静的かつ動的な式を許すような部分評価器のための束縛時解析

浅井 健一

本稿では，関数型言語の部分評価器で構造データの扱いを向上すべく，静的かつ動的な値を許すような束縛時解析を提案する．静的かつ動的な値を使うと，静的に構造データにアクセスしつつ，その構造データを（動的な値として）コードに残すことができるようになる．この方法は実質的に，従来，許されていなかった構造データの lift を許すようにした，と見ることもできる．本稿で提案する束縛時解析は，型システムとして定式化され，型推論の際に生成される束縛時に関する制約を解くことで結果を得る．さらに，アルゴリズムの正当性を示すとともに，制約の解消が従来の効率的な束縛時解析と同じくほぼ線形時間でできることを示す．

1 はじめに

部分評価 (partial evaluation) [12] とは，プログラムと引数の一部を受け取って，その既知の引数の情報から計算できる部分を先に計算してしまうことでプログラムの効率をあげるプログラム変換である．部分評価の技法を使うとインタプリタからコンパイラを生成できる [7]，など応用が広いため，特に関数型言語の世界では多くの研究がなされ，実際にいくつかのシステムが動くに至っている [3]．また，近年ではオブジェクト指向言語の最適化等にも利用されるようになってきている [6] [4]．

部分評価の中でも offline の部分評価と呼ばれる方法では，部分評価をプログラム解析のフェーズ（束縛時解析 BTA と呼ばれる）とプログラム特化のフェーズにわけて行う．解析のフェーズでは，プログラムを部分評価時に実行可能（静的 static）な部分と不可能（動的 dynamic）な部分とにわけ，特化のフェーズでは静的な部分を実行する．具体的な例を見てみよう．例えば

```
(lambda (x) ((lambda (y) y) x))
```

という式を考えてみる．この式を束縛時解析に通すと，最初の `(lambda (x) ...)` の部分はまだ実行できないが，その中の `((lambda (y) y) x)` の部分は実行できる，という結果が返ってくる．そこで束縛時解析の結果として，実行できる部分は実行し，実行できない部分は実行時に実行されるようなコード（あるいはプログラムテキスト）[†1] を出力する．これは簡単には以下のように quasiquote と unquote を使えば良い．

```
`(lambda (x) ,((lambda (y) y) 'x))
```

特化のフェーズでは，単にこれを実行すれば，部分評価の結果である `(lambda (x) x)` というプログラムテキストを得ることができる．このとき，`(lambda (x) ...)` の部分は quasiquote が付いたただのリスト（プログラムテキスト）であるのに対し，`(lambda (y) y)` は実行されて closure が作られていることに注意しよう．

Binding-Time Analysis for a Specializer that Allows Both Static and Dynamic Expressions

Kenichi ASAI, 東京大学大学院理学系研究科情報科学専攻 / 科学技術振興事業団 さきがけ研究 21 「情報と知」領域グループ, Department of Information Science, Graduate School of Science, University of Tokyo / "Information and Human Activity," PRESTO, Japan Science and Technology Corporation (JST)

コンピュータソフトウェア, Vol.17, No.3 (2000), pp.20–37.
[論文]1999年5月22日受付.

†1 本稿では「コード（表現）」と「プログラムテキスト」を同じ意味で使用する．これらは Scheme の場合にはリストである．

```
(let ((lst (list (cons 1 2) (cons 3 4)
                            (cons 5 6))))
     ; ペアのリストで表された点列
  (lambda (r)
    (define (near? pair rr)
      ; (x座標)^2 + (y座標)^2 < r^2 かを判定する
      (< (+ (* (car pair) (car pair))
            (* (cdr pair) (cdr pair)))
         rr))
    (define (take-near l rr)
      ; l の中から原点に近いものを抽出する
      (cond ((null? l) '())
            ((near? (car l) rr)
             (cons (car l)
                   (take-near (cdr l) rr)))
            (else (take-near (cdr l) rr))))
    (take-near lst (* r r))))
```

図1 原点から距離 r 以内の点列を返すプログラム

```
(let ((c0 (cons 1 2)) (c1 (cons 3 4))
                      (c2 (cons 5 6)))

  (lambda (r)
    (let ((rr (* r r)))
      (cond ((< 5 rr)
             (cond ((< 25 rr)
                    (if (< 61 rr)
                        (list c0 c1 c2)
                        (list c0 c1)))
                   ((< 61 rr) (list c0 c2))
                   (else (list c0))))
            ((< 25 rr)
             (if (< 61 rr) (list c1 c2)
                           (list c1)))
            ((< 61 rr) (list c2))
            (else '())))))
```

図2 特化した結果

このように offline の方法は，解析のフェーズであらかじめ実行可能な部分を分離し，それらを特化時に（シンボリックに実行するのではなく）直接実行してしまうため，特化を効率的に行えるという特徴を持つ．その一方で，直接実行されるときの実行時データとそのコード表現が食い違うため，分離がきれいにできないと十分な特化ができない，という問題点がある．例えば，

```
((lambda (f) (cons f (f 3)))
 (lambda (x) x))
```

という式を考えてみよう．ここでは部分評価時に `(f 3)` を実行したいので `cons` のみを動的 `(lambda (x) x)` を含む他の部分を静的として以下のようにしたい．

```
((lambda (f) `(cons ,f ,(f 3)))
 (lambda (x) x))
```

ところがこの式を実行すると

```
(cons #[procedure #x1359A08] 3)
```

のような結果が返ってくる．実行時データとそのコード表現が異なるため，`(lambda (x) x)` というコードが残って欲しいところに実行時の値である `#[procedure #x1359A08]` が出てきてしまうのである．ここでの問題点は `(lambda (x) x)` を静的に使うため closure を作成したが，実は同時にそのコード表現も必要であった，という点である．このような場合，従来の部分評価器では（closure からそのプログラムテキストを復元するのが容易でないため）`(lambda (x) x)` は動的となり，従って `(f 3)` の実行も行われない．

同様の問題は closure 以外の構造データについても起きる．例えば，図 1 の例を見てみよう．このプログラムは，cons セルのリストで表現された点列 `lst` のうち，原点から距離 `r` 以内にある点を抽出するものである．ここでは，このプログラムで点列 `lst` が具体的に与えられた場合の部分評価を考えている．

今，`lst` が具体的に与えられているので `take-near` の再帰や `near?` における $(x\text{座標})^2 \times (y\text{座標})^2$ の計算などを実行して，図 2 のような結果を得たい．これは，各点に関してあらかじめ距離の計算を行っておき，後で `r` が与えられたときに，素早く結果を返せるようにした，と見ることができる．

しかし，図 1 のプログラムを従来の束縛時解析にかけると `lst` の各要素が全て動的となってしまうため `take-near` の再帰は展開されるが $(x\text{座標})^2 \times (y\text{座標})^2$ の計算は一切行われない．`lst` の各要素が動的になってしまう理由は，`take-near` 中の `cond` 文の 2 つ目の節（四角で囲まれている部分）で，`(car l)` が最終結果に出力されているためである．`(car l)` には `lst` 中の各要素がきうるが，それらを最終結果のコードに残すため，そのコード表現が必要となるのである．このことは，仮に `lst` の各要素を静的と分類し，最終結果に残る `(car l)` を静的のまま，特化を実行して見るとわかる．その結果は，図 2 とほぼ同じものが得られるが，第 1 行目が

`(let ((c0 (1 . 2))(c1 (3 . 4))(c2 (5 . 6)))` という誤ったものになってしまう．各要素が静的のため `(cons 1 2)` のようにコードにはならず `(1 . 2)` という値になってしまうのである．Lift 命令を使って `(1 . 2)` という値を `(cons 1 2)` というコードに変換することができれば問題が解決するように感じられるかも知れない．しかし，lift は通常，定数にのみ認められ，構造データには使用することができない．これは構造データに lift を使うと，データ構造の定義が複製される可能性があること，closure のようなデータ構造は値からコードに変換するのが自明ではないこと，などのためである．

ここでの問題は，`lst` の各要素が静的な計算に使われると同時に，コードとして最終結果にも残る（動的にも使われている），という点である．このような式を，本稿では静的かつ動的 (both static and dynamic) な式と呼ぶことにする．静的かつ動的な式については，部分評価時にその値を使った計算をしつつ，最終結果に残る部分についてはコード表現を出力したい．これを実現するため，静的に使われる値とそのコード表現の両方を渡す，という方法を提案する．この手法によって，従来，最終結果に残るために動的となっていたデータ構造に対しても，部分評価時にアクセスできるようになる．先の例では，`lst` の各要素に部分評価時にアクセスしつつ，そのコード表現を最終結果に残し，図 2 の結果を得ることができるようになる．

しかし，このように値とコード表現の両方を渡すようにすると，従来の片方しか渡さなかった場合に比べてオーバーヘッドがかかる．従って，可能な全ての値を静的かつ動的と扱うのは好ましくない．本稿では，束縛時解析を型システムとして定式化し，実際に静的かつ動的に使われる可能性のある部分を特定することで，オーバーヘッドをなるべく抑えるようにしている．

同じ式に複数の束縛時を許す一般的な束縛時解析としては polyvariant な束縛時解析があげられる．しかし，polyvariant な解析はコストが非常に高いことが知られている [2]．ここでは扱える束縛時を静的かつ動的なものまでに限ることで monovariant な解析の効率を損なうことなく，構造データの扱いを向上することを目指している．実際，本稿で示す束縛時解析は，従来のものと同じくほぼ線形でできることがわかっている．

以下，2 節で本稿で扱う言語を説明し，3 節では特化器を示す．4 節で束縛時型を導入した後，5 節に注釈のついた言語に対する型規則（型チェック規則）を示し，6 節でその健全性を示す．7 節ではこの規則を注釈のついていない言語のものに変換し，制約生成の規則を得る．型チェック規則と制約生成規則の等価性は 8 節で示される．9 節で，制約解消時に使う型の解釈を示し，10 節で実際に制約解消アルゴリズムを述べる．11 節で制約解消の例を示した後，12 節でそのアルゴリズムの正当性を示すとともに，13 節ではその計算量も述べる．関連研究を 14 節で述べ，15 節でまとめる．

2 言語

本稿で扱う言語は，以下に示すような関数型言語の一部で，高階関数，関数適用，primitive の適用，そして構造データとして *cons*, *car*, *cdr* を扱う．ここでは簡単のため再帰を扱っていないが，再帰が入っても本稿の枠組は問題なく拡張できることがわかっている．

$$\begin{aligned} L \quad = \quad & x \mid \lambda x.\, L \mid L@L \mid \mathit{cons}\, L\, L \mid \\ & \mathit{car}\, L \mid \mathit{cdr}\, L \mid \text{定数} \mid L + L \end{aligned}$$

言語 L のプログラムを受け取ると，束縛時解析は注釈の付いたプログラムを返す．従来の束縛時解析では，注釈は S（静的）または D（動的）の 2 種類であったが，これに加えて本稿では B（静的かつ動的）も許す．以下に，注釈の付いた言語の定義を示す．

$$\begin{aligned} M \quad = \quad & x \mid \lambda^S x.\, M \mid \lambda^D x.\, M \mid \lambda^B x.\, M \mid \\ & M@^S M \mid M@^D M \mid \\ & \mathit{cons}^S M\, M \mid \mathit{cons}^D M\, M \mid \mathit{cons}^B M\, M \mid \\ & \mathit{car}^S M \mid \mathit{car}^D M \mid \mathit{cdr}^S M \mid \mathit{cdr}^D M \mid \\ & \text{定数}^S \mid \text{定数}^D \mid \text{定数}^B \mid \\ & M +^S M \mid M +^D M \mid M{\downarrow}_S^B \mid M{\downarrow}_D^B \end{aligned}$$

$\lambda^B x.\, M$ および $\mathit{cons}^B M\, M$ がそれぞれ静的かつ動的な関数，ペアを返す命令である．また，$M{\downarrow}_S^B$ と $M{\downarrow}_D^B$ はそれぞれ，静的かつ動的な値を受け取ったら，その静的な部分，動的な部分を返す命令である．この 2 つを本稿では型変換命令と呼ぶ．

