平成21年度 修士論文

ページ遷移を考慮した Webアプリケーション記述言語の 設計と実装

電気通信大学 大学院情報システム学研究科

情報システム基盤学専攻

0853021　三島　航

指導教官　小宮 常康　准教授
多田 好克　教授
古賀 久志　准教授

提出日　平成22年1月28日

目 次

図 目 次

表 目 次

第 1 章

はじめに

近年，Web アプリケーションの存在はますます重要なものになってきている．Web アプリケーションとは，IE や Firefox といった Web ブラウザを通して，サービスや動的コンテンツを提供するアプリケーションである．Web アプリケーションの身近な例として，掲示板やブログ，銀行のオンラインバンク，通信販売サイト等があり，これらは多くの人が普段の生活で利用している．

Web アプリケーションの実行形態の種類としてスクリプト，アプレット，サーブレット，CGI 等がある．その中で，CGI による Web アプリケーションは一般的に広く普及している．CGI とは，Common Gateway Interface の略であり，Web サーバから送られてきたデータをプログラムで処理させ，処理結果を Web サーバに送り返す仕組みのことをいう．CGI による Web アプリケーションは，Web サーバに処理結果を送り返す際，言い換えるとユーザに Web ページを出力する際，処理を終了する．この仕組みにより，CGI による Web アプリケーションはページ遷移を行うことで，開発者の予期しない結果やエラーを表示する恐れがあり，多くの論文 [1–7] で問題視されている．

Web アプリケーションでは，Web ブラウザのバックボタン等を用いることで，ユーザが自由にページを遷移させることができる．しかし，全てのページ遷移で Web アプリケーションは正常に動作するとは限らない．開発者が想定していないページ遷移により，予期しない結果やエラーが表示される恐れがある．開発者の想定していないページ遷移をなくすためには，アプリケーション実行中に起こるペー

ジ遷移を開発者が全て把握する必要がある．しかし，従来の言語でこれを行うのは煩雑である．

そこで本研究では，Scheme を拡張することでソースプログラムの記述から開発者がページ遷移を容易に読み取ることができ，完成後のソースプログラムからアプリケーション実行中に起こりうる全てのページ遷移を静的に解析できるプログラミング言語の設計と実装を行う．

第 2 章

Web アプリケーション開発における問題点

Web アプリケーション開発において，他人の作成した Web アプリケーションを改良する場合，開発者は Web アプリケーションの実行中に起こるページ遷移の設計を念頭に置かない状態で開発を行いがちである．また，インタラクションの多い Web アプリケーションを作成する場合，開発当初に思い描いていたページ遷移がぼやけて，ページ遷移の設計において曖昧な部分がある状態で開発を行いがちである．このようにして作成される Web アプリケーションは開発者の想定していないページ遷移を含んでいるため，開発者の予期しない出力やエラーを表示する恐れがある．なぜなら，全てのページ遷移において Web アプリケーションは開発者の予期した結果を返すとは限らないからである．ページ遷移を行うことで問題が起こる例として以下のようなものがある．

ページ遷移を行うことで問題が起こる例を，２つの数字をユーザから入力してもらい，その合計を出力する Web アプリケーションを用いて説明する．このアプリケーションを**図 2.1** のように動作させることで，問題が起こる．

① ページ p1 を複製し，ページ p1’ を作成する．

② ページ p1 で１を入力する．

③ ページ p2 からページ p1’ にページを切り替える．

④ ページ p1’ にて 5 を入力する．

⑤ ページ p2’ からページ p2 にページを切り替える．

⑥ ページ p2 にて 2 を入力する．

この結果，ページ p1，ページ p2 で入力された値の合計は 3 のはずなのに，合計を出力するページには 7 と出力される．これはページを遷移させることで，ページ p1 で入力した値である 3 が，ページ p1’ で入力した値である 5 に上書きされることで起こる．このように Web アプリケーションは，全てのページ遷移で開発者の意図通り動作するとは限らないので，開発者の想定していないページ遷移は予期しない結果やエラーとなる恐れがある．

開発者は，想定していないページ遷移による予期しない結果やエラーをなくすために，作成中の Web アプリケーションで起こりうる全てのページ遷移を把握した上で，開発を行う必要がある．Web アプリケーション実行中に起こる全てのページ遷移は 2 種類に分けることができ，本論文では，**制御フローに基づくページ遷移**，**ブラウザによるページ遷移**と呼び区別して扱う．

図 2.2 は，ユーザ登録アプリケーションにおける制御フローに基づくページ遷移とブラウザによるページ遷移を表したものである．このアプリケーションは左上からスタートし，氏名，年齢，住所を順番に入力し，確認ページが表示される．確認ページで，訂正を押すことで最初から入力をし直し，確認を押すことで登録

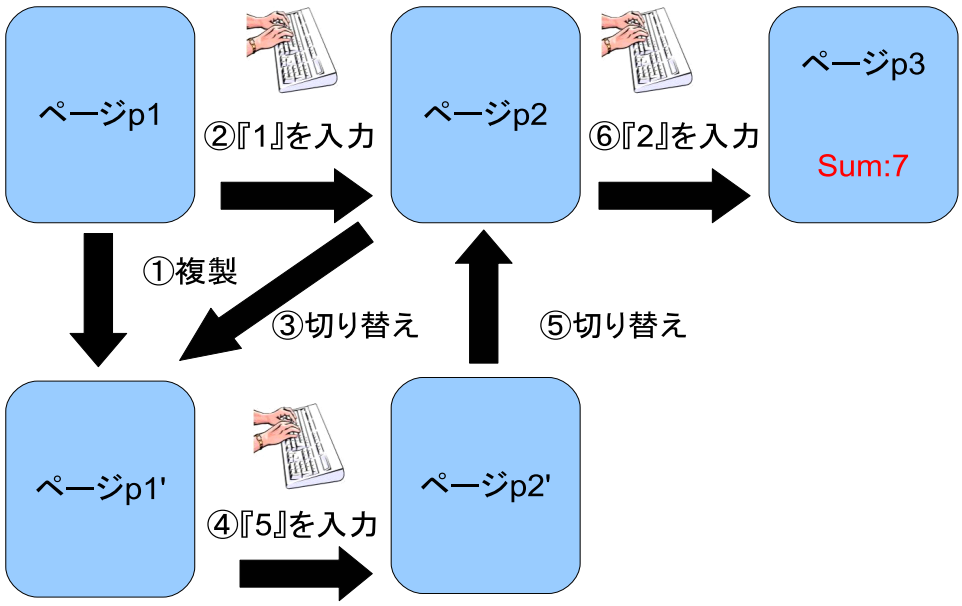

図 2.1: ページ遷移により予期しない結果を表示する例

完了のページを出力し処理を終了する．このようにアプリケーションの処理が進むにつれて，起こる遷移を**制御フローに基づくページ遷移**といい，プログラムの記述により決定される．これは，**図 2.2** では実線で表されている．また，バックボタン，ブックマークを利用することで，年齢，住所の入力に戻り入力し直すこともできる．このように，ブラウザの持つバックボタン，更新ボタン，ブックマーク等を用いることで起こる遷移を**ブラウザによるページ遷移**といい，ユーザが自由に行うことが可能である．**図 2.2** では点線で表されている．ページ遷移を全て把握した上で開発を行うためには，この 2 種類のページ遷移を両方とも把握する必要がある．

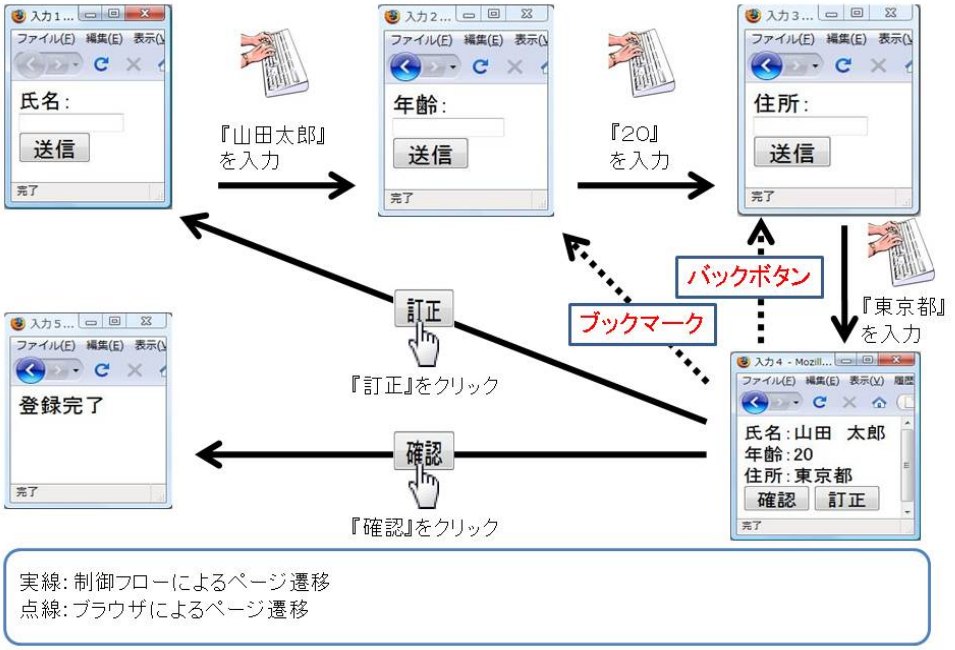

図 2.2: ユーザ登録アプリケーションにおける制御フローに基づくページ遷移，ブラウザによるページ遷移

2.1 制御フローに基づくページ遷移の把握

制御フローに基づくページ遷移を把握する方法について考察する．制御フローに基づくページ遷移はプログラムの記述により決定するため，ソースプログラムから読み取ることが可能である．しかし，従来の言語において，ソースプログラムから制御フローに基づくページ遷移を読み取ることは煩雑である．以下で煩雑であることを示す．

Web アプリケーションには，出力するページごとにプログラムを分割しているものとページごとに分けず，複数のページを 1 つのプログラムで記述したものがある．両方の場合において，制御フローに基づくページ遷移をソースプログラムからどのように読み取るかを考察する．

まず，ページごとにプログラムを分割するケースを考えてみる．このようなケー

スでは，下記に記述したHTMLのformタグの属性であるactionやancorタグの属性であるhrefから次にどのページへ遷移するかを読み取ることができる．

```
<a href="http://localhost/plus1.html">リンク先にジャンプ</a>
<form method="POST" action="http://localhost/plus1.html">
```

しかし，どのページから遷移してきたかという情報はないため，１つのソースファイルからは，どのページへ遷移するかしか読み取ることはできない．アプリケーションにおけるページの全遷移を読み取るためには，全てのファイルを調べる必要があるため，解析しない限りページ遷移の読み取りは煩雑である．

次に，プログラムをページごとに分割せず，１つのソースプログラムで複数のページを出力するケースをcookieを用いた例をもとに考えてみる．下記はPHPで記述したWebアプリケーションにおいて，ページ遷移を判断する部分を抜き出したものである．

```
if(!isset($ _COOKIE["STEP"])) p1();
else if($ _COOKIE["STEP"]=="FIRST") p2();
else if($_COOKIE["STEP"]=="SECOND") p3();
                                .
                                .
function p1(){...setcookie("STEP","FIRST");}
function p2(){...setcookie("STEP","SECOND");}
function p3(){...}
```

このようなケースでは，アプリケーションにおけるページ遷移はプログラムの制御フローと各ページで書き出されるcookieから読み取ることができる．ページごとにソースファイルを分割した場合と比べ，全てのファイルを見る必要がないので，その点では読みやすいといえる．しかし，制御フローと各ページからcookieに書き出される値を調べ，この両方を照らし合わせない限り，ページ遷移を読み取るこ

