ウェブアプリケーション開発のための PHP オンライン部分評価器の試作

A Prototype of PHP Online Partial Evaluator for Web Application Development

吉田 敦 * 蜂巣 吉成 † 阿草 清滋 ‡

あらまし PHP を用いたページ記述は，ページの参照時の状態によってページ内の構成が変わることが多い．本論文は，特定の状態のときのページ内容の確認を支援するためのオンライン部分評価方法を提案する．

1 はじめに

ウェブアプリケーションは，複数の言語を組み合わせて開発されることが多いが，主要な言語の一つが PHP であり，HTML のページ記述の中に PHP のコードが埋め込まれる．ページ記述の中に条件分岐や繰り返しが含まれ，ユーザからの入力やデータベースの実行時の状態などによって，実際に返すページ記述が変わる．開発時には，各状態ごとのページ記述を確認したいが，ページの出力には，もともと HTML として記述された文字列と PHP の命令が出力した文字列が混在し，問題を発見しにくい．そこで，部分実行を利用し，特定の状態を決めるパラメタに特殊化したページ記述に変換することを考える．特殊化後のページであれば，文字列の混在の問題を回避しやすく，誤りやその原因を探しやすい．

PHP に対する部分評価としては，最適化への応用があるが [1] ライブラリやフレームワークを含めた束縛時解析が困難であり，実用的ではない．一方，プログラムの実行をトレースしながら部分評価を行うオンライン手法 [2] であれば，束縛時解析が不要になるが，PHP への適用例は知られていない．また，PHP は eval で自己評価ができ，また，スーパーグローバル変数など定義済みの要素も存在するので，部分評価器に PHP 自体を用いることで，実装が容易になると期待できる．

一般に，部分評価では，コメントや改行など，実行時には意味を持たない要素は考慮しない．しかし，特殊化後のページ記述で発見した誤りに対する修正は，特殊化前のページ記述に対して行うので，特殊化前後のページ記述間の比較が容易にできるよう，部分評価によって置き換わる式や文以外の要素は，できるだけ元の記述を保持すべきである．また，部分評価の結果は，その時点での変数の状態に依存する．部分評価では通常残らない計算可能な代入文も残すことで，変数の状態や特殊化された理由が理解しやすくなる．さらに，代入文が残ることで，特殊化前後の対応も取りやすくなる．

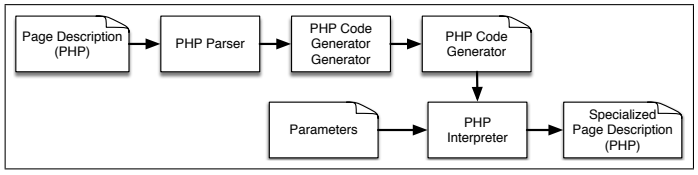

図 1 部分評価の流れ

そこで，本論文では，PHP 解釈器自体で動作し，空白などの意味を持たない要素や代入文などを残す PHP の部分評価器を提案する．図 1 に示すように，ページ記述を，その記述自体を出力する PHP のコード生成器に変換する．この生成器は，部分評価器そのものであり，パラメタを与えることで，計算可能な箇所は評価結果を出力し，そ

*Atsushi Yoshida, 南山大学
†Yoshinari Hachisu, 南山大学
‡Kiyoshi Agusa, 南山大学

れ以外は元の記述をそのまま出力する．

以下では，部分評価器の構成と具体的な事例に適用した結果を示す．また，考察として，この手法の適用可能な範囲や問題点について議論する．

2 部分評価器の構成

2.1 PHP コード生成器への変換

部分評価は，対象となるページ記述を，それ自体を出力する PHP コード生成器に変換することで実現する．変換にあたっては，まず，PHP に対する抽象構文木を構成する．PHP タグの外側の HTML 記述は，各断片ごとに PHP の出力命令として扱う．次に，各構文要素を，実行評価順に，それ自体を出力する処理に変換していく．