以下では，注釈の付いた式 M から，全ての注釈と型変換命令を取り除いた式を $|M|$ と書くことにする．

$$
\begin{array}{rcl}
\mathcal{S} : \mathit{Exp} \to \mathit{Env} & \to & W + \{\mathsf{error}\} \\
\mathcal{S}\,[\![x]\!]\,\rho & = & \rho(x) \\
\mathcal{S}\,[\![\lambda^S x.\,M]\!]\,\rho & = & \lambda x'.\,\mathcal{S}\,[\![M]\!]\,\rho[x'/x] \\
\mathcal{S}\,[\![\lambda^D x.\,M]\!]\,\rho & = & \mathit{build}\text{-}\lambda(x^\diamond, \mathcal{S}\,[\![M]\!]\,\rho[x^\diamond/x]) \\
\mathcal{S}\,[\![\lambda^B x.\,M]\!]\,\rho & = & (\mathit{build}\text{-}\lambda(x^\diamond, \mathcal{D}(\mathcal{S}\,[\![M]\!]\,\rho[x^\diamond/x])), \lambda x'.\,\mathcal{S}\,[\![M]\!]\,\rho[x'/x]) \\
\mathcal{S}\,[\![M_1 @^S M_2]\!]\,\rho & = & (\mathcal{S}\,[\![M_1]\!]\,\rho)@(\mathcal{S}\,[\![M_2]\!]\,\rho) \\
\mathcal{S}\,[\![M_1 @^D M_2]\!]\,\rho & = & \mathit{build}\text{-}@(\mathcal{S}\,[\![M_1]\!]\,\rho, \mathcal{S}\,[\![M_2]\!]\,\rho) \\
\mathcal{S}\,[\![\mathit{cons}^S M_1\ M_2]\!]\,\rho & = & (\mathcal{S}\,[\![M_1]\!]\,\rho, \mathcal{S}\,[\![M_2]\!]\,\rho) \\
\mathcal{S}\,[\![\mathit{cons}^D M_1\ M_2]\!]\,\rho & = & \mathit{build}\text{-}\mathit{cons}(\mathcal{S}\,[\![M_1]\!]\,\rho, \mathcal{S}\,[\![M_2]\!]\,\rho) \\
\mathcal{S}\,[\![\mathit{cons}^B M_1\ M_2]\!]\,\rho & = & \textbf{let}\ (m_1, m_2) = (\mathcal{S}\,[\![M_1]\!]\,\rho, \mathcal{S}\,[\![M_2]\!]\,\rho) \\
 & & \textbf{in}\ (\mathit{build}\text{-}\mathit{cons}(\mathcal{D}(m_1), \mathcal{D}(m_2)), (m_1, m_2)) \\
\mathcal{S}\,[\![\mathit{car}^S M]\!]\,\rho & = & \#_1(\mathcal{S}\,[\![M]\!]\,\rho) \\
\mathcal{S}\,[\![\mathit{car}^D M]\!]\,\rho & = & \mathit{build}\text{-}\mathit{car}(\mathcal{S}\,[\![M]\!]\,\rho) \\
\mathcal{S}\,[\![\text{定数}^S]\!]\,\rho & = & \text{定数} \\
\mathcal{S}\,[\![\text{定数}^D]\!]\,\rho & = & \mathit{lift}(\text{定数}) \\
\mathcal{S}\,[\![\text{定数}^B]\!]\,\rho & = & (\mathit{lift}(\text{定数}), \text{定数}) \\
\mathcal{S}\,[\![M_1 +^S M_2]\!]\,\rho & = & (\mathcal{S}\,[\![M_1]\!]\,\rho) + (\mathcal{S}\,[\![M_2]\!]\,\rho) \\
\mathcal{S}\,[\![M_1 +^D M_2]\!]\,\rho & = & \mathit{build}\text{-}+(\mathcal{S}\,[\![M_1]\!]\,\rho, \mathcal{S}\,[\![M_2]\!]\,\rho) \\
\mathcal{S}\,[\![M{\downarrow}^B_D]\!]\,\rho & = & \#_1(\mathcal{S}\,[\![M]\!]\,\rho) \\
\mathcal{S}\,[\![M{\downarrow}^B_S]\!]\,\rho & = & \#_2(\mathcal{S}\,[\![M]\!]\,\rho)
\end{array}
$$

図3 特化器

3 特化器

図 3 と図 4 に注釈の付いた言語 M に対する特化器を示す．図中で **let** $x = a$ **in** M は $(\lambda x.\,M)@a$ の略記，$\mathcal{D}(M)$ は M が静的かつ動的だった場合にそのコード部分を取り出す命令[†2]，$x^\diamond$ は（Scheme 等の処理系にある `gensym` 命令によって）新たに生成された変数（のコード表現），$\#_1$ と $\#_2$ はペア (A, B) の *car* 部分，*cdr* 部分をとってくる命令である．

$$\#_1(A, B) = A, \quad \#_2(A, B) = B$$

また *lift* は定数やシンボルなどの非構造データの値をプログラムテキストに変換する命令，*build-* はコードを出力する命令で，例えば Scheme のコードを出力するなら図 5 のようになる．

†2 ここでは簡単のため特化時に M の束縛時をチェックするように書いているが，M の束縛時は束縛時解析が終了した時点でわかっているので，それに従って規則を（M が静的かつ動的な場合と動的な場合の）2 つに分ければ，特化時に束縛時をチェックする必要はない．

$$
\begin{array}{rcl}
W & = & W_S + W_D + W_B \\
W_S & = & \mathsf{Int} + (W \to W) + (W \times W) \\
W_D & = & \mathit{Code} \\
W_B & = & W_D \times W_S \\
\mathit{Env} & = & \mathit{Var} \to W
\end{array}
$$

$$
\mathcal{D}(M) = \begin{cases} \#_1(M) & \text{if } M \text{ is both} \\ M & \text{if } M \text{ is dynamic} \end{cases}
$$

図4 Semantic Domain と関数 $\mathcal{D}(\cdot)$

$$
\begin{array}{lcl}
\mathit{build}\text{-}\lambda(x^\diamond, N) & = & \texttt{(lambda (}x^\diamond\texttt{) }N\texttt{)} \\
\mathit{build}\text{-}@(N_1, N_2) & = & \texttt{(}N_1\ N_2\texttt{)} \\
\mathit{build}\text{-}\mathit{cons}(N_1, N_2) & = & \texttt{(cons }N_1\ N_2\texttt{)} \\
\mathit{build}\text{-}\mathit{car}(N) & = & \texttt{(car }N\texttt{)} \\
\mathit{build}\text{-}+(N_1, N_2) & = & \texttt{(+ }N_1\ N_2\texttt{)}
\end{array}
$$

図5 Scheme 用の *build-* 関数

図 3 の特化器はごく標準的なものである．静的な命令については普通のインタプリタと全く同じように実行し，動的な命令については，そのコードを出力している．従来の特化と異なるのは，静的かつ動的な命令で，この場合は，後に静的にも動的にも使われる可能性があるので，両方を作りペアにして渡している．これらは，その後の特化で必要に応じて型変換命令 $M{\downarrow}_D^B$ や $M{\downarrow}_S^B$ によってその動的な部分，静的な部分が取り出されることとなる．ここで使われるペアは *cons* 命令で作られるペアと同じものを使用しているが，これらは後に述べる型システムで明確に区別されるので，混同されることはない．

本稿では，定数も静的，動的，静的かつ動的の 3 種類に厳格に区別しているが，これは，定数を他の構造データと同じく統一的に扱うためである．しかし，必ずしも，図 3 にある通り静的かつ動的な定数の表現として，静的な定数と動的な定数の両方を持っておく必要はない．静的な定数のみを持っておき，

$$\mathcal{S}[\![\text{定数}{\downarrow}_D^B]\!]\,\rho \quad = \quad \mathit{lift}(\text{定数})$$
$$\mathcal{S}[\![\text{定数}{\downarrow}_S^B]\!]\,\rho \quad = \quad \text{定数}$$

とすることで，従来と同様に定数を扱うことができる．

図 3 では簡単のため，コードを残す際に let 文の挿入を行っていない．そのため，プログラムによってはコードの複製，消去，実行順の入れ換えが起きる可能性がある．この問題を避けるためには，従来通りの方法を使って，コードが生成されるたびに（具体的には *build-* 命令が呼ばれるたびに）let 文を挿入すれば良い．

図 3 の特化器は，関数以外のものを適用したり，コード以外のものから *build-* によってコードを作ろうとするとエラーを起こす．6 節では，プログラムの注釈がきちんと付けられていれば，そのプログラムの特化でエラーが起きることはないことを見る．

プログラム M が与えられたときに，その特化を行うには，初期環境 ρ_{init} を使って $\mathcal{S}[\![M]\!]\,\rho_{init}$ を計算する．ここで初期環境 ρ_{init} は，全ての変数に対して未定義である（エラーを返す）ような環境である．(特化する式が閉じた式ではなく，あらかじめ定義された大域変数等がある場合には，それに対する束縛が ρ_{init} に入ってもよい.)

4 束縛時型と型に対する制限

本稿で扱う束縛時解析は，型システムとして定式化されており，そこで使われる各式の束縛時の値 π と型 τ は以下のように定義される．

$$\pi \quad = \quad S \mid D \mid B$$
$$\tau \quad = \quad \mathsf{Int}^{\pi} \mid \tau \to^{\pi} \tau \mid \tau \times^{\pi} \tau$$

束縛時値 π は S（静的），D（動的），または B（静的かつ動的）である．また，束縛時型 τ は，定数（ここでは代表として整数を扱う），関数，またはペアである．各束縛時型 τ には，束縛時値 π が肩に付与されており，これで束縛時値が π であるような束縛時型を示す．以下，束縛時型 τ の束縛時値が π であることを明示する必要があるときには τ^{π} と書き，π を τ の（トップレベルの）束縛時値と呼ぶ．

ここでは，簡単のため部分評価されるプログラムは，束縛時値を取り除いた型に関して simply typed である，と仮定する．すると，束縛時解析は，従来通りの手法を使って（束縛時値なしの）型を求めたあと，各型につく束縛時値を決定するのが主要な仕事となる．

上記で定義される束縛時型は，全てが有効な束縛時型である，というわけではない．例えば，動的なペア型 $\tau_1 \times^D \tau_2$ に現れる τ_1, τ_2 はともに束縛時値 D を持たなくてはいけない．これは，ペア全体が最終結果にコードとして残るなら，ペアの各要素も残るためである．このように構造データ（関数，ペア）については，全体の束縛時値と要素の束縛時値の間に制約がある．そこで，有効な束縛時型を以下のように定義する．

定義 1 束縛時型は以下の場合にのみ有効である．

1. Int^{π} は（π の値にかかわらず）有効.
2. $\tau_1^{\pi_1} \to^{\pi} \tau_2^{\pi_2}$ は，$\pi \triangleright' \pi_1$ かつ $\pi \triangleright \pi_2$ なら有効.
3. $\tau_1^{\pi_1} \times^{\pi} \tau_2^{\pi_2}$ は，$\pi \triangleright \pi_1$ かつ $\pi \triangleright \pi_2$ なら有効.

ここで，$\pi_1 \triangleright \pi_2$ と $\pi_1 \triangleright' \pi_2$ が構造データに関する制約で，以下のように定義される．

$S \triangleright S$　$S \triangleright D$　$S \triangleright B$　　$D \triangleright D$　　$B \triangleright D$　$B \triangleright B$

$S \triangleright' S$　$S \triangleright' D$　$S \triangleright' B$　　$D \triangleright' D$　　$B \triangleright' D$

π_1 が S のときは π_2 は何でも良いが，π_1 が D のときはπ_2 も D でなくてはならない．これは，構造データのトップレベルの束縛時値が S ならその各要素は何でも良いが，トップレベルの束縛時値がD になったら各

要素も D になる，ということを意味している．ここまでは，従来の場合と同様である．

π_1 が B だったときには，π_2 として D と B を許すもの $(\triangleright)$ と D のみしか許さないもの $(\triangleright')$ の 2 種類がある．どちらの場合も π_2 として S が許されないのは，構造データの束縛時値が B になったら，その各要素のコード表現が必要となるためである．例えば，図 3 の $cons^B M_1\, M_2$ の規則を見てみよう．この規則では，将来，コード表現が必要になるのに備えて，全体のコード表現を *build-cons* によって作っているが，このとき M_i の束縛時値が S だと，そのコード表現が得られないため全体のコードを作ることができなくなってしまう．一方，M_i の束縛時値が D あるいは B なら，必ず M_i のコード部分を取ってくることができるので大丈夫である．これが，ペア型 $\tau_1^{\pi_1} \times^{\pi} \tau_2^{\pi_2}$ に対して $\pi \triangleright \pi_1, \pi \triangleright \pi_2$ という制約がついている理由である．

一方，関数型 $\tau_1^{\pi_1} \to^{\pi} \tau_2^{\pi_2}$ の引数部分については，より強力な制約 $\pi \triangleright' \pi_1$ がついている．これは「関数の束縛時値が B になったら，その引数の束縛時値は D でなくてはいけない」ということを意味している．この制限は，本稿の枠組からやむを得ず出てくるものである．

まず，この制限の意味するところを見るため，

$$P = (\lambda f.\, cons\, f\, (f@2))@(\lambda x.\, x + 3)$$

という式を考えてみよう．$\lambda x.\, x + 3$ は静的に適用されるとともに最終結果に残るので静的かつ動的である．そこで静的な部分は計算し，動的な部分は残して $cons\, (\lambda x.\, x + 3)\, 5$ を結果として得たい．しかし，上の制限から静的かつ動的な関数 $\lambda x.\, x + 3$ の引数 x は動的となってしまうのである．これは $\lambda x.\, x + 3$ が動的な値として使われる場合に備えてそのコード表現を用意する必要があるが，そのときは x に関する静的な情報が何もないためである．結果として $+$ も動的となり $cons\, (\lambda x.\, x + 3)\, (2 + 3)$ しか得ることができないのである†3.

