- This slide was
  - a material for the "Reading PLDI Papers (PLDIr)" study group
  - written by Kazuhiro Inaba (www.kmonos.net), under my own understanding of the papers published at PLDI
    - So, it may include many mistakes etc
- For your correct understanding, please consult the original paper and/or the authors' presentation slide!

k.inaba (稲葉 一浩), reading the following paper:

PLDIr #3 :: Nov 3, 2009

とある型の修飾理論

A Theory of Type Qualifiers

メタ情報

- Citation Count (ACM) : 59
- Authors
  - J. S. Foster
    - “Static Type Inference for Ruby”, 2009
    - Type Qualifier の研究もずっと継続しているぽい
  - M. Fähndrich
    - MS Research で Spec#, Singularity OS など
  - A. Aiken
    - 型を使ったりSAT Solver使ったりでプログラム検証

論文の概要

- Type Qualifier = 要はC言語のconst
  - 例) const int : 読み取り専用の整数
- Qualifierという概念を型理論的に整理
  - const に限らず汎用に
  - C に限らず他の型システムにPlug-inしやすく
  - Qualifier Polymorphism, Qualifier Inference

Type Qualifier の例

- const
  - 書き換えできない

```
const char* s = "abc";
*s = "A"; // エラー!
```

- nonnull
  - null にならない

```
void f( nonnull void* p);
f( null ); // エラー!
```

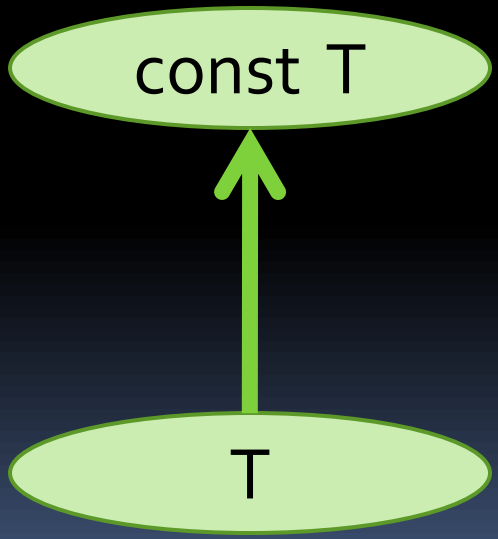
- static
  - コンパイル時定数

```
void g(static int N);
g( 42 );          // 大丈夫
g( getchar() );   // エラー!
```

Qualifierによるサブタイプ関係

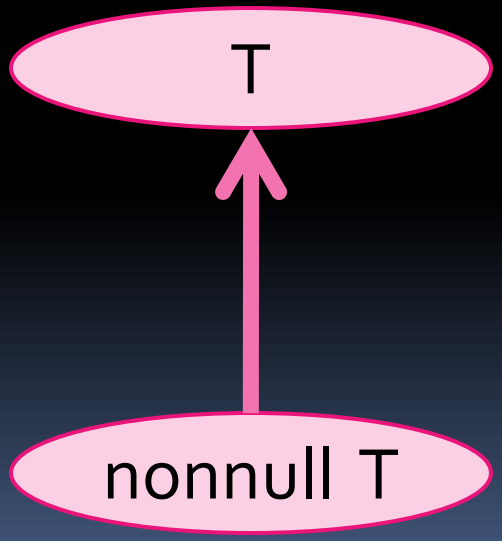
- const
  - 書き換えできない

```
void  paramConst( const char* p );
void  paramNonConst( char* p );

char* ms = ...;
paramConst( ms ); // OK

const  char*  cs = ...;
paramNonConst ( cs ); // エラー
```