式の部分評価結果は，式が計算可能だった場合は，その評価結果の値，そうでない場合は，その式に対応する PHP コードのテキスト断片を持つオブジェクトで表現する．ただし，計算不可能な場合に，内部の計算可能な部分式は評価値に置き換わっている．ここで，計算可能とは，その式の直下の部分木がすべて計算可能であることである．ただし，葉にあたる変数参照は変数に値が値が代入されている場合に，定数は無条件に計算可能と扱う．

```
 1  <html>
 2    <head> <title>Sample page</title>
 3      <link rel="stylesheet" type="text/css"
 4            href="tr_even_odd.css" /> </head>
 5  <body> <h1>Sample page</h1>
 6
 7  <?php
 8  #   $max = $_REQUEST["max"];
 9  #   $id = $_REQUEST["id"];
10    $max = $argv[1];
11    $id = $argv[2];
12    if ($max <= 0) {
13      echo '<b>Error: illegal request</b>';
14    } else {
15  ?>
16  <table>
17  <?php $i = 1; while($i < $max) { ?>
18    <tr class="<?= tr_class($i) ?>">
19      <td> <?= $i ?> </td>
20      <td> <?= get_value($i, $id) ?> </td>
21    </tr>
22  <?php
23      $i = $i + 1;
24    }
25  ?>
26  </table>
27
28  <?php
29    }
30
31    function tr_class($x) {
32      return $x % 2 ? "even" : "odd";
33    }
34  ?>
35
36  </body></html>
```

図 2　ページ記述の例

2.2 部分評価の例

図 2 に評価対象のページ記述の例を示す．これは開発途中のものとし，後述する誤りを含む．関数 get_value は未定義である．変数 $_REQUEST はコマンドラインからの実行では扱えないので，代わりに $argv から読み込むように修正している．変換して得られる PHP コード生成器を図 3 に示す．引数に何も与えないで図 3 を実行すると，図 2 と空白等も含め完全に一致するページ記述を出力する．

コマンドラインの 2 つの引数のうち，最初の方のみを指定した実行結果を図 4 と図 5 に示す．図 4 は，正常な入力があった場合のページ構成となり，繰り返し文が展開されている．図 5 は，不正な入力として 0 が与えられており，エラーメッセージを出力するページ記述になっている．図 4 では，tr タグの属性 class が偶数行と奇数行で逆に設定されていることから，図 2 に誤りがあることがわかる．

2.3 対象とする構文要素

部分評価の開発支援への適用可能性の確認を目的に，変数への代入，論理演算以外の 2 項演算子を用いた式，if 文，while 文，関数呼び出し，break 文，出力コマンド (echo と printf) のみを対象とする．これらは，構造化プログラミングにおける基本要素である順次，反復，分岐を確認するために最低限必要なものである．また，代入は右辺の評価値が整数，実数，文字列のいずれかである場合に限定する．