この静的かつ動的なλ式に対する制限は，本稿で扱う束縛時解析が monovariant であることによる．λ式を最終結果に残すためには，そのλ式の本体部分を解析するときに引数に関して何も仮定できないので，引数は動的とせざるを得ない．monovariant な解析ではプログラム各部に 1 つの束縛時しか割り当てられないので，このλ式が別のところで静的に使われ，そこで静的な引数を渡されたとしても，それは動的とせざるを得ないのである．

解析を monovariant にしているのは効率のためである．polyvariant な解析はコストが非常に高いことが知られている [2] ので，ここでは，より効率的に行える monovariant な解析を採用した．実際，13 節ではこの解析が効率的にほぼ線形時間で行えることを見る．また，上のような制限が付いていても，従来，静的かつ動的なλ式は全て動的と扱われてきていたので，従来の monovariant な部分評価と比べればより部分評価できるようになっている．

5 型規則

図 6 に注釈の付いた言語に対する型規則（型チェック規則）を示す．図中 $\Gamma \vdash_1 M : \tau^{\pi}$ は，型環境 Γ のもとで注釈の付いた式 M が型 τ^{π} を持つことを示す．(Γ が空の型環境のときは Γ を省略して $\vdash_1 M : \tau^{\pi}$ と書く．)

静的かつ動的な値の型の有効性にだけ注意しておけば，これらの規則はいずれも標準的なものである．$\tau{\downarrow}_D^B$ の規則に出てくる $[\tau]^D$ というのは，τ^D 中に出てくる全ての束縛時値を D にしたものである．図中，唯一，特殊なのは静的な加算の結果の型で，これらはいつも静的とはせず，その使われ方によって何でも良いことにしている．これは静的な定数はいつでも動的な定数に（従来の lift 命令によって）変換できるからである．

定義 2 注釈の付いたプログラム p_{ann} に対し $\vdash_1 p_{ann} : \tau^D$ が導けるとき，その注釈は有効である，という．

定義 3 同じプログラム p に対する有効な注釈 p_{ann} のうちで，以下の *2* つの性質を満たす注釈 p^*_{ann} を最も望ましい注釈と呼ぶ．

1. p の全ての部分式について，その束縛時値が p_{ann} において S なら，p^*_{ann} においても S である．
2. p_{ann} と p^*_{ann} で束縛時値が S である部分式が同一の場合には，それ以外の全ての部分式について，その束縛時値が p^*_{ann} において B なら，p_{ann} にお

†3 x を静的かつ動的とすることも不可能である．静的かつ動的な値は常に静的にも動的にも使用可能な値だが，λ式がコードに残るときの x は動的な情報しか持っていない．

$$\frac{\Gamma, x:\tau_1^{\pi_1} \vdash_1 M:\tau_2^{\pi_2}}{\Gamma \vdash_1 \lambda^S x.M:\tau_1^{\pi_1} \to^S \tau_2^{\pi_2}} \qquad \frac{\Gamma, x:\tau_1^{D} \vdash_1 M:\tau_2^{D}}{\Gamma \vdash_1 \lambda^D x.M:\tau_1^{D} \to^D \tau_2^{D}} \qquad \frac{\Gamma, x:\tau_1^{D} \vdash_1 M:\tau_2^{\pi_2} \quad \pi_2 \in \{B,D\}}{\Gamma \vdash_1 \lambda^B x.M:\tau_1^{D} \to^B \tau_2^{\pi_2}}$$

$$\Gamma, x:\tau^{\pi} \vdash_1 x:\tau^{\pi} \qquad \frac{\Gamma \vdash_1 M_1:\tau_2^{\pi_2} \to^S \tau_1^{\pi_1} \quad \Gamma \vdash_1 M_2:\tau_2^{\pi_2}}{\Gamma \vdash_1 M_1 @^S M_2:\tau_1^{\pi_1}} \qquad \frac{\Gamma \vdash_1 M_1:\tau_2^{D} \to^D \tau_1^{D} \quad \Gamma \vdash_1 M_2:\tau_2^{D}}{\Gamma \vdash_1 M_1 @^D M_2:\tau_1^{D}}$$

$$\frac{\Gamma \vdash_1 M_1:\tau_1^{\pi_1} \quad \Gamma \vdash_1 M_2:\tau_2^{\pi_2}}{\Gamma \vdash_1 cons^S M_1\, M_2:\tau_1^{\pi_1} \times^S \tau_2^{\pi_2}} \qquad \frac{\Gamma \vdash_1 M:\tau_1^{\pi_1} \times^S \tau_2^{\pi_2}}{\Gamma \vdash_1 car^S M:\tau_1^{\pi_1}} \qquad \frac{\Gamma \vdash_1 M_1:\mathsf{Int}^S \quad \Gamma \vdash_1 M_2:\mathsf{Int}^S}{\Gamma \vdash_1 M_1 +^S M_2:\boxed{\mathsf{Int}^{\pi}}}$$

$$\frac{\Gamma \vdash_1 M_1:\tau_1^{D} \quad \Gamma \vdash_1 M_2:\tau_2^{D}}{\Gamma \vdash_1 cons^D M_1\, M_2:\tau_1^{D} \times^D \tau_2^{D}} \qquad \frac{\Gamma \vdash_1 M:\tau_1^{D} \times^D \tau_2^{D}}{\Gamma \vdash_1 car^D M:\tau_1^{D}} \qquad \frac{\Gamma \vdash_1 M_1:\mathsf{Int}^D \quad \Gamma \vdash_1 M_2:\mathsf{Int}^D}{\Gamma \vdash_1 M_1 +^D M_2:\mathsf{Int}^D}$$

$$\frac{\Gamma \vdash_1 M_1:\tau_1^{\pi_1} \quad \Gamma \vdash_1 M_2:\tau_2^{\pi_2} \quad \pi_i \in \{B,D\}}{\Gamma \vdash_1 cons^B M_1\, M_2:\tau_1^{\pi_1} \times^B \tau_2^{\pi_2}} \qquad \frac{\Gamma \vdash_1 M:\tau^B}{\Gamma \vdash_1 M{\downarrow}_S^B:\tau^S} \qquad \frac{\Gamma \vdash_1 M:\tau^B}{\Gamma \vdash_1 M{\downarrow}_D^B:[\tau]^D} \qquad \Gamma \vdash_1 \text{定数}^{\pi}:\mathsf{Int}^{\pi}$$

図6　注釈の付いた言語に対する型規則（型チェック規則）

$$\dfrac{\dfrac{\dfrac{\Gamma_1 \vdash_1 f:D \to^B D}{\Gamma_1 \vdash_1 f{\downarrow}_S^B:D \to^S D} \quad \Gamma_1 \vdash_1 2^D:D}{\Gamma_1 \vdash_1 f{\downarrow}_D^B:D \to^D D} \quad \Gamma_1 \vdash_1 f{\downarrow}_S^B\ @^S 2^D:D}{\Gamma_1 \vdash_1 cons^D f{\downarrow}_D^B\ (f{\downarrow}_S^B\ @^S 2^D):(D \to^D D) \times^D D} \qquad \dfrac{\dfrac{\Gamma_2 \vdash_1 x:D \quad \Gamma_2 \vdash_1 3^D:D}{\Gamma_2 \vdash_1 x +^D 3^D:D}}{\vdash_1 \lambda^B x.x +^D 3^D:D \to^B D}}{\vdash_1 \lambda^S f.cons^D f{\downarrow}_D^B\ (f{\downarrow}_S^B\ @^S 2^D):(D \to^B D) \to^S (D \to^D D) \times^D D}$$

$$\vdash_1 (\lambda^S f.cons^D f{\downarrow}_D^B\ (f{\downarrow}_S^B\ @^S 2^D))@^S(\lambda^B x.x +^D 3^D):(D \to^D D) \times^D D$$

図7　型チェックの例（$\Gamma_1 = f:D \to^B D$, $\Gamma_2 = x:D$, Int^{π} を π と表記）

いても B である．

直観的に解釈すると，第1点は S となっている部分がより多い，と言っており，これは，より多くの部分を部分評価時に実行できることを示している．また，第2点は，S となっている部分が同じだったら，B となっている部分がより少ない，と言っており，これは，部分評価時に静的かつ動的な値を作るオーバーヘッドがより少ないことを示している．

束縛時解析は，注釈の付いていないプログラム p が与えられたときに，有効な注釈の付いたプログラム p_{ann} のうち，上の意味で最も望ましい注釈を返すもの，ということができる．

型チェックの例として前節に出てきた

$$P = (\lambda f.\, cons\ f\ (f@2))@(\lambda x.x+3)$$

という項を考えてみよう．この式は $\lambda x.x+3$ を静的かつ動的として図7のように注釈を付けることができる．この注釈は，これ以上 S を増やすことができず，また，現在 S になっている部分を D にすることなく B を減らすことはできないので最も望ましい注釈である．

6　型システムの健全性

図3の特化器 $\mathcal{S}$ は，注釈の付いたプログラムに対して，図4に示されているドメイン上の表示的意味記述を与えている，と見ることができる†4．図6に示された型システムが，この表示的意味記述に対して健全である，とは，ある式が型システムで型 τ^{π} を持つことがわかったら $\mathcal{S}$ は（エラーを起こすことなく）その型の値を返す，と定義される．

ここで $V_{\tau^{\pi}}$ を図8のように定義しよう．すると，型システムの健全性は，型 τ^{π} の値の特化の結果が $V_{\tau^{\pi}}$ に含まれている，と言い換えることができる．ここでは，

†4　新たな変数の生成があるので，厳密には表示的意味記述ではないが，各関数に変数生成用の引数を1つ余計に渡すという標準的な方法でこの問題は回避できる．

$$
\begin{aligned}
V_{\mathsf{Int}^S} &= \mathsf{Int} \\
V_{\tau_1^{\pi_1} \to^S \tau_2^{\pi_2}} &= V_{\tau_1^{\pi_1}} \to V_{\tau_2^{\pi_2}} \\
V_{\tau_1^{\pi_1} \times^S \tau_2^{\pi_2}} &= V_{\tau_1^{\pi_1}} \times V_{\tau_2^{\pi_2}} \\
V_{\tau^D} &= \mathit{Code} \\
V_{\tau^B} &= V_{\tau^D} \times V_{\tau^S}
\end{aligned}
$$

図8 V_{τ^π} **の定義**

これを証明していく．

以下，型環境 Γ で x が τ^π 型であるとき $\tau^\pi = \Gamma(x)$ と書く．

定義 4 特化時の環境 ρ が型環境 Γ に適合している，とは，$\rho(x)$ が定義されている全ての x について $\rho(x) \in V_{\Gamma(x)}$ が成り立つことを言う．

型規則の健全性は以下のように述べられる．

定理 1（型規則の健全性） 図 6 の規則で $\Gamma \vdash_1 M : \tau^\pi$ が導けるなら，Γ に適合する全ての ρ について，$\mathcal{S}[\![M]\!]\rho \in V_{\tau^\pi}$ が成り立つ．

証明： $\Gamma \vdash_1 M : \tau^\pi$ の証明木の深さに関する帰納法で証明する．以下，M として $x, \lambda^S x.M, \lambda^D x.M, \lambda^B x.M$ の場合について証明する．他も同様である．

（x の場合） 証明木は $\Gamma, x : \tau^\pi \vdash_1 x : \tau^\pi$ となる．$\Gamma, x : \tau^\pi$ に適合するような ρ に対しては $\rho(x) \in V_{\tau^\pi}$ が成り立つ．従って，$\mathcal{S}[\![x]\!]\rho = \rho(x) \in V_{\tau^\pi}$

（$\lambda^S x.M$ の場合） 証明木の最後は

$$\frac{\Gamma, x : \tau_1^{\pi_1} \vdash_1 M : \tau_2^{\pi_2}}{\Gamma \vdash_1 \lambda^S x.M : \tau_1^{\pi_1} \to^S \tau_2^{\pi_2}}$$

となる．ρ が Γ に適合しているとしよう．$x' \in V_{\tau_1^{\pi_1}}$ なる全ての x' に対して $\rho[x'/x]$ は $\Gamma, x : \tau_1^{\pi_1}$ に適合するので，帰納法の仮定により $\mathcal{S}[\![M]\!]\rho[x'/x] \in V_{\tau_2^{\pi_2}}$ が成り立つ．従って，

$$
\begin{aligned}
\mathcal{S}[\![\lambda^S x.M]\!]\rho &= \lambda x'.\mathcal{S}[\![M]\!]\rho[x'/x] \\
&\in V_{\tau_1^{\pi_1}} \to V_{\tau_2^{\pi_2}} \\
&= V_{\tau_1^{\pi_1} \to^S \tau_2^{\pi_2}}
\end{aligned}
$$

（$\lambda^D x.M$ の場合） 証明木の最後は

$$\frac{\Gamma, x : \tau_1^D \vdash_1 M : \tau_2^D}{\Gamma \vdash_1 \lambda^D x.M : \tau_1^D \to^D \tau_2^D}$$

となる．ρ が Γ に適合しているとしよう．$x^\diamond \in V_{\tau_1^D}$ なる全ての $x^\diamond$ に対して $\rho[x^\diamond/x]$ は $\Gamma, x : \tau_1^D$ に適合するので，帰納法の仮定により $\mathcal{S}[\![M]\!]\rho[x^\diamond/x] \in V_{\tau_2^D}$ が成り立つ．従って，*build*-λ の引数がコードとなっているので

$$
\begin{aligned}
\mathcal{S}[\![\lambda^D x.M]\!]\rho &= \mathit{build}\text{-}\lambda(x^\diamond, \mathcal{S}[\![M]\!]\rho[x^\diamond/x]) \\
&\in \mathit{Code} \\
&= V_{\tau_1^D \to^D \tau_2^D}
\end{aligned}
$$

（$\lambda^B x.M$ の場合） 証明木の最後は

$$\frac{\Gamma, x : \tau_1^D \vdash_1 M : \tau_2^{\pi_2} \quad \pi_2 \in \{B, D\}}{\Gamma \vdash_1 \lambda^B x.M : \tau_1^D \to^B \tau_2^{\pi_2}}$$

となる．ρ が Γ に適合しているとしよう．$x' \in V_{\tau_1^D}$ なる全ての x' に対して $\rho[x'/x]$ は $\Gamma, x : \tau_1^D$ に適合するので，帰納法の仮定により $\mathcal{S}[\![M]\!]\rho[x'/x] \in V_{\tau_2^{\pi_2}} (\pi_2 \in \{B, D\})$ が成り立つ．従って，

$$
\begin{aligned}
\lambda x'.\mathcal{S}[\![M]\!]\rho[x'/x] &\in V_{\tau_1^D} \to V_{\tau_2^{\pi_2}} \\
&= V_{\tau_1^D \to^S \tau_2^{\pi_2}}
\end{aligned}
$$

