1F4-2

# 直接操作を取り入れた自然言語プログラミング

金子 望*1 Nozomu KANEKO　鬼沢 武久*1 Takehisa ONISAWA

*1 筑波大学大学院 システム情報工学研究科
Systems and Information Engineering, University of Tsukuba

In this paper, a programming system for text-editing task that uses both natural language text and direct manipulation is proposed. The system is based on the idea of *programming by paraphrasing*, which enables users with little knowledge of computer programming to make computer program by inputting natural language text because paraphrasing is usually used as human everyday language communication. The case-based reasoning (CBR) method is used for paraphrasing so that the system can acquire the meaning of unknown natural language text inputted by users. This paper also shows a running example using the system.

## 1. はじめに

自然言語は我々の日常的なコミュニケーションにおける主要な手段の一つであり, そのためプログラミングに自然言語を用いることができれば, プログラミングに関する特別な知識を持たないエンドユーザでもプログラミングを行うことが可能になる. このような背景から, エンドユーザ・プログラミングの分野では自然言語を用いたプログラミングシステムが盛んに研究されてきている [1, 2, 3]. しかし, これらのシステムでは従来のプログラミング言語と同様に, 語彙文法などの言語の要素を設計者があらかじめ定義するというアプローチが取られている. これに対し, ユーザとの対話を通して入力された自然言語テキスト (以下単に「テキスト」と呼ぶ) の意味を獲得することのできる自然言語プログラミングシステムが提案されている [4]. このシステムは, プログラミングの手段として「言い換え」を用いる “言い換えによるプログラミング” の考え方に基づいたもので, 言い換えには事例ベース推論を用いている. ユーザの入力したテキストからシステムにとって既知のテキストへと半自動的に言い換えを行うことで入力されたテキストの意味を推論し, システムにとって未知の入力テキストに対してはユーザ自身が言い換えを追加することが可能である. ただし, “言い換えによるプログラミング” では, 言い換えによってテキスト同士の関係を表すことはできるが, テキストと処理内容との対応関係を翻訳事例として事前に定義する必要がある. これは, 記号としての言語と現実世界の非言語的な実体をどのように対応づけるかという, 記号接地 (シンボル・グラウンディング)[5] 問題の一種である. そこで, 本論文では翻訳と言い換えを用いたテキストのみの方法に加えて, 直接操作を導入する. 直接操作とは, 画面上のアイコンやボタン, ウィンドウなどを実物と同じような感覚で操作するインタフェースのことで [6], エンドユーザ・プログラミングの分野では直接操作からプログラムを生成する研究も行われている [7, 8]. 直接操作を導入することによって, ユーザは直接操作とテキストのどちらか好きな方を使って指示を入力することができる. また, 直接操作による入力とテキストによる入力を併用することで, テキストの意味を言い換えによって別のテキストで表すだけでなく, 直接操作によっても表すことができる.

本論文では, テキスト編集ドメインを対象として, 言い換えを用いた自然言語プログラミングシステムに直接操作を導入したシステムを構築する. さらにシステムの実行例を示す.

## 2. 自然言語によるプログラミングと直接操作

図 1 は言い換え, 翻訳, 直接操作の関係を表したものである. ただし, 図において各層は, 上から順にテキストの集合, 操作の集合, プログラムの集合を表している. また, 矢印は要素間の関係を表しており, 特に実線の矢印はユーザが入力可能な関係である. 言い換えはテキスト間の関係性を表しており, このようにテキスト全体が一つの体系を構成する. また, 翻訳はテキストとプログラムの対応関係を記述したものである. そのため, ユーザの入力した未知のテキストは, 言い換えと翻訳を通じてプログラムへと変換される. 一方, 直接操作による入力とテキストによる入力を統合することで, ユーザはテキストと直接操作の対応関係を表現することができる. 直接操作は容易にプログラムへと変換されるため, このようにして直接操作によって表現されたテキストからプログラムへの変換が可能である.