2.4 実行状態の管理

実行トレース時は，変数とその値の表を実行状態として管理する必要がある．実装の簡単化のために，表の管理は PHP の解釈器の本来の機能に任せ，変数の代入は，左辺の変数の参照を受け取って処理する．例えば，図 3 の 6 行目の Synx::assign() が代入を表し，第 3 引数が変数の参照である．

```
<?php
include "Synx.php";
$_c = [];  # for if-condition
$_v = [];  # for assigned variables
print "<html>\n  <head> <title>Sample page</title>\n    <link rel=\"stylesheet\" type=\"text/
    css\"\n      href=\"tr_even_odd.css\" /> </head>\n<body> <h1>Sample page</h1>\n\n<?php\n#
      \$max = \$_REQUEST[\"max\"];\n#  \$id = \$_REQUEST[\"id\"];\n  ";
Synx::stmt(Synx::assign($_c, '$max', $max, Synx::arr1('$argv', $argv, Synx::constref('1', 1),
    '#_#[#_#]'), '#_# = #_#'), '#_#;');
print "\n  ";
Synx::stmt(Synx::assign($_c, '$id', $id, Synx::arr1('$argv', $argv, Synx::constref('2', 2), '
    #_#[#_#]'), '#_# = #_#'), '#_#;');
print "\n  ";
$_c = [ Synx::op2('<=', Synx::varref('$max', $max), Synx::constref('0', 0), '#_# <= #_#'),
    $_c ];
if (!$_c[0]->isVal()) { $_v = [ [(isset($i)?$i:NULL)], $_v ]; }
Synx::ctrl($_c[0], 'if (#_#) ');
if (!$_c[0]->isVal() || $_c[0]->val) {
print "{\n    ";
Synx::stmt_cmd('echo', Synx::constref("'<b>Error: illegal request</b>'", '<b>Error: illegal
    request</b>'), '#_# #_#;');
print "\n  ";
print "}";

}
if (!$_c[0]->isVal()) {
  $_v0 = [(isset($i) ? $i : NULL)];
  list($i) = $_v[0];
  $_v[0] = $_v0;
}
Synx::ctrl($_c[0], ' else ');
if (!$_c[0]->isVal() || !$_c[0]->val) {
print "{\n?>\n<table>\n<?php ";
Synx::stmt(Synx::assign($_c, '$i', $i, Synx::constref('1', 1), '#_# = #_#'), '#_#;');
print " ";
$_c = [ Synx::op2('<', Synx::varref('$i', $i), Synx::varref('$max', $max), '#_# < #_#'), $_c
    ];
if (!$_c[0]->isVal()) {
if (isset($i)) { $i = NULL;}
$_c[0] = Synx::op2('<', Synx::varref('$i', $i), Synx::varref('$max', $max), '#_# < #_#');
}
while (1) {
    Synx::ctrl($_c[0], 'while(#_#) ');
    if (!$_c[0]->isVal() || $_c[0]->val) {
print "{ ?>\n  <tr class=\"<?=";
Synx::stmt_cmd('', Synx::call("tr_class", function($a1)  { return tr_class($a1); }, [Synx::
    varref('$i', $i)], '#_#(#_#)'), '#_# #_# ');
print "?>\">\n    <td> <?=";
Synx::stmt_cmd('', Synx::varref('$i', $i), '#_# #_# ');
print "?> </td>\n    <td> <?=";
Synx::stmt_cmd('', Synx::call("get_value", function($a1, $a2)  { return get_value($a1, $a2);
    }, [Synx::varref('$i', $i),Synx::varref('$id', $id)], '#_#(#_#, #_#)'), '#_# #_# ');
print "?> </td>\n  </tr>\n<?php\n    ";
Synx::stmt(Synx::assign($_c, '$i', $i, Synx::op2('+', Synx::varref('$i', $i), Synx::constref(
    '1', 1), '#_# + #_#'), '#_# = #_#'), '#_#;');
print "\n  ";
print "}";

    }
    if (!$_c[0]->isVal() || !$_c[0]->val) break;
    $_c[0] = Synx::op2('<', Synx::varref('$i', $i), Synx::varref('$max', $max), '#_# < #_#');
}
$_c = $_c[1];
print "\n?>\n</table>\n\n<?php\n  ";
print "}";

}
if (!$_c[0]->isVal()) {
  list($i0) = $_v[0];
  if ($i0 != $i) { $i = NULL; }
  $_v = $_v[1];}
$_c = $_c[1];
print "\n\n  ";
print "function tr_class(\$x) {\n    return \$x % 2 ? \"even\" : \"odd\";\n  }";
function tr_class($x) {
    return $x % 2 ? "even" : "odd";
  }
print "\n?>\n\n</body></html>\n";
?>
```

図 3　部分評価用の PHP コード生成器の例

2.5　非構文要素の保持

部分評価を最適化に用いた場合，コメントや空白など，抽象構文木には含まれない要素は保存する必要がない．そのような要素を「非構文要素」と呼ぶ．理解支援に用いる場合は，部分評価前後でのページ記述を見比べるので，部分評価で置き換わる部分以外は元のテキストがそのまま残るよう，非構文要素を保持する．