とができない．それゆえ，ソースプログラムからページ遷移を読み取るのは煩雑である．

このように読み取りが煩雑なのは，CGI の仕組みが関係している．CGI による Web アプリケーションでは，ユーザとインタラクションを行う際，次に行う処理を cookie 等で指定し，プログラムを終了する．そして，ユーザから送信されてきたページの cookie の記述に従い，次の処理を行う．これはラベルにジャンプする GOTO 文と同じ構造である．そのため，Web アプリケーションの制御フローは断続的なものとなり，ページ遷移を読み取るのが煩雑になる．

Web アプリケーション開発の手法として，継続を用いた Web アプリケーション開発 [1–3] がある．継続とはある時点での残りの処理である．継続を用いた Web アプリケーションでは，継続サーバ [1] という特殊なサーバを用いて，ユーザとインタラクションを行う際，継続サーバが残りの処理である継続を捕まえ保存し，プログラムを終了する．そして，ユーザから送信されてきたページに応じて，継続サーバが適切な継続を呼び出すことで，次の処理を行う．このため，GOTO 文のようなプログラムでなく，通常のプログラムのように Web アプリケーションを記述できるので，Web アプリケーションの制御フローは連続したものとなり，処理の実行順序は読み取りやすい．しかし，従来の CGI のように，ページを出力する部分での処理に切れ目がないため，どこまでの処理がページを作成する処理かの区別がつきにくく，ソースプログラムからページ遷移を読み取るのは煩雑である．

そこで本研究では，制御フローに基づくページ遷移をソースプログラムの制御フローから開発者が容易に読み取ることができるプログラミング言語の設計と実装を行う．開発スタイルを**図 2.3** に示す．まず，①ある程度制御フローを作成した段階で制御フローを読み取る．次に，②制御フローからページ遷移を把握する．そして，③プログラムを修正する．このように，プログラム全体が未完成でも，どのようにページが遷移していくかを開発者は読み取ることができるので，アプリケーション実行中に起こる想定していないページ遷移を減らすことができる．

2.2 ブラウザによるページ遷移の把握

ブラウザによるページ遷移を把握する方法について考察する．ブラウザによるページ遷移は，ユーザが自由に行えるページ遷移であるため，ソースプログラムから直接調べることができない．原理的には，ブラウザによるページ遷移は全ページ遷移と制御フローに基づくページ遷移の差分によって求めることができる．しかし，開発者がソースプログラムから制御フローに基づくページ遷移を読みとり，全遷移から制御フローに基づくページ遷移を除くことで，ブラウザによるページ遷移を把握するというのは煩雑であり，遷移を見落とす可能性がある．また，インタラクションの多い Web アプリケーションにおいて，アプリケーション実行中に使用される Web ページの数は多くなる．そのため，作成中のソースプログラムから制御フローに基づくページ遷移を読み取るのが煩雑になる可能性がある．

そこで本研究で作成する言語は，完成後のソースプログラムからアプリケーション実行中に起こりうる制御フローに基づくページ遷移とブラウザによるページ遷移を静的に解析できるようにし，解析するためのツールを実装する．解析ツールを用いた開発スタイルを**図 2.4** に示す．まず，　完成したソースプログラムから，ページ遷移解析ツールによりページ遷移解析結果が表示される．次に，　ページ遷移解析結果から，開発者はページ遷移を読み取る．そして，　開発者の想定していないページ遷移が予期しない結果やエラーを表示される場合，プログラムを修正する．これを繰り返すことで，想定していないページ遷移が発生しない Web アプリケーションを作成することができる．

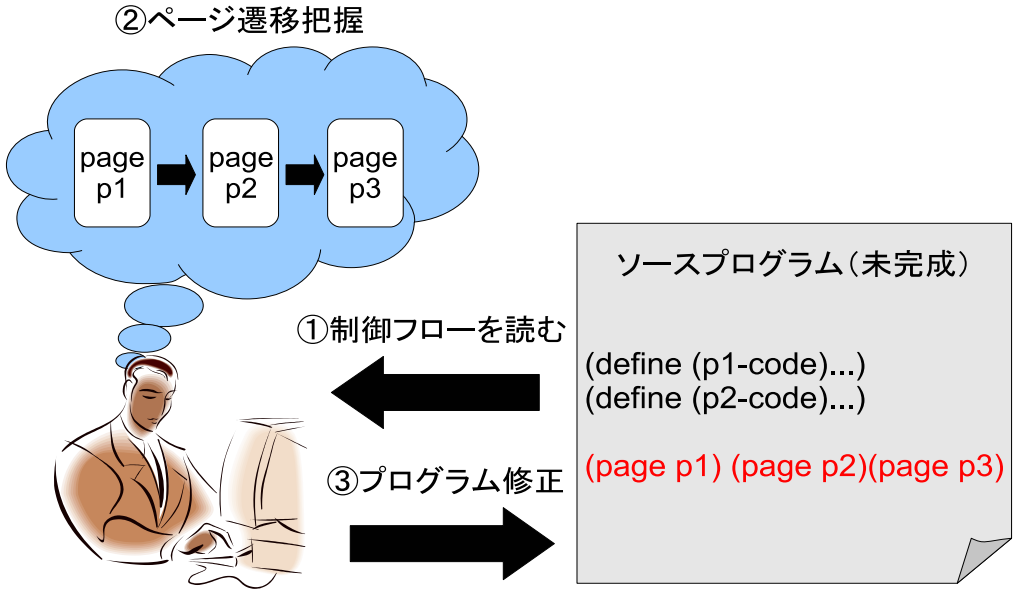

図 2.3: 本研究の言語による Web アプリケーション開発スタイル 1

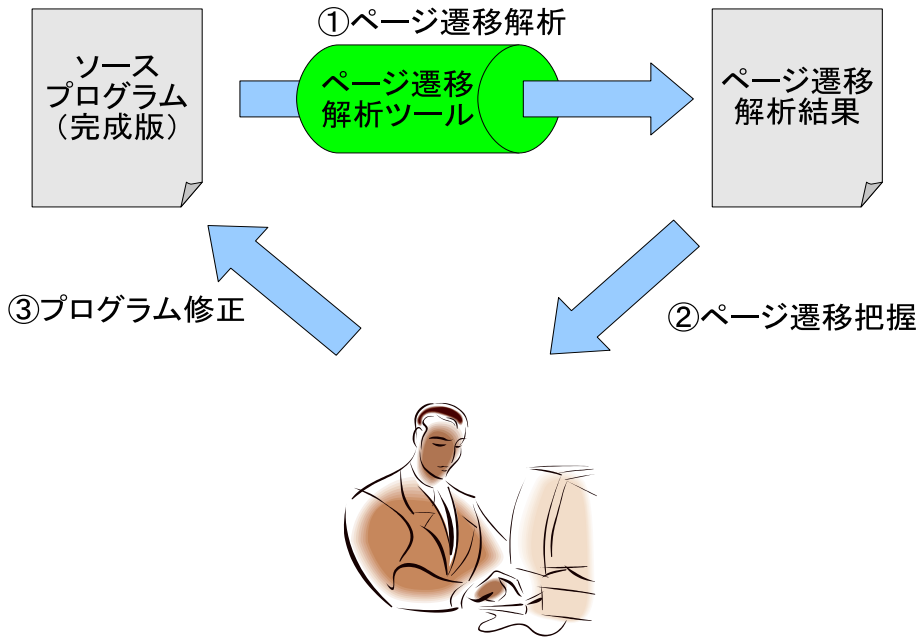

図 2.4: 本研究の言語による Web アプリケーション開発スタイル 2

第 3 章

要求事項

ページ遷移を把握した上で Web アプリケーションを開発するために，本研究の言語には以下の 2 つの特徴が必要であると考えた．

- 作成中のソースプログラムから開発者がページ遷移を読み取り易い
- 完成したソースプログラムからページ遷移を静的に解析可能

この章では，本研究の言語において，この 2 つの特徴をどのように実現するかを考察する．そして，開発者の想定していないページ遷移を発見後，プログラムの修正を容易に行うために必要な機能を考察する．

3.1 ページ遷移の読み取り易さ

従来の CGI だと制御フローが断続的なため，ソースプログラムにおいて，どこまでの記述がページを作成する部分かの区別はつきやすいが，処理の実行順序が分かりにくい．継続サーバを用いた Web アプリケーションでは，処理の実行順序は分かりやすいが，どこまでの記述がページを作成する部分かの区別がつきにくい．そこで本研究の言語では，基盤に継続を用い，ページ遷移を読み取りやすくするために，ソースプログラムにおいて，個々のページを作成する部分をページとする．そして，そのページを定義するために，ページ定義構文を言語に導入する．また，ページの実行順序に関する制御フローを記述するために，ページ遷移構文を言語

に導入する．こうすることで，作成中のソースプログラムからページ遷移を読み取った上で，開発が行えるため，アプリケーション作成の段階で，開発者の想定していないページ遷移を減らすことができる．

3.2 ページ遷移の解析

次に，ソースプログラムからどのようにページ遷移を静的に解析するかについて考察する．ブラウザによるページ遷移はソースプログラムから直接調べることができないので，ソースプログラムから制御フローに基づくページ遷移を解析し，ブラウザによるページ遷移は，ページの全遷移との差分として導き出す．

プログラムの制御フローから解析を行う上で，以下のことが問題となる．解析を行うプログラムの制御フローにおいて高階関数を用いた処理や eval 関数の呼び出しが存在すると，制御フローが動的に変化するため，解析精度は著しく低下する．

そこで本研究の言語では，ページを記述する部分と制御フローを記述する部分で用いる言語を分け，プログラムの制御フローを本研究の言語のみで記述する．そして，制御フローからページ遷移が静的に決定できるようページ定義構文，ページ遷移構文を設計する．ここで，静的に決定するとはプログラムの実行前に起こりうる全遷移が決定することを意味している．単純な条件分岐や繰り返しで，ページ遷移が実行時に確定する場合であれば，取りうる遷移は解析可能であるため，静的に決定するといえる．こうすることで，ソースプログラムからページ遷移を静的に解析することができるため，開発者は，解析結果からアプリケーション実行中に起こる全てのページ遷移を把握することができるようになる．

3.3 プログラム修正

開発者は，想定していないページ遷移を発見後，開発者の意図通り動作するようプログラムを修正していく．開発者の想定していないページ遷移が問題となる状

況として，次の 2 種類が挙げられる．

- プログラムを実行するのに必要なデータが揃っているが，実行すると開発者の予期しない結果になる．
- プログラムを実行するのに必要なデータが揃っていないため，実行するとエラーになる．

必要なデータが全部揃っているのに，実行すると開発者の予期しない結果になる場合，ユーザがブラウザによるページ遷移を行うことで，プログラムの実行順序がプログラムに記述された制御フローとは異なるものになるのが原因である．ショッピングサイトにおいて，ブラウザの持つバックボタン，更新ボタンを用いることにより，商品の 2 重注文，誤った商品の注文をしてしまう場合が，これに該当する．この問題は，ページ遷移が原因で起こるため，ページ遷移を制限することによって解決することができる．従来の CGI にはトランザクショントークンという手法があり，これを用いることで，ページ遷移を制限することができる．しかし，トランザクショントークンによるページ遷移の制限はページの中で行われるため，制御フローに表れない．そのため，解析結果に反映されず，解析結果として表示されるページ遷移と実際にアプリケーション実行中に起こるページ遷移は異なるものになってしまう．また，ソースプログラムからページ遷移を読み取る際，各ページの処理を調べ，ページ遷移が制限されているかを調べていくのは煩雑である．これを解決するために，ページ遷移の制限はソースプログラムから容易に読み取れる形かつページ遷移解析結果に容易に反映できる形で行う必要がある．そこで本研究では，ページ定義構文にどのようなページ遷移を許可するかを記述したパラメータをもたせ，このパラメータをチェックすることでページ遷移を制限する機能を言語に導入した．ソースプログラムからページ遷移を読み取る際，ページ定義構文のパラメータも読み取り，解析を行う際は，ページ定義構文のパラメータも解析する．