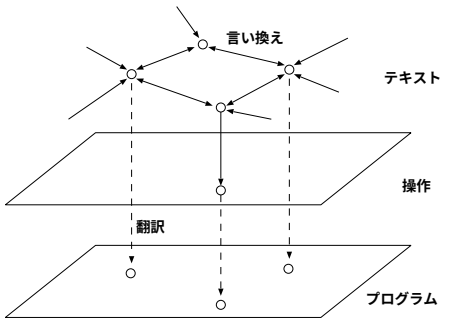


図 1: 言い換えと翻訳

## 3. システム構成

本システムの構成を図 2 に示す. システムはまず, 入力されたテキストに対して形態素解析と構文解析を行い, 構文木を得る. 次に構文木の構造に基づいて事例ベース推論を行う. 本システムでは翻訳事例と言い換え事例の 2 種類の事例を用いて事例ベース推論を行っており, 類似事例が見つかった場合には見つかった事例に応じて翻訳, または言い換えを行う. 類似事例が見つからなかった場合はユーザに言い換えを求め, 言い換え事例として追加する. 言い換えられた結果は再度システムへと入力され, このような言い換えを再帰的に行うことで, 任意の入力テキストに対して最終的にはプログラムが出力, 実行される. また, 直接操作による入力は翻訳事例を用いてテキストへと変換されるが, 翻訳事例に存在しない直接操作が入力された場合は, ユーザが対応するコマンドをテキストで入力することで翻訳事例を追加することが可能である.

なお, 本システムの実装は Java 言語および Scheme 言語により行い, Scheme 処理系として Java 言語で実装された Kawa[9] を用いる. また形態素解析に MeCab[10] を用いている.

A: 〒 305-8573 茨城県つくば市天王台 1-1-1
E-mail: nozom@fhuman.esys.tsukuba.ac.jp

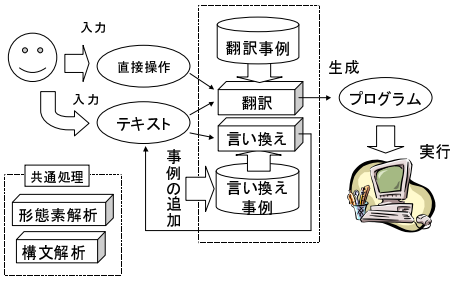


図 2: システム構成

### 3.1 事例ベース推論による翻訳と言い換え

本システムの事例ベース推論は以下のように行われる．まず, システムは類似した構造を持つテキストを含む事例を検索する．事例の類似度は, 翻訳事例ではパターンマッチングによる一致度により, 言い換え事例では構文木の畳み込みカーネル [11] により, それぞれ計算する [4]．類似事例が複数見つかった場合, 最も類似した事例を用いて出力が生成される．翻訳による出力はプログラムであり, 言い換えによる出力は別のテキストである．類似事例が見つからなかった場合, システムはユーザに元のテキストを別のテキストへ言い換えるように求める．元のテキストと言い換え後のテキストの組が新しい言い換え事例となり, 言い換え事例データベースに追加される．

類似事例が翻訳事例データベース中で見つかった場合は翻訳が行われ, テキストからプログラムを生成する．本システムで用意されている翻訳事例を表 1 に示す．ただし翻訳事例中で “?x<型>” のような形式で表された部分は変数であり, 数字, 文字列, 埋め込み文の 3 種類いずれかの型を持つ．

本システムは句点で区切られた範囲を文として扱い, 複数の文からなるテキストも入力として受け付ける．また述語とそれに係る 0 個以上の補語の組を節と呼び, 文は複数の節を含むことができる．これは「～ して、～ する」のような並列節と,「～ なら ～ だ」「～ だったら ～ する」のような従属節の両方がある．従属節の中で述語に仮定形が用いられている節は条件を表す節としてプログラム中では if 節に翻訳される．同様に, 述部に助動詞「ない」を含む「～（で）ない」という節は否定表現として扱われ, プログラム中では真偽値が反転される．さらに, 繰り返しを表すために, 表 1 に示されるように “『?$x$<埋め込み文>』を『?$y$<埋め込み文>』まで繰り返す” のような表現が翻訳事例として用意されている．

