アンダースタンディング・コンピュテーション　第 4 章
プッシュダウン・オートマトン

岡本 吉央
okamotoy@uec.ac.jp

電気通信大学

2019 年 7 月 19 日

最終更新：2019 年 7 月 19 日　15:25

スケジュール

2 プログラム意味論 (5 月)
3 有限オートマトン (6 月)
4 プッシュダウン・オートマトン (7 月)
5 チューリング機械
6 ラムダ計算
7 万能性
8 決定可能性
9 抽象解釈／静的意味論

能力を高める

今からやること

「有限オートマトン」に簡単な「記憶 (メモリ)」の機能を追加して，有限オートマトンの能力を高める

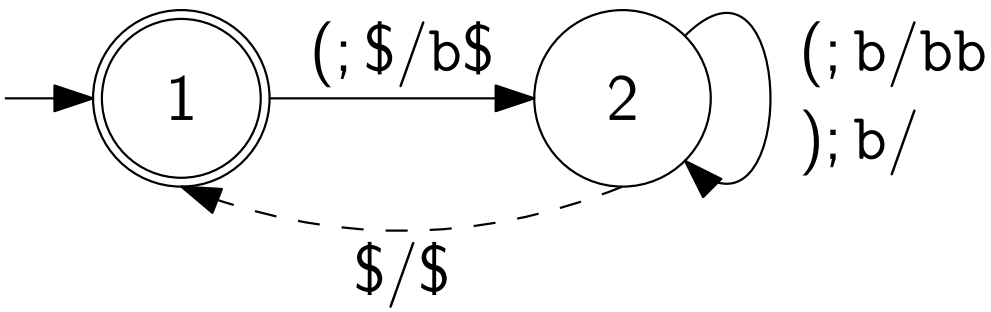
- $\rightsquigarrow$ プッシュダウン・オートマトン

目次

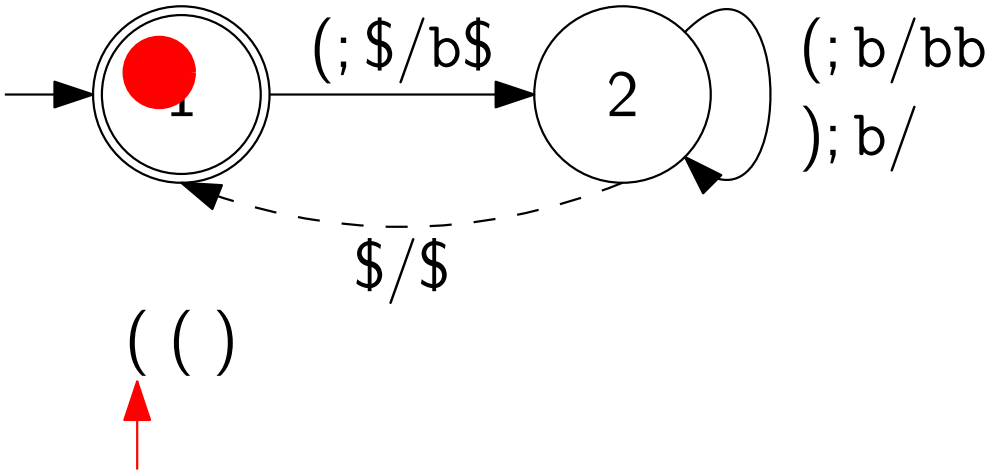
標語的にいえば

プッシュダウン・オートマトン = 有限オートマトン + スタック

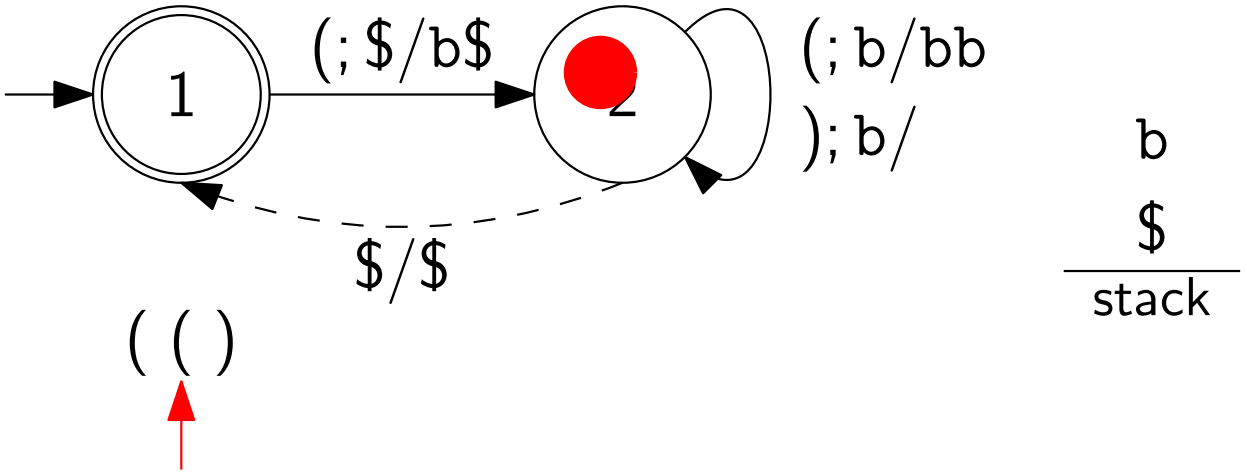
スタック = 後入れ先出し (last-in first-out) のデータ構造



