数理解析研究所講究録 1325 巻 2003 年 104-109

再帰型をもつオブジェクト指向計算モデルにおける例外処理の型付

堀江 美保子　酒井 正彦　坂部 俊樹
名古屋大学工学研究科
〒 464-8603 名古屋市千種区不老町
mihoko@sakabe.nuie.nagoya-u.ac.jp {sakai,sakabe}@nuie.nagoya-u.ac.jp

Typing Exception in an Object Oriented Calculus with Recursive Types

Mihoko HORIE　Masahiko SAKAI　Toshiki SAKABE
Graduate School of Engneering, Nagoya University
Furo-cho, Chikusa-ku, Nagoya, Japan, 464-8603

概　要

object calculus[1] はオブジェクト指向計算モデルであるが，多くのプログラミング言語が持つ例外処理機能を表現することができない．そこで著者らはこれまでに object calculus のもっともシンプルなモデルである $\mathrm{Ob}_{1<:}$ を拡張し例外処理機能を追加するとともに操作意味論と型システムを与え，型システムが操作意味論に対して健全であることを証明した [9]．本稿では，$\mathrm{Ob}_{1<:}$ を拡張し再帰を扱えるようにした $\mathrm{Ob}_{1<:\mu}$ に例外処理機能を追加し，操作意味論と型システムを与え，型システムが操作意味論に対して健全であることを証明する．この型システムにより，$\mathrm{Ob}_{1<:\mu}$ のプログラムの各部分で発生する例外の型を静的に検査できる．キーワード　object calculus，オブジェクト指向言語，例外処理，型システム

1　はじめに

オブジェクト指向計算を形式化した object calculus [1] には，シンプルなものから複雑なものまでいろいろな種類のがあり，Java などのオブジェクト指向言語の多くの機能を表現することができる．しかし，計算の途中で例外が発生したときに，計算を止めて例外処理に移るような例外処理機能は表現できない．

本稿では，object calculus の中で，再帰型をもち簡単なモデルである $\mathrm{Ob}_{<:1\mu}$ に，例外を発生させる throw 構文と，処理する例外を型によって指定できる try catch 構文を追加し例外処理の機能を組み入れる．構文の追加に伴い操作意味論，型システムの拡張も行い，型システムが操作意味論に対して健全であることの証明を行う．この型システムで推論が完成すれば，どんな例外が発生するか実行前に知ることができる．

操作意味論の拡張では，例外の型のサブタイプ関係を調べて処理するかどうか判断するため，サブタイプ判定式の決定可能性を示す．これによって，計算結果が決定できることが保証される．型システムの拡張で，型は通常型と起こりうる例外の型を列挙した例外型の組だと考える．

2　object calculus $\mathrm{Ob}_{<:1\mu}$ への例外処理機能の追加

object calculus $\mathrm{Ob}_{<:1\mu}$ に処理する例外を例外の型によって指定できる例外処理の機能を加えて拡張する．この拡張した $\mathrm{Ob}_{<:1\mu}$ を $\mathrm{Ob}_{<:1\mu\langle\rangle}$ と表記する．拡張した部分には☆印を，改良した部分には★印を表記する．

2.1　$\mathrm{Ob}_{<:1\mu\langle\rangle}$ の構文

$\mathrm{Ob}_{<:1\mu}$-calculus の項は，変数，オブジェクト，メソッド実行，メソッド更新，fold，unfold からなる (表 1)．オブジェクト $[l_i = \varsigma(x_i:A_i)b_i{}^{i\in 1..n}]$ は，l_i がメソッド名，b_i がメソッド本体，$\varsigma(x_i:A_i)$ はメソッド本体に現れる変数 x_i が self 変数，x_i の型が A_i であることを表す．fold は再帰をするためにオ

ブジェクトに再帰型を関連づける．また unfold は fold されたオブジェクトから元のオブジェクトを取り出すものである．これに例外を発生と捕捉を表す 2 つの構文を追加する．