また，$x^\diamond \in V_{\tau_1^D}$ なる全ての $x^\diamond$ に対して $\rho[x^\diamond/x]$ は $\Gamma, x : \tau_1^D$ に適合するので，帰納法の仮定により $\mathcal{S}[\![M]\!]\rho[x^\diamond/x] \in V_{\tau_2^{\pi_2}} (\pi_2 \in \{B, D\})$ が成り立つ．従って $\mathcal{D}(\cdot)$ の引数の束縛時値が B あるいは D となっているので $\mathcal{D}(\mathcal{S}[\![M]\!]\rho[x^\diamond/x]) \in \mathit{Code}$. よって

$$
\begin{aligned}
\mathit{build}\text{-}\lambda(x^\diamond, \mathcal{D}(\mathcal{S}[\![M]\!]\rho[x^\diamond/x])) &\in \mathit{Code} \\
&= V_{\tau_1^D \to^D \tau_2^{\pi_2}}
\end{aligned}
$$

以上より

$$
\begin{aligned}
& \mathcal{S}[\![\lambda^B x.M]\!]\rho \\
= \; & (\mathit{build}\text{-}\lambda(x^\diamond, \mathcal{D}(\mathcal{S}[\![M]\!]\rho[x^\diamond/x])), \\
& \lambda x'.\mathcal{S}[\![M]\!]\rho[x'/x]) \\
\in \; & (V_{\tau_1^D \to^D \tau_2^{\pi_2}}) \times (V_{\tau_1^D \to^S \tau_2^{\pi_2}}) \\
= \; & V_{\tau_1^D \to^B \tau_2^{\pi_2}} \qquad \square
\end{aligned}
$$

この定理より $\vdash_1 M : \tau^D$ が導けるなら，必ずその特化の結果は $V_{\tau^D} = \mathit{Code}$ となり，エラーを起こすことなく，部分評価の結果のコードを得られることがわかる．

系 1 $\vdash_1 M : \tau^D$ が導けるなら $\mathcal{S}[\![M]\!]\rho_{init} \in V_{\tau^D}$

証明： ρ_{init} が空の型環境に適合しているため． $\square$

$$\Gamma \vdash_2 \text{定数} : \mathsf{Int}^{\pi} \rightsquigarrow \text{定数}^{\pi}$$

$$\Gamma, x : \tau^{\pi} \vdash_2 x : \boxed{[\tau]^{\pi'}} \rightsquigarrow x{\downarrow}^{\pi}_{\pi'} \quad \{\boxed{\pi \geq \pi'}\}$$

$$\frac{\Gamma, x : \tau_1^{\pi_1} \vdash_2 L : \tau_2^{\pi_2} \rightsquigarrow M}{\Gamma \vdash_2 \lambda x.\, L : \tau_1^{\pi_1} \rightarrow^{\pi} \tau_2^{\pi_2} \rightsquigarrow \lambda^{\pi} x.\, M} \ \{\pi \rhd' \pi_1,\ \pi \rhd \pi_2\}$$

$$\frac{\Gamma \vdash_2 L_1 : \tau_2^{\pi_2} \rightarrow^{\pi} \tau_1^{\pi_1} \rightsquigarrow M_1 \quad \Gamma \vdash_2 L_2 : \tau_2^{\pi_2} \rightsquigarrow M_2}{\Gamma \vdash_2 L_1 @ L_2 : \boxed{[\tau_1]^{\pi_1'}} \rightsquigarrow (M_1 @^{\pi} M_2){\downarrow}^{\pi_1}_{\pi_1'}} \left\{ \begin{array}{ll} \boxed{\pi_1 \geq \pi_1'}, & \pi \rhd' \pi_2, \\ \pi \in \{S, D\}, & \pi \rhd \pi_1 \end{array} \right\}$$

$$\frac{\Gamma \vdash_2 L_1 : \tau_1^{\pi_1} \rightsquigarrow M_1 \quad \Gamma \vdash_2 L_2 : \tau_2^{\pi_2} \rightsquigarrow M_2}{\Gamma \vdash_2 \mathit{cons}\, L_1\, L_2 : \tau_1^{\pi_1} \times^{\pi} \tau_2^{\pi_2} \rightsquigarrow \mathit{cons}^{\pi} M_1\, M_2} \ \{\pi \rhd \pi_1,\ \pi \rhd \pi_2\}$$

$$\frac{\Gamma \vdash_2 L : \tau_1^{\pi_1} \times^{\pi} \tau_2^{\pi_2} \rightsquigarrow M}{\Gamma \vdash_2 \mathit{car}\, L : \boxed{[\tau_1]^{\pi_1'}} \rightsquigarrow (\mathit{car}^{\pi} M){\downarrow}^{\pi_1}_{\pi_1'}} \left\{ \begin{array}{ll} \boxed{\pi_1 \geq \pi_1'}, & \pi \rhd \pi_1, \\ \pi \in \{S, D\}, & \pi \rhd \pi_2 \end{array} \right\}$$

$$\frac{\Gamma \vdash_2 L_1 : \mathsf{Int}^{\pi_1} \rightsquigarrow M_1 \quad \Gamma \vdash_2 L_2 : \mathsf{Int}^{\pi_2} \rightsquigarrow M_2}{\Gamma \vdash_2 L_1 + L_2 : \mathsf{Int}^{\pi} \rightsquigarrow M_1 +^{\pi} M_2} \left\{ \begin{array}{l} \pi_1 \rhd \pi,\ \pi_2 \rhd \pi,\ \pi_1 \rhd \pi_2,\ \pi_2 \rhd \pi_1, \\ \pi_1 \in \{S, D\},\ \pi_2 \in \{S, D\} \end{array} \right\}$$

図9　注釈の付いていない言語に対する型規則（制約生成規則）

7　制約の生成

図 6 の規則は注釈の付いた言語に対する型規則なので，これを使って注釈の付いていないプログラムに注釈を付けることはできない．注釈の付いていないプログラムに注釈を付けるためには，図 6 の型規則を注釈の付いていないプログラムに対するものに書き換える必要がある．この書き換えは（1）同じ命令に対する規則をまとめ，（2）B から S や D への型変換規則を他の規則に埋め込むこと，で行われる．書き換えた結果を図 9 に示す．この規則を本稿では制約生成規則と呼ぶ．図中 $\Gamma \vdash_2 L : \tau^{\pi} \rightsquigarrow M$ は，型環境 Γ のもとで注釈の付いていない式 L が型 τ^{π} を持ち，その結果，もとの式 L が M という注釈の付いた式に変換されることを示す．また $[\tau]^{\pi}$ は，$\pi = D$ のときは τ^{π} 中に出てくる全ての束縛時値を D にしたもの，$\pi \neq D$ のときは τ^{π} そのものを表す．

図 9 の規則の中で四角で囲んである部分が型変換が起きる可能性のある場所である．（型変換がこれだけで十分なことは 8 節の補題 1 で見る．）ここに出てくる $\pi \geq \pi'$ という制約は型の変換可能性を示すもので以下のように定義される．（$S > D$ は成り立たないことに注意．）

$$B > S, \quad B > D$$

$$\pi \geq \pi' \Longleftrightarrow (\pi > \pi') \vee (\pi = \pi')$$

束縛時解析終了時に $\pi \geq \pi'$ の制約の等号が成り立たず $\pi > \pi'$ となったときには，π' に対応する項に注釈として ${\downarrow}^{B}_{\pi'}$ が付く．逆に $\pi = \pi'$ の場合は $M{\downarrow}^{\pi}_{\pi'} = M$ と解釈する．

制約生成規則は注釈の付いていない言語に対する規則となっているが，各式を具体的にどういう束縛時値にすれば良いかは明らかではない．束縛時解析は，まず制約生成規則で束縛時値に関する制約を生成し，それを解くことで行われる．

例えば，項 P に対しては，従来通りの型推論を行って図 10 のような推論木を作ることができる．この推論木の各推論を制約生成規則と照らしあわせると以下のような制約を得ることができる．

$$\dfrac{\dfrac{\dfrac{\Gamma_1 \vdash_2 f : \pi_1 \to^{\pi_4} \pi_3}{\Gamma_1 \vdash_2 f\,@2 : \pi_3'} \quad \dfrac{\Gamma_1 \vdash_2 f : \pi_1 \to^{\pi_4''} \pi_3 \quad \Gamma_1 \vdash_2 2 : \pi_1}{\Gamma_1 \vdash_2 f\,@2 : \pi_3'}}{\Gamma_1 \vdash_2 \mathit{cons}\, f\,(f@2) : (\pi_1 \to^{\pi_4'} \pi_3) \times^{\pi_5} \pi_3'} \quad \dfrac{\dfrac{\Gamma_2 \vdash_2 x : \pi_1' \quad \Gamma_2 \vdash_2 3 : \pi_2}{\Gamma_2 \vdash_2 x + 3 : \pi_3}}{\vdash_2 \lambda x.\, x + 3 : \pi_1 \to^{\pi_4} \pi_3}}{\vdash_2 (\lambda f.\, \mathit{cons}\, f\,(f@2))@(\lambda x.\, x + 3) : (\pi_1 \to^{\pi_4'} \pi_3) \times^{\pi_5'} \pi_3'}$$

図 10　型推論の例（$\Gamma_1 = f : \pi_1 \to^{\pi_4} \pi_3$, $\Gamma_2 = x : \pi_1$, Int^{π} を π と表記．$\rightsquigarrow M$ の部分は省略）

$$\begin{array}{lll}
\pi_1 \geq \pi_1', & \pi_4 \rhd' \pi_1, & \pi_3 \geq \pi_3', \\
\pi_1' \rhd \pi_3, & \pi_4 \rhd \pi_3, & \pi_5 \rhd \pi_4', \\
\pi_2 \rhd \pi_3, & \pi_4 \geq \pi_4', & \pi_5 \rhd \pi_3', \\
\pi_1' \rhd \pi_2, & \pi_4 \geq \pi_4'', & \pi_6 \rhd' \pi_4, \\
\pi_2 \rhd \pi_1', & \pi_4'' \rhd' \pi_1, & \pi_6 \rhd \pi_5, \\
\pi_1' \in \{S, D\}, & \pi_4'' \rhd \pi_3, & \pi_5 \geq \pi_5', \\
\pi_2 \in \{S, D\}, & \pi_4'' \in \{S, D\}, & \pi_6 \in \{S, D\}
\end{array}$$

この制約は（9 節以降で説明する方法を使って）$\pi_4' = D$ という初期条件のもとで解くと

$$\pi_4'' = \pi_6 = S, \quad \pi_4 = B,$$
$$\pi_1 = \pi_1' = \pi_2 = \pi_3 = \pi_3' = \pi_4' = \pi_5 = \pi_5' = D$$

という解が得られ，これより

$(\lambda^S f.\, \mathit{cons}^D f{\downarrow}_D^B\ (f{\downarrow}_S^B\ @^S 2^D))@^S(\lambda^B x.\, x +^D 3^D)$

という注釈が得られる．ここで $\pi_4 \geq \pi_4'$ と $\pi_4 \geq \pi_4''$ の 2 つの制約の等号が成り立たなかったので，それに対応する項に ${\downarrow}_D^B$ と ${\downarrow}_S^B$ が挿入されている．

8　型チェック規則と制約生成規則の等価性

本節では，図 6 の規則（型チェック規則）と図 9 の規則（制約生成規則）の等価性を見ていく．

次の定理は，制約生成規則で得られる注釈は型チェック規則でも得ることができることを示している．

定理 2 $\Gamma \vdash_2 L : \tau^{\pi} \rightsquigarrow M$ ならば $\Gamma \vdash_1 M : \tau^{\pi}$.