例えば, “『行の長さ』が 0 より大きいなら、『行末』まで『右に移動する』を繰り返す” の翻訳は次のように行われる．1) 最初の節は “?$x$<数>が?$y$<数>より大きい” とマッチする．ただし $x$ と $y$ は変数であり, それぞれ「『行の長さ』」,「0」と対応している．2) 変数に対して `let` 式による局所的な束縛が作られる．3) 2 番目の節は “?$x$<埋め込み文>まで?$y$<埋め込み文>を繰り返す” とマッチする．この場合, 変数 $x$, $y$ の値は単純な数値や文字列ではなく, それぞれ “行末 (だ)”, “右に移動する” というテキストに対応した値であるため, 束縛が作られる代わりに, 変数の出現位置にテキストの翻訳結果が展開される．4) それぞれの節が翻訳された結果が合成され, 図 3 のようなプログラムが得られる．このとき最初の節が仮定形の特徴を持つ従属節であるため, テキスト全体は if 節へと変換される．

表 1: 翻訳事例

| カーソルの移動に関係する指示 |
|---|
| 左に移動する |
| ?x<数>文字左に移動する |
| 右に移動する |
| ?x<数>文字右に移動する |
| 前の行に移動する |
| ?x<数>行上に移動する |
| 次の行に移動する |
| ?x<数>行下に移動する |
| 前の単語に移動する |
| 次の単語に移動する |
| 行頭に移動する |
| 行末に移動する |
| 文頭に移動する |
| 文末に移動する |

| 文字列の挿入削除と選択に関係する指示 |
|---|
| ?x<文字列>を挿入する |
| 1 文字削除する |
| ?x<数>文字削除する |
| 前に 1 文字削除する |
| 前に?x<数>文字削除する |
| コピーする |
| 貼り付ける |
| 切り取る |
| 改行する |
| 選択を開始する |
| 選択を解除する |
| 選択範囲を削除する |
| 全選択する |

| 真偽値を取る命題 |
|---|
| 行頭だ |
| 行末だ |
| 文頭だ |
| 文末だ |
| ?x<数>が?y<数>より大きい |
| ?x<数>が?y<数>より小さい |
| ?x<数>が?y<数>以上 |
| ?x<数>が?y<数>以上 |
| ?x<数>が?y<数>と等しい |

| 式の値が数値である命題 |
|---|
| 行の長さ |

| 繰り返しに関係する指示 |
|---|
| ?x<埋め込み文>を?y<数>回繰り返す |
| ?x<埋め込み文>を?y<埋め込み文>のあいだ繰り返す |
| ?x<埋め込み文>を?y<埋め込み文>まで繰り返す |
| ?x<埋め込み文>あいだ?y<埋め込み文>を繰り返す |
| ?x<埋め込み文>まで?y<埋め込み文>を繰り返す |

```
(if (let ((x (- (line-end-position)
                (line-beginning-position)))
          (y 0))
      (> x y))
    (while (not (eol?))
      (forward-char)))
```

図 3: 翻訳例

### 3.2 画面構成

本システムのメインウィンドウ画面は図 4 のようになっている．左の領域が編集対象のテキストを表示するテキスト編集領域である．右はテキスト一覧と入力履歴を表示するテキスト表示領域である．下はテキスト入力領域で, ここにテキストを入力して「実行」を押すとプログラムが生成, 実行される．

テキスト一覧には, その時点でシステムが理解できる全てのテキストが表示されている．また, 入力履歴にはそれまでにユーザが入力したテキストが表示されている．テキスト表示領域とテキスト入力領域の間にあるボタンを押すと, テキスト表示領域で選択中のテキストがテキスト入力領域に追加される．これによってテキスト入力の際にテキスト表示領域に含まれるテキストを容易に再利用することができるようになっている．

ユーザがテキスト編集領域でカーソルを移動したり, 直接, 文字を入力すると, システムは翻訳事例を用いて操作に対応したテキストを求め, テキストによって入力された場合と同様に入力履歴に追加する．ただし, 翻訳事例として定義されていない操作が入力された場合は, 図 5 に示すようなダイアログを表示して, ユーザがその操作に対応するテキストを入力できるようにする．