具体的には，構文要素ごとに，その元のテキストを表すテンプレートを保持する．

テンプレートは，構文要素の直下の部分木の部分は穴にしたテキストである．図 3 において，各構文要素に対応するメソッド呼び出しの最後尾の実引数がテンプレートであり，“**#_#**” が 1 つの穴を表す．この穴に，各部分木の評価値を当てはめることで，計算不可能な構文要素に対するテキストが構成される．

2.6 制御文

制御文については，条件式が計算不可能な場合は，その制御文全体を出力する．ただし，内部に含む計算可能な式や文は，評価結果に置き換える．その際，制御文内の代入の影響を考慮する．if 文の場合，制御の流れが分岐し，合流する時点で，変数には各分岐で異なる値が代入されている可能性がある．等しい値を持つ場合はその値を保持し，それ以外は変数を未定義にする．図 3 の 11 行目では，分岐の直前の変数 $i の値を保存する．20〜24 行目では，else 節の評価開始のために，then 節直後の変数の値を退避し，if 文実行前の状態に戻す．58〜61 行目は，分岐の合流処理として，変数の値の保持，または，未定義を意味する NULL の代入を行う．

繰り返し文の中では，変数の代入が，それより前の変数参照に影響する可能性がある．そこで，繰り返しの実行直前で，条件式が計算不能であれば，内部で代入される可能性のある変数はすべて未定義にする．図 3 において，20 行目で条件式の計算可能性を調べ，32 行目で代入されうる変数 $i に NULL を代入する．

制御文は複合文を伴うことが多いが，条件式が計算可能であれば，文全体の評価結果としては複合文の波括弧は不要になることが多い．例えば，図 4 の表の列を表すタグを囲む波括弧は削除できる．しかし，削除の可能性の評価は前後の文脈の解析が必要であることや，制御文の内部であることを示唆するために，そのまま残す．

繰り返しの展開により，図 4 では，各繰り返しの複合文の終り (左波括弧) の直後に次の始まり (右波括弧) が存在している．文の範囲に直後の改行が含めることで，可読性を向上できるので，検討が必要である．

```
 1  <html>
 2    <head> <title>Sample page</title>
 3      <link rel="stylesheet" type="text/css"
 4            href="tr_even_odd.css" /> </head>
 5  <body> <h1>Sample page</h1>
 6
 7  <?php
 8  #   $max = $_REQUEST["max"];
 9  #   $id = $_REQUEST["id"];
10    $max = '4';
11    $id = $argv[2];
12    {
13  ?>
14  <table>
15  <?php $i = 1; { ?>
16    <tr class="<?= 'even' ?>">
17      <td> <?= 1 ?> </td>
18      <td> <?= get_value(1, $id) ?> </td>
19    </tr>
20  <?php
21      $i = 2;
22    }{ ?>
23    <tr class="<?= 'odd' ?>">
24      <td> <?= 2 ?> </td>
25      <td> <?= get_value(2, $id) ?> </td>
26    </tr>
27  <?php
28      $i = 3;
29    }{ ?>
30    <tr class="<?= 'even' ?>">
31      <td> <?= 3 ?> </td>
32      <td> <?= get_value(3, $id) ?> </td>
33    </tr>
34  <?php
35      $i = 4;
36    }
37  ?>
38  </table>
39
40  <?php
41    }
42
43    function tr_class($x) {
44      return $x % 2 ? "even" : "odd";
45    }
46  ?>
47
48  </body></html>
```

図 4　引数が 4 のときの適用結果

```
 1  <html>
 2    <head> <title>Sample page</title>
 3      <link rel="stylesheet" type="text/css"
 4            href="tr_even_odd.css" /> </head>
 5  <body> <h1>Sample page</h1>
 6
 7  <?php
 8  #   $max = $_REQUEST["max"];
 9  #   $id = $_REQUEST["id"];
10    $max = '0';
11    $id = $argv[2];
12    {
13      echo '<b>Error: illegal request</b>';
14    }
15
16    function tr_class($x) {
17      return $x % 2 ? "even" : "odd";
18    }
19  ?>
20
21  </body></html>
```