- $throw\ (a : U)$
  型 U の項 a を例外として発生させる．
- $try\ a\ catch(x : U)b$
  項 a を実行中に型 U のサブタイプの例外が発生した場合，それを変数 x に束縛して項 b を実行する．

本稿の例外処理では，型によって例外を捕捉するかどうか判断する．try の中で発生した例外が指定された型のサブタイプならば，処理を行うがそうでなければそのまま例外は発生したままである．型の構文は表 3 のように定められる．(2.3 節で詳しく述べる．)

表 1: 項の構文

$a, b ::=$	項
x	変数
$[l_i = \varsigma(x_i : A_i)b_i{}^{i\in 1..n}]$	オブジェクト (l_i 別)
$a.l$	メソッド実行
$a.l \Leftarrow \varsigma(x:A)b$	メソッド更新
$fold(U, a)$	recursive fold
$unfold(a)$	recursive unfold
$throw\ (a:U)$	例外発生☆
$try\ a\ catch(x:U)b$	例外捕捉☆

例 2.1 e を型 $U\langle\rangle$ の *recursive fold* で表される式 $fold(U, [print = \varsigma(x_1 : U'\langle\rangle)x_1])$, a を式 $[l = \varsigma(x_2 : A)throw\ (e : U)]$, e' を式 $try\ a.l\ catch(x_3 : U)unfold(x_3).print$ とする．このとき e' を実行すると，メソット呼出の式 $a.l$ の実行で $throw\ (e : U)$ により型 U の例外 e が発生する．この例外 e は，*catch* の後ろで変数 x_3 に束縛され，$unfold(e).print$ が実行される．

2.2 $\mathrm{Ob}_{<:1\mu\langle\rangle}$ の操作意味論

計算結果の項を表 2 のように定める．

操作意味論は，項を結果に関連づける判定式: $a \rightsquigarrow v$ を推論するための規則で与えられる．例外処理機能を表現するために，$\mathrm{Ob}_{<:1\mu}$ に対して，以下の推論規則を追加する．

表 2: 操作意味論の記法 (結果の項)

$v ::=$	結果
$[l_i : \varsigma(x_i : A_i)b_i{}^{i\in 1..n}]$	通常の結果
$fold(U, v)$	通常の結果
$throw\ (u : U)$	例外発生する結果☆

ただし，u は通常の結果．

(Red Throw1) ☆

$$\frac{\vdash a \rightsquigarrow u}{\vdash throw(a : U) \rightsquigarrow throw(u : U)}$$

where $u \in \{[l_i = \varsigma(x_i : A_i)b_i{}^{i\in 1..n}], fold(U, u)\}$
a の結果が u の時，$throw\ (a : U)$ の結果は，$throw(u : U)$ なる．これは，例外オブジェクトの結果として例外が発生していることを表す．

(Red Throw2) ☆

$$\frac{\vdash a \rightsquigarrow throw(u : U)}{\vdash throw(a : U') \rightsquigarrow throw(u : U)}$$

すでに例外が発生しているので，その例外の結果 $throw(u : U)$ をそのまま $throw(a : U')$ の結果する．

(Red Try1) ☆

$$\frac{\vdash a \rightsquigarrow throw(u : U) \quad \vdash b\{\!\{u\}\!\} \rightsquigarrow v \quad \vdash U <: U'}{\vdash try\ a\ catch(x : U')b \rightsquigarrow v}$$

例外が発生し，例外処理を行う場合である．a の結果が例外オブジェクトの結果 $throw(u : U)$ で U が U' のサブタイプであるとき，$try\ a\ catch(x : U')b$ の結果は b の結果 v とする．サブタイプ判定式 $\vdash U <: U'$ が評価規則の中に含まれているのでこの決定可能性を 2.4 節で示す．

(Red Try2) ☆

$$\frac{\vdash a \rightsquigarrow throw(u : U) \quad \nvdash U <: U'}{\vdash try\ a\ catch(x : U')b \rightsquigarrow throw(u : U)}$$

