number

- 가수의 첫 숫자가 0이 아닌 수
  - 같은 숫자 - 다른 표현 중 대표
  - 예) 0.1 : $0.01 \times 10^{1}, 1.00 \times 10^{-1}$

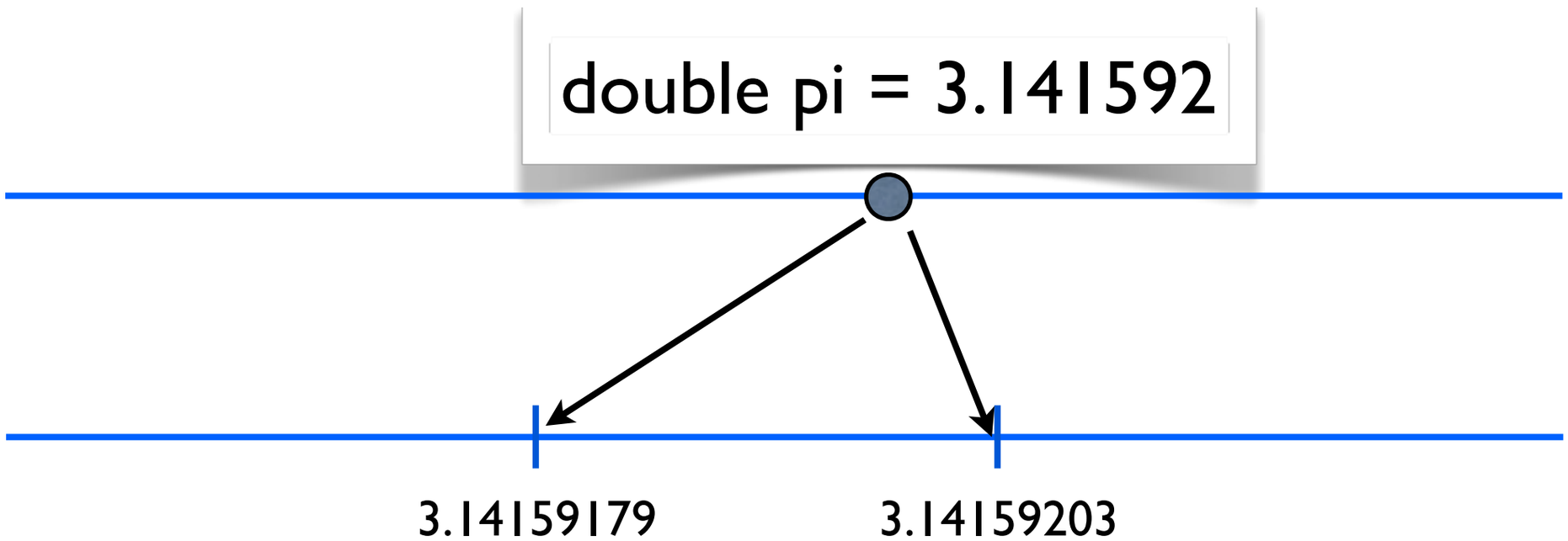
- 단, $0 = 1.0 \times \beta^{e_{min}-1}$

Rounding Modes



- round toward +∞ : 올림
- round toward -∞ : 내림
- round toward 0 : 버림
- round toward nearest : 반올림

Rounding Modes



- round toward +∞ : 올림
- round toward -∞ : 내림
- round toward 0 : 버림
- round toward nearest : 반올림

r(2.5) = 2
r(-2.5) = -2

연산

- 정확하게 계산한 후 라운딩
  - 예 : $2.15 \times 10^{12} - 1.25 \times 10^{-5}$

$$
\begin{aligned}
x &= 2.15000000000000000000 \times 10^{12} \\
y &= 0.00000000000000000125 \times 10^{12} \\
x - y &= 2.14999999999999999875 \times 10^{12} \\
&\rightarrow 2.15 \times 10^{12}
\end{aligned}
$$

특별한 값

- 특별한 상황의 결과
  - 매끄러운 예외처리를 위해

지수	가수	의미
$e = e_{min} - 1$	$f = 0$	± 0
$e = e_{min} - 1$	$f \neq 0$	$0.f \times 2^{e_{min}}$
$e_{min} \leq e \leq e_{max}$	$-$	$1.f \times 2^{e}$
$e = e_{max} + 1$	$f = 0$	$\pm\infty$
$e = e_{max} + 1$	$f \neq 0$	`NaN`

NaN

- *Not a Number*
- 중간결과가 NaN이면 최종결과도 NaN

연산	NaN 발생
$+$	$\infty + (-\infty)$
$\times$	$0 \times \infty$
$/$	$0/0,\ \infty/\infty$
$\%$	$x\%0, \infty\%y$
$\sqrt{\ }$	$\sqrt{x}$ (when $x < 0$)