図 5　引数が 0 のときの適用結果

2.7 代入式

代入式は，変数に値を代入するのみなので，計算可能な場合は削除して構わない．しかし，繰り返し文で変数を未定義にしたときに，元の値を失わず，また，実行時の状態を把握できるよう，代入文を残す．繰り返し文では，内部で代入される変数を未定義にするが，最初の代入までの変数参照の評価には，繰り返し開始時の変数の値が必要となる．プログラムの実行状態は変数の値で定義されるので，どのような理由で部分評価結果が得られたのかがわかりやすくなる．例えば，繰り返し回数が指定された繰り返し文が展開されると，回

数を数える変数への代入式が残り，繰り返し条件の変化がわかりやすい．

図 4 では，各繰り返しの前後に変数 $i の代入が存在する．最初の代入は，繰り返しの初期化式であり，それ以外は，$i の値を 1 増やす．これにより，各繰り返しでの変数の状態が確認しやすくなる．なお，代入式を出力に残す場合，Object 型の表現方法が問題となる．

```
1  <?php
2  class Synx {
3      public $val = NULL;
4      public $str = NULL;
5
6      # ... omitted ...
7
8      static public function varref($v, &$vp) {
9          $e = new Synx();
10         if (!isset($vp)) {
11             $e->str = $v;
12         } else {
13             $e->val = $vp;
14         }
15         return $e;
16     }
17
18     # ... omitted ...
19
20     static public function assign($_c, $lhs_n, &$lhs, $rhs, $tmp) {
21         $v = new Synx();
22         $v->str = Synx::rep($tmp, [ $lhs_n, $rhs->str() ]);
23         if (isset($_c[0]) && !$_c[0]->isVal()) {
24             $lhs = NULL;  # ignores old values.
25         }
26         if ($rhs->isVal()) {
27             $lhs = $rhs->val;
28         }
29         return $v;
30     }
31
32     # ... omitted ...
33
34     static public function op2($op, $e1, $e2, $tmp) {
35         $e = new Synx();
36         if ($e1->isVal() && $e2->isVal()) {
37             eval("\$f = function(\$a, \$b){ return \$a $op \$b; };");
38             $e->val = $f($e1->val, $e2->val);
39         } else {
40             $e->str = Synx::rep($tmp, [ $e1->str(), $e2->str() ]);
41         }
42         return $e;
43     }
44
45     # ... omitted ...
46
47 }
```

図 6　コード生成器 Synx.php の一部

2.8　出力コマンド

echo や print などの出力コマンドは，その評価結果がページを構成する記述になる．ページ記述にもともと含まれる HTML 記述と区別するために，計算可能でも，出力コマンド自体は評価実行せず，そのまま残す．

2.9　関数

関数定義は部分評価の対象としない．これは，各呼び出しの実引数に合わせて特殊化することで，特殊化された関数が大量に生成され，逆に理解しくくなることや，ページ記述自体には，関数定義が記述されることは少ないと想定されるからである．この結果として，再帰呼び出しで実行トレースが停止しない問題を回避できる．

関数呼び出しについては，次のように評価する．

- 関数名が計算可能で，その名前の関数定義が存在し，かつ，実引数がすべて計算可能であれば，関数を呼び出す．評価結果は，その関数の返却値になる．
- それ以外は，計算可能な部分のみが評価値に置き換わった呼び出し式とする．

ここでの「関数定義が存在する」とは，ライブラリ関数も含め，呼び出しが可能な状態にあるという意味であり，部分実行の対象ファイル内における関数定義の存在の有無は関係しない．図 3 の 39 行目の Synx::call() が関数呼び出しであり，第 2 引数は関数を呼び出すための無名関数である．関数名に変数を用いる可変関数も，関数名が計算可能であれば，同様の条件で呼び出せる．

図 4 では，関数 tr_rec の定義がそのまま部分評価後に残り，また，その関数呼び出しは，実引数が値として与えられることから，評価結果に置き換わっている．図 3 の 39 行目に関数 tr_rec の呼び出し処理が記述されており，また，65〜67 行目に，関数定義そのものも存在し，部分評価時に呼び出される．一方，関数 get_value は，未定義であるので，図 4 では，関数呼び出しのまま残っている．