例外が発生しても，例外処理を行わない場合である．項 a の結果が例外オブジェクトの結果 $throw(u : U)$ で型 U が型 U' のサブタイプでないとき，a の結果 $throw(u : U)$ がそのまま $try\ a\ catch(x : U')b$ の結果である．サブタイプ判定式 $\nvdash U <: U'$ が評価規則の中に含まれているのでこの決定可能性を 2.4 節で示す．

(Red Try3) ☆

$$\frac{\vdash a \rightsquigarrow v}{\vdash try\ a\ catch(x:U')b \rightsquigarrow v}$$

where $v \in \{[l_i = \varsigma(x_i:A_i)b_i{}^{i\in 1..n}], fold(U,u)\}$
例外が発生しない場合である．項 a の結果が通常のオブジェクトの結果であれば，a の結果 u を $try\ a\ catch(x:U')b$ の結果とする．

(Red Select2) ☆

$$\frac{\vdash a \rightsquigarrow throw(u:U)}{\vdash a.l_j \rightsquigarrow throw(u:U)}$$

メソッド実行の中の a で例外が発生した場合である．a の例外オブジェクトの結果 $throw(u:U)$ を $a.l_j$ の結果とする．

(Red Update2) ☆

$$\frac{\vdash a \rightsquigarrow throw(u:U)}{\vdash a.l_j \Leftarrow \varsigma(x:A)b \rightsquigarrow throw(u:U)}$$

メソッド更新の中の a で例外が発生した場合である．a の例外オブジェクトの結果 $throw(u:U)$ を $a.l_j \Leftarrow \varsigma(x:A)b$ の結果とする．

(Red Fold2) ☆

$$\frac{\vdash a \rightsquigarrow throw(u:U')}{\vdash fold(U,a) \rightsquigarrow throw(u:U')}$$

再帰を含む fold の a で例外が発生した場合である．再帰を抜け出し，a の例外オブジェクトの結果 $throw(u:U')$ を $fold(U,a)$ の結果とする．

(Red Unfold2) ☆

$$\frac{\vdash a \rightsquigarrow throw(u:U)}{\vdash unfold(a) \rightsquigarrow throw(u:U)}$$

unfold の a で例外が発生した場合である．a の例外オブジェクトの結果 $throw(u:U)$ を $unfold(a)$ の結果とする．

例 2.2 例 2.1 の式 e' をが次のようになることがこの操作意味論で判定できる．

$$\vdash try\ a.l\ catch(x_3:U)unfold(x_3).print \rightsquigarrow [print = \varsigma(x_1:U'\langle\rangle)x_1]$$

2.3 $\mathrm{Ob}_{<:1\mu\langle\rangle}$ の型システム

$\mathrm{Ob}_{<:1\mu}$ における型は通常型と呼び，U, V と表記する．$\mathrm{Ob}_{<:1\mu\langle\rangle}$ の型は，通常型と例外型の組 $U\langle Ue\rangle$ で定義する．ここに，例外型 Ue は，通常型のリストである．項 a の型 $U\langle Ue\rangle$ であること ($a:U\langle Ue\rangle$ と書く) は，例外が発生していなければ U の型であり，例外が発生するとすればその型は Ue 中の型のいずれかであることを意味する．例外の型は，Ue，もしくは，$U_1, U_2, ..., U_m$ というリストで表す．また ϵ は，必ず例外が発生し，通常の型がないことを表す．型の構文を表 3 のように定める．オブジェクト型 $[l_i:A_i^{i\in 1..n}]$ は，l_i がメソッド名，A_i がメソッドの返り値の型を表す．再帰型 $\mu(X)U\{X\}$ は，再帰を行う際 $U\{X\}$ を X に束縛することを表す．再帰型を $\mu(X)A$ とせず $\mu(X)U$ としたのは，例外まで束縛して再帰しなくても十分であると考えたこと，ならび，計算モデルをなるべく簡単にしたかったことによる．

表 3: 型の構文

$A, B ::= U\langle Ue\rangle$	型 (通常型と例外型の組) ☆
$U, V ::=$	通常型
X	タイプ変数
K	グランド型
$\top$	最大の型
ϵ	最小の型☆ (必ず例外が発生する)
$[l_i : A_i^{i\in 1..n}]$	オブジェクト型
$\mu(X)U$	再帰型
$Ue ::=$	例外型☆
U, Ue	
U	