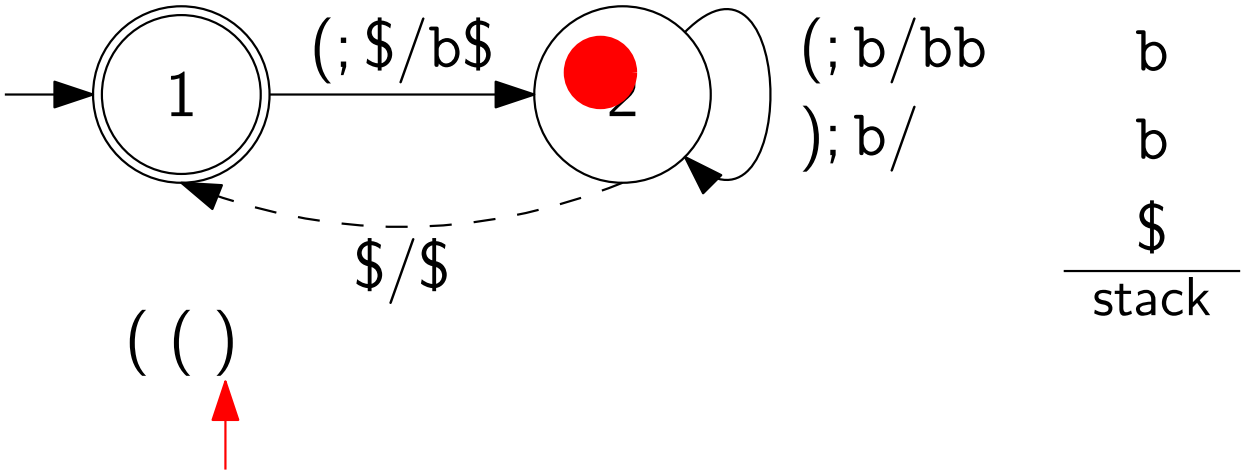
棄却する例



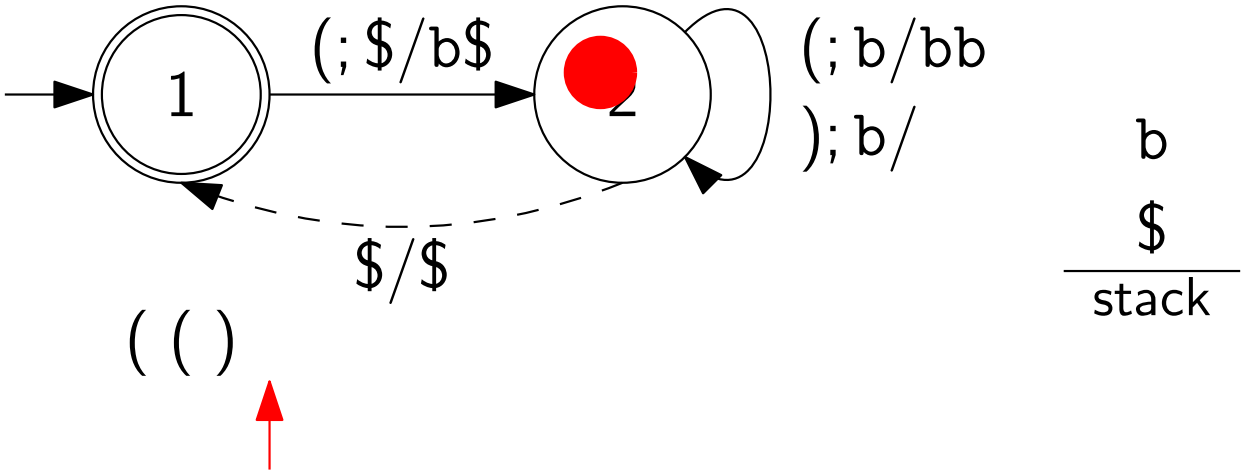
棄却する例