3　実装

構文解析には，C 言語用の TEBA [3] を応用し，字句系列の書換え規則を用いたボトムアップ型の解析器として実装した．未実装の構文要素があるが，本論文の部分評価で取り扱う構文要素は解析できる．構文解析の結果は，構文木を表現する字

句系列で表され，それを PHP コード生成器に変換する．変換の前には，各制御文の中で代入される変数を静的に解析し，制御文にその情報を付加する．構文解析器と PHP コード生成器の生成器は，Perl で実装した．

PHP コード生成器が用いる構文要素ごとの評価実行処理をクラス Synx として実装した．例として，変数参照，代入，2 項演算に関する処理を図 6 に示す．クラス Synx は構文要素とその評価値を表すオブジェクトである．$val と $str の 2 種類の属性を持ち，計算可能だった場合には $val に値を，そうでなければ $str に PHP コードのテキスト断片を持つ．代入の処理 assign は，変数の参照を受け取り，その変数の代入を行う．2 項演算子の評価関数は op2 として定義し，eval を利用して簡素化している．なお，論理和や論理積は op2 では評価できない．これらの演算子は評価順序を考慮する必要があるからである．

4 考察

小規模なアプリケーションでは，ページの構成やスタイルなどを統一するために，1 つのファイルに複数のページの内容が盛り込むスタイルが用いられる．そのような場合の開発に，本論文の手法は効果的である．また，開発時の典型的な課題は，提示するページが HTML としての妥当性の検証とそれに基づく修正の支援である．これに対し，自動修正手法が提案されている [4] [5]．これらでは，テストケースに基づいて PHP を実行し，出力結果とページ記述との対応関係を取る方法が用いられている．テストケースが適用可能な状態まで実装した後を想定しており，実装途中での適用は想定していない．部分評価は，実装の途中でも適用できるので，実装途中の早い段階で問題を発見できる．また，静的な解析によって修正できる場合があり [4]，部分評価と組み合わせることで，より精度を高められる．なお，本論文の手法は，HTML だけでなく，PHP 自体の誤りの検証にも利用できる．

実用的なアプリケーションでは，フレームワークが利用される．フレームワーク自体への部分実行の適用は技術的には可能であるが，理解支援としては不要であり，むしろ避けるべきである．フレームワークの取り扱いは，提案手法を現実的なものとするうえで，大きな課題である．その他，実用化に向けては，対象となるデータ構造の拡大や，参照渡し，変数に対する副作用への対応なども課題である．

5 おわりに

特定の状態に合わせたページ記述の理解支援のために，PHP の部分実行手法を提案した．必要最小限の部分のみ実装しており，未対応の構文要素への対応や，フレームワークを用いた実用的なアプリケーションへの適用は今後の課題である．

謝辞 本研究の一部は，JSPS 科研費 26350344，2016 年度南山大学パッヘ奨励金 I-A-2 の助成を受けて実施した．

参考文献

[1] 竹辺靖昭. PHP-Mix: 部分評価による動的 web ページの高速化. コンピュータ ソフトウェア, Vol. 24, No. 2, pp. 2_88–2_91, 2007.

[2] Erik Ruf and Daniel Weise. Opportunities for online partial evaluation. Vol. Technical Report, No. CSL-TR-92-516, 1992.

[3] Atsushi Yoshida. Teba. `http://tebasaki.jp/src/`.

[4] Hesam Samimi, Max Schäfer, Shay Artzi, Todd Millstein, Frank Tip, and Laurie Hendren. Automated repair of html generation errors in php applications using string constraint solving. In *Proceedings of the 34th International Conference on Software Engineering*, ICSE '12, pp. 277–287, Piscataway, NJ, USA, 2012. IEEE Press.

[5] H. V. Nguyen, H. A. Nguyen, T. T. Nguyen, and T. N. Nguyen. Auto-locating and fix-propagating for html validation errors to php server-side code. In *Automated Software Engineering (ASE), 2011 26th IEEE/ACM International Conference on*, pp. 13–22, Nov 2011.