型システムは次の 4 つの判定式の型推論規則によって構成される．

- Well-formed environment 判定式: $E \vdash \diamond$
- Well-formed type 判定式: $E \vdash A$
- Subtyping 判定式: $E \vdash A <: B, E \vdash U <: V$
- Term typing 判定式: $E \vdash a : A$

追加した型 $U\langle Ue\rangle$, ϵ について判定式 $E \vdash A$, $E \vdash A <: B, E \vdash U <: V$ の規則を次のように定義する．

(Type Excep) ☆

$$\frac{E \vdash U \quad E \vdash V \quad \forall j \in 1..m}{E \vdash U\langle V_j{}^{j\in 1..m}\rangle}$$

$U\langle V_j{}^{j\in 1..m}\rangle$ を構成するすべての型が Well-formed ならば，$U\langle V_j{}^{j\in 1..m}\rangle$ も Well-formed である．

(Sub Excep) ☆

$$\frac{E \vdash U\langle {V_i}^{i\in 1..m}\rangle \quad E \vdash U'\langle {V'_j}^{j\in 1..n}\rangle \quad \forall i \in 1..m \quad E \vdash U <: U' \quad E \vdash V_i <: V'_{k_i} \quad k_i \in 1..n}{E \vdash U\langle {V_i}^{i\in 1..m}\rangle <: U'\langle {V'_j}^{j\in 1..n}\rangle}$$

型 $U\langle {V_i}^{i\in 1..m}\rangle$ が，型 $U'\langle {V'_j}^{j\in 1..n}\rangle$ のサブタイプであるということは，型 U が U' のサブタイプであり，${V_i}^{i\in 1..m}$ のそれぞれについて，${V'_j}^{j\in 1..n}$ のどれか一つのサブタイプになっていることである．例外の型では，サブタイプで発生する例外は，スーパータイプでも発生できることを意味する．

次に，throw,catch 構文について項の型付け判定式 $E \vdash a : A$ の規則を追加する．

(Val Throw) ☆

$$\frac{E \vdash a : U\langle Ue\rangle}{E \vdash throw(a : U) : \epsilon\langle U, Ue\rangle}$$

$throw(a : U)$ は，必ず例外を発生するので通常の型は空という意味で ϵ となる．a で例外が発生していなければ，$throw(a : U)$ は，型 U の例外を発生させ，$\epsilon\langle U\rangle$ となる．例外が発生していれば，それが伝わるので，a の例外の型を $throw(a : U)$ の例外の型に加える．

(Val Try) ☆

$$\frac{E \vdash a : U\langle {V_j}^{j\in 1..m}\rangle \quad E, x : U' \vdash b : U\langle Ve'\rangle}{E \vdash try\ a\ catch(x : U')b : U\langle {V_j}^{j\in\{k\mid k\in 1..m, E\vdash V_k \not<: U'\}}, Ve'\rangle}$$

$try\ a\ catch(x : U')b$ は，a または b が実行されるものである．$try\ a\ catch(x : U')b$ の値を使うことは，a と b の値を同じように扱うことである．そこで $try\ a\ catch(x : U')b$ の通常の型は，a と b の共通な型 U とする．

その他 (Type ϵ)，(Sub ϵ) の規則を定義し，例外の型を外側に伝えるために以下の規則の変更を行う．

(Val Fold) ★

$$\frac{E \vdash b : U\{\!\{A\}\!\}\langle Ve\rangle}{E \vdash fold(A, b) : A}$$

where $A \equiv \mu(X)U\{X\}\langle Ve\rangle$

(Val Unfold) ★

$$\frac{E \vdash a : A}{E \vdash unfold(a) : U\{\!\{A\}\!\}\langle Ve\rangle}$$

where $A \equiv \mu(X)U\{X\}\langle Ve\rangle$

(Val Object) ★

$$\frac{E, x_i : U\langle\rangle \vdash b_i : B_i \quad \forall i \in 1..n}{E \vdash [l_i = \varsigma(x_i : U\langle\rangle)b_i{}^{i\in 1..n}] : U\langle\rangle}$$