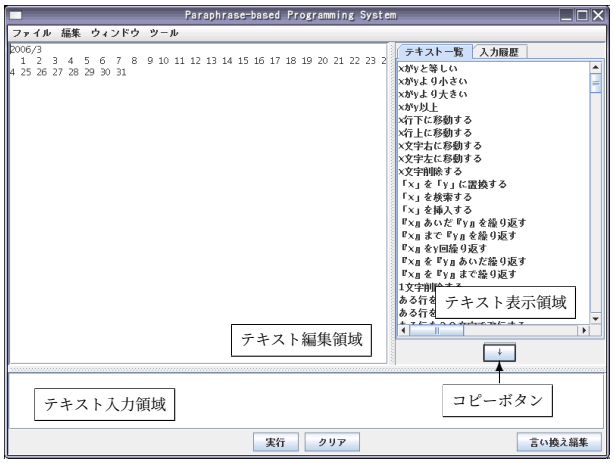


図 4: メインウィンドウ画面

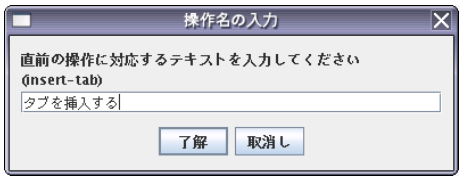


図 5: 操作名の入力画面

### 3.3 言い換えの選択と追加

入力したテキストに類似した言い換え事例が複数見つかった場合は, 図 6 に示すような言い換えのリストが表示される. このリストの中に正解があれば選択して「了解」ボタンを押す. どれも違う場合は, 「新規」にチェックを入れて「了解」ボタンを押すことで, 言い換えの追加に移る. すると図 7 のような言い換えの追加画面が表示される. 左上の領域に入力したテキストが表示されているので, これを言い換えたものを左下の領域に入力する. その際, 右側のテキスト表示領域からコピーすることもできる.

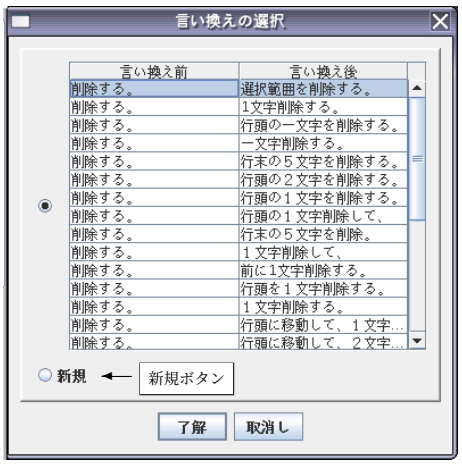


図 6: 言い換えの選択画面

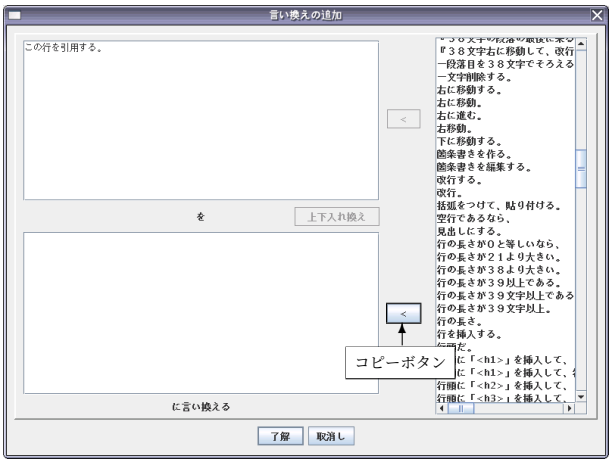


図 7: 言い換えの追加画面

言い換えの前後で同じ数字や文字列が存在する場合, それらは対応しているとみなされる. 例えば, 「3 文字切り取る」→「選択を開始して 、3 文字右に移動して 、切り取る」という言い換えは下線部が対応しているため, 「5 文字切り取る」→「選択を開始して 、5 文字右に移動して 、切り取る」という言い換えにも自動的に拡張される. 一方, そのような対応がない場合には数字や文字列に対して特別な処理を行わず, その違いは無視される. そのため 「1 行空ける」→「改行する」という言い換えが存在するとき, 「2 行空ける」も「3 行空ける」も「改行する」へと言い換えられる.