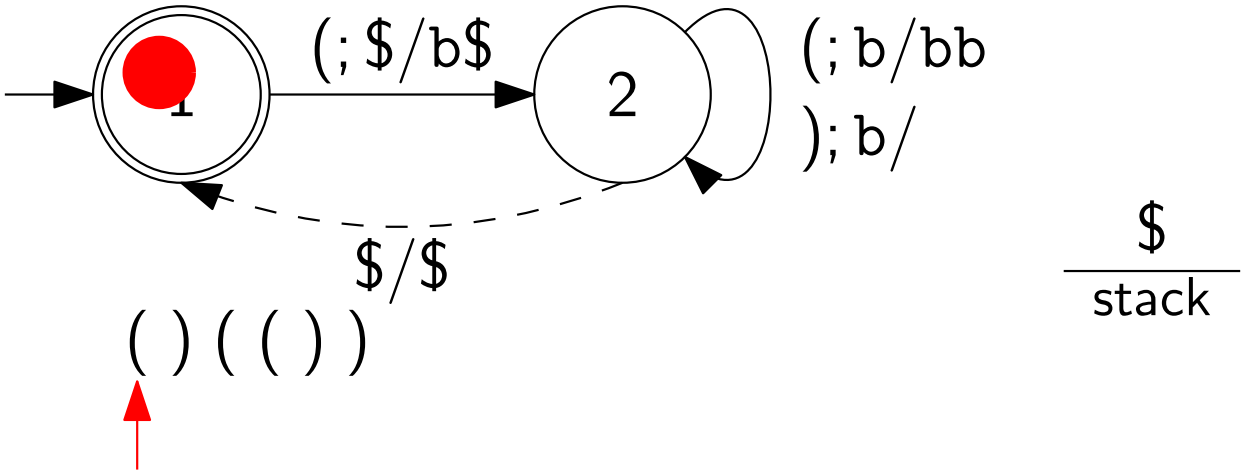
棄却する例



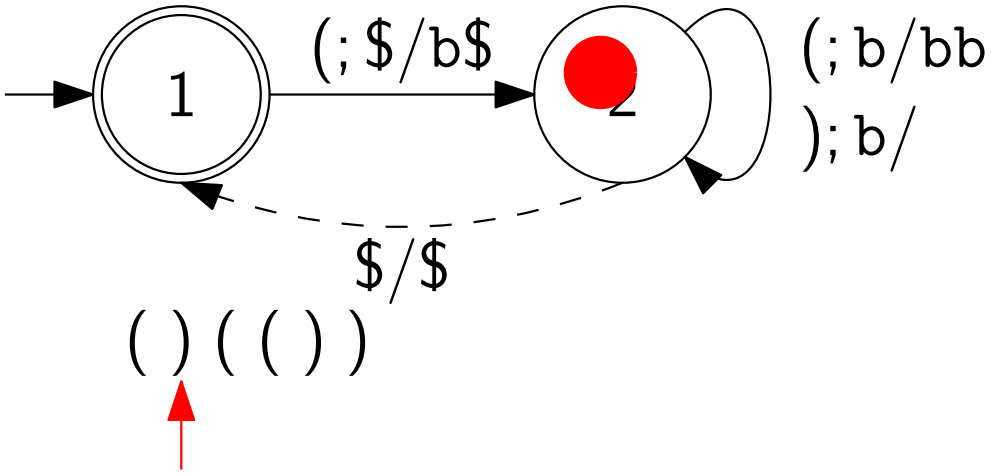
棄却する例