무한대

- 표현가능한 가장 큰/작은 수보다 큰/작은 경우
  - 예) $\beta = 10, p = 3, e_{max} = 127, x = 5.00 \times 10^{100}$
    $x^2 = \infty$

- 0이 아닌 수를 0으로 나누는 경우

±0

- Normalized Number는 0을 표현 못함

±0

- 무한대의 부호를 보존하기 위하여
  - 예) $x = \pm\infty, 1/(1/x) = x?$

$$1/(1/\infty) = \infty \quad 1/(1/(-\infty)) \neq \infty$$

- 0 에서 불연속 함수인 경우
  - 예) $\log(+0) = -\infty, \log(-0) = NaN$

Denormalized Number

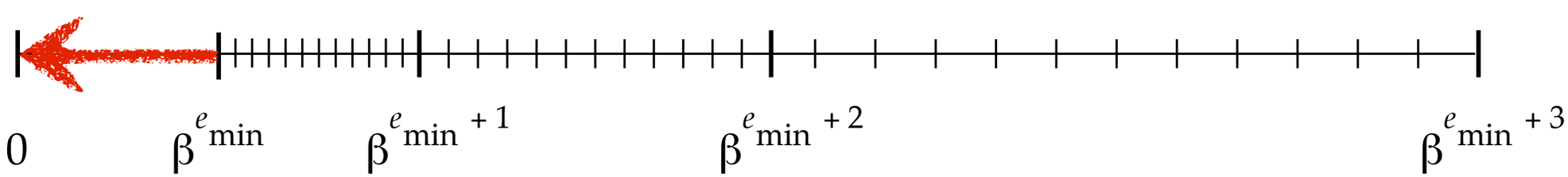
- Normalized Number가 못다루는 작은 수

Denormalized Number

- 예) Normalized number만 사용하면
  - $\beta = 10, p = 3, e_{min} = 98$

    $x = 6.87 \times 10^{-97}, y = 6.81 \times 10^{-97}$

$$x - y = 6.0 \times 10^{-99} \Rightarrow 0$$



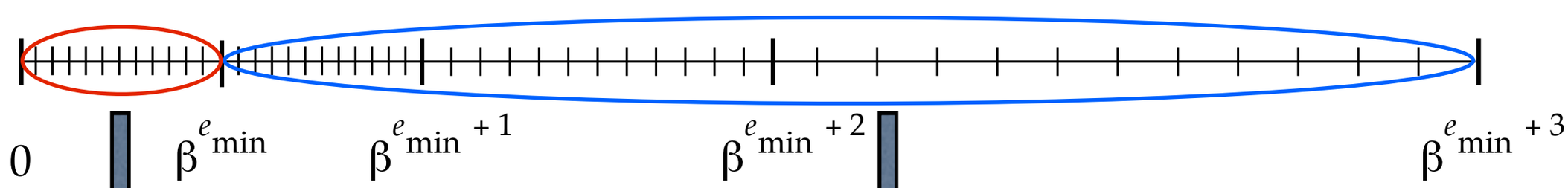
Denormalized Number

- 예) Normalized number만 사용하면
  - $\beta = 10, p = 3, e_{min} = 98$

    $x = 6.87 \times 10^{-97}, y = 6.81 \times 10^{-97}$

```
if(x != y)
  z = 1/(x-y);
```

Denormalized Number



$$e = e_{min} - 1, f \neq 0$$

$$0.f \times 2^{e_{min}}$$

$$e_{min} \leq e \leq e_{max}$$

$$1.f \times 2^{e}$$

현실

애매모호

- IEEE754는 하드웨어 관점
  - PL의 의미를 강제하지는 않음
  - 프로그래머, 컴파일러는 ‘그저 그러려니’

위험한 프로그램

- 애매한 타입 변환 (C 언어)
  - double - float 변형시 오차 발생

```
float q = 3.0/7.0;
if(q == 3.0/7.0){
  printf(“equal”);
}else{
  printf(“not equal”);
}
```

위험한 프로그램

- 애매한 타입 변환 (C 언어)
  - double - float 변형시 오차 발생

```
float q = 3.0/7.0;
if(q == 3.0/7.0){
  printf("equal");
}else{
  printf("not equal");
}
```

```
$./a.out
not equal
```