証明： 制約生成規則を使って得られた証明木が型チェック規則を使って得られる証明木に変換できることを示す．以下 $\lambda x.\, L$, $L_1@L_2$, $L_1 + L_2$ の 3 つの規則についての変換を示す．他も同様である．

（$\lambda x.\, L$ の場合） $\pi = S, D, B$ だった場合に，それぞれ $\lambda^S x.\, M$, $\lambda^D x.\, M$, $\lambda^B x.\, M$ の規則を使えば良い．そのとき $\pi \rhd' \pi_1$, $\pi \rhd \pi_2$ の 2 つの制約が型の有効性を保証するために用いられる．$\pi = S$ のときは，これらの制約は制約にはならないので $\lambda^S x.\, M$ の規則が得られ，$\pi = D$ のときは $\pi_1 = \pi_2 = D$ となって $\lambda^D x.\, M$ の規則が得られ，$\pi = B$ のときは $\pi_1 = D$, $\pi_2 \in \{B, D\}$ となって $\lambda^B x.\, M$ の規則が得られる．

（$L_1@L_2$ の場合） $\pi = S, D$ だった場合に，それぞれ $M_1@^S M_2$, $M_1@^D M_2$ の規則を使えば良い．そのとき $\pi \rhd' \pi_2$, $\pi \rhd \pi_1$ の 2 つの制約が型の有効性を保証するために用いられる．また，$\pi_1 > \pi_1'$ が成り立った場合には，直後に ${\downarrow}_{\pi_1'}^B$ の規則を使えば良い．具体的には，

$$\dfrac{\Gamma \vdash_2 L_1 : \tau_2^{\pi_2} \to^S \tau_1^B \rightsquigarrow M_1 \quad \Gamma \vdash_2 L_2 : \tau_2^{\pi_2} \rightsquigarrow M_2}{\Gamma \vdash_2 L_1@L_2 : \tau_1^{\pi_1'} \rightsquigarrow (M_1@^S M_2){\downarrow}_{\pi_1'}^B}$$

を

$$\dfrac{\dfrac{\Gamma \vdash_1 M_1 : \tau_2^{\pi_2} \to^S \tau_1^B \quad \Gamma \vdash_1 M_2 : \tau_2^{\pi_2}}{\Gamma \vdash_1 M_1@^S M_2 : \tau_1^B}}{\Gamma \vdash_1 (M_1@^S M_2){\downarrow}_{\pi_1'}^B : [\tau_1]^{\pi_1'}}$$

に変換すればよい．

（$L_1 + L_2$ の場合） $\pi_1 = \pi_2 = S$ だった場合に $M_1 +^S M_2$ の規則を，$\pi_1 = \pi_2 = D$ だった場合に $M_1 +^D M_2$ の規則を使えば良い．それ以外の場合は，$\pi_1 \rhd \pi_2, \pi_2 \rhd \pi_1$ の 2 つの制約によって除外されている．また，$\pi_1 \rhd \pi, \pi_2 \rhd \pi$ の 2 つの制約によって $\pi_1 = \pi_2 = D$ だったら $\pi = D$ を保証している． □

補題 1 最も望ましい注釈の導出木において，型変換規則は，x, $M_1@^{\pi} M_2$, $\mathit{car}^{\pi} M$（または $\mathit{cdr}^{\pi} M$）に対してのみ使われ，定数$^{\pi}$, $\lambda^{\pi} x.\, M$, $\mathit{cons}^{\pi} M_1\, M_2$，または $M_1 +^{\pi} M_2$ に対して使われることはない．

証明： $\lambda^{\pi} x.\, M$ の場合について証明する．（定数$^{\pi}$,

$cons^{\pi}\,M_1\,M_2$，$M_1 +^{\pi} M_2$ の場合も同様に証明できる.）$\lambda^{\pi}x.\,M$ に対して型変換規則が使われる場合というのは，$(\lambda^{B}x.\,M)\!\downarrow_{D}^{B}$ または $(\lambda^{B}x.\,M)\!\downarrow_{S}^{B}$ のどちらかである．いずれの場合も，型変換規則を使わずに $\lambda^{D}x.\,M$ または $\lambda^{S}x.\,M$ とすれば，λ式の束縛時値が（他の束縛時値を変えることなく）B から D あるいは S になっているのでより望ましい注釈となっている．従って，最も望ましい注釈においては，型変換規則がλ式に直接，使われることはない． □

次の定理は，型チェック規則を使って得られる注釈で，それが最も望ましい注釈になっているならば，その注釈は制約生成規則でも得ることができることを示している．

定理 3 $\Gamma \vdash_1 M : \tau^{\pi}$ かつ M の注釈が最も望ましいものであるならば $\Gamma \vdash_2 |M| : \tau^{\pi} \rightsquigarrow M$.

証明： 型チェック規則を使って得られた証明木が，最も望ましい注釈に対するものであったなら，制約生成規則を使って得られる証明木に変換できることを示す.

型変換以外の規則については，対応する制約生成規則をそのまま使えば良い．問題になるのは，型変換規則が使われた場合である．補題 1 より，注釈が最も望ましいものであった場合，型変換の規則が使われるのは x，$M_1 @^{\pi} M_2$，$car^{\pi}M$（または $cdr^{\pi}M$）のいずれかである．これらに対応する制約生成規則にはいずれも $\pi \geq \pi'$ の制約がついているので，型変換規則をあわせて制約生成規則に対応させれば良い． □

9 型のペアによる解釈

本稿で行う制約の解消は，まず束縛時値 π を (s,d) という真偽値のペアで表現する．そして π 上の制約をこれらの変数の上の制約に変換し，そこで制約解消を行う．S, D, B と (s,d) の関係は以下のように定義される．

	$d = \mathsf{false}$	$d = \mathsf{true}$
$s = \mathsf{false}$	$\bot$	D
$s = \mathsf{true}$	S	B

$s = \mathsf{true}$ は静的に使われ得る，すなわち S あるいは B であることを示し，$d = \mathsf{true}$ は動的に使われ得る，すなわち D あるいは B であることを示す．$s = d = \mathsf{false}$ の場合は，その値が静的に使われることもなければ最終結果にも残らないことを示す．この場合は，その値をどう扱っても部分評価の結果には影響しないのだが，ここでは $\bot$ とした†5.

このように束縛時値を解釈すると，最も望ましい注釈の定義は次のように言い換えることができる.

定義 5 同じプログラム p に対する有効な注釈 p_{ann} のうちで，以下の 2 つの性質を満たす注釈 p^{*}_{ann} を最も望ましい注釈と呼ぶ.

1. p の全ての部分式について，その d の値が p_{ann} において false なら， p^{*}_{ann} においても false である.
2. p_{ann} と p^{*}_{ann} の全ての部分式で d の値が同一の場合には，s の値が p_{ann} において false なら，p^{*}_{ann} においても false である.

より直観的に述べれば，d の値はなるべく false の方が良く，d の値で区別できないときは s の値がなるべく false の方が良い，ということである.

定義 5 は $s = d = \mathsf{false}$ の場合の解釈を $\bot$ ではなく S とすれば，定義 3 と一致する．あえて $\bot$ を使用しているのは，部分評価の結果に影響を与えない部分に柔軟性を持たせることで後の制約解消をやりやすくするためである．このためにある式の束縛時値が $\bot$ になってしまった場合は，特化時にその式を（void 値などの）適当な定数と解釈すればよい．以下では，定義 5 を最も望ましい注釈の定義として使用する.

(s,d) を使って，図 9 に出てくる π 上の制約を以下のように (s,d) 上の制約に変換する.（以下では $\pi = (s,d)$, $\pi_i = (s_i, d_i)$ とする.）

$$
\begin{aligned}
\pi_1 \triangleright \pi_2 :&\quad (d_1 = \mathsf{true}) \Rightarrow (d_2 = \mathsf{true}) \quad \wedge \\
&\quad (s_1 = \mathsf{false}) \Rightarrow (s_2 = \mathsf{false}) \\
\pi_1 \triangleright' \pi_2 :&\quad (d_1 = \mathsf{true}) \Rightarrow (d_2 = \mathsf{true}) \quad \wedge \\
&\quad (d_1 = \mathsf{true}) \Rightarrow (s_2 = \mathsf{false}) \\
\pi_1 \geq \pi_2 :&\quad (d_2 = \mathsf{true}) \Rightarrow (d_1 = \mathsf{true}) \quad \wedge \\
&\quad (s_1 = \mathsf{false}) \Rightarrow (s_2 = \mathsf{false}) \\
\pi \in \{S, D\} :&\quad (s = \mathsf{false}) \Rightarrow (d = \mathsf{true}) \quad \wedge \\
&\quad (d = \mathsf{true}) \Rightarrow (s = \mathsf{false})
\end{aligned}
$$

いずれの制約も（型が $\bot$ となる場合を除けば）もとの制約と同じ制約を表している．このことは (s,d) の真偽

†5 例えば $car\,(cons\,1\,2)$ の 2 など．束縛時値が $\bot$ であるような式は，特化時には任意の定数，例えば void 値に置き換えることができる.

値の全ての場合を確かめれば確認できる．ここで，上の制約中には $d = \mathsf{true}$ と $s = \mathsf{false}$ しか出てきておらず，$d = \mathsf{false}$ や $s = \mathsf{true}$ は出てきていないことに注意しよう．このことは制約解消アルゴリズムの正当性を見る上で重要な役割を果たす．以降の節ではこれらの制約を $(p = x) \Rightarrow (p' = x')$ のように表現する．ここで，p, p' は s または d であり，x, x' は true または false である．

10 制約の解消

プログラムが与えられたら前節までの方法に従って制約を生成する．そして，そのプログラム自体の束縛時値（トップレベルの束縛時値）を $\pi_{top} = (s_{top}, d_{top})$ とすれば $d_{top} = \mathsf{true}$ という制約を加えた上で制約解消を行う．この $d_{top} = \mathsf{true}$ という制約は，このプログラムを部分評価した結果が全体としてコードになる，というもので，初期条件のようなものである．

制約の解消は，より望ましい注釈の２条件にそって２段階で行う．どちらも，制約を書き換え規則と見なして順次，解を更新していくことで制約解消を行う．アルゴリズムを直観的に説明すると以下のようになる．

第１段階 まず（初期条件に使われる d_{top} を除く）全ての d に false，全ての s に true を代入する．そして，制約 $(p = x) \Rightarrow (p' = x')$ を「$p = x$ が成り立っていたら $p' = x'$ が成り立つように値を変更する」と解釈して適用し，これを変化がなくなるまで続ける．

第２段階 d の値は第１段階で確定したものを使用し，s の値は全て false として制約の適用を変化がなくなるまで行う．この際，制約は全て対偶をとって解釈する．つまり $(p = x) \Rightarrow (p' = x')$ を「$p' = x'$ が成り立っていなかったら $p = x$ が成り立たないように値を変更する」と解釈する．

第１段階では $s = \mathsf{true}$, $d = \mathsf{false}$ すなわち全ての値をまず S と仮定し，その中で $d = \mathsf{true}$ すなわち最終結果に残すため，そのコード表現が必要な部分を特定する．直観的には，全ての値が静的に使用可能（$s = \mathsf{true}$）と仮定しても，最終結果にコードとして残さざるを得ない部分を特定している．逆からとらえると，ここで $d = \mathsf{true}$ と特定されなかった部分はコード表現が不要ということなので S とすることができる．

第１段階で d だけではなく s も考慮に入れているのは s の値が d の値に影響を及ぼすことがあるからである．それは，具体的には（関数適用など）$\pi \in \{S, D\}$ の制約が現れるところである．関数適用は S か D のどちらかでなくてはならないが，S とはなり得ない場合にのみ D としたい．そこで，まず S または B（つまり $s = \mathsf{true}$）という仮定からはじめ，S または B とはなり得ない（$s = \mathsf{false}$）ことがわかったときのみ D とするのである．

ここで「関数適用等は可能なら S としたい」という要求と「D としてしか使われていない値を不必要に B にはしたくない」という要求は相容れないことに注意しよう．前者は s をなるべく true にし，後者は s をなるべく false にする方向である．このため，本稿の制約解消アルゴリズムは２段階に分かれている．２つの要求のうちより重要な前者が第１段階で満たされ，後者の要求は第２段階で満たされる．

第２段階では，第１段階で B となったもののうち，実際に静的かつ動的に使われているものを特定する．そのためにまず，第１段階で B（または D）と判断したものを全て D として，B としては使われていない（$s = \mathsf{false}$）と仮定する．そして，その中で S の計算を行うために必要なものを B に戻している（$s = \mathsf{true}$）．

制約解消のアルゴリズムをより形式的に書くと以下のようになる．ここで，C_{init} は 7 節で生成された π 上の制約を 9 節で述べた方法で変換した s と d 上の制約，C^*_{init} は C_{init} 中の制約の対偶をとったものとする．（$(p_1 = x_1) \Rightarrow (p_2 = x_2)$ の対偶は，$(p_2 = \neg x_2) \Rightarrow (p_1 = \neg x_1)$ で与えられる．）また，プログラム全体の束縛時値を (s_{top}, d_{top}) とする．

第１段階 初期制約集合 C_{I} と初期解 θ_{I} を以下のように定義する．

$$
\begin{aligned}
C_{\mathrm{I}} &= C_{init} \\
\theta_{\mathrm{I}} &= \{d_{top} = \mathsf{true}\} \\
&\quad \cup \{d = \mathsf{false} \mid \text{全ての } d(\neq d_{top}) \text{ について}\} \\
&\quad \cup \{s = \mathsf{true} \mid \text{全ての } s \text{ について}\}
\end{aligned}
$$

$\langle C_{\mathrm{I}}, \theta_{\mathrm{I}} \rangle$ に対して，以下の書き換え規則を適用できなくなるまで適用する．

$$\langle\{(p=x)\Rightarrow(p'=x')\}\cup C,\ \{p=x,p'=x''\}\cup\theta\rangle \to_{\mathrm{I}} \langle C,\{p=x,p'=x'\}\cup\theta\rangle$$

すなわち，$p=x$ が成り立っており，$(p=x)\Rightarrow(p'=x')$ という制約があったら，その制約を取り除き，p' の値を更新する．最終的に得られた制約集合と解を $\langle\tilde{C_{\mathrm{I}}},\tilde{\theta_{\mathrm{I}}}\rangle$ とする．

第２段階 初期制約集合 C_{II} と初期解 θ_{II} を以下のように定義する．

$$\begin{aligned} C_{\mathrm{II}} &= C^*_{init} \\ \theta_{\mathrm{II}} &= \{d=x \mid d=x\in\tilde{\theta_{\mathrm{I}}}\} \\ &\cup \{s=\mathsf{false} \mid \text{全ての } s \text{ について}\} \end{aligned}$$

$\langle C_{\mathrm{II}},\theta_{\mathrm{II}}\rangle$ に対して第１段階と同じ書き換え規則を適用できなくなるまで適用する．（第２段階の書き換えは $\to_{\mathrm{II}}$ で表す．）最終的に得られた制約集合と解を $\langle\tilde{C_{\mathrm{II}}},\tilde{\theta_{\mathrm{II}}}\rangle$ とする．