必要なデータが揃っていないため，実行するとエラーになる場合，原因が 2 パターンある．1 つは，ユーザがブラウザの持つブックマーク，URL 直接入力を使用し，経由すべきページを飛ばしてページを遷移させることで，必要なデータが揃っていない状態でプログラムが実行されるのが原因である．これは，前述した通りページ遷移を制限することで解決する．そしてもう 1 つは，ユーザがデータを入力していないか，開発者がページの cookie にデータを埋め込んでいないため，データが送信されてこない，あるいは送信されてきたデータの型が一致しない場合である．これを解決するために，ページ間でデータを受け渡す際，データが送られてきたか，データの型があっているかをチェックする必要がある．そこで本研究では，ページ定義構文にページを実行するのに必要なデータ型の宣言，ページが次のページに送るデータをパラメータとしてもたせ，このパラメータをチェックすることで，ページからデータが送信されてきていない場合，ページから送信されてきたデータの型が必要なデータの型と異なる場合，エラーページを表示する機能を言語に導入した．

3.4 言語機能の比較

表 3.1 は，CGI 機能を持つ言語，継続機能を持つ言語，本研究の言語の特徴を比較したものである．

従来の CGI や継続サーバを用いた Web アプリケーションでは，作成中のソースプログラムからページ遷移を読み取ることが煩雑，あるいは不可能であった．そこで本研究の言語では，これを容易に行えるようにした．

次に，従来の CGI や継続サーバを用いた Web アプリケーションにおけるページ遷移の解析は，制御フローにおいて高階関数や eval 関数が存在する場合，解析精度は著しく低下する．そこで本研究の言語では，解析部分である制御フローを本研究の言語がもつ構文のみで記述し，動的に変化する部分を含まないよう構文を

設計することで，ページ遷移の解析を容易に行えるようにした．

最後に，ページ遷移を制限する機能として従来の CGI には，トランザクショントークンという機能がある．これを用いることで，バックボタンや更新ボタンによるページ遷移を制限することができる．しかし，本研究の言語ではページ遷移を解析するので，制限することで変化するページ遷移を解析結果に反映させる必要がある．そこで，本研究の言語独自のページ遷移制限機能を取り入れた．

表 3.1: 各言語の特徴比較

	CGI	継続	本言語
ソースからのページ遷移の読み取り	△/×	△/×	○
ソースからのページ遷移の解析	△/×	△/×	×
ページ遷移の制限機能	○	×	○

第 4 章

設計

前章までより，以下の構文，機能を言語に導入する．

- ページ遷移構文
- ページ定義構文
- 解析範囲の限定化

図 4.1 は，ページ p1 で入力をもらい，その入力を条件式で評価しページ p2，ページ p3 を出力するプログラムを本研究の言語で書いたものである．**図 4.1** において，青い枠で囲った部分であるページを作成するコードとページ遷移構文の条件式を Scheme で記述する．そして，ページを作成するコードの名前の宣言をページ定義構文で記述する．ページ定義構文で名前を付けたページを評価する順序をページ遷移構文で記述する．このように，本研究の言語で記述する部分と Scheme で記述する部分を明確に分け，ページ遷移構文に単純な条件分岐と繰り返しの構文以外に動的に変化する仕組みを持たせないことで，ページ遷移を静的に決定することができる．

また，開発者の想定していないページ遷移の発見後，予期しない結果やエラーを容易に取り除くために，以下の 2 つの機能が言語に必要である．

- ページ遷移を動的にチェックするための機能
- ユーザ入力をチェックするための機能

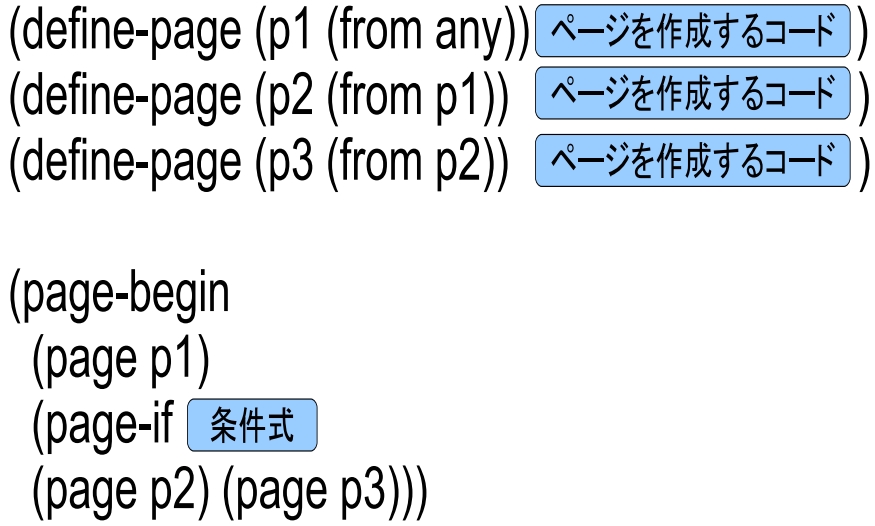

図 4.1: 解析範囲の明確化

この章では，これらの機能の設計を述べる．

4.1 ページ定義構文

ページ定義構文はページを作成するためのコードに名前を付けるために使用する．ページ定義構文を用いることで，プログラムの制御フローはページ遷移構文で記述することができるため，ソースプログラムからページ遷移の読み取りが容易になる．また，ページは静的な解析で使用されるため，再定義できない仕様となっている．

本研究の言語ではどういうページ遷移してきたか，そしてページ間で受け渡されるデータから定義したページを実行するかどうかをチェックする機能を言語に取り入れるため，チェックに用いるパラメータをページ定義構文にもたせる．下記がページ定義構文であり，各種パラメータの説明を**表 4.1** に示す．

`(define-page (` *page-tag* `[(from...)(parameter...)(input...)])` *page-code*`)`

表 4.1: ページの持つ各種パラメータ

パラメータ	用途
page-tag	ページの識別を宣言
(parameter ((*page-tag* (*type name*)...)...))	ページで使用する型と変数，これらを送るページの *page-tag* を宣言
(input *name*...)	ユーザ入力を宣言
(from *page-tag*...)	ページ遷移を宣言

page-tag は，ページの識別子である．ページは再定義を禁止しており，*page-tag* により一意に決定する．parameter は，ページ内で使用する変数名と型，そして変数を渡してくれるページの *page-tag* を宣言する．型には，int，float，string が指定でき，それぞれ，整数型，浮動小数点型，文字列を表している．parameter で *page-tag* を宣言していない場合，どのページから受け取っても良い．input は，このページが次のページに渡すデータを宣言する．from は，ページ遷移のチェックに使用する．from には，遷移を許可するページの *page-tag*，あるいは自分以外の他のページ全てからの遷移を許可する any が入る．全てのページからの遷移を許可する場合，何も記述しない．ページを出力するコードは，ページで表示する内容を作成し，HTML ページにして出力するコードである．これは，Scheme で記述する．

4.2 ページ遷移構文

ページ遷移構文は，プログラムの制御フローを記述するのに使用する．ページ遷移構文のみを用いて制御フローを記述することで，ソースプログラムからページ遷移を静的に解析できるようにページ遷移構文を設計している．**図 4.2** はペー

表 4.2: ページ遷移構文

構文名	用途
page-begin	複数のページ遷移構文を左から実行
page-if	単一分岐を作成
page-while	ループを作成
page-cond	多重分岐を作成
page	ページ定義構文で定義したページ

```
<ページ遷移式>::=(page-begin <ページ遷移式>+)
        |(page-if <条件式> <ページ遷移式> <ページ遷移式>)
        |(page-while <条件式> <ページ遷移式>)
        |(page-cond (<条件式> <ページ遷移式>)*(else <ページ遷移式>))
        |(page <page-tag>)
<条件式>::=Scheme の式
```

図 4.2: ページ遷移構文

ジ遷移構文の定義を示す.

表 4.2 に本研究の言語に導入するページ遷移構文を示す．page-begin は，複数のページ遷移式を要素として取ることができ，要素を左から実行する．page-if は，第 2 要素である条件式を評価し，真ならば第 3 要素のページ遷移式を評価，偽ならば第 4 要素のページ遷移式を評価する．page-while は，第 2 要素である条件式を評価し，真である限り，第 3 要素のページ遷移式を評価し続ける．page-cond は，複数のリストを要素として取り，各リストの第 1 要素である条件式を評価し，真となるリストの要素であるページ遷移式を評価する．どの条件式にも一致しない場合，第 1 要素が else 節であるリストの要素であるページ遷移式を評価する．page は，ページ定義構文で定義済みである *page-tag* を要素として取り，*page-tag* の指す

ページを評価する．page-if，page-while，page-cond の条件式は Scheme の式である．

4.3 解析範囲の限定化

本研究ではページ遷移を把握するために，プログラムの制御フローを解析する．しかし，制御フローにおいて本研究の言語と Scheme による記述が混在する場合，本研究の言語と Scheme で記述した部分を全て解析する必要があり，解析は困難なものになる．

そこで本研究の言語では，プログラムの制御フローからページ遷移の解析を容易に行うために，Scheme で記述する部分と本研究の言語で記述する部分を明確に区別する．解析に用いるプログラムの制御フロー以外の部分を (SCM ...) で囲み，Scheme で記述し，(SCM ...) で囲われていない部分を本研究の言語で記述し，各処理系で評価する．

こうすることで，(SCM...) で囲まれた部分でページ遷移構文の記述はできない仕様となる．それゆえ，ソースプログラムの解析範囲を本研究の言語を用いている部分に限定することができ，ページ遷移の解析が容易になる．また，ソースプログラムは**図 4.3** のように，ページ遷移に関係する部分とそうでない部分に明確に分けることができるため，作成中のソースプログラムのどの部分を見ることでページ遷移を読み取ることができるかが明確になる．

4.4 ページ間におけるデータの受け渡し方法

ページ間で自由に値の受け渡しを可能にするためにデータの受け渡し方法を設計した．ページでユーザに入力された全ての値は，どのページで入力された値であるかを page-tag で区別された状態で大域環境に保存する．**図 4.4** のように，HTML のフォームの name で大域環境で保存する．ページの評価で使用する際，このこ

図 4.3: 解析範囲の明確化

とを意識する必要がある．この大域環境から，ページを評価するのに必要な値を `form-let` 構文で束縛し，ページを評価する際，変数のように扱えるようにする．

4.4.1 form-let 構文

ページでユーザに入力された値を，ページ定義式の *page-code* において，変数として扱える様にするための構文である．ページ評価に使用する *name* をページ定義式の `parameter` で宣言する．

値を環境から取りだされる際，値の型が `parameter` の宣言通りかチェックされ，宣言通りなら大域環境から値が取りだされ，型が異なる場合，エラーページが表示される．