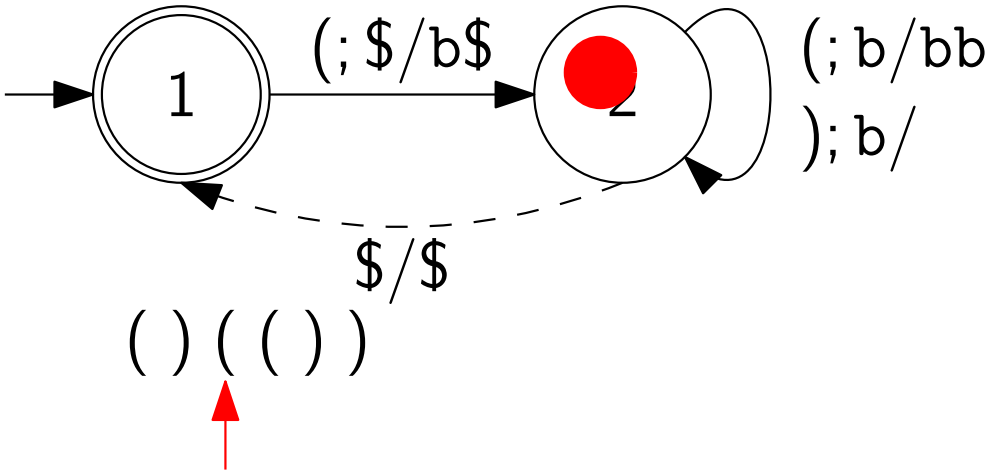
受理する例



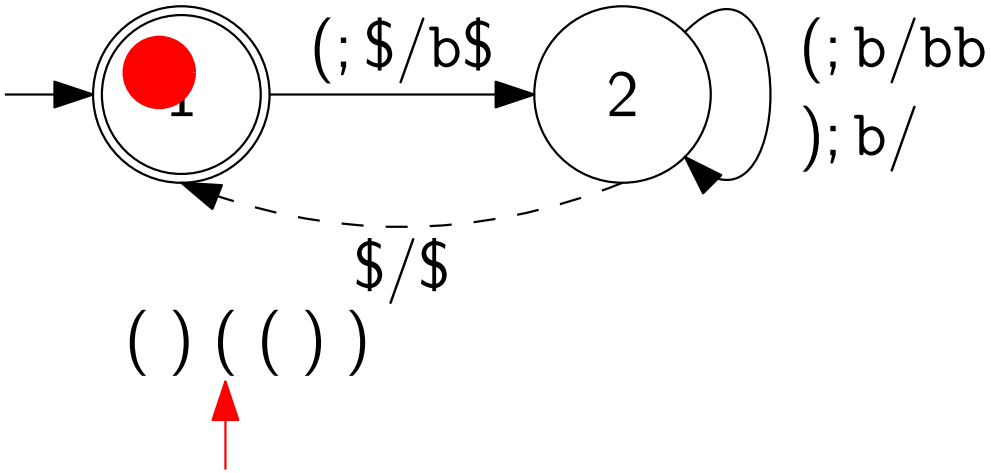
受理する例



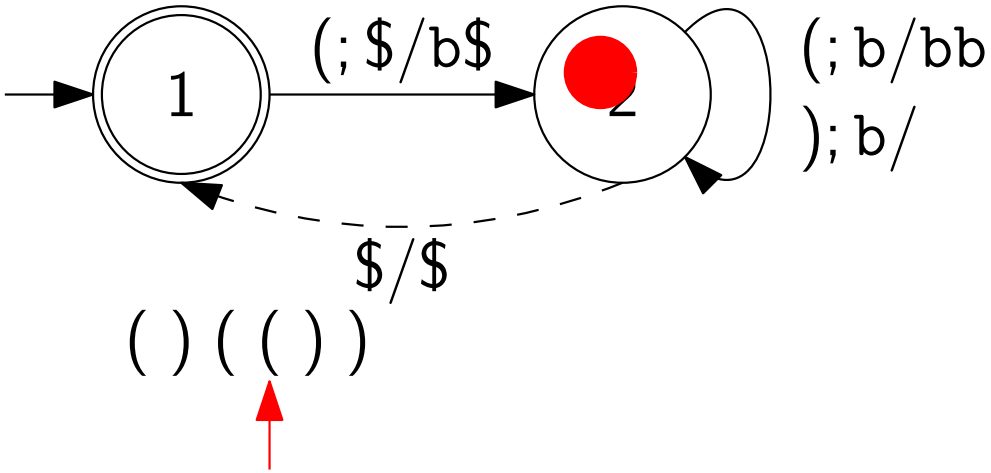