初期解と書き換え規則の作り方から θ には同じ p に対して複数の $p=x$ が含まれることはない．従って θ が求まれば，制約生成規則の $\rightsquigarrow M$ の部分から注釈を求めることができる．

以下 $p=x$ が θ に含まれるとき $\theta(p)=x$ と書く．また θ から $p=x'$ を取り除いて $p=x$ を加えて得られる解を $\theta[p\mapsto x]$ と書く．書き換え規則を適用する時，どの制約を取り除いたかが重要となる場合には，$c:(p=x)\Rightarrow(p'=x')$ のように制約に名前を付けた上で，$\langle C,\theta\rangle \xrightarrow{c}_{\mathrm{I}} \langle C',\theta'\rangle$ のように書く．

11 制約解消の例

7 節で得られた項 P に対する制約を例にとって制約解消を実際にやってみよう．まず π に関する制約を (s,d) に関する制約に変換すると以下のようになる．ここでは true を t，false を f と書いている．

$$\begin{array}{rl} \pi_1\ge\pi'_1: & \boxed{d'_1=\mathsf{t}\Rightarrow d_1=\mathsf{t}} \wedge \boxed{s_1=\mathsf{f}\Rightarrow s'_1=\mathsf{f}} \\ \pi'_1\triangleright\pi_3: & \boxed{d'_1=\mathsf{t}\Rightarrow d_3=\mathsf{t}} \wedge \boxed{s'_1=\mathsf{f}\Rightarrow s_3=\mathsf{f}} \\ \pi_2\triangleright\pi_3: & \boxed{d_2=\mathsf{t}\Rightarrow d_3=\mathsf{t}} \wedge \boxed{s_2=\mathsf{f}\Rightarrow s_3=\mathsf{f}} \\ \pi'_1\triangleright\pi_2: & \boxed{d'_1=\mathsf{t}\Rightarrow d_2=\mathsf{t}} \wedge \boxed{s'_1=\mathsf{f}\Rightarrow s_2=\mathsf{f}} \\ \pi_2\triangleright\pi'_1: & \boxed{d_2=\mathsf{t}\Rightarrow d'_1=\mathsf{t}} \wedge \boxed{s_2=\mathsf{f}\Rightarrow s'_1=\mathsf{f}} \end{array}$$

$$\begin{array}{rl} \pi'_1\in\{S,D\}: & \boxed{s'_1=\mathsf{f}\Rightarrow d'_1=\mathsf{t}} \wedge \boxed{d'_1=\mathsf{t}\Rightarrow s'_1=\mathsf{f}} \\ \pi_2\in\{S,D\}: & \boxed{s_2=\mathsf{f}\Rightarrow d_2=\mathsf{t}} \wedge \boxed{d_2=\mathsf{t}\Rightarrow s_2=\mathsf{f}} \\ \pi_4\triangleright'\pi_1: & \boxed{d_4=\mathsf{t}\Rightarrow d_1=\mathsf{t}} \wedge \boxed{d_4=\mathsf{t}\Rightarrow s_1=\mathsf{f}} \\ \pi_4\triangleright\pi_3: & \boxed{d_4=\mathsf{t}\Rightarrow d_3=\mathsf{t}} \wedge s_4=\mathsf{f}\Rightarrow s_3=\mathsf{f} \\ \pi_4\ge\pi'_4: & \boxed{d'_4=\mathsf{t}\Rightarrow d_4=\mathsf{t}} \wedge s_4=\mathsf{f}\Rightarrow s'_4=\mathsf{f} \\ \pi_4\ge\pi''_4: & d''_4=\mathsf{t}\Rightarrow d_4=\mathsf{t} \wedge s_4=\mathsf{f}\Rightarrow s''_4=\mathsf{f} \\ \pi''_4\triangleright'\pi_1: & d''_4=\mathsf{t}\Rightarrow d_1=\mathsf{t} \wedge d''_4=\mathsf{t}\Rightarrow s_1=\mathsf{f} \\ \pi''_4\triangleright\pi_3: & d''_4=\mathsf{t}\Rightarrow d_3=\mathsf{t} \wedge s''_4=\mathsf{f}\Rightarrow s_3=\mathsf{f} \\ \pi''_4\in\{S,D\}: & s''_4=\mathsf{f}\Rightarrow d''_4=\mathsf{t} \wedge d''_4=\mathsf{t}\Rightarrow s''_4=\mathsf{f} \\ \pi_3\ge\pi'_3: & \boxed{d'_3=\mathsf{t}\Rightarrow d_3=\mathsf{t}} \wedge \boxed{s_3=\mathsf{f}\Rightarrow s'_3=\mathsf{f}} \\ \pi_5\triangleright\pi'_4: & \boxed{d_5=\mathsf{t}\Rightarrow d'_4=\mathsf{t}} \wedge s_5=\mathsf{f}\Rightarrow s'_4=\mathsf{f} \\ \pi_5\triangleright\pi'_3: & \boxed{d_5=\mathsf{t}\Rightarrow d'_3=\mathsf{t}} \wedge s_5=\mathsf{f}\Rightarrow s'_3=\mathsf{f} \\ \pi_6\triangleright'\pi_4: & d_6=\mathsf{t}\Rightarrow d_4=\mathsf{t} \wedge d_6=\mathsf{t}\Rightarrow s_4=\mathsf{f} \\ \pi_6\triangleright\pi_5: & d_6=\mathsf{t}\Rightarrow d_5=\mathsf{t} \wedge s_6=\mathsf{f}\Rightarrow s_5=\mathsf{f} \\ \pi_5\ge\pi'_5: & \boxed{d'_5=\mathsf{t}\Rightarrow d_5=\mathsf{t}} \wedge s_5=\mathsf{f}\Rightarrow s'_5=\mathsf{f} \\ \pi_6\in\{S,D\}: & s_6=\mathsf{f}\Rightarrow d_6=\mathsf{t} \wedge d_6=\mathsf{t}\Rightarrow s_6=\mathsf{f} \end{array}$$

初期条件は $d'_5=\mathsf{true}$ である．それ以外の全ての d に対して $d=\mathsf{false}$，全ての s に対して $s=\mathsf{true}$ を代入し，上の制約を使って $d'_5=\mathsf{true}$ を伝搬していくと，四角で囲んだ制約が適用されて，以下のような結果を得る．

$$d_1=d'_1=d_2=d_3=d'_3=d_4=d'_4 = d_5=d'_5=\mathsf{true}$$

$$d''_4=d_6=\mathsf{false}$$

$$s_4=s'_4=s''_4=s_5=s'_5=s_6=\mathsf{true}$$

$$s_1=s'_1=s_2=s_3=s'_3=\mathsf{false}$$

これを π 上の制約に戻すと

$$\pi''_4=\pi_6=S,\quad \pi_4=\pi'_4=\pi_5=\pi'_5=B,$$
$$\pi_1=\pi'_1=\pi_2=\pi_3=\pi'_3=D$$

となり，これは以下のような注釈を得たことに相当する．

$$(\lambda^B f.(cons^B f{\downarrow}^B_D\ (f{\downarrow}^B_S\ @^S 2^D)){\downarrow}^B_D){\downarrow}^B_S\ @^S(\lambda^B x.x+^D 3^D)$$

第１段階は $d=\mathsf{true}$ すなわち D あるいは B となる部分を特定するが，逆に見るとこれは S となる部分を特定していることになる．上の結果を見ると，きちんと全体の関数適用と f の適用は静的とされていることがわかる．しかし，最初のλ式と $cons$ 文については無駄に静的かつ動的とされていることがわかる．この無駄な部分は第２段階で取り除かれる．第２段階は s の値を

false に戻した以下の状態から始める．

$$d_1 = d_1' = d_2 = d_3 = d_3' = d_4 = d_4' = d_5 = d_5' = \mathsf{true}$$
$$d_4'' = d_6 = \mathsf{false}$$
$$s_4 = s_4' = s_4'' = s_5 = s_5' = s_6 = \boxed{\mathsf{false}}$$
$$s_1 = s_1' = s_2 = s_3 = s_3' = \mathsf{false}$$

また，適用する制約は全て対偶が取られて以下のようになる．

$$\begin{array}{rrcl}
\pi_1 \geq \pi_1': & d_1 = \mathsf{f} \Rightarrow d_1' = \mathsf{f} & \wedge & s_1' = \mathsf{t} \Rightarrow s_1 = \mathsf{t} \\
\pi_1' \rhd \pi_3: & d_3 = \mathsf{f} \Rightarrow d_1' = \mathsf{f} & \wedge & s_3 = \mathsf{t} \Rightarrow s_1' = \mathsf{t} \\
\pi_2 \rhd \pi_3: & d_3 = \mathsf{f} \Rightarrow d_2 = \mathsf{f} & \wedge & s_3 = \mathsf{t} \Rightarrow s_2 = \mathsf{t} \\
\pi_1' \rhd \pi_2: & d_2 = \mathsf{f} \Rightarrow d_1' = \mathsf{f} & \wedge & s_2 = \mathsf{t} \Rightarrow s_1' = \mathsf{t} \\
\pi_2 \rhd \pi_1': & d_1' = \mathsf{f} \Rightarrow d_2 = \mathsf{f} & \wedge & s_1' = \mathsf{t} \Rightarrow s_2 = \mathsf{t} \\
\pi_1' \in \{S, D\}: & d_1' = \mathsf{f} \Rightarrow s_1' = \mathsf{t} & \wedge & s_1' = \mathsf{t} \Rightarrow d_1' = \mathsf{f} \\
\pi_2 \in \{S, D\}: & d_2 = \mathsf{f} \Rightarrow s_2 = \mathsf{t} & \wedge & s_2 = \mathsf{t} \Rightarrow d_2 = \mathsf{f} \\
\pi_4 \rhd' \pi_1: & d_1 = \mathsf{f} \Rightarrow d_4 = \mathsf{f} & \wedge & s_1 = \mathsf{t} \Rightarrow d_4 = \mathsf{f} \\
\pi_4 \rhd \pi_3: & d_3 = \mathsf{f} \Rightarrow d_4 = \mathsf{f} & \wedge & s_3 = \mathsf{t} \Rightarrow s_4 = \mathsf{t} \\
\pi_4 \geq \pi_4': & d_4 = \mathsf{f} \Rightarrow d_4' = \mathsf{f} & \wedge & s_4' = \mathsf{t} \Rightarrow s_4 = \mathsf{t} \\
\pi_4 \geq \pi_4'': & d_4 = \mathsf{f} \Rightarrow d_4'' = \mathsf{f} & \wedge & \boxed{s_4'' = \mathsf{t} \Rightarrow s_4 = \mathsf{t}} \\
\pi_4'' \rhd' \pi_1: & d_1 = \mathsf{f} \Rightarrow d_4'' = \mathsf{f} & \wedge & s_1 = \mathsf{t} \Rightarrow d_4'' = \mathsf{f} \\
\pi_4'' \rhd \pi_3: & d_3 = \mathsf{f} \Rightarrow d_4'' = \mathsf{f} & \wedge & s_3 = \mathsf{t} \Rightarrow s_4'' = \mathsf{t} \\
\pi_4'' \in \{S, D\}: & \boxed{d_4'' = \mathsf{f} \Rightarrow s_4'' = \mathsf{t}} & \wedge & \boxed{s_4'' = \mathsf{t} \Rightarrow d_4'' = \mathsf{f}} \\
\pi_3 \geq \pi_3': & d_3 = \mathsf{f} \Rightarrow d_3' = \mathsf{f} & \wedge & s_3' = \mathsf{t} \Rightarrow s_3 = \mathsf{t} \\
\pi_5 \rhd \pi_4': & d_4' = \mathsf{f} \Rightarrow d_5 = \mathsf{f} & \wedge & s_4' = \mathsf{t} \Rightarrow s_5 = \mathsf{t} \\
\pi_5 \rhd \pi_3': & d_3' = \mathsf{f} \Rightarrow d_5 = \mathsf{f} & \wedge & s_3' = \mathsf{t} \Rightarrow s_5 = \mathsf{t} \\
\pi_6 \rhd' \pi_4: & d_4 = \mathsf{f} \Rightarrow d_6 = \mathsf{f} & \wedge & \boxed{s_4 = \mathsf{t} \Rightarrow d_6 = \mathsf{f}} \\
\pi_6 \rhd \pi_5: & d_5 = \mathsf{f} \Rightarrow d_6 = \mathsf{f} & \wedge & s_5 = \mathsf{t} \Rightarrow s_6 = \mathsf{t} \\
\pi_5 \geq \pi_5': & d_5 = \mathsf{f} \Rightarrow d_5' = \mathsf{f} & \wedge & s_5' = \mathsf{t} \Rightarrow s_5 = \mathsf{t} \\
\pi_6 \in \{S, D\}: & \boxed{d_6 = \mathsf{f} \Rightarrow s_6 = \mathsf{t}} & \wedge & \boxed{s_6 = \mathsf{t} \Rightarrow d_6 = \mathsf{f}}
\end{array}$$

これに対して $d_4'' = d_6 = \mathsf{false}$ を伝搬させていくと，上の四角で囲んだ制約が適用されて以下の結果を得る．

$$d_1 = d_1' = d_2 = d_3 = d_3' = d_4 = d_4' = d_5 = d_5' = \mathsf{true}$$
$$d_4'' = d_6 = \mathsf{false}$$
$$s_4 = s_4'' = s_6 = \mathsf{true}$$
$$s_1 = s_1' = s_2 = s_3 = s_3' = s_4' = s_5 = s_5' = \mathsf{false}$$

d の値は第1段階終了時から変化していないことに注意しよう．上の結果を π 上の制約に戻すと

$$\pi_4'' = \pi_6 = S, \quad \pi_4 = B,$$
$$\pi_1 = \pi_1' = \pi_2 = \pi_3 = \pi_3' = \pi_4' = \pi_5 = \pi_5' = D$$

となり，これで以下の最も望ましい注釈を得る．

$$(\lambda^S f.\, cons^D f{\downarrow}_D^B\ (f{\downarrow}_S^B\ @^S 2^D))@^S(\lambda^B x.\, x +^D 3^D)$$

12 アルゴリズムの正当性

ここでは前節のアルゴリズムが実際，期待通り動くことを見ていく．以下では真偽値の間の大小関係を $\mathsf{true} > \mathsf{false}$ とする．

第1段階では，解 θ に対して $\theta \leq_{\mathrm{I}} \theta' \stackrel{\mathrm{def}}{\Longleftrightarrow} \forall s.\theta(s) \geq \theta'(s) \wedge \forall d.\theta(d) \leq \theta'(d)$ という大小関係を考える．すると解 θ は，この大小関係のもとで束となる．次の補題は，第1段階の制約の適用はこの束のもとで解が減少しないことを示している．

補題 2 $\langle C, \theta\rangle \rightarrow_{\mathrm{I}} \langle C', \theta'\rangle$ なら $\theta \leq_{\mathrm{I}} \theta'$.