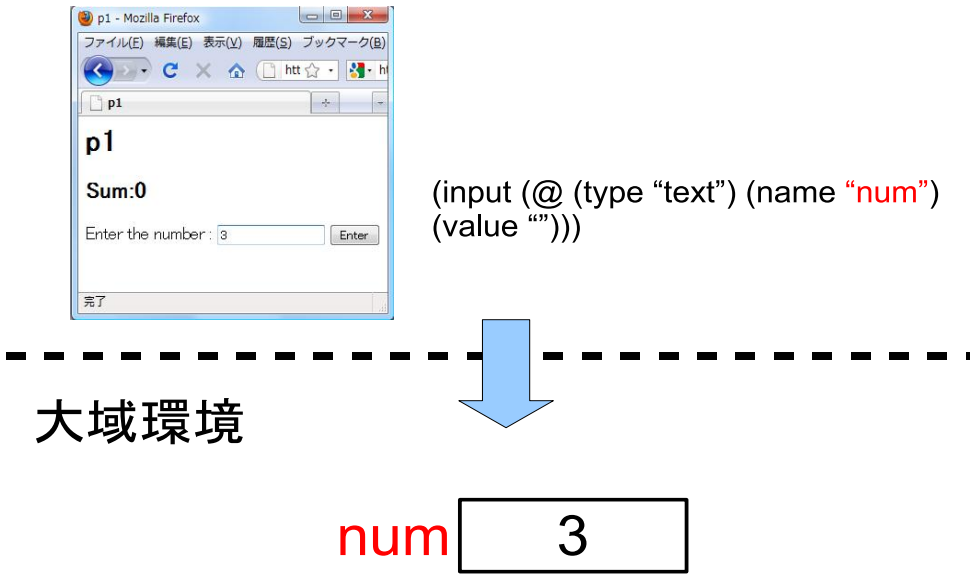

図 4.4: ページ定義構文

4.4.2 ユーザ入力チェック

期待しない値が送信されることによる予期しないエラーをなくすために，ユーザ入力を必要とするページで，ユーザから入力を受け取っているか，また，受け取った入力が処理に必要なデータ型であるかをチェックする機能を言語に導入した．ユーザ入力が送信されてくる際，送信されてきたページのページ定義式が持つ (`input` *name*) をチェックし，*name* がページから送信されてきた場合，大域環境に保存する．送信されてこなかった場合，エラーページを表示する．

また，ページを作成するために大域環境から値を取りだす場合，ページ定義式が持つ (`parameter` (*type variable*)) をチェックし，型が一致する場合，取りだす．異なる場合，エラーページを表示する．

4.5 ページ遷移チェック機能

バックボタンや更新ボタンを用いたページ遷移における予期しない結果やエラーをなくすために，特定のページ遷移以外を実行時にエラーと見なす機能を言語に導入した．ユーザにページを出力する際，出力するページの *page-tag* をサーバに保存する．そして，ユーザからリクエストされるページのページ定義構文が持つパラメータである (`from` *page-tag*...) をチェックし，許可されていないページからの遷移の場合，エラーページを表示する．

第 5 章

実装

5.1 実装環境

本研究の言語は Scheme [8] というプログラミング言語を用いて実装した．プログラムの行数は 1200 行程度である．Scheme を用いたのは，アルゴリズムを簡潔に記述でき，マクロで構文を書けるなど，言語開発に適しているからである．また，Scheme には数多くの処理系があるが，今回はドキュメントも多く，Web まわりの機能の豊富な Gauche [9] を使用した．

5.2 評価機

`Scheme` と本研究の言語を評価するために，以下のように動作する評価機を実装した．評価機に (`SCM...`) が読み込まれた場合，Scheme が持つ関数 `eval` に読み込んだ式を渡し，実行する．

ページ定義構文が読み込まれた場合，ページ定義構文の *page-tag*，`from`，`input`，`parameter`，*page-code* を 1 つのリストに格納したものが保存される．保存する際，同じ *page-tag* を持つ前述のリストがすでに保存されていないかチェックする．保存されていない場合，リストが保存され定義完了となる．保存されていた場合，エラーとなる．

ページ遷移構文が読み込まれた場合，読み込まれた式は，page-begin 式の要素として入れられる．(page *page-tag*) が読み込まれた場合，*page-tag* を用いて，ページ定義構文で定義済みであるかを確認する．

以上の処理が済むと，Web アプリケーションは実行可能になるため，サーバをリクエスト待機の状態で立ち上げる．サーバがユーザからのリクエストを受信すると，継続を実行する．ページ遷移構文を要素として取り入れた page-begin 式を評価機で実行する．page-if，page-cond，page-while が実行される場合，これらの式が持つ条件式の Scheme の eval で評価し，次に評価する要素を決定する．(page *page-tag*) が実行される場合，残りの処理を継続として捕まえ，サーバに保存する．その後，実行されるページの *page-tag* を用いて，ページ定義構文読み込み時に保存した *page-tag*，from，input，parameter，*page-code* のリストを取りだす．ここから，以下のような順序でページを出力することになる．まず，ページ定義式の from を用いて，ページ遷移をチェックする．次に，form-let 構文がページ定義式の parameter で宣言した値を環境から取り出し，ページ定義式の page-code で変数のように扱えるようにする．そして，Scheme の eval で page-code を実行し HTML ページを出力する．

5.3 ページ定義構文

下記はページ定義構文の使用例である．

(define-page (*page-tag* (from any) (parameter (int num)) (input x)) *page-code*)

from，parameter，input は省略可能である．ソースファイル読み込み時，上記リストが読み込まれ，下記のリストが保存される．

((p1 (from any) (parameter (int num)) (input x)) (SCM...))

上記リストが保存される際，同じ *page-tag* でリストが既に保存されていた場合，エラーとなる．

ページ実行時，ページ定義式の持つ (`from` *variable*) により，ページ遷移がチェックされる．そして，`form-let` 関数がページ定義式の `parameter` で宣言した値を環境から取り出し，ページで変数のように扱えるようにする．これらの処理を経て，`page-code` が実行され HTML ページを出力する．

5.4 ページ遷移構文

下記がページ遷移構文を用いて記述したプログラムである．

```
(page p1)
(page-if 条件式 (page p2) (page p3))
(page-while 条件式 (page p4))
(page-cond  ((条件式 (page p5) (page p6)))
            (else (page p7)))
```

`define-page` で定義したページは (`page` *page-tag*) で記述される．ソースファイル読み込み時，上記の式は `page-begin` 式の要素となり下記の形で評価機に渡される．

```
(page-begin
(page p1)
(page-if 条件式 (page p2) (page p3))
(page-while 条件式 (page p4))
(page-cond  ((条件式 (page p5) (page p6)))
(else (page p7))))
```

このプログラムは以下の通りにページを出力する．まず，ページ `p1` を出力する．次に，`page-if` の条件式を評価し，真ならばページ `p2`，偽ならばページ `p3` が出力

される．そして，page-while の条件式が真である限り，ページ p4 を出力し続け，最後に，page-cond の条件式が真となればページ p5，ページ p6 を出力し，偽であればページ p7 を出力する．このように，制御フローからページ遷移を容易に読み取ることができる．

ページ遷移構文で記述したプログラムを評価機に渡す際，**表 4.1** で示したページ遷移構文のみで記述されているか，そして，(page *page-tag*) において，*page-tag* の指すページはページ定義構文で定義済みかをチェックする．

5.5 ページ間のデータの受け渡し

ページ間で自由に値の受け渡しを可能にするために以下のようなデータの受け渡し方法を実装した．まず，ページが出力される際，ページ定義式の持つ (input *name*...) の *name* は *page-tag* とともにグローバル変数*input*に保存される．*input*は下記のようなリストとなる．

input=((*page-tag*1 *name*1...) (*page-tag*2 *name*2...)...)

次に，ユーザ入力がサーバに送信される際，get メソッド，あるいは post メソッドでデータを受け渡す．get メソッド，post メソッドで受け取ったデータから Gauche の関数である cgi-parse-parameter 関数を用いて，変数名と値によるリストを作成し，グローバル変数*bids*に束縛する．*bids*は下記のように変数と値のリスト構造になっている．

bids=((*name*1 *value*1) (*name*2 *value*2)..(*key page-tag*))

*input*には各ページでユーザに入力してほしい値が入っており，*bids*には page-tag のページでユーザが入力した値が入っている．この 2 つを用いて，ページ定義式の input で宣言した値が送られてきたかをチェックし，その後，環境である*env*に保存する．input で宣言したデータが送信されてこない場合，エラーページが表示される．環境は下記のような構造となっている．

((*env-id* (*page-tag* (*name value*)...)))

env-id は，環境の識別子である．*page-tag* は，データを受け取ったページのページタグである．

環境が作成されるまでの一連のメカニズムを説明する．ページが出力される際に環境の *env-id* をページに埋め込む．ページがサーバに送信されてくる際，ページに埋め込まれた *env-id* と環境が持つ *env-id* を比較する．ブラウザによるページ遷移を行わない場合，*env-id* はページと環境で一致する．この場合，環境の *env-id* を新しい値に更新する．

ページが複製される，もしくはバックボタンで戻る等，ブラウザによるページ遷移を行う場合，ページと環境の *env-id* は異なるものとなる．この場合，新しい *env-id* を作成し，送信されてきたページの *page-tag* によるリストから後ろのリスト全てと *env-id* で新しい環境を作成する．環境は下記のようになる．

env=((*env-id*1...) (*env-id*2...))

env-id をチェックし，一致しない場合環境を新たに作るのは，作成するアプリケーションによっては，環境を複製する必要があるからである．環境を複製する必要がある具体例として，ユーザに２つの数字を入力してもらい，その合計を出力するアプリケーションが挙げられる．このアプリケーションを**図 5.1**，**図 5.2** で記述したように以下の順序で動作させる．

①ページ p1 を複製し，ページ p1’ を作成する．

②ページ p1 で１を入力する．

③ページ p2 からページ p1’ にページを切り替える．

④ページ p1’ にて５を入力する．

⑤ページ p2’ からページ p2 にページを切り替える．

⑥ページ p2 にて２を入力する．

このように動作させることで，環境を複製した場合としない場合で結果が変わってくる．

環境を複製しない場合，**図 5.1** のように，ページからサーバに渡された値は，全て同じ環境に格納されていく．その結果，ページ p3 では Sum: 7 が表示される．複製したページであるページ p1' での入力とページ p2 での入力の合計値が出力されているが，これはユーザの意図した結果ではない．

環境を複製する場合，**図 5.2** のように，ページを複製するたびに環境は複製されるので，ページ p3 で Sum: 3 が表示される．ページ p1 での入力とページ p2 での入力の合計値が出力されており，ユーザの意図した結果であるといえる． 本研究の環境では，**図 5.2** のように，ページを複製する，あるいはバックボタンを用いるたびに環境を複製する．そして，どのページがどの環境に属するかを判別するために，*env-id* をページに持たせる．ただ，アプリケーションによっては，環境を複製しない方がいい場合がある．そこで，開発する Web アプリケーションに応じて環境を複製するか，しないかを開発者が選択できるようにしようと考えているが，その部分に関して本研究の言語では未実装である．

使用例として図 5.3，図 5.4 はユーザから数字入力をもらい，今までの入力の合計を出力するアプリケーションにおいて，ページ p1 からページ p2 へのデータの受け渡し方法を示したものである．ページ p1 は一番最初に数字入力をもらうページであり，合計の初期値は 0 である．

まず，図 5.3 の説明を行う．①の make-input-list 関数は，ページ定義構文の (input *value*) を (*page-tag variable*) の形にして，*input*のリストに追加する関数である．各ページはページ定義構文の (input value) により，サーバに送るデータを宣言するが，全てのページの (input value) を格納したのが*input*である．図 5.3 の一番上の式であるページ p1 のページ定義構文を見ると，(input num) となっている．これは，ページ p1 から num というデータが送られてくるという意味であり，make-input-list 関数により，(p1 num) の形で*input*に追加される．この*input*は，p1 から num が送られてきたかをチェックする際に用いる．