Qualifierによるサブタイプ関係

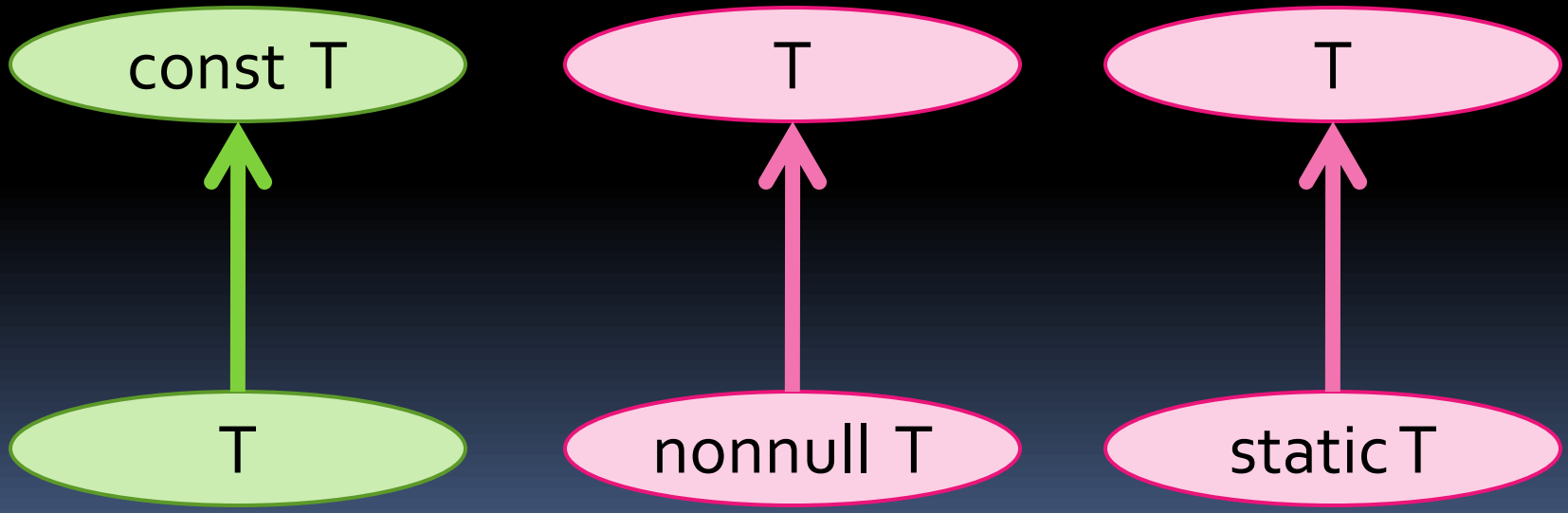
- nonnull
  - nullにならない



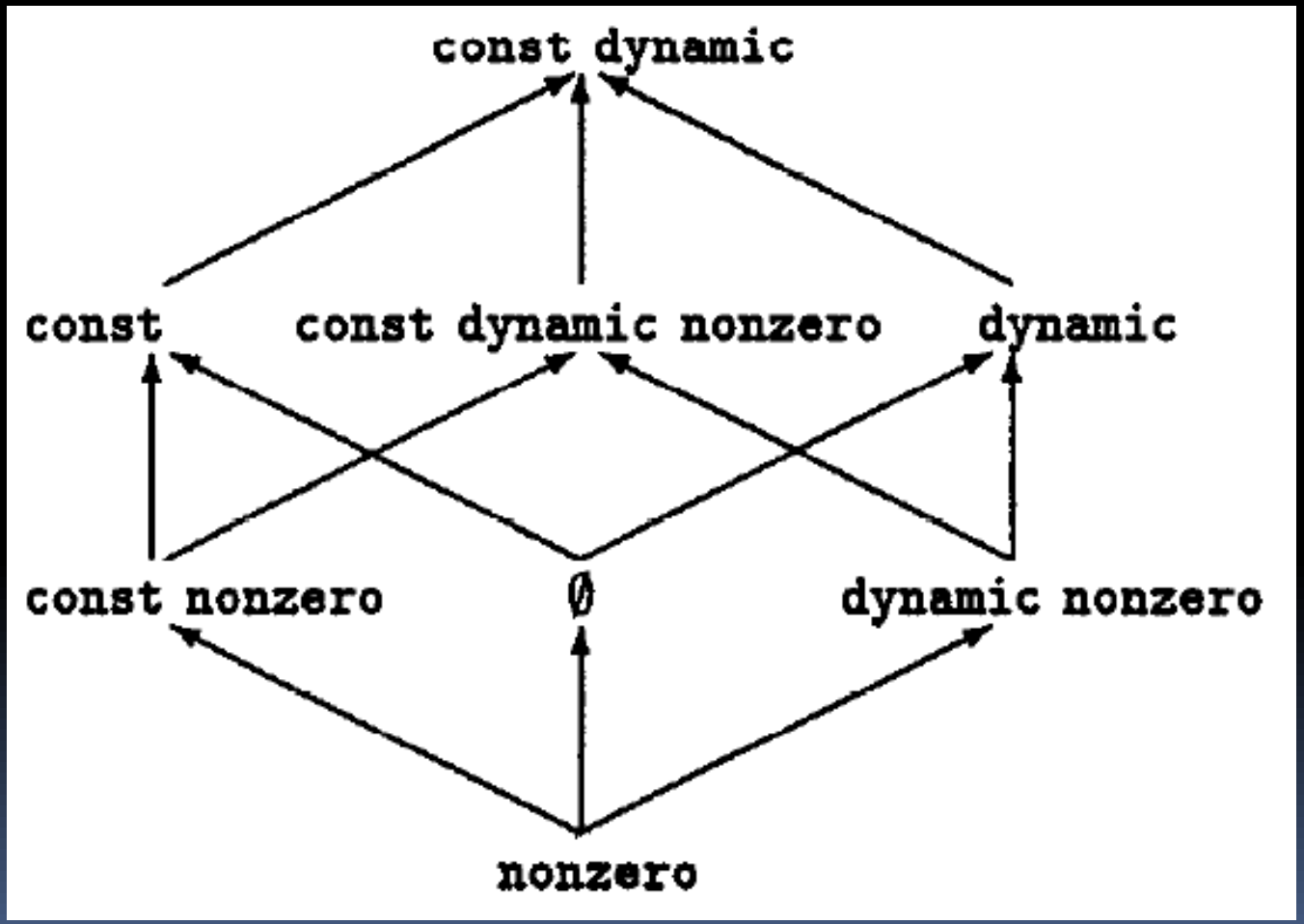

```
void  paramNN( nonnull char* p );
void  paramNullKamo( char* p );

char*  nullpo = ...;
paramNN( nullpo ); // エラー

nonnull   char*  nnp = ...;
paramNullKamo( nnp ); // OK
```