証明： $\theta = \theta'$ なら補題は満たされている．$\theta \neq \theta'$ としよう．$\theta \neq \theta'$ となるのは，ある s について $\theta(s) = \mathsf{true}$ かつ $\theta'(s) = \mathsf{false}$ となっているか，ある d について $\theta(d) = \mathsf{false}$ かつ $\theta'(d) = \mathsf{true}$ となっているかのどちらかである．これは制約 $(p = x) \Rightarrow (p' = x')$ の $p' = x'$ の部分には $s = \mathsf{false}$ あるいは $d = \mathsf{true}$ という形しか現れていないためである．どちらの場合も $\theta \leq_{\mathrm{I}} \theta'$ となっているので補題が満たされている． □

補題 3 第1段階終了時の解 $\tilde{\theta}_{\mathrm{I}}$ は C_{init} 中の制約を全て満たしている．

証明： C_{init} 中の制約 $(p = x) \Rightarrow (p' = x')$ は，第1段階終了時の制約集合 $\tilde{C}_{\mathrm{I}}$ に含まれているものとそうでないものに分けられる．前者は $\tilde{\theta}_{\mathrm{I}}$ において制約の前提条件である $p = x$ が満たされなかった，ということを示しているので，この制約は自明に満たされている．後者の場合は，制約適用の過程で

$$\langle \{(p = x) \Rightarrow (p' = x')\} \cup C, \{p = x, p' = x''\} \cup \theta\rangle \rightarrow_{\mathrm{I}} \langle C, \{p = x, p' = x'\} \cup \theta\rangle$$

のような書き換えが行われ，制約 $(p = x) \Rightarrow (p' = x')$ が取り除かれるとともに，解が $p' = x'$ のように更新された，ということを示している．ここで，制約 $(p = x) \Rightarrow (p' = x')$ の $p' = x'$ の部分には $s = \mathsf{false}$ あるいは $d = \mathsf{true}$ という形しか現れていないことに

注意しよう．従って，ここで解は$s = \mathsf{false}$ あるいは$d = \mathsf{true}$ という形に書き換えられている．第１段階中これらは，補題 2 により $s = \mathsf{true}$ あるいは$d = \mathsf{false}$ となることはない．これより，第１段階終了時にも制約 $(p = x) \Rightarrow (p' = x')$ の $p' = x'$ が成り立っている，すなわちこの制約が満たされていることがわかる． □

次の補題は，2 つの制約を適用できるときには，それらをどのような順番で適用しても結果は変わらないことを示している．

補題 4 $\langle C_0, \theta_0\rangle \xrightarrow{c_1}_{\mathrm{I}} \langle C_1, \theta_1\rangle$ かつ $\langle C_0, \theta_0\rangle \xrightarrow{c_2}_{\mathrm{I}} \langle C_2, \theta_2\rangle$ なら，ある $\langle C_3, \theta_3\rangle$ が存在して $\langle C_1, \theta_1\rangle \xrightarrow{c_2}_{\mathrm{I}} \langle C_3, \theta_3\rangle$ かつ $\langle C_2, \theta_2\rangle \xrightarrow{c_1}_{\mathrm{I}} \langle C_3, \theta_3\rangle$.

証明： c_1, c_2 をそれぞれ $(p_1 = x_1) \Rightarrow (p_2 = x_2), (p_3 = x_3) \Rightarrow (p_4 = x_4)$ とおく．$\langle C_0, \theta_0\rangle$ からは c_1 も c_2 も適用可能なので $\theta_0(p_1) = x_1$ かつ $\theta_0(p_3) = x_3$ が成り立っている．

まず c_1 を先に適用する場合を考えよう．$\langle C_0, \theta_0\rangle \xrightarrow{c_1}_{\mathrm{I}} \langle C_1, \theta_1\rangle$ より $\theta_1 = \theta_0[p_2 \mapsto x_2]$ である．ここで $p_2 \neq p_3$ なら $\theta_1(p_3) = \theta_0(p_3) = x_3$ なので $\langle C_1, \theta_1\rangle$ に c_2 を適用することができ，$\theta_3 = \theta_1[p_4 \mapsto x_4] = \theta_0[p_2 \mapsto x_2][p_4 \mapsto x_4]$ となる．また，$p_2 = p_3$ の場合は（制約に出てくる $p_i = x_i$ は $d = \mathsf{true}$ あるいは$s = \mathsf{false}$ のどちらかの形なので）$x_2 = x_3$ となり，$\theta_1(p_3) = \theta_1(p_2) = x_2 = x_3$ から，やはり c_2 を適用することができ，同じ $\theta_3 = \theta_0[p_2 \mapsto x_2][p_4 \mapsto x_4]$ を得る．

全く同様にして c_2 を先に適用する場合を考えると，c_1 も続いて適用できることがわかり，その結果 $\theta'_3 = \theta_2[p_2 \mapsto x_2] = \theta_0[p_4 \mapsto x_4][p_2 \mapsto x_2]$ を得る．$p_2 \neq p_4$ の場合は，$\theta'_3 = \theta_0[p_4 \mapsto x_4][p_2 \mapsto x_2] = \theta_0[p_2 \mapsto x_2][p_4 \mapsto x_4] = \theta_3$．また，$p_2 = p_4$ の場合は（制約に出てくる $p_i = x_i$ は $d = \mathsf{true}$ あるいは$s = \mathsf{false}$ のどちらかの形なので）$x_2 = x_4$ となり，$\theta'_3 = \theta_0[p_4 \mapsto x_4][p_2 \mapsto x_2] = \theta_0[p_2 \mapsto x_2] = \theta_0[p_2 \mapsto x_2][p_4 \mapsto x_4] = \theta_3$ となる．

また，C_3 については，どちらの経路を通っても $C_0 \setminus \{c_1, c_2\}$ となるので等しい．以上により証明された． □

次の補題は，第１段階で得られる解は一意的であることを示している．ここで $\to^*$ は $\to$ の０回以上の繰り返しである．

補題 5 $\langle C, \theta\rangle \to^*_{\mathrm{I}} \langle \tilde{C}_{\mathrm{I}}, \tilde{\theta}_{\mathrm{I}}\rangle$ かつ $\langle C, \theta\rangle \to^*_{\mathrm{I}} \langle \tilde{C}'_{\mathrm{I}}, \tilde{\theta}'_{\mathrm{I}}\rangle$ ならば $\langle \tilde{C}_{\mathrm{I}}, \tilde{\theta}_{\mathrm{I}}\rangle = \langle \tilde{C}'_{\mathrm{I}}, \tilde{\theta}'_{\mathrm{I}}\rangle$.

証明： $\to_{\mathrm{I}}$ の適用回数に関する帰納法で証明する．一度も適用しない場合は，$\langle C, \theta\rangle = \langle \tilde{C}_{\mathrm{I}}, \tilde{\theta}_{\mathrm{I}}\rangle$ かつ $\langle C, \theta\rangle = \langle \tilde{C}'_{\mathrm{I}}, \tilde{\theta}'_{\mathrm{I}}\rangle$ なので成り立つ．

複数回，適用する場合を考える．$\langle C, \theta\rangle \to^*_{\mathrm{I}} \langle \tilde{C}_{\mathrm{I}}, \tilde{\theta}_{\mathrm{I}}\rangle$ で最初に適用する制約を $c : (p = x) \Rightarrow (p' = x')$ としよう．$\langle C, \theta\rangle$ においてこの制約が適用可能なので $\theta(p) = x$ である．ここで $p = x$ は $d = \mathsf{true}$ あるいは $s = \mathsf{false}$ のどちらかの形であり，かつ C 中の他の制約の結論部分も $d = \mathsf{true}$ あるいは $s = \mathsf{false}$ という形なので，θ で成り立っている $p = x$ という関係は，その後の制約の適用で満たされなくなることはない．従って，$\langle C, \theta\rangle$ から始まる全ての書き換えにおいてずっと $p = x$ が成り立っている．これより $\langle C, \theta\rangle \to^*_{\mathrm{I}} \langle \tilde{C}'_{\mathrm{I}}, \tilde{\theta}'_{\mathrm{I}}\rangle$ においても，制約 c はいつでも適用可能で，かつ必ずどこかで適用されていることがわかる．どこで適用されていても，補題 4 を複数回使用することで，$\langle C, \theta\rangle \xrightarrow{c}_{\mathrm{I}} \langle C_1, \theta_1\rangle \to^*_{\mathrm{I}} \langle \tilde{C}'_{\mathrm{I}}, \tilde{\theta}'_{\mathrm{I}}\rangle$ のように最終状態を変えることなく c を最初に適用するようにすることができる．$\langle C, \theta\rangle \xrightarrow{c}_{\mathrm{I}} \langle C_1, \theta_1\rangle \to^*_{\mathrm{I}} \langle \tilde{C}_{\mathrm{I}}, \tilde{\theta}_{\mathrm{I}}\rangle$ であったので，$\langle C_1, \theta_1\rangle$ に対して帰納法の仮定を使うことができ $\langle \tilde{C}_{\mathrm{I}}, \tilde{\theta}_{\mathrm{I}}\rangle = \langle \tilde{C}'_{\mathrm{I}}, \tilde{\theta}'_{\mathrm{I}}\rangle$ が言える． □

このアルゴリズム（の第１段階）は，解として全ての s が false，d が true の場合を考えると，自明に全ての制約を満たしているので，解は存在する．第１段階のアルゴリズムは，最小の解から必要なもののみについて単調に解を更新することで制約を満たしているので，いずれ最小の解に到達する．従って，結果は期待通り制約を満たすもののうちで d が false となっているものが最も多いものとなっていることがわかり，次の定理が導かれる．

定理 4 第１段階終了時の解 $\tilde{\theta}_{\mathrm{I}}$ から得られる注釈は，定義 5 の最も望ましい注釈の条件のうち，最初の条件を満たしたものとなっている．

次に第２段階である．今度は $s = \mathsf{true}$，d は第１段階で得られたものに設定して制約（の対偶）を適用する．次の補題は，第２段階では d はもはや変化しないことを示す．

補題 6 全ての d について $\theta_{\mathrm{II}}(d) = \tilde{\theta}_{\mathrm{II}}(d)$.

証明： 書き換えの長さに関する帰納法で，第2段階中全ての θ について $\theta(d) = \theta_{\mathrm{II}}(d)$ であることを証明する．書き換えがなかった場合には $\theta_{\mathrm{II}}(d) = \tilde{\theta_{\mathrm{II}}}(d)$ なので成り立つ．今，書き換えが $\langle C^*_{init}, \theta_{\mathrm{II}}\rangle$ から $\langle C, \theta\rangle$ まで進んだところで補題が成り立っている，すなわち，全ての d について $\theta(d) = \theta_{\mathrm{II}}(d)$ である，としよう．このとき，その θ からもう一度，書き換え $\langle C, \theta\rangle \xrightarrow{c} \langle C', \theta'\rangle$ を行っても $\theta'(d) = \theta(d) = \theta_{\mathrm{II}}(d)$ であることを示す．

第2段階で使われる制約 $(p_1 = x_1) \Rightarrow (p_2 = x_2)$ は，第1段階の制約の対偶となっているので $p_i = x_i$ の部分は $s = \mathsf{true}$ または $d = \mathsf{false}$ のどちらかの形である．このうち，制約の適用の結果 $\theta(d) \neq \theta'(d)$ となり得るのは，制約の結論部分が $d = \mathsf{false}$ の形，すなわち $(d_1 = \mathsf{false}) \Rightarrow (d_2 = \mathsf{false})$ または $(s = \mathsf{true}) \Rightarrow (d = \mathsf{false})$ のどちらかの形である．それぞれについて考えてみよう．

第1段階終了時の解 $\tilde{\theta_{\mathrm{I}}}$ は C_{init} 中の全ての制約を満たしているので，その対偶をとった C^*_{init} 中の制約も全て満たしている．d の値については $\tilde{\theta_{\mathrm{I}}}(d) = \theta_{\mathrm{II}}(d)$ であり，かつ帰納法の仮定から $\theta(d) = \theta_{\mathrm{II}}(d)$ なので θ は $(d_1 = \mathsf{false}) \Rightarrow (d_2 = \mathsf{false})$ の形の制約については全て満たしている．すなわち $\theta(d_1) = \mathsf{true}$ であるか $\theta(d_1) = \mathsf{false}$ かつ $\theta(d_2) = \mathsf{false}$ である．前者の場合はこの制約が適用不可能であり，後者の場合は適用により $\theta' = \theta[d_2 \mapsto \mathsf{false}] = \theta$ である．従って $(d_1 = \mathsf{false}) \Rightarrow (d_2 = \mathsf{false})$ の場合は証明された．