where $U \equiv [l_i : {B_i}^{i\in 1..n}]$

(Val Select) ★，(Val Update) ★についても同様に規則の変更を行う．

例 2.3 例 2.1 において，$U \triangleq \mu(X)[print : X]$，$U' \triangleq [print : U]$ としたとき，e' の型は $U\langle\rangle$ になることが型システムで判定できる．このことは式 e' の例外が外部に波及しないことを意味する．

2.4 サブタイプ判定式:$E \vdash U <: V$ の決定可能性

操作意味論 (2.2 節) の推論規則 (Red Try1)，(Red Try2) において $\vdash U <: V$，$\not\vdash U <: V$ が評価規則に含まれている．そこで環境 E が空のときの $\phi \vdash U <: V$ の決定可能性を示す．$\phi \vdash U <: V$ が決定可能性であることは，その否定の $\phi \not\vdash U <: V$ が決定可能であることを含意する．

$\phi \vdash U <: V$ の決定可能性を示すために，サブタイプであることを判定する項書換え系 R_{st} を作成し，付録に示す．そして以下の 2 つの補題を示すことにより，定理 2.3 を得た．

補題 2.1 U, V を型とし，それらをコード化した項を u, v とする．このとき，
サブタイプ判定式 $\phi \vdash U <: V$ が推論可能 *iff* $subtype?(u, v)$ が R_{st} で $true$ に到達可能．
(⇐) ステップ数に関する帰納法により証明される．
(⇒) 導出木の深さに関する帰納法により証明される． □

補題 2.2 サブタイプ判定項書換え系が停止性を持つ．
辞書式経路順序により証明される． □

定理 2.3 $\phi \vdash U <: V$ は，決定可能である．
補題 *2.1*, *2.2* による． □

項書換え系の停止性証明ツールを利用することにより，補題 2.2 の証明を簡単に行うことができた．項書換え系を使わなくてもサブタイプ判定式の導出木に対して，葉に向かって判定式のサイズが小さくなるかまたは 2 つの型がサブタイプとして近づくことを用いて，決定可能性は証明できると考えられる．しかしながら，項書換え系へのコーディングにより証明を容易にすることができた．

2.5 型システムに対する操作意味論の健全性

$Ob_{<:1\mu\langle\rangle}$ において，次の補題が成り立つ．

補題 2.4 $E, X <: U, E' \vdash \Im, E \vdash U' <: U$ ならば，$E, X <: U', E' \vdash \Im$.
$E, x : U\langle\rangle, E' \vdash \Im, E \vdash U' <: U$ ならば，$E, x : U'\langle\rangle, E' \vdash \Im$.

$\Im$ の構造に関する帰納法で証明される． □

補題 2.5 $E, X <: U, E'\{X\} \vdash \Im\{X\}, E \vdash U' <: U$ ならば，$E, E'\{\!\{U'\}\!\} \vdash \Im\{\!\{U\}\!\}$.
$E, x : U\langle\rangle, E' \vdash \Im\{x\}$ $E \vdash d : U\langle\rangle$ ならば，$E, E' \vdash \Im\{\!\{d\}\!\}$.
$\Im\{x\}$ の構造に関する帰納法で証明される． □

これらの補題を利用して，次の定理が成り立つ．

定理 2.6 $\vdash a \rightsquigarrow v$ のとき，$\phi \vdash a : C$ ならば，$\phi \vdash v : C$.
判定式 $\vdash a \rightsquigarrow v$ の証明木の深さに関する帰納法によって証明される． □

この定理から，計算結果の型は計算する前と同じであるか，もしくはサブタイプの型になることがわかる．

3 関連研究

本稿は Java の例外処理を基に，オブジェクト指向計算モデル上で例外処理機能のモデル化を行った．例外処理機能については，多くの研究がなされている [2],[3],[4],[5],[8]．Java，Nakano[5]，Kameyama and Sato[4]，Common Lisp における例外処理機構の構文を表 4 にまとめる．