本論文での Type Qualifier とは

「型を修飾して、
どっちか向きのサブタイプ関係を作る物」



複数Qualifierの組み合わせも可



Qualifierを入れるための型システムと言語の拡張

- 型規則（必要ならもっと増やしてOK）

$$\frac{\vdash Q \sqsubseteq Q' \qquad \vdash \rho_i = \rho_i' \quad i \in [1..n]}{\vdash Q\ c(\rho_1, \ldots, \rho_n) \preceq Q'\ c(\rho_1', \ldots, \rho_n')}$$

たとえば
Q = constなし
Q′ = constあり

ならば

“e の修飾子は ℓ 以下”

イプ
（c(...)は元の言語の型）

- 言語

$$\begin{array}{lcl} e & ::= & \cdots \\ & | & e|_\iota \\ & | & \iota\, e \end{array}$$

“e の修飾子は ℓ 以上”

言語の拡張の使い道

- 例：
  - x = expr;

という文を型チェッカに渡す前に

  - $x|_{\neg const}$ = expr;

と変換しておき、

  - null

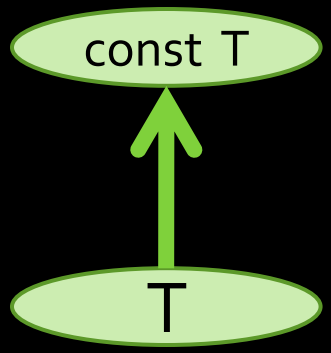
という式を型チェッカに渡す前に

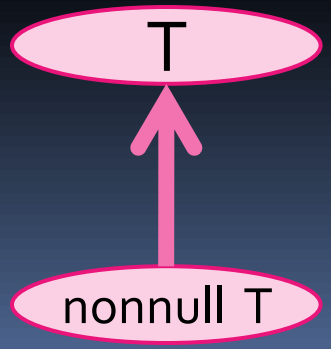
  - ¬nonnull　null

と変換しておく

“≦ constなし”

“≧ nonnullなし”





この論文のやったこと

- このようなQualifierの入った言語の型チェック規則を作った（詳細略）
- このようなQualifierの入った言語の型推論規則も作った（詳細略）
- Qualifierに関するPolymorphismも扱える

Qualifier Polymorphism とは

- Windows API にこんな関数があります。

```
char* CharNext(  const char* p  );
```

  - 現在の文字コードで１文字、ポインタを進める
  - 明らかに型がおかしい

  - 本当はこうしたかったに違いない

```
char*       CharNext( char* p  );
const char* CharNext( const char* p );
```

Qualifier Polymorphism とは

- 本当はこうしたかったに違いない

```
char*       CharNext( char* p  );
const char* CharNext( const char* p );
```

でも、こうするには、全く同じ実装を
２カ所に書かないといけない
→ そこで Qualifier Polymorphism!

```
Q char*  CharNext<Q>(  Q  char* p  )
  { return p+1; }
```

※構文と実装はイメージです

まとめ

- Type Qualifier
  - サブタイプ関係をどっちか向きに作る物
- 型システム作った
  - Qualifier Polymorphism もある
- 型推論もできる
  - 実際にcで書かれたベンチマークを推論してconstにできる箇所を探す実験もした

Quickly Reading:

Simon Peyton Jones
Alastair Raid
Fergus Henderson
Tony Hoare
Simon Marlow

A Semantics for Imprecise Exceptions

Haskell に綺麗に例外を入れたい

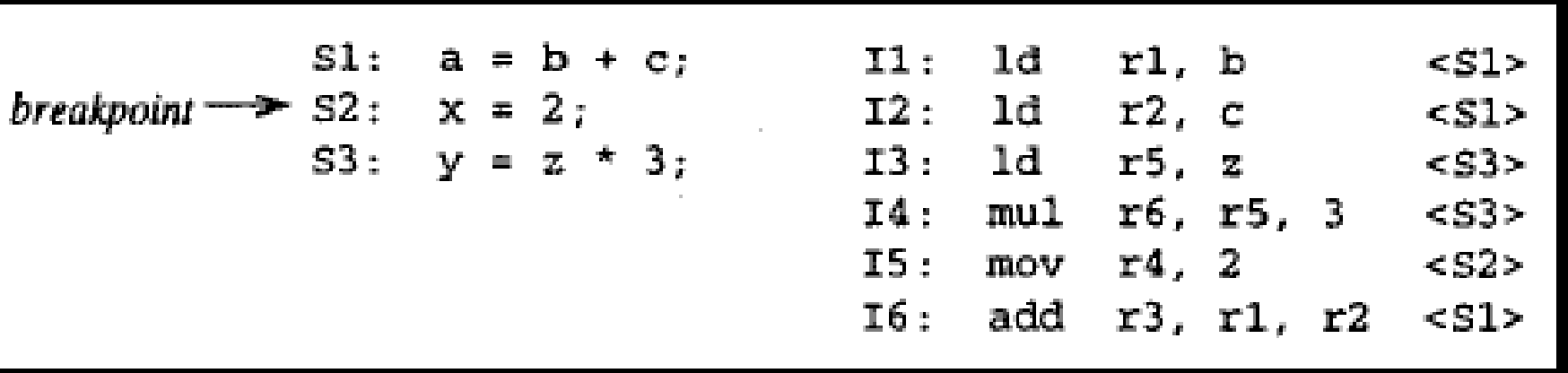
- 遅延評価のことをちゃんと考える
  - “制御フロー”ではなく“値”が例外を運ぶ
    - c.f. 浮動小数点数の NaN
- プログラム変換に対し頑強・参照透明性
  - (1/0)+(raise “foo”)　vs　(raise “foo”)+(1/0)
  - “起こりえた例外すべての集合” を運ぶ
    - “例外全てを求める計算”をIOモナドにして運ぶとベター
- 例外を使わない時はゼロオーバーヘッド
  - あたかも↑の意味論で動くように見せつつ、実際の実装は例外ハンドラへのジャンプで可能

Quickly Reading:

Le-Chun Wu
Rajiv Mirani
Harish Patil
Bruce Olsen
Wen-mei W. Hwu

A New Framework for Debugging Globally Optimized Code

命令再配置に対応できるデバッガ



- 基本アイデア
  - BPより後ろ由来の命令の先頭 (I3) でストップ
  - BPより前由来の命令 (I6) をエミュレート実行
- 実現方法
  - ソース→命令の対応を Anchor, Interception, Finish, Escape の4種のマッピングで管理して頑張る
    ※ 基本ブロック越え再配置 も扱える ×:ループ最適化