예

- 무심코 지나치는 타입 변환

```
int main(){
    double x, a;
    double y, z, f;
    x = 1125899973951488.0;
    a = 1.0;
    y = (x+a);
    z = (x-a);
    f = y - z;
    printf("%f\n", f);
}
```

```
int main(){
    double x, a;
    float y, z, f;
    x = 1125899973951488.0;
    a = 1.0;
    y = (x+a);
    z = (x-a);
    f = y - z;
    printf("%f\n", f);
}
```

```
int main(){
    double x, a;
    float y, z, f;
    x = 1125899973951487.0;
    a = 1.0;
    y = (x+a);
    z = (x-a);
    f = y - z;
    printf("%f\n", f);
}
```

*P. Cousot, ROSAEC seminar slide

예

- 무심코 지나치는 타입 변환

```
int main(){
    double x, a;
    double y, z, f;
    x = 1125899973951488.0;
    a = 1.0;
    y = (x+a);
    z = (x-a);
    f = y - z;
    printf("%f\n", f);
}
```

2.0

```
int main(){
    double x, a;
    float y, z, f;
    x = 1125899973951488.0;
    a = 1.0;
    y = (x+a);
    z = (x-a);
    f = y - z;
    printf("%f\n", f);
}
```

```
int main(){
    double x, a;
    float y, z, f;
    x = 1125899973951487.0;
    a = 1.0;
    y = (x+a);
    z = (x-a);
    f = y - z;
    printf("%f\n", f);
}
```


*P. Cousot, ROSAEC seminar slide

예

- 무심코 지나치는 타입 변환

```
int main(){
    double x, a;
    double y, z, f;
    x = 1125899973951488.0;
    a = 1.0;
    y = (x+a);
    z = (x-a);
    f = y - z;
    printf("%f\n", f);
}
```

2.0

```
int main(){
    double x, a;
    float y, z, f;
    x = 1125899973951488.0;
    a = 1.0;
    y = (x+a);
    z = (x-a);
    f = y - z;
    printf("%f\n", f);
}
```

134217728.0

```
int main(){
    double x, a;
    float y, z, f;
    x = 1125899973951487.0;
    a = 1.0;
    y = (x+a);
    z = (x-a);
    f = y - z;
    printf("%f\n", f);
}
```

*P. Cousot, ROSAEC seminar slide

예

- 무심코 지나치는 타입 변환

```
int main(){
    double x, a;
    double y, z, f;
    x = 1125899973951488.0;
    a = 1.0;
    y = (x+a);
    z = (x-a);
    f = y - z;
    printf("%f\n", f);
}
```

2.0

```
int main(){
    double x, a;
    float y, z, f;
    x = 1125899973951488.0;
    a = 1.0;
    y = (x+a);
    z = (x-a);
    f = y - z;
    printf("%f\n", f);
}
```

134217728.0

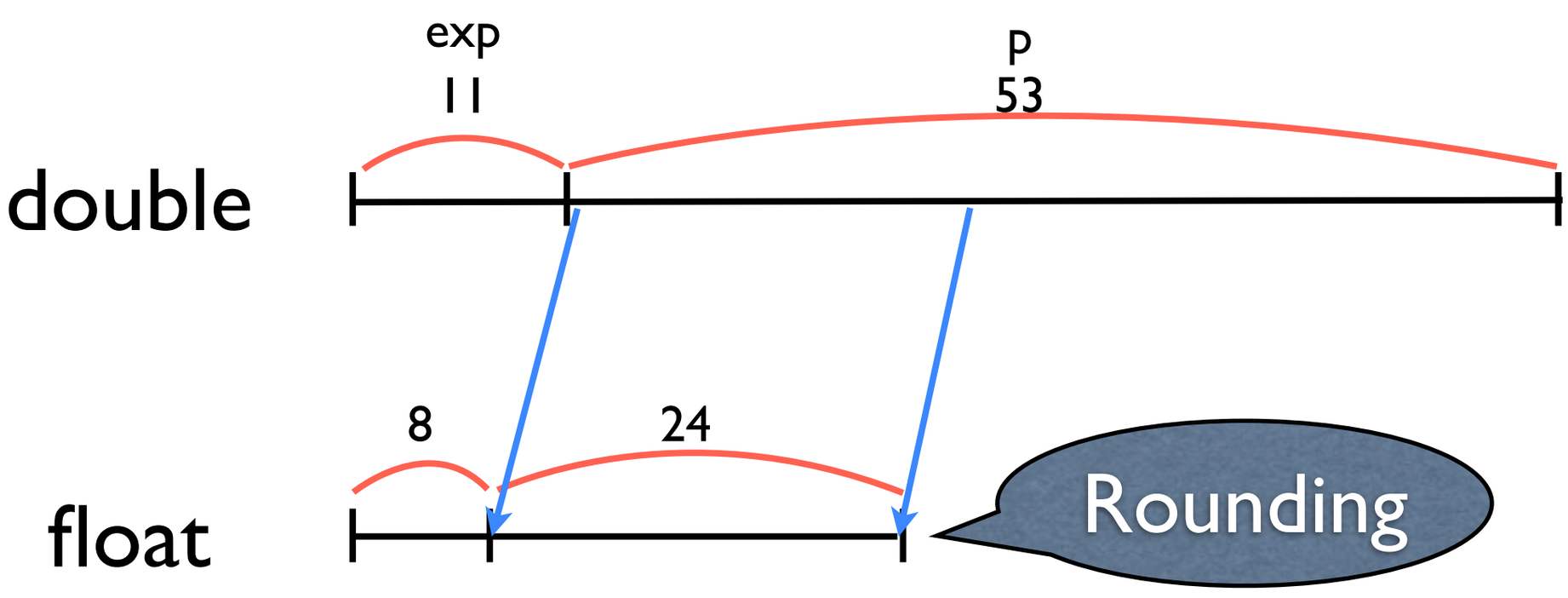
```
int main(){
    double x, a;
    float y, z, f;
    x = 1125899973951487.0;
    a = 1.0;
    y = (x+a);
    z = (x-a);
    f = y - z;
    printf("%f\n", f);
}
```