②の parameter-check 関数は，env-id のチェックとユーザ入力のチェックを行う．

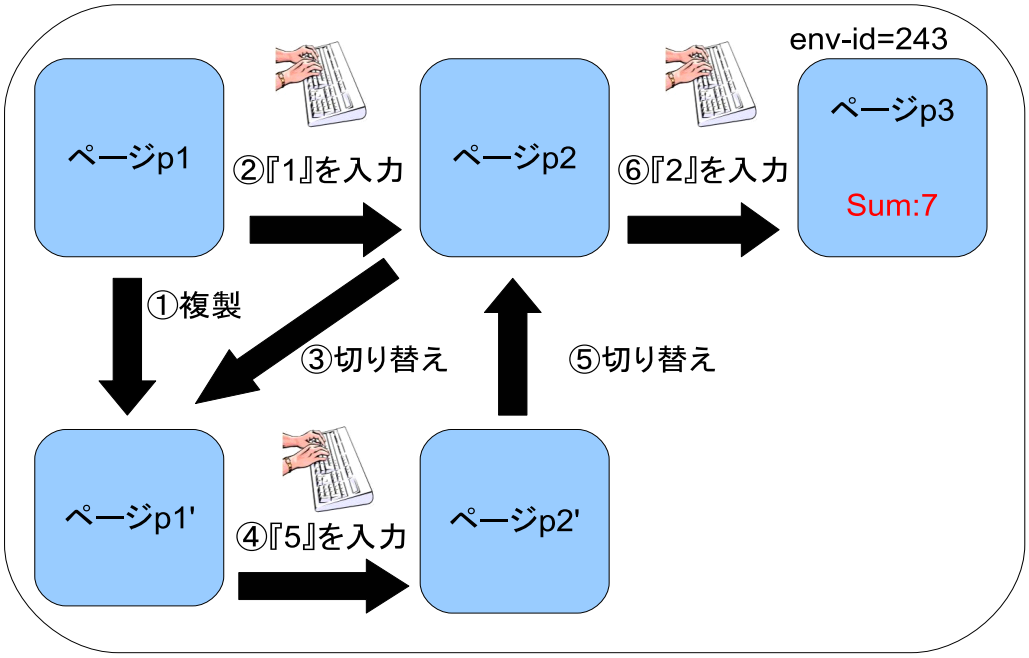

図 5.1: 環境を複製しない場合

env-id のチェックにより，env-id の値が一致するかチェックする．図 5.4 のキーボードの絵の下にある*bids*の中の env-id と*env*の中の env-id を数値を比較すると，この場合 2864 同士で一致するため，環境は複製しない．新しい*env-id*である 8820 が発行され，スコープに入れられる．ユーザ入力のチェックにより，*bids*が*input*に格納されている第 1 要素と同じページからの送信である場合，どのページからのデータ送信であるかは*bids*の key から判断する．そして，*input*の第 2 要素以降にある変数を*bids*から取りだし，*env*に保存する．ページ p1 からの送信なので，*input*内の (p1 num) という情報を元に，*bids*から (num 3) が取りだされ*env*に (p1 (num 3)) が保存される． ここで，図 5.3 での処理は終了であり，ここから図 5.4 の処理に移る．③の type-check 関数は，受け取ったデータの型がページを作成するのに必要なデータの型かチェックする．このチェックにはページ定義構文の持つパラメータである parameter を用いる．図 5.4 の一

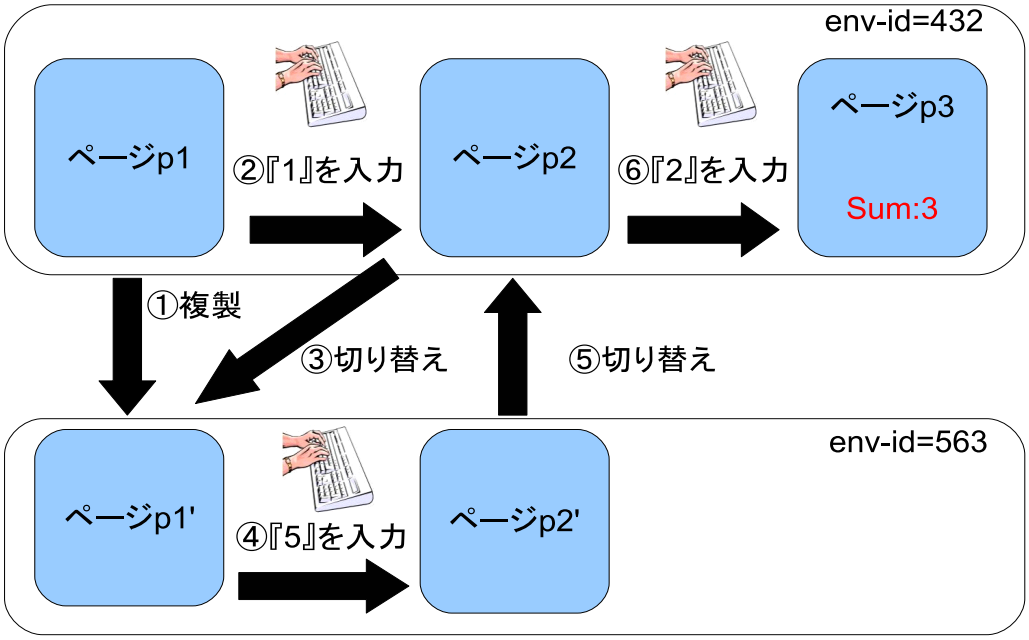

図 5.2: 環境を複製した場合

番上の式の parameter をみるとページ p2 が必要としているデータは (int num)，つまり整数である num が (page-plus "p2") を実行するのに必要であることが分かる．*page-tag* による指定がないため，左から*env*を調べていったとき一番最初に見つかる num が値として取り出される．よっと*env*から (num 3) が取り出される．

④の form-let 関数は，type-check が*env*から受け取った (*variable value*) のリストをページを作成するコードで使えるようにする．この場合は，(page-plus "p2") で num を変数のように扱うことができるようにする．

⑤の (page-plus "p2") により，(+ num sum) という式で合計を出し，ページの Sum で表示している．１つ目の値の入力であるため，sum は 0 であり，num はページ p1 で受け取った値である 3 なため，ページ p2 の Sum の欄に 3 と出力される．図 5.4 の一番下にあるページの絵から Sum:3 と表示されているのが確認できる．

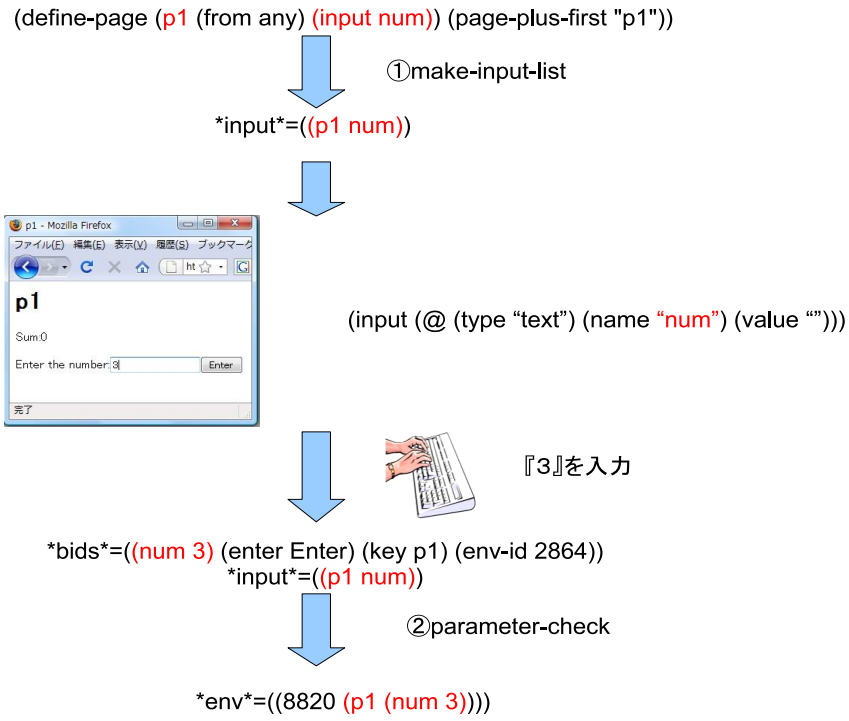

図 5.3: ページ間のデータの受け渡し方法 1

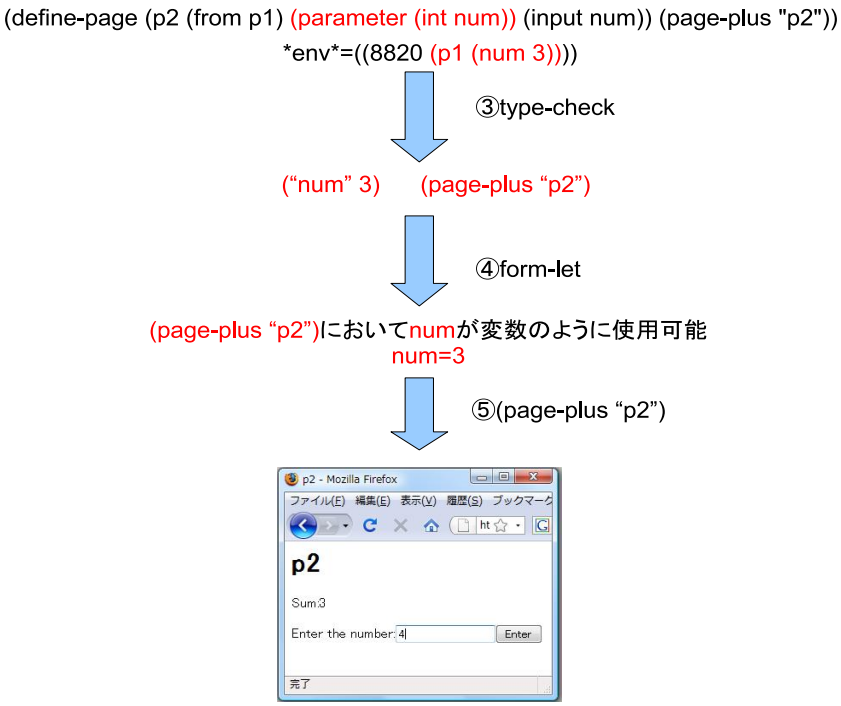

図 5.4: ページ間のデータの受け渡し方法 2

```
(define-page (p1 (from any) (input num)) (page-plus-first "p1"))
```

図 5.5: ユーザ入力が行われなかった時に表示されるエラーページ

この処理において，ページ p1 で入力が行われなかった場合，図 5.5 のエラーページが表示される．また入力した値が整数型でない場合，図 5.6 のエラーページが表示される．

5.6 ページ遷移チェック機能

サーバからユーザにページが出力される際，ページ遷移をチェックするために，出力するページの *page-tag* をグローバル変数*pre-page*に束縛する．その後，ユーザからリクエストされたページのページ定義構文が持つパラメータ from とサーバに保存された*pre-page*を比較する．from の値によって，以下の動作を行う．

- (from) の場合リクエストされたページを出力する．
- (from any) の場合リクエストされたページの *page-tag* と*pre-page*に束縛された *page-tag* を比較し，同じ場合エラーページを表示し，異なる場合リク

```
(define-page (p2 (from p1) (parameter (int num)) (input num))
                                                  (page-plus "p2"))
```