## 4. 実行例

本システムの実行例を示す. 本実行例では, 図 8 のような文書が読み込まれた状態から始まり, 図 9 のような状態になるまでの一連の操作をテキストで入力するという, カレンダー作成課題を行う. 課題を行う際には, 表 2 に示すような全部で 115 の言い換え事例を用意しておく. これらは事前に行った被験者実験で得られた言い換え事例を元にしたものである.

```
2006/3
  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20
21 22 23 24 25 26 27 28 29 30 31
```

図 8: 初期状態

```
          2006/3
                1  2  3  4
 5  6  7  8  9 10 11
12 13 14 15 16 17 18
19 20 21 22 23 24 25
26 27 28 29 30 31
```

図 9: 終了状態

図 10 に示すテキストが順に入力されると, システムはそれぞれの入力に対して実行を行い, 結果をユーザに提示する. このとき追加された言い換え事例を表 3 に示す. 入力テキスト中で先頭に ⋆ が付いたものは, 直接操作により入力されたものである. ただし, 本課題の目的は文書の状態を初期状態から終了状態へと変化させる一連の操作をテキストで入力することであるため, 最終的にそのようなテキストが得られるまで, 途中で何度か文書の状態を初期状態に戻す操作を行っている. 1 行目から 21 行目では主に直接操作を使い, 目的のプログラムに必要な操作を入力している. 22 行目から 28 行目では, 言い換えを追加することによって直接操作で行った指示を一般化している. このとき下線で示した部分についてシステムは言い換えを求め, それに対しユーザは言い換えを追加している. ここまでの実行結果でユーザは先頭行の空白の個数が間違っていたことに気づいたため, 29 行目から 40 行目で再び直接操作を用いて修正し, 最終的に 41 行目で目的の動作を実現するテキストが得られている. このように, 本システムでは直接操作とテキスト入力を柔軟に組み合わせてプログラミングが行えることがわかる.

## 5. おわりに

本論文では, 直接操作を組み込んだ自然言語テキストによるプログラミングシステムを提案した. またテキスト編集を例にして, 実行例を示した. 本システムは直接操作とテキスト入力を組み合わせて柔軟にプログラミングを行うことができる.

現在のシステムは, 入力されたテキストの意味は状況によらず一定であるという仮定に基づいて処理を行っているが, 実際にはテキストの意味を考える場合にコンテキストの影響は無視できない場合が多い. そのため今後はコンテクストの一つとして編集中の文書の状態を考慮するつもりである.

## 参考文献

[1] 木村明, 片桐明. 日本語プログラミング言語『Mind』について – その概要と、日本語プログラミングの実用性. 情報処理学会 研究報告 プログラミング言語, PL-16-4, pp. 25–32, 1988.

表 2: 初期言い換え事例 (一部)