0.0

*P. Cousot, ROSAEC seminar slide

예

- 이유
  - *Float* : 24 bit 유효 숫자
  - *Double* : 53 bit 유효 숫자



예

```
int main(){
   double x;
   x = 112589973951488.0;
   printf("%f\n", f);
}
```

112,589,973,951,488

```
int main(){
   float x;
   x = 112589973951487.0;
   printf("%f\n", f);
}
```

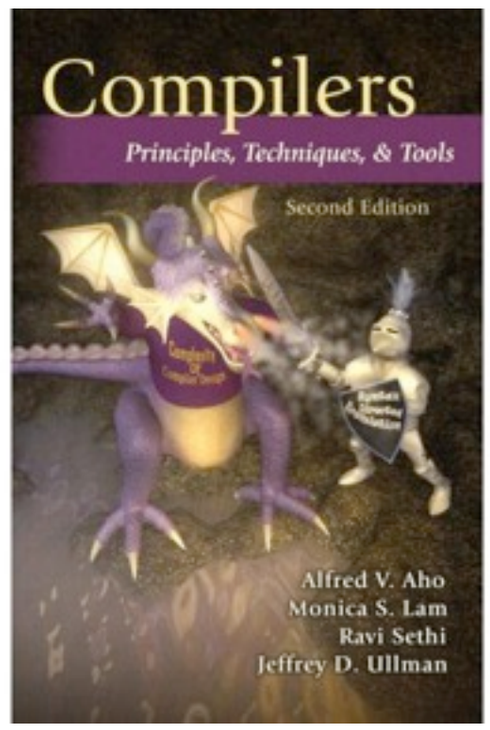
```
int main(){
   float x;
   x = 112589973951488.0;
   printf("%f\n", f);
}
```

```
int main(){
   float x;
   x = 112589973951489.0;
   printf("%f\n", f);
}
```

1,125,899,906,842,624 = 1,125,899,906,842,624 << 1,125,900,041,060,352

위험한 최적화

- 코드 최적화 단계에서 의미가 바뀔수도




```
x / 10.0
```

↓

```
0.1 * x
```

소수점 값 분석

- 문제
  - 분석기 자체의 소수점 값 오차
  - 분석기 만드는 언어와 분석 대상 언어의 괴리

유용한 도구

- 유용한 라이브러리 : APRON*
  - 부동 소수점 비롯 여러 수학적 도메인

*http://apron.cri.ensmp.fr

APRON

문자열 : 라운딩 오차 방지

C float 타입 : 가수 24비트

```
let s = “3.1415926” in
let l = Mpfr.init2 24 in
let u = Mpfr.init2 24 in
let ub = Mpfr.set_str x s Mpfr.up in
let lb = Mpfr.set_str x s Mpfr.down in
...
```

라운딩 모드 : 올림, 내림

마무리

- 소수점 값 분석 :
  소.무.연.의 좋은 연구 대상
  - 프로그래머의 예상 밖
  - 소수점 숫자를 사용하는 중요한 SW가 산재

참고자료

- David Goldberg. What every computer scientist should know about floating-point arithmetic, *ACM Computing Surveys*.1991
- Eric Goubault. Static analysis of floating-point operations. *SAS01*
- W. Kahan. IEEE Standard 754 for Binary floating-Point Arithmetic, *Lecture Notes on the Status of IEEE 754*. 1995

참고자료

- B. Jeannet and A. Miné. APRON: A library of numerical abstract domains for static analysis. *CAV'09*
- *http://apron.cri.ensmp.fr/*

고맙습니다.