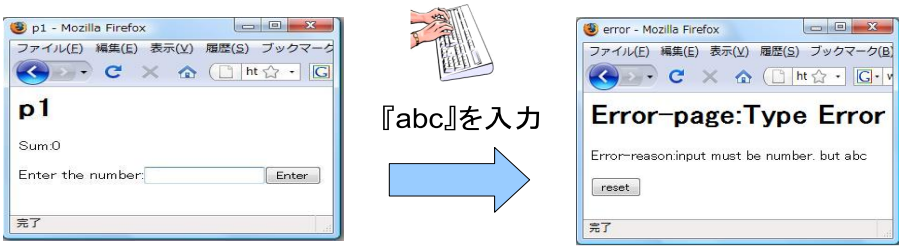

図 5.6: ユーザ入力の型が求めるものと異なる場合に表示されるエラーページ

エストされたページを表示する．

- (from *page-tag*...) の場合 (from *page-tag*...) に*pre-page*に束縛された *page-tag* が記述されている場合，リクエストされたページを出力し，記述されていない場合，エラーページを表示する．

エラーページが出力される例として，以下のようなコードで更新ボタンが押された場合を考えてみる．

```
(define-page (p1 (from any) (input num)) (page-plus-first "p1"))
```

ページ p1 の from には any が入力されているため，ページ p1 からの遷移は許可されていない．そのため，ページ p1 で更新ボタンを押すと，**図 5.7** のようなエラーページが出力される．

アプリケーション実行時のページ遷移チェック機能の使用例を**図 5.8** に示す．ページ p3 からバックボタンを用いて，ページ p2 に戻り，そこで，サーバにページ p3 を

図 5.7: ページ遷移によるエラーページ

リクエストする場合，サーバに保存されているカレントページが p3 であり，ページ定義構文 p2 の持つパラメータは，(from p2) であるため，エラーページが表示される．

ページ p3 からバックボタンを用いて，ページ p1 に戻り，そこで，サーバにページ p2 をリクエストする場合，サーバに保存されているカレントページは p3 であり，ページ定義構文 p1 の持つパラメータは (from any)，つまり自分以外の遷移は許可されるので，ページ p2 が表示される．

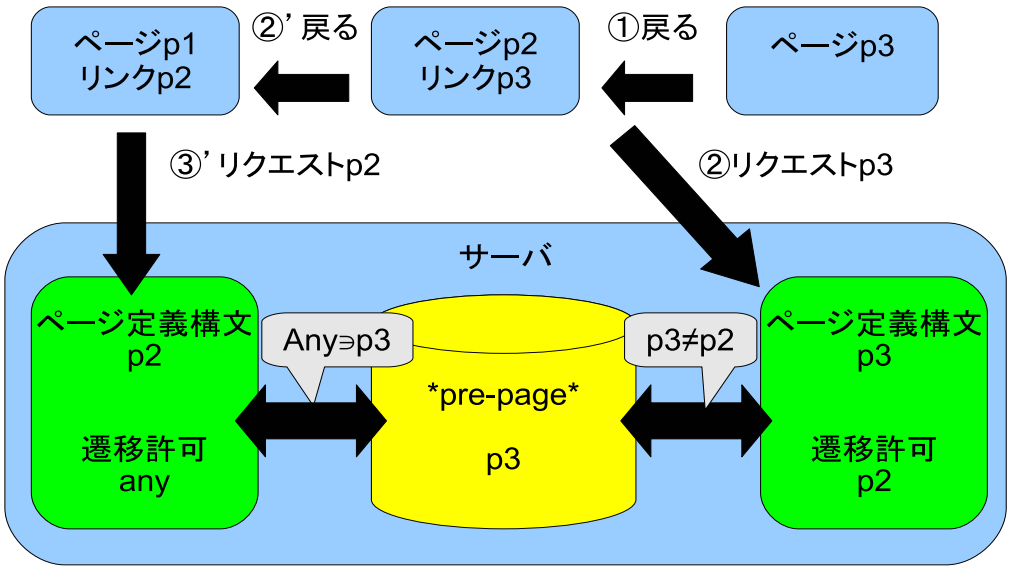

図 5.8: ページ遷移チェック機能の使用例

第 6 章

ページ遷移解析ツール

ソースプログラムから Web アプリケーションの実行中に起こる全てのページ遷移を解析するために，本研究ではページ遷移を解析ツールを言語に導入した．本研究の言語では，制御フローに基づくページ遷移とブラウザによるページ遷移の２種類のページ遷移を解析するので，解析ツールも制御フローに基づくページ遷移を解析するツール，ブラウザによるページ遷移を解析するツールの２つを実装した．また，制御フローに基づくページ遷移解析結果を用いて，Web アプリケーション開発をサポートするツールを実装した．

下記のプログラムから制御フローに基づくページ遷移とブラウザによるページ遷移を解析する過程を通じて，ツールの動作を説明していく．

```
(page-begin
  (page p1)
  (page-if 条件 (page p2) (page p3))
  (page-while 条件 (page p4))
  (page-cond  ((条件 (page p5) (page p6)))
  (else (page p7))))
```

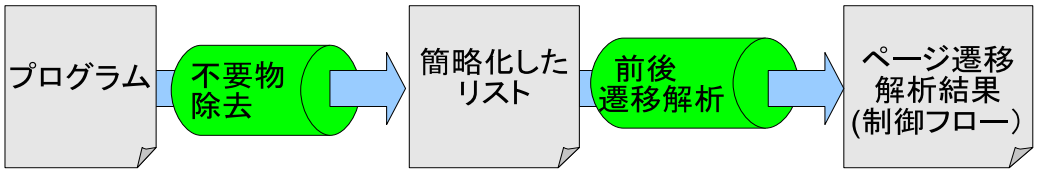

図 6.1: 制御フローに基づくページ遷移解析ツールの動作

6.1 制御フローに基づくページ遷移解析ツール

図 6.1 が制御フローに基づくページ遷移解析ツールの動作である．まず，本研究の言語で記述したプログラムから解析に必要のない部分を除去することで，簡略化したリストを作成する．そして，簡略化したリストから制御フローに基づくページ遷移解析を行う．

6.1.1 不要物除去プログラム

ページ遷移解析を行いやすくするために，本研究の言語で記述したプログラムから解析に必要のない部分を除去することで，簡略化したリストを作成する．

不要物除去の動作について説明する．評価機に読み込まれたページ遷移式を要素として取り入れている `page-begin` 式を簡略化する．各々の式は以下の簡略化ルー

ルに従って簡略化される．

```
(page p1) => p1
(page-begin (page p1) (page p2)) => (p1 p2)
(page-begin (page-begin (page p1) (page p2))) => (p1 p2)
(page-if 条件 (page p1) (page p2)) => (if p1 p2)
(page-while 条件 (page p1)) => (while p1)
(page-cond (条件 (page p1) (page p2)) (条件 (page p3)))
 => (cond (p1 p2) (p3))
(page-cond (条件 (page p1) (page p2)) (条件 (page p3)))
=> (if (p1 p2) (p3))
```

page 式は，*page-tag* に簡略化される．page-begin 式は，各要素を個別に簡略化したものを要素としたリストに簡略化される．**図 4.2** のページ遷移構文の構文定義に記述したが，本研究の言語では page-begin をいくつも重ねたプログラムを作成することができる．そのようなプログラムのページ遷移解析は煩雑であるため，現在注目している page-begin 式の外側の式も page-begin 式の場合，これを取り除く．

page-if 式は，第 2 要素である条件式を取り外し，第 3 要素，第 4 要素を簡略化し，page-if を if に置き換えたリストに簡略化される．

page-while 式は，第 2 要素である条件式を取り外し，第 3 要素を簡略化し，page-while を while に置き換えたリストに簡略化される．

page-cond 式は，各要素であるリストから条件式を取り外し，残りのリストを簡略化する．これを page-cond の全ての要素で行い，page-cond を cond に置き換えたリストに簡略化される．page-cond に else 節が含まれる場合，cond リストではなく page-if 式と同じように簡略化される．これは，page-cond が条件式のみで構成されている場合，page-cond はどの節も選ばれない可能性がある．それに対して else 節が含まれる場合，page-cond は必ず実行されるので，page-cond は else 節を含むときと含まないときで動作が異なる．この違いを解析結果に反映さ

せるために，else 節を含む page-cond を同じ意味の先頭の要素が if のリストに置き換えた．

以下が実行例である．

```
gosh>(route-list
                (page-begin (page p1)
                    (page-if cond (page p2) (page p3))
                    (page-while cond (page p4))
                    (page-cond  (cond (page p5) (page p6))
                                (else (page p7)))))

(p1 (if p2 p3) (while p4) (if (p5 p6) (p7)))
```

できあがるリストは，*page-tag*，if，while，cond のみで構成されている．

6.1.2 前後遷移解析

不要物除去処理により本研究の言語で記述したプログラムを簡略化したリストを作成した．簡略化したリストから各ページの前後に出力されるページを調べることで，制御フローに基づくページ遷移解析を行う．

プログラムを簡略化したリストから各ページの前後に出力されるページを調べ上げる from-to 関数の動作について説明する．from-to 関数は，簡略化したリストをもとに，各ページの前後に出力されるページを調べることで，制御フローに基づくページ遷移を全て調べあげる．遷移を調べるページを current，current より前に表示されるページを from，後に表示されるページを to とすると from-to 関数は，簡略化したリストから current を決定し，次に，その current の to を決定し，最後に current の from を決定することで，ページ遷移を調べていく．

from-to 関数はリストにある要素の遷移を調べる page-begin-from-to 関数と

if，while，cond の要素の遷移を調べる func-arg-from-to 関数で構成されている．page-begin-from-to 関数は引数として渡されたリストの第 1 要素を current として決定し，current の前後の遷移を調べる．page-begin-from-to 関数は current として決定されたものにより，処理が異なるので，それを以下で説明する．current が *page-tag* の場合，to の決定処理を行い，from の決定処理に移る．current が if，while，cond の場合，to の決定処理を行い，if，while，cond の要素と current の to が func-arg-from-to 関数に渡す．そして，リストから current を取り除いたものを引数とし，再び from-to 関数を呼び出す．current が空の場合，処理は終了する．

func-arg-from-to 関数は page-begin-from-to 関数から渡された if，while，cond の要素を current に決定し，前後の遷移を調べる．func-arg-from-to 関数も current により，処理が異なる．current がリストの場合，具体的には (p1 p2) 等の場合，このリストと page-begin-from-to 関数から渡された to が page-begin-from-to に関数に渡され遷移が調べられる．current が if，while，cond の場合，to の決定処理を行い，if，while，cond の要素と page-begin-from-to から渡された to が再び func-arg-from-to に渡される．current が *page-tag* の場合，page-begin-from-to 関数から渡された to が current の to として決定され，from の決定処理に移る．

to の決定処理

各ページの to を決定する．to は to-decide という関数によって決定される．リストにおいて，current の要素の 1 つ後ろの要素を調べていくわけだが，後ろの要素によって to-decide 関数の処理は異なる．後ろの要素がない場合，to は end と記述される．to の結果は，後述する from の決定処理で使用するため保存される．下記が後ろの要素ごとの to-decide の動作である．

- if の場合

第 2 要素，第 3 要素を to-decide に渡す．2 つの引数の to-decide 実行結果をリストにしたものが to となる．具体例として，(p1 (if p2 (p3 p4))) において，p1 の to は (p2 p3) である．

- while の場合

  while の要素によって処理が異なる．while の要素がリストの場合，もう 1 つ後ろの要素とリストの先頭の要素を to-decide で調べた結果をリストにしたものが，リストの最後尾の要素の to として決定される．while の要素が *page-tag* の場合，もう 1 つ後ろの要素を to-decide で調べた結果と while の要素の *page-tag* をリストにしたものが to として決定される．これは，条件によっては実行されない可能性があり，また，while なのでループする可能性があるからである．具体例として，((while (p1 p2)) p3) において，p2 の to は，p3，p1 である．(p1 (while p2) p3) において，p1 の to は，p2，p3 である．

- cond の場合

  全ての引数を to-decide に渡し，その結果をリストにしたものにもう 1 つ後ろの要素を to-decide した結果を追加したものが to として決定される．cond は条件次第で実行されない可能性があるので，さらに後ろの要素を調べる．具体例として，(p1 (cond (p1 p2) (p3 p4))) において，p1 の to は (p1 p3) である．

- リストの場合

  第 1 要素を取りだして返す．

- シンボルの場合

  to として決定される．具体例として，(p1 p2) において，p1 の to は p2 である．p2 の to は end である．

from の決定処理

次に，from を決定する．from は to から求める．あるページの from を調べる場合，各ページの to を調べ，to にあれば，そのページの *page-tag* を from に加える．全てのページの to になかった場合，そのページより前に表示されるページがないので，from は start になる．こうして，from，to が決定し，from を第 1 要素，to を第 2 要素とした以下のようなリストが解析結果として保存される．

((*page-tag*...) (*page-tag*...))