| 言い換え前 | 言い換え後 |
|---|---|
| １行上へ移動する。 | １行上に移動する。 |
| ２０文字ある。 | 『行の長さ』が２０以上。 |
| ２１で改行。 | 『右移動』を２１回繰り返して、改行する。 |
| ３０文字で改行する。 | ある行を３０文字で改行する。 |
| ３８文字にそろえる。 | 『３８文字の段落の最後に来る』まで『３８文字で改行する』を繰り返す。下に移動する。 |
| ３８文字の段落の最後に来る。 | 『行の長さ』が３８より小さい。 |
| ３８文字以下の短行はそのままにする。 | 『行の長さ』が３８より小さいなら、次の行に移動。 |
| ３８文字以上の長行は改行する。 | 『行の長さ』が３８より大きいなら、ある行を３８文字で改行する。 |
| ３文字空白を入れる。 | 「␣␣␣」を挿入する。 |
| ある行を３８文字で改行する。 | 行頭に移動。３８文字右に移動。右に移動。改行。 |
| ある段落を３８文字でそろえる。 | 段落の幅を３８文字にする。 |
| ここから行末まで切り取る。 | 選択を開始する。行末に移動する。切り取る。 |
| この後を全部削除する。 | 選択を開始する。文末に移動する。選択範囲を削除する。 |
| カレンダーにする。 | 行頭に「␣␣␣」を挿入して、行頭に移動する。『行末』まで『２１で改行』を繰り返す。 |
| カレンダーの短行はそのままにする。 | 『行の長さ』が２０より小さいなら、次の行に移動。 |
| カレンダーの長行は改行する。 | 『行の長さ』が２０より大きいなら、ある行を２０文字で改行する。 |
| カレンダーを作成する。 | 行頭に「␣␣␣」を挿入。文頭に移動。『文末だ』まで『カレンダーの長行は改行する。カレンダーの短行はそのままにする』を繰り返す。 |
| カレンダー作成する。 | 「␣␣␣」を挿入して、文頭に移動して、『行の長さが２１より大きい』あいだ『２１文字右に移動して、改行する』を繰り返す。 |
| 一段落目を３８文字でそろえる。 | ある段落を３８文字でそろえる。 |
| 右に進む。 | 右に移動する。 |
| 右移動。 | 行末で無いなら、１文字右に移動する。 |
| 下に移動する。 | １行下に移動する。 |
| 空行であるなら、 | 行の長さが０と等しいなら、 |
| 行の長さが０と等しいなら、 | 『行の長さ』が０と等しいなら、 |
| 行の長さが３８より大きい。 | 『行の長さ』が３８より大きい。 |
| 行の長さが３９以上である。 | 『行の長さが３８より大きい』あいだ『３８文字右に移動して、改行する』を繰り返す。 |
| 行の長さが３９文字以上。 | 『行の長さ』が３９文字以上。 |
| 行の長さが３９文字以上である。 | 行の長さが３９文字以上。 |
| 行を挿入する。 | 行頭に移動する。改行する。上に移動する。 |
| 行頭に「␣␣␣」を挿入。 | 行頭に移動して、「␣␣␣」を挿入する。 |
| 行頭の１文字を削除する。 | 行頭に移動して、１文字削除する。 |
| 行頭の一文字を削除する。 | 行頭の１文字を削除する。 |
| 行末で無いなら、 | 行末でないなら、 |
| 行末の５文字を削除。 | 行末に移動。５文字左に移動。５文字削除。 |
| 行末の５文字を削除する。 | 行末に移動して、５文字削除する。 |
| 最初に戻る。 | 文頭に移動する。 |
| 最初の行の行末に移動する。 | 最初に戻る。行末に移動する。 |
| 削除する。 | 選択範囲を削除する。 |
| 次の行の行頭に移動。 | 次の行に移動して、行頭に移動。 |
| 上に移動する。 | １行上に移動する。 |
| 折り返す。 | 『３８文字で改行する』を6回繰り返して、１行下に移動して、『３０文字で改行する』を6回繰り返す。 |
| 短行はそのままにする。 | 『行の長さ』が３８より小さいなら、次の行に移動。 |
| 段落の幅を３８文字にする。 | 『３８文字右に移動して、改行する』を『３８文字ある』あいだ繰り返す。 |
| 長行は３８文字で改行する。 | 『行の長さ』が３８より大きいなら、ある行を３８文字で改行する。 |
| 長行は改行する。 | 『行の長さ』が３８より大きいなら、ある行を３８文字で改行する。 |
| 文頭へ移動して、 | 文頭に移動する。 |