本稿と Java はサブタイプ関係を用いて例外捕捉することは，同じである．しかし，本稿では throw に対し型情報 U を記述しなければならないのに対して，Java ではその必要はない．型情報をはずし型推論を行うことが考えられるが，$Ob_{1<:\mu\langle\rangle}$ の型推論アルゴリズムはまだ与えられていない．ただし，$Ob_{1<:\mu}$ の型推論アルゴリズムはすでに与えられている [6]．また，$Ob_{1<:\mu\langle\rangle}$ は簡単な Object calculus に例外処理機能を追加したものであるが，複雑な Object calculus では，型推論が NP 完全であることもすでに示されている [7]．このことから，型情報

表 4: 例外処理機構の比較

本稿 $Ob_{1<:\mu\langle\rangle}$	Common Lisp
$throw\ (a : U)$	$(throw\ 'u, b)$
$try\ a\ catch(x : U)b$	$(catch\ 'u, a)$
Java	
$throw\ a$	
$try\ a\ catch(U_1\ x_1)b_1..catch(U_n\ x_n)b_n$	
$try\ a\ catch(U_1\ x_1)b_1..catch(U_n\ x_n)b_n\ finally\ b_{n+1}$	
$L_{c/t}/L^K_{c/t}$[5]	$NJ_{c/t}/NK_{c/t}$[4]
$throw\ (u, b)$	$!u.b$
$catch\ (u, a)$	$?u.a$
u はタグ変数	

U を記述した形での定式化を行った．また Java では，Throwable のサブクラスの値を例外として発生させるが，$Ob_{1<:\mu\langle\rangle}$ では投げられる例外値の型に制限はない．Java の *try catch ... catch* は，$Ob_{1<:\mu\langle\rangle}$ の *try catch* をネストすることで表現することができるが，try-catch-finally は，残念ながら $Ob_{1<:\mu\langle\rangle}$ では表現することはできない．ただし，$Ob_{1<:\mu\langle\rangle}$ に逐次実行の機能を追加すれば表現可能である．Common Lisp，SML，[4]，[5] は例外を例外名あるいはタグで管理して捕捉している．

4 まとめ

object calculus $Ob_{<:1\mu}$ を拡張して，例外処理機能を追加した $Ob_{<:1\mu\langle\rangle}$ を形式化し，構文，型システム，操作意味論を与えた．操作意味論の評価規則に含まれているサブタイプ判定式に対して，その決定可能性を示した．また，操作意味論の型に関する健全性の証明を行った．さらに複雑な object calculus への例外処理機能の追加は今後の課題である．

謝辞

本研究を進めるに際して，熱心にご討論頂いた坂部研の皆様に感謝致します．本研究は一部，日本学術振興会科学研究基盤 (C)11680352，柏森情報科学振興財団の補助を受けています．

参考文献

[1] Abadi,M., Cardelli,L.: *A THEORY OF OBJECTS*, Springer, 1998.

[2] Drossopoulou,S, Valkevych,T : Java Exceptions Throw no Surprises http://www-

doc.ic.uk/project/ slurp/,2000.

[3] Jacobs,B.：A Formalization of Java's Exception Machanism, *ESOP2001*, *LNCS2028*, 2001, pp. 284-301.

[4] Kameyama,Y., Sato,M.：Non-deterministic Catch/Throw Calculi, *Theoretical Computer Science*, Vol. 272, 2002, pp. 223-245.

[5] Nakano,H.：Logical Structure of the Catch and Throw Mechanism,, ph.D thesis , University of Tpkyo, 1995.

[6] Palsberg,J.：Efficient Inference of Object Types, *Information and Computation*, Vol. 123, No. 2, 1994, pp. 198-209.

[7] Palsberg,J., Jim,T.：Type inferrence with Simple Selftypes is NP-complete , *Nordic Journal of Computing*, Vol. 4, No. 3, 1997, pp. 259-286.