そして，次の要素に移る．これをプログラムを簡略化したリストの全要素で行うことで，プログラム実行中に起こる全てのページ遷移を調べることができる．

下記が実行結果である．

```
>(from-to '(p1 (if p2 p3) (while p4) (if (p5 p6) (p7))))

**page-transition-Control Flow**
p1:from start to (p2 p3)
p2:from p1 to (p4 p5 p7)
p3:from p1 to (p4 p5 p7)
p4:from (p2 p3 p4) to (p4 p5 p7)
p5:from (p2 p3 p4) to p6
p6:from p5 to end
p7:from (p2 p3 p4) to end
```

ページ遷移結果はページごとに保存される．そして，route-search 関数を用いることで，個別に参照することができるようにした．

```
>(route-search 'p1)
p1:from start to (p2 p3)
```

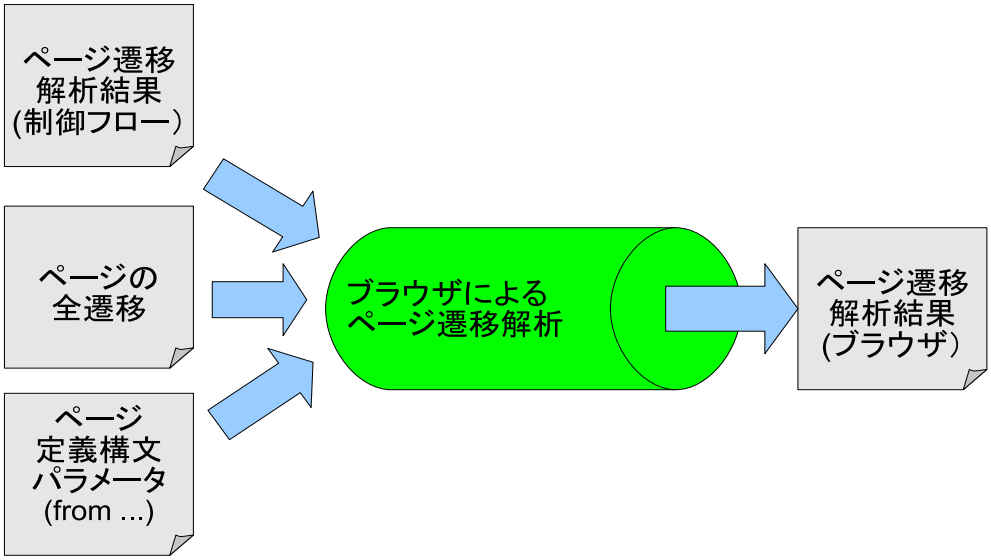

図 6.2: ブラウザによるページ遷移解析ツールの動作

6.2 ブラウザによるページ遷移を解析するツール

図 4.2 がブラウザによるページ遷移を解析するツールの動作である．制御フローに基づくページ遷移解析結果とページ定義構文がもつパラメータを用いて解析を行うことで，ブラウザによるページ遷移解析結果を出力する．

6.2.1 from-to2 関数

`from-to2` 関数は，制御フローに基づくページ遷移解析結果と個々のページがパラメータとして持つ (`from` *variable*) からブラウザによるページ遷移解析結果を作成する．`from-to2` 関数の動作について説明する．`from-to` 関数と同様，遷移を調べるページを `current`，`current` より前に表示されるページを `from`，後に表示されるページを `to` とする．`from-to` を行う際，グローバル変数`*page-list*`に遷移

を調べたページの page-tag を保存し，*page-list*に入っている全てのページの前後の遷移，言い換えると from と to を決定することで，ブラウザによるページ遷移を解析する．

from の決定

ページ定義構文が持つ (from *variable*) が各ページの from になる．(from p1) なら p1，(from any) なら自分以外の全てのページ，空リストなら全てのページが from として決定される．下記は具体例である．

(define-page (p1 (from p2))...) の場合，p1 の from は p2 となる．
(define-page (p1 (from any))...) の場合，p1 の from は p2，p3 となる．
(define-page (p1)...) の場合，p1 の from は p1，p2，p3 となる．

to の決定

全てのページの from から to を決定する．*page-list*から from を調べ，from にある場合，to に追加する．これを全ページに関して行うことで，to を決定できる．下記が具体例である．

```
p1:from (p2 p3)
p2:from (p1 p3)
p3:from (p1 p2)
```

p1 を全ページの from から調べると，p2 と p3 にあるので，p1 の to は p2，p3 となる．p2 を全ページの from から調べると，p1 と p3 にあるので，p2 の to は p1，p3 となる．p3 を全ページの from から調べると，p1 と p2 にあるので，p3 の to は p1，p2 となる．

こうして出来上がる from，to から制御フローに基づくページ遷移で解析した各ページの from，to を引いたものがブラウザによるページ遷移として出力される．

from が start，to が end である場合，制御フローの入り口と出口であるため from 同士，to 同士で引く行為は行われない．ブラウザによるページ遷移がない場合，空リストが表示される．

下記が，ブラウザによるページ遷移である．

```
>(from-to2 '(p1 (if p2 p3) (while p4) (cond (p5 p6) (p7))))


**page-transition-Browser**
p1:from (p1 p2 p3 p4 p5 p6 p7) to (p1 p4 p5 p6 p7)
p2:from (p2 p3 p4 p5 p6 p7) to (p1 p2 p3 p6)
p3:from (p2 p3 p4 p5 p6 p7) to (p1 p2 p3 p6)
p4:from (p1 p5 p6 p7) to (p1 p2 p3 p6)
p5:from (p1 p5 p6 p7) to (p1 p2 p3 p4 p5 p7)
p6:from (p1 p2 p3 p4 p6 p7) to (p1 p2 p3 p4 p5 p6 p7)
p7:from (p1 p5 p6 p7) to (p1 p2 p3 p4 p5 p6 p7)
```

下記は，制御フローに基づくページ遷移であるが，上記と比較してみる．

```
**page-transition-Control Flow**
p1:from start to (p2 p3)
p2:from p1 to (p4 p5 p7)
p3:from p1 to (p4 p5 p7)
p4:from (p2 p3 p4) to (p4 p5 p7)
p5:from (p2 p3 p4) to p6
p6:from p5 to end
p7:from (p2 p3 p4) to end
```

ページ遷移は制限していないので，全遷移から制御フローに基づくページ遷移を引いたものが，ブラウザによるページ遷移として表示されている．例えば，p1 の

from 欄は start であるため，(p1 p2 p3 p4 p5 p6 p7) となる．p1 の to 欄は制御フローに基づく遷移が (p2 p3) であるため，(p1 p4 p5 p6 p7) となる．

この結果はページごとに保存され，route-search2 関数を用いることで，個別に参照することができるようにした．

```
>(route-search2 'p1)
p1:from start to (p2 p3)
```

6.3 ページ遷移解析結果を利用したツール

アプリケーションの実行中に起こるページ遷移の読み取り以外に，ページ遷移解析結果の利用法を考察し，以下のツールを作成した．

6.3.1 ページ遷移制限の自動化ツール

ブラウザによるページ遷移を禁止したアプリケーションを作成するために，ページ遷移制限の自動化ツールを作成した．制御フローに基づくページ遷移解析結果の from 欄の部分をページのパラメータである (from *variable*) に設けることで，ブラウザによるページ遷移を禁止したアプリケーションを自動で作成することができる．from 欄が start の場合，from のパラメータは (from any) となる．ブラウザによるページ遷移により起こる開発者の予期しない出力やエラーを考慮する必要がなくなるので，開発者にとって便利である．しかし，ブラウザによるページ遷移が一切行えないので，ユーザにとって不便である．

6.3.2 静的ユーザ入力チェック

ページを実行するのに必要なデータが，前のページから送られてくる可能性があるのかを静的にチェックすることで，を作成した．制御フローに基づくページ遷移

解析結果，ページ定義構文が持つパラメータである input，parameter を用いて，静的なチェックを行う．

静的ユーザ入力チェックの動作について説明する．まず，ページ定義式から parameter が宣言されているページを全て探し出す．次に，制御フローに基づくページ遷移結果から parameter が宣言されているページの前に表示されるページを調べる．そして，parameter が宣言されているページの前のページで input が宣言されているかをチェックする．宣言されていれば，**parameter-check-success** と表示され，宣言されていなければエラーとともに宣言されていないページが表示される．

第 7 章

評価と考察

7.1 各手法におけるページ遷移の読み取り

本研究の言語と既存の言語におけるソースプログラムからのページ遷移の読み取りやすさを比較するために，本研究の言語を用いて，２つの数字をユーザに入力してもらい，その合計を出力するアプリケーションを作成した．プログラムの記述からの p1，p2，p3 というページ遷移の読み取りやすさを比較する．

7.1.1 CGI による Web アプリケーション開発におけるページ遷移読み取り

２章で述べたように，Web アプリケーションには，出力するページごとにプログラムを分割しているものとページごとに分けず，複数のページを１つのプログラムで記述したものがある．両方の場合において，制御フローに基づくページ遷移をソースプログラムからどのように読み取るかを考察する．

まず，ページごとにプログラムを分割するケースを考えてみる．このケースでは，HTML の form,a タグの属性である action,href から次にどのページへ遷移するかしか読み取ることができない．そのため，アプリケーションにおけるページの全遷移を読み取るためには，全てのファイルを調べる必要があるため，ページ遷移の読み取りは煩雑である．

```
<a href="http://localhost/plus1.html">リンク先にジャンプ</a>
<form method="POST" action="http://localhost/plus1.html">
```

次に，プログラムをページごとに分割せず，1つのソースプログラムで複数のページを出力するケースを cookie を用いた例をもとに考えてみる．下記は PHP で記述した Web アプリケーションにおいて，ページ遷移を判断する部分を抜き出したものである．

```
if(!isset($_COOKIE["STEP"])) p1();
else if($_COOKIE["STEP"]=="FIRST_STEP") p2();
else if($_COOKIE["STEP"]=="SECOND_STEP") p3();

function p1(){...setcookie("STEP","FIRST_STEP");}
function p2(){...setcookie("STEP","SECOND_STEP);}
```

実際は，ソースプログラムから p1()，p2() の記述箇所を探し，p1()，p2() の内部の記述から，setcookie の部分で入力された値を探し，if 式の条件式と照らし合わせなければならないので，読み取りは煩雑である．

7.1.2 継続を用いた Web アプリケーション開発におけるページ遷移読み取り

継続を用いた Web アプリケーションにおいて足し算のプログラムは以下のように記述できる．

```
(web-display
    (+ (web-read "First number:")
       (web-read "Second number:")))
```

web-display がページを出力する関数で，web-read がユーザ入力を受け取る関数である．web-read 内でも web-display が呼ばれている．プログラムの内容として，