表 3: 追加された言い換え事例

| 言い換え前 | 言い換え後 |
|---|---|
| ３日分空ける。 | 『「␣␣␣」を挿入する』を３回繰り返す。 |
| ２０文字で改行する。 | ある行を２０文字で改行する。 |
| ４文字空白を入れる。 | 『「␣」を挿入する』を４回繰り返す。 |
| カレンダーを作る。 | 行頭に移動する。『行の長さが２０文字以上である』あいだ『２０文字で改行する』を繰り返す。 |
| 空白を７文字入れる。 | ７文字空白を入れる。 |

```
1  ★ 「␣」を挿入する。
2  ★ 「␣」を挿入する。
3  ★ 「␣」を挿入する。
4  ★ 「␣」を挿入する。
5  ★ 次の行に移動する。
6  ★ 行頭に移動する。
7  ★ 「␣」を挿入する。
8  ★ 「␣」を挿入する。
9  ★ 「␣」を挿入する。
10 ★ 「␣」を挿入する。
11 ★ 「␣」を挿入する。
12 ★ 「␣」を挿入する。
13 ★ 「␣」を挿入する。
14 ★ 「␣」を挿入する。
15 ★ 「␣」を挿入する。
16 ★ 行頭に移動する。
17 ある行を２０文字で改行する。
18 ある行を２０文字で改行する。
19 ある行を２０文字で改行する。
20 ある行を２０文字で改行する。
21 ★ 文頭に移動する。
   (ここで文書の状態を初期状態に戻す)
22 『「␣」を挿入する』を 4 回繰り返す。
23 ★ 次の行に移動する。
24 ★ 行頭に移動する。
25 ３日分空ける
26 行頭に移動する。
27 ２０文字で改行する
28 『２０文字で改行する』を『行の長さが２０文字以上である』あいだ繰り返す
   (ここで文書の状態を初期状態に戻す)
29 ４文字空白を入れる。次の行の行頭に移動する。３日分空けて、カレンダーを作る。
30 ★ 前の行に移動する。
31 ★ 前の行に移動する。
32 ★ 前の行に移動する。
33 ★ 前の行に移動する。
34 ★ 前の行に移動する。
35 ★ １文字削除する。
36 ★ １文字削除する。
37 ★ １文字削除する。
38 ★ １文字削除する。
39 空白を７文字入れる
40 ★ 「␣」を挿入する。
   (ここで文書の状態を初期状態に戻す)
41 ８文字空白を入れる。次の行の行頭に移動する。３日分空けて、カレンダーを作る。
```

図 10: 入力テキスト


[2] 兼宗進, 御手洗理英, 中谷多哉子, 福井眞吾, 久野靖. 学校教育用オブジェクト指向言語「ドリトル」の設計と実装. 情報処理学会論文誌, Vol. 42, No. SIG11, pp. 78–90, 2001.

[3] クジラ飛行机. 日本語プログラミング言語「なでしこ」公式ガイドブック. 毎日コミュニケーションズ, 2005.

[4] Nozomu Kaneko and Takehisa Onisawa. An experimental study on computer programming with linguistic expressions. In *Lecture Notes in Computer Science*, Vol. 3681, pp. 911–917. Springer-Verlag, 2005.

[5] 今井むつみ. 言語獲得におけるシンボルグラウンディング. 人工知能学会誌, Vol. 18, No. 5, pp. 580–585, 2003.

[6] 増井俊之. インターフェイスの街角. ASCII, 2005.

[7] Allen Cypher and David Canfield Smith. KidSim: End user programming of simulations. In *Proc. of the SIGCHI conference on Human factors in computing systems*, pp. 27–34, New York, NY, USA, 1995. ACM Press.

[8] Robert C. Miller and Brad A. Myers. LAPIS: Smart editing with text structure. In *Extended abstract for CHI 2002 formal demonstration*, pp. 496–497, 2002.

[9] Per Bothner. The Kawa language framework. `http://www.gnu.org/software/kawa/`.

[10] 工藤拓. MeCab: Yet another part-of-speech and morphological analyzer. `http://mecab.sourceforge.jp/`.

[11] Tetsuro Takahashi, Kozo Nawata, Kentaro Inui, and Yuji Matsumoto. Effects of structural matching and paraphrasing in question answering. *IEICE Transactions on Information and Systems*, Vol. E86-D, No. 9, pp. 1677–1685, 2003.