受理する例



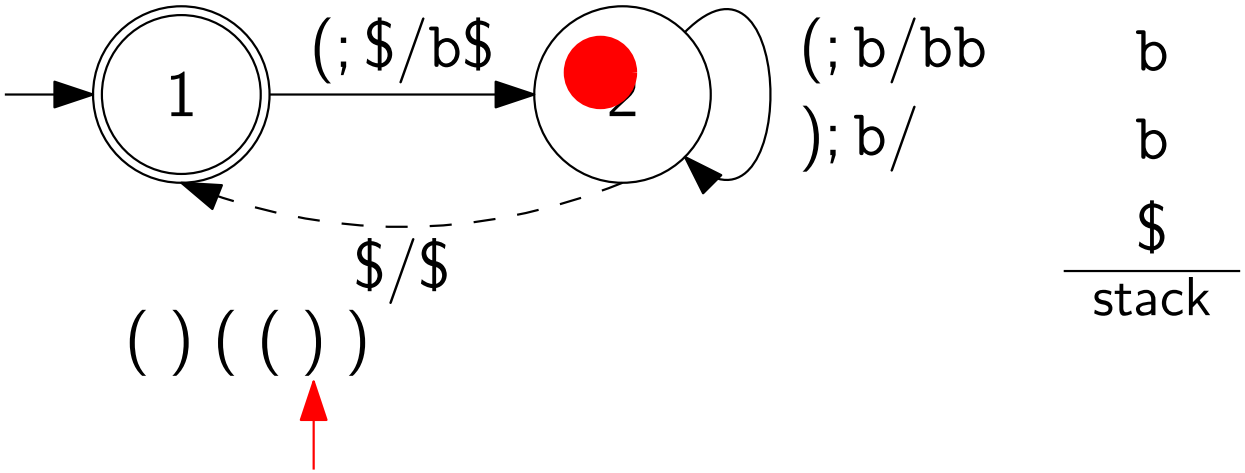
受理する例



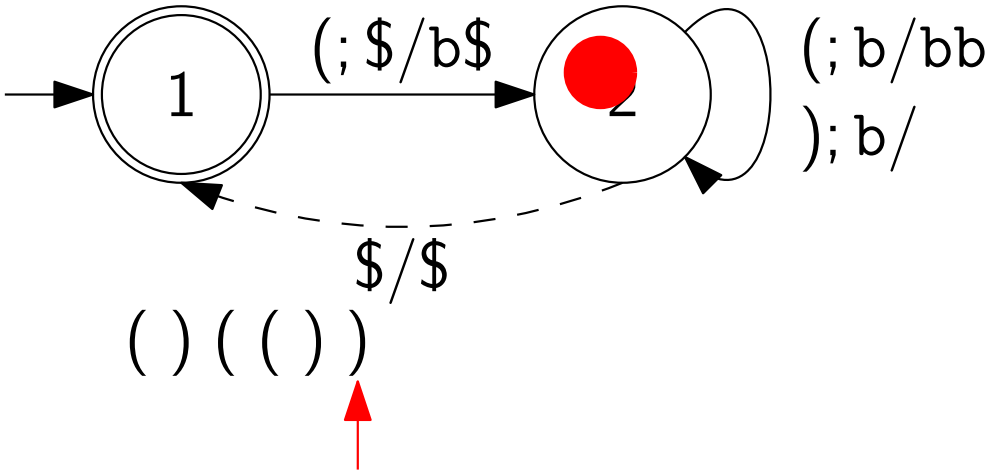