まず，(web-read "First number:") で 1 つ目の入力をもらう．次に，(web-read "Second number:") で 2 つ目の入力をもらう．最後に，これらを足し合わせたものを (web-display...) で表示するプログラムである．継続を用いることで，ページ出力する箇所で制御フローが断続的にならないため，どこまでの処理が 1 ページ分なのかが分かりづらい．そのため直感的に p1 → p2 → p3 というページ遷移を把握しやすいとはいえないと考えられる．

7.1.3 本研究の言語を用いた Web アプリケーション開発におけるページ遷移読み取り

最後に本研究の言語の場合を見てみるとする．

```
(define-page (p1 (from any)(input num))...)
(define-page (p2 (from p1)(parameter (int num))(input num2))...)
(define-page (p3 (from p2)(parameter (int num2)))...)

(page p1) (page p2) (page p3)
```

ページ定義構文による記述は，読み込まれた順番によって page-begin 式に格納されていくため，先に記述したものが必ず先に実行される．プログラムの記述から p1 → p2 → p3 というページ遷移を容易に読み取りできる．また，各ページ定義式の from を見ることで，ページ p1 は自分以外の全てのページから遷移が許可されている．ページ p2 は p1 からのページ遷移のみ許可されている．ページ p3 は p2 からのページ遷移のみ許可されているというページ遷移の制限が読み取れる．

7.2 ページ遷移解析

従来の言語において，ソースプログラムからページ遷移を解析するのは困難である．例えば，下記のような高階関数を用いたプログラムを考える．

```
(define (f g)
            ...(g)...)
```

f，g をページを出力する関数と仮定する．この場合，g にはメモリに格納されているあらゆる関数が呼ばれる可能性があるため，ページ遷移解析の精度は著しく低下する．

本研究の言語では，従来の言語で行うことが不可能であった，ソースプログラムから静的にページ遷移の解析を行うことができる．これにより，開発者は Web アプリケーションで実行中に起こるページ遷移を全て把握することで，開発の段階から開発者の想定していないページ遷移を発見できるため，ページ遷移による予期しない結果やエラーの起こらない Web アプリケーション開発に貢献できると考えられる．

7.3 考察

本研究の言語を用いて，Web アプリケーション開発を行う利点として，Web アプリケーション実行中に起こる全てのページ遷移を静的に解析することで，アプリケーションを実行せずにページ遷移を把握できることにある．しかし，そのためには静的にページ遷移が決定される必要があるので，動的にページ遷移が変化するプログラムの作成には向かない．例えば，Wikipedia のように動的にページを作成するプログラムにおいて，ページ遷移は動的に変化してしまうため，本研究の言語で実装したページ遷移解析ツールを使用し，ページ遷移解析を行うことはできない．つまり，この言語を用いて作成することで，恩恵を受ける Web アプリ

ケーションは動的にページが生成されないものに限定されてしまう．

この言語を用いてWebアプリケーション開発を行う利点がない場合がある．例えば，ユーザが１からWebアプリケーションを作成する場合，開発者はページ遷移を念頭に置き，そこからWebアプリケーションを作成していくのが一般的であると考えられる．また，インタラクションの少ないWebアプリケーションを作成する場合，ページ遷移の設計が曖昧になることは少ないと考えられる．このような状況下では，Webアプリケーションにおいて開発者の想定していないページ遷移が含まれる可能性は非常に低く，開発者は，本研究の言語を使うまでもなく，ページ遷移を把握することができる．つまり，他人の書いたWebアプリケーションの改良やインタラクションの多いWebアプリケーションの作成等で本研究の言語を利用することで，静的にページ遷移を把握することができるという特徴が大いに活かされる．また，本研究ではページ遷移解析結果を用いて，ブラウザによるページ遷移を禁止したプログラムを自動で作成，ページ間の値の受け渡しによるエラーを静的に解析することができるため，アプリケーション実行中に起こるページ遷移を把握する目的以外で，本研究の言語で開発を行う利点はあると考えられる．

第 8 章

関連研究

ブラウザのバックボタン等によるページ遷移が Web アプリケーションの動作に影響を与えることは多くの論文で問題視されている [1–7]．インタラクティブな Web アプリケーションにおいて，Web ページを出力する際プログラムが終了するという従来の CGI の仕組みにより，制御フロー，ステート管理が複雑になり開発者の負担となっているため，これを取り除くために，様々な研究が行われている．CGI の仕組みにより複雑化する制御フロー，ステート管理を簡単にするために，継続サーバを用いた Web アプリケーション開発 [1] がある．ただ，継続を用いた Web アプリケーション開発は一般にはあまり行われていないため，従来の CGI アプリケーションを継続を用いた Web アプリケーションに自動で再構築する研究 [3] もある．また，インタラクティブな Web アプリケーションのモデルを作成し，エラーを動的にチェックする研究 [4,5] やブラウザによるページ遷移がステートに与える影響を回避するために，動的スコープの新しい形を提案，実装した研究 [6] がある．本研究は，ブラウザによるページ遷移により影響を受ける制御フローやステートに対して何かを行うのではなく，全てのページ遷移を把握することで，Web アプリケーションの動作に影響を与えるページ遷移を探すところまでを考えたものである．

また，ページ遷移を把握しつつ Web アプリケーションを作成する研究として，専用のエディタで Web ページの遷移図を設計し，その遷移図から Web アプリケーションを自動で作成する研究がある [10,11]．こうして作成される Web アプリケー

ションは，cookie，セッション管理を使用することができる標準レベルの安全性を備えている．

ページ遷移を把握した上で，Web アプリケーションを開発していくことができるが，この研究は，遷移図からページ遷移を把握し，本研究ではプログラムの記述からページ遷移の把握を試みたことに違いがあるといえる．

第 9 章

結論

9.1 まとめ

本研究では，開発者がソースプログラムの記述からページ遷移を読み取ることができ，ソースプログラムからページ遷移を静的に解析できるプログラミング言語の設計と実装を行った．

ソースプログラムからページ遷移の解析を容易に行えるようにするため，本研究の言語で記述する部分と Scheme で記述する部分を区別し，Scheme で本研究の言語を使用することはできない仕様とした．また，プログラムの制御フローからページ遷移を静的に決定できるように，本研究の言語であるページ定義構文，ページ遷移構文を設計，実装した．こうすることで，本研究の言語で記述した部分からページ遷移が静的に決定するため，ページ遷移を解析することが容易になる．そしてページ定義構文，ページ遷移構文で記述されたプログラムの制御フローから静的にページ遷移を解析するためのツールを実装した．

開発者の想定していないページ遷移発見後，ページ遷移を制限するために，ページ遷移チェック機能，ユーザ入力チェック機能を言語に実装した．そしてページ遷移解析結果を用いて，アプリケーションにページ遷移制限を自動で加えていく機能を言語に実装した．

この言語を用いて Web アプリケーション開発を行うことで，プログラム実行前にアプリケーションで起こる全てのページ遷移を把握できることを示し，Web ア

プリケーションにおいて，開発者の想定していないページ遷移がもたらす問題を容易に解決できる可能性を見出した．

9.2 今後の課題

Web アプリケーションは，ブラウザの持つページの複製機能を利用することで，予期しない結果をもたらす場合がある．この問題を，ページの複製が行われるたびに，動的に変化するスコープを取り入れることで解決した研究がある [6]．本研究では 5 章で紹介したように，ページの複製機能を利用することで動的にスコープを変化させる機能を取り入れた．しかし，アプリケーションによっては，このようなスコープ変化をしない方がいい場合がある．そこで，作成する Web アプリケーションに応じて適切にスコープを使い分けれる機能を取り入れることで，開発者はエラーの少ない Web アプリケーション開発が行えると考えられる．

本研究の言語ではページ遷移を制限する際に，ユーザに表示されるエラーページを処理系で用意しているが，これを開発者が自由にエラーページで表示される内容を決定できるようにし，エラーページの遷移もページ遷移解析の結果として表示可能にすることで，Web アプリケーション開発の自由度が上がると考えられる．

本研究の言語は言語開発に適しているという理由で Scheme で実装した．しかし，Scheme は Web アプリケーション開発において主要に使われている言語とは言えない．そのため本研究の言語の実用性について考察する場合，本研究の言語機能やページ遷移解析ツールを PHP や Perl といった言語で実装し，改めて評価する必要がある．

謝辞

本研究を遂行するにあたって，いろいろな方々にお世話になりました．指導教員の小宮常康先生には本論文の執筆に当たって日頃から熱心なご指導，ご鞭撻を賜りました．ご多忙中にもかかわらず，論文の草稿を丁寧に読んで下さり，大変貴重なご助言を頂きました．ここに厚く御礼申し上げます．多田好克先生，佐藤喬先生には，研究の方向性の決定，そして研究を進める上で貴重なご助言を頂きました．大変感謝しております．石橋崇氏には，研究，論文に対し，日頃から数多くのご助言頂きました．大変感謝しております．そして，本研究が行えたことは，研究方針や方法論について日頃より活発に議論をし，共に充実した研究生活を送ってきた多田研，小宮研，村山研，水野研の学生諸氏のおかげでもあります．これらの学生諸氏に深く感謝致します．

参考文献

[1] S. Krishnamurthi, P. W. Hopkins, J. Mccarthy, P. T. Graunke, G. Pettyjohn, and M. Felleisen.: "Implementation and use of the PLT scheme Web server," HOSC, 20(4):pp.431–460, 2007.

[2] Christian Queinnec, Inverting back the inversion of control or, continuations versus page-centric programming, *ACM SIGPLAN Notices*, v.38 n.2 pp.57–64, February 2003.

[3] Jacob Matthews, Robert Bruce Findler, et -al.: "Automatically Restructuring Programs for the Web," *Automated Software Engineering*, Vol. 11, Number. 4, pp. 337–364, Oct.2004.

[4] S. Krishnamurthi, R. B. Findler, P. Graunke, and M. Felleisen.: "Modeling Web interactions and errors," In D. Goldin, S. Smolka,and P. Wegner, editors, *Interactive Computation: The New Paradigm*, Springer Lecture Notes in Computer Science. Springer-Verlag, 2006. To appear.

[5] D. R. Licata and S. Krishnamurthi.: "Verifying interactive Web programs.," In *IEEE International Symposium on Automated Software Engineering*, pp.164–173, Sept. 2004.

[6] Jay McCarthy and Shriram Krishnamurthi.: "Interaction-safe state for the Web.," *In Scheme and Functional Programming*, September 2006.

[7] Christian Queinnec.: "The influence of browsers on evaluators or, continuations to program web servers.," In *International Conference on Functional Programming*, pp.23–33, 2000.

[8] Michael Sperber, R. Kent Dybvig, Matthew Flatt, and Anton van Straaten. Revised (5.97) report on the algorithmic language Scheme, June 2007b. URL `http://www.r6rs.org/versions/r5.97rs.pdf`. With R. Kelsey, W. Clinger, J. Rees, R. B. Findler, and J. Matthews.

[9] Gauche, http://practical-scheme.net/gauche/index-j.html

[10] M.Taguchi, T.Susuki, T.Tokuda: Generation of Server Page TypeWeb Applications from Diagrams. *Proc. of the 12th Conference on Information Modelling and Knowledge Bases*, pp.117–130 2002.

[11] K.Jamroendararasame, T.Matsuzaki, T.Suzuki, T.Tokuda: Generation of Secure Web Applications from Web Transition Diagrams. *Proc. of the IASTED International Symposia Applied Informatics*, pp.496–501, 2001.