$(s = \mathsf{true}) \Rightarrow (d = \mathsf{false})$ の場合を考えよう．この制約が適用されるためには $\theta(s) = \mathsf{true}$ でなければならないが，第2段階の開始時の解 θ_{II} では全ての s は false である．$\theta(s)$ が true となるのは，ある d について $\theta(d) = \mathsf{false}$ のもとで $(d = \mathsf{false}) \Rightarrow (s = \mathsf{true})$ の形の制約を適用した場合と，その後 $(s = \mathsf{true}) \Rightarrow (s' = \mathsf{true})$ の形の制約を適用した場合のみである．今，第2段階開始時において $\theta_{\mathrm{II}}(d_0) = \mathsf{false}$ が成り立っていると仮定しよう．そのもとで $(d_0 = \mathsf{false}) \Rightarrow (s_0 = \mathsf{true})$, $(s_{i-1} = \mathsf{true}) \Rightarrow (s_i = \mathsf{true})$ $(1 \leq i \leq n)$ という順番で制約を適用して $\theta(s_n) = \mathsf{true}$ となり，ここで $(s_n = \mathsf{true}) \Rightarrow (d_n = \mathsf{false})$ を適用したとしよう．すると実はすでに $\theta(d_n) = \mathsf{false}$ であり，制約を適用しても d_n の値は変化しないことが証明できる（以下を参照）．従って $\theta' = \theta[d_n \mapsto \mathsf{false}] = \theta$ となり $(s = \mathsf{true}) \Rightarrow (d = \mathsf{false})$ の形の場合も証明できた．

$\theta(d_n) = \mathsf{false}$ であることは，第1段階終了時の解 $\tilde{\theta_{\mathrm{I}}}$ が全ての制約を満たしている（補題 3）ことからわかる．仮に $\theta(d_n) = \mathsf{true}$ であったとしよう．帰納法の仮定より $\tilde{\theta_{\mathrm{I}}}(d_n) = \theta_{\mathrm{II}}(d_n) = \theta(d_n) = \mathsf{true}$ である．上で適用した制約の対偶を考えると $(d_n = \mathsf{true}) \Rightarrow (s_n = \mathsf{false})$, $(s_i = \mathsf{false}) \Rightarrow (s_{i-1} = \mathsf{false})$ $(n \geq i \geq 1)$, $(s_0 = \mathsf{false}) \Rightarrow (d_0 = \mathsf{true})$ となり，これらは第1段階終了時にはみな満たされていたので，$\tilde{\theta_{\mathrm{I}}}(d_0) = \mathsf{true}$ であったはずだが，これは上で $\tilde{\theta_{\mathrm{I}}}(d_0) = \theta_{\mathrm{II}}(d_0) = \mathsf{false}$ と仮定したことに矛盾する． □

これにより，第2段階では s についてのみ考えれば良い．これは「第2段階では無駄な B を D にする」という直観にも合致している†6．

第2段階では，解 θ に対して $\theta \leq_{\mathrm{II}} \theta' \overset{\mathrm{def}}{\iff} \forall s.\theta(s) \leq \theta'(s) \wedge \forall d.\theta(d) = \theta'(d)$ という大小関係を考える．s に関する大小関係が第1段階とは逆になっていることに注意しよう．また，第2段階では制約の対偶をとるので，制約の $p = x$ の部分に現れるのは $s = \mathsf{true}$ あるいは $d = \mathsf{false}$ という形である．これらのことに注意すれば，以下の4つの補題は第1段階と全く同じようにして証明することができる．

補題 7 $\langle C, \theta\rangle \to_{\mathrm{II}} \langle C', \theta'\rangle$ なら $\theta \leq_{\mathrm{II}} \theta'$．（第2段階の制約の適用で解は減少しない．）

補題 8 第2段階終了時の解 $\tilde{\theta_{\mathrm{II}}}$ は C^*_{init} 中の制約を全て満たしている．（従って C_{init} 中の制約も全て満たしている．）

補題 9 $\langle C_0, \theta_0\rangle \xrightarrow{c_1}_{\mathrm{II}} \langle C_1, \theta_1\rangle$ かつ $\langle C_0, \theta_0\rangle \xrightarrow{c_2}_{\mathrm{II}} \langle C_2, \theta_2\rangle$ なら，ある $\langle C_3, \theta_3\rangle$ が存在して $\langle C_1, \theta_1\rangle \xrightarrow{c_2}_{\mathrm{II}} \langle C_3, \theta_3\rangle$ かつ $\langle C_2, \theta_2\rangle \xrightarrow{c_1}_{\mathrm{II}} \langle C_3, \theta_3\rangle$．（制約の適用順によって結果は変わらない．）

補題 10 $\langle C, \theta\rangle \to^*_{\mathrm{II}} \langle \tilde{C_{\mathrm{II}}}, \tilde{\theta_{\mathrm{II}}}\rangle$ かつ $\langle C, \theta\rangle \to^*_{\mathrm{II}} \langle \tilde{C'_{\mathrm{II}}}, \tilde{\theta'_{\mathrm{II}}}\rangle$ ならば $\langle \tilde{C_{\mathrm{II}}}, \tilde{\theta_{\mathrm{II}}}\rangle = \langle \tilde{C'_{\mathrm{II}}}, \tilde{\theta'_{\mathrm{II}}}\rangle$．（第2段階で得られる解は一意的である．）

第1段階の結果である $\tilde{\theta_{\mathrm{II}}}$ を持ってくれば，C^*_{init} 中の制約は全て満たされているので，解は存在する．s について最小の解から始めて，必要なもののみについて単

†6 加えて，無駄な S を $\bot$ にもしている．

調に解を更新することで制約を満たしているので，いずれ最小の解に到達する．d の値は第１段階で得られるものと同じなので，第２段階の結果は期待通り，制約を満たすもののうちで d が false となっているものが最も多く，その中で s が false となっているものが最も多いものとなっていることがわかる．よって次の定理が導かれる．

定理 5 第２段階終了時の解 $\tilde{\theta}_{\mathrm{II}}$ から得られる注釈は，定義 5 の最も望ましい注釈の条件を満たしたものとなっている．

13 計算量

ここでは，アルゴリズムがプログラムの長さに対してほぼ線形時間であることを見ていく．まず，生成される制約の数は各命令に対して定数なので，全体としてプログラムの長さに対して線形である．しかし，制約を生成する過程で変数の単一化が起きるため，そのときに union/find アルゴリズム [15] が必要となる．union/find アルゴリズムの計算量は，union/find が適用される回数を m，union/find の対象となる変数の数を n，アッカーマン関数の逆関数を α とすると $O(m\alpha(m,n))$ である [15]．プログラムの大きさを N とすると，m も n も N で押えられるので，union/find アルゴリズムに必要な計算量は全体で $O(N\alpha(N,N))$ であるが，実際的なプログラムに対しては $\alpha(N,N)$ は 4 以下であることがわかっている [15] ので，ほぼ線形と言うことができる．

第１段階は，初期条件の $d_{top} = \mathsf{true}$ から始めて，以後 $d = \mathsf{true}$ あるいは $s = \mathsf{false}$ を順に伝搬していくことで線形時間で終了する．これは，各制約が一度適用されたらその後はずっと満たされ続けるため，高々制約の数しか適用する必要がないためである．第２段階も同様で，$d = \mathsf{false}$ であるもの全てについて s の値を true にするか調べ，それを伝搬していくことで，線形時間で終了する．

14 関連研究

本研究は，Hornof [11] らが C で行った束縛時解析と密接な関係がある．彼らは，抽象解釈に基づいて構造データの使用と定義を前向き解析と後向き解析を使って求めている．これは，ちょうど本稿で s と d を使って２段階で制約を解消しているのに相当している．本研究では，彼らの仕事を型システムとして定式化するとともに，高階関数についても制限付きながら同様のことが行えることを示した．

本稿で示した型システムとその制約解消系は，Henglein [9] の作成した従来の束縛時解析を静的かつ動的な束縛時を許すように拡張したものと見ることができる．静的かつ動的な値が入ったことで制約解消は２段階になったが，制約解消の計算量は従来のものと同じくほぼ線形で行うことができる．

Sperber [13] は，静的，動的，そして未知，という３つの束縛時を扱う部分評価器を作成している．これは，静的，動的以外の値を扱う，という点では本研究と類似しているが，内容的には異なっている．Sperber の部分評価器は，条件分岐のそれぞれの束縛時が異なるときに，その条件文の束縛時を動的とはせずに未知としておき，online にその束縛時を判断する，というもので，「静的または動的」な式を online の手法で扱おうというものである．

「静的かつ動的」な式を扱う部分評価器は，筆者が [1] において発表している．本稿では，そのうちコアとなる部分について正当性の証明を行った．本稿の仕事をより発展させたものとして，住井ら [14] による束縛時解析があげられる．彼らは，本稿の枠組と Sperber の枠組を統合する形でより一般的な束縛時解析手法を提案している．

15 おわりに

本稿では，関数型言語の部分評価器で構造データの扱いを向上すべく，静的かつ動的な値を許すような束縛時解析を提案した．静的かつ動的な値を使うと，静的に構造データにアクセスしつつ，その構造データをコードとして最終結果に残すことができるようになる．この方法は実質的に，従来許されていなかった構造データの lift を許すようにした，と見ることもできる．本稿で提案した束縛時解析は，型システムとして定式化され，型推論の際に生成される束縛時に関する制約を解くことで結果を得る．さらに，型システムの正当性を示すとともに，制約の解消が従来の効率的な束縛時解析と同じくほぼ線形時間でできることを示した．これに基づいて処理系を

作成中であり，現在ほぼ動くに至っている．

ここで示した枠組には「静的かつ動的な関数の引数は動的」という強い制限がある．現在，この制限を取り除くべく，本稿の仕事と多相束縛時解析[10][5]との関係を調査中である．また，本稿ではベースとなる言語はsimply typedであると仮定したが，そこに多相性を導入するとどうなるか，というのも興味深い．その他には，実際に大きなプログラムを使った部分評価の実験と評価，枠組の多段の部分評価[8]への拡張などが今後の課題としてあげられる．

謝辞

日頃より本研究について多くのコメントを頂いている増原英彦氏，およびいろいろなコメントを頂いた査読者の方々に感謝いたします．

参考文献

[1] Asai, K.：Binding-Time Analysis for Both Static and Dynamic Expressions, SAS'99, Static Analysis (LNCS 1694), 1999, pp. 117–133.
[2] Ashley, J. M. and Consel, C.：Fixpoint Computation for Polyvariant Static Analyses of Higher-Order Applicative Programs, *ACM Transactions on Programming Languages and Systems*, Vol. 16, No. 5 (1994), pp. 1431–1448.
[3] Bondorf, A.：*Similix 5.0 Manual*, DIKU, University of Copenhagen, 1993.
[4] Dean, J., Chambers, C. and Grove, D.：Identifying Profitable Specialization in Object-Oriented Languages, ACM SIGPLAN Workshop on Partial Evaluation and Semantics-Based Program Manipulation (PEPM '94), 1994, pp. 85–96.
[5] Dussart, D., Henglein, F. and Mossin, C.：Polymorphic Recursion and Subtype Qualifications: Polymorphic Binding-Time Analysis in Polynomial Time, SAS'95, Static Analysis (LNCS 983), 1995, pp. 118–135.
[6] 藤波順久：オブジェクト指向言語を利用した自動的かつ効率的な実行時コード生成，コンピュータソフトウェア，Vol. 15, No. 5 (1998), pp. 25–37.
[7] Futamura, Y.：Partial evaluation of computation process – an approach to a compiler-compiler, *Systems, Computers, Controls*, Vol. 2, No. 5 (1971), pp. 45–50.
[8] Glück, R. and Jørgensen, J.：An Automatic Program Generator for Multi-Level Specialization, *Lisp and Symbolic Computation*, Vol. 10, No. 2 (1997), pp. 113–158.
[9] Henglein, F.：Efficient Type Inference for Higher-Order Binding-Time Analysis, Functional Programming Languages and Computer Architecture (LNCS 523), 1991, pp. 448–472.
[10] Henglein, F. and Mossin, C.：Polymorphic Binding-Time Analysis, ESOP'94, 5th European Symposium on Programming (LNCS 788), 1994, pp. 287–301.
[11] Hornof, L., Consel, C. and Noyé, J.：Effective Specialization of Realistic Programs via Use Sensitivity, SAS'97, Static Analysis (LNCS 1302), 1997, pp. 63–73.
[12] Jones, N. D., Gomard, C. K. and Sestoft, P.：*Partial Evaluation and Automatic Program Generation*, Prentice-Hall, New York, 1993.
[13] Sperber, M.：Self-Applicable Online Partial Evaluation, Partial Evaluation (LNCS 1110), 1996, pp. 465–480.
[14] Sumii, E. and Kobayashi, N.：Online-and-Offline Partial Evaluation: A Mixed Approach, ACM SIGPLAN Workshop on Partial Evaluation and Semantics-Based Program Manipulation (PEPM '00), 2000, pp. 12–21.
[15] Tarjan, R.：Data Structures and Network Flow Algorithms, *Regional Conference Series in Applied Mathematics*, Vol. CMBS 44 (1983), SIAM.