受理する例



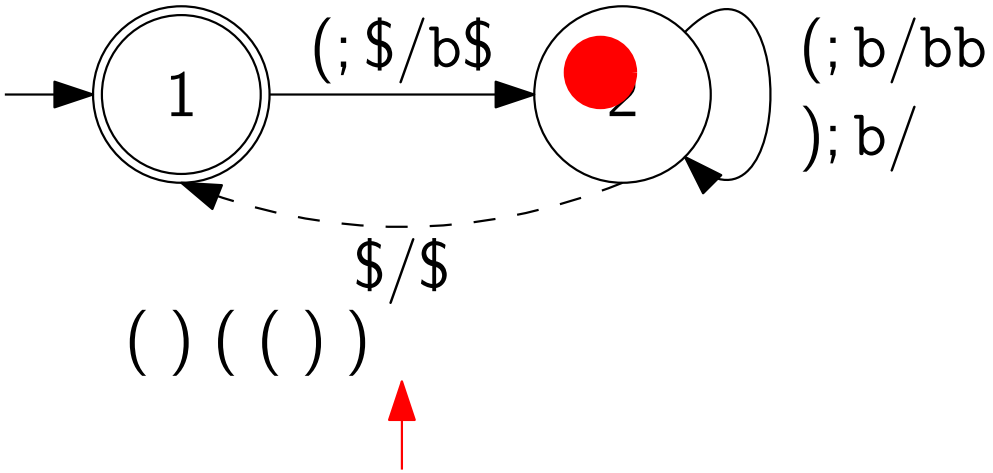
受理する例



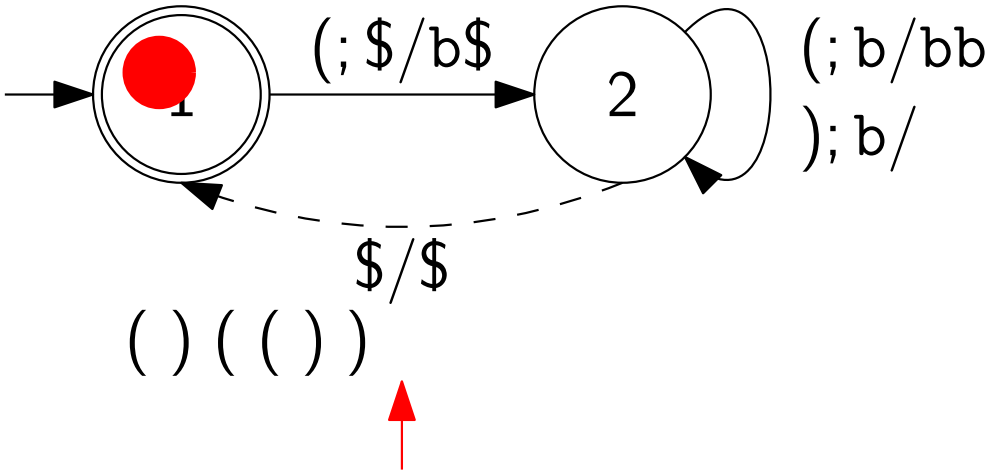
受理する例