[8] 寺澤啓司，酒井正彦，坂部俊樹：オブジェクト指向計算モデルにおける例外処理の形式化，修士論文，名古屋大学大学院工学研究科情報工学専攻，2000.

[9] 堀江美保子，酒井正彦，坂部俊樹：オブジェクト指向計算モデルにおける例外処理の型付，日本ソフトウェア科学会第 19 回大会論文集，2002.

付録:サブタイプ判定項書換え系 R_{st}

```
 / subtype?(a,b): a<:b の時 true,subtype かどうかを判定する. /
// sub e sub top
subtype?(E,E) → true
subtype?(E,TOP) → true
subtype?(E,OB(_)) → true
subtype?(X,TOP) → true // 使いたい変数に対して作る
subtype?(E,X) → true  // 使いたい変数に対して作る
subtype?(X,X) → true  // 使いたい変数に対して作る
subtype?(MU(_,_),TOP) → true
subtype?(E,MU(_,_)) → true
subtype?(TOP,TOP) → true
subtype?(OB(_),TOP) → true

// sub Excep
subtype?(TY(n1,ns1),TY(n2,ns2))
→ subtype?(n1,n2) and subtype1?(ns1,ns2)

// sub Object
subtype?(OB(CONS(fs1,fss1)),OB(CONS(fs2,fss2)))
→ same?(fs1,fs2) and subtype?(OB(fss1),OB(fss2))
subtype?(OB(x),OB(NIL)) → true

// sub Rec
subtype?(MU(X,nt1),MU(Y,nt2))
→ subtype+(nt1,nt2)
☆このルールを呼ぶたびに subtype?(NT1,NT2) のルールに subtype+?(X,Y) → ture を追加した subtype+?を作る。
//例
subtype+?(X,Y) → true
subtype+?(X,TOP) → true // 使いたい変数に対して作る
subtype+?(E,X) → true  // 使いたい変数に対して作る
subtype+?(MU(_,_),TOP) → true
subtype+?(E,MU(_,_)) → true
subtype+?(E,E) → true
subtype+?(E,TOP) → true
subtype+?(E,OB(_)) → true
subtype+?(TOP,TOP) → true
subtype+?(OB(_),TOP) → true
subtype+?(OB(CONS(fs1,fss1)),OB(CONS(fs2,fss2)))
→ same?(fs1,fs2) and subtype+?(OB(fss1),OB(fss2))
subtype+?(OB(x),OB(NIL)) → true

// subtype1?(as,bs):例外の型 as,bs において，
[]<as><:[]<bs>の時 true.
subtype1?(NIL,ns2) → true
subtype1?(CONS(n1,ns1),ns2)
→ subtype2?(n1,ns2) and subtype1?(ns1,ns2)

// subtype2?(a,CONS(b,bs)): 通常の型 a の
supertype が CONS(b,bs) に存在する時，true.
subtype2?(n1,CONS(n2,ns2))
→ subtype?(n1,n2)or subtype2?(n1,ns2)

// same?(x,y): x と y が等しい時，true
same?(OB(CONS(fs1,fss1)),OB(CONS(fs2,fss2)))
→  same?(fs1,fs2) and same?(OB(fss1),OB(fss2))
same?(FS(l1,ty1),FS(l2,ty2))
→  same?(l1,l2) and same?(ty1,ty2)
same?(TY(n1,ns1),TY(n2,ns2))
→  same?(n1,n2) and same?(ns1,ns2)
same?(CONS(x1,xs1),CONS(x2,xs2))
→  same?(x1,x2) and same?(xs1,xs2)
same?(MU(x1,nt1),MU(x2,nt2))
→  same+?(nt1,nt2)
☆このルールを呼ぶたびに same+?(x1,x2) → true を same に追加した same+?を作る。
same?(E,E)  → true    same?(TOP,TOP)  → true
same?(l1,l1)  → true  same?(l2,l2)  → true
same?(NIL,NIL)  → true

 //and と or
ture and ture  → true
_ or ture  → true      ture or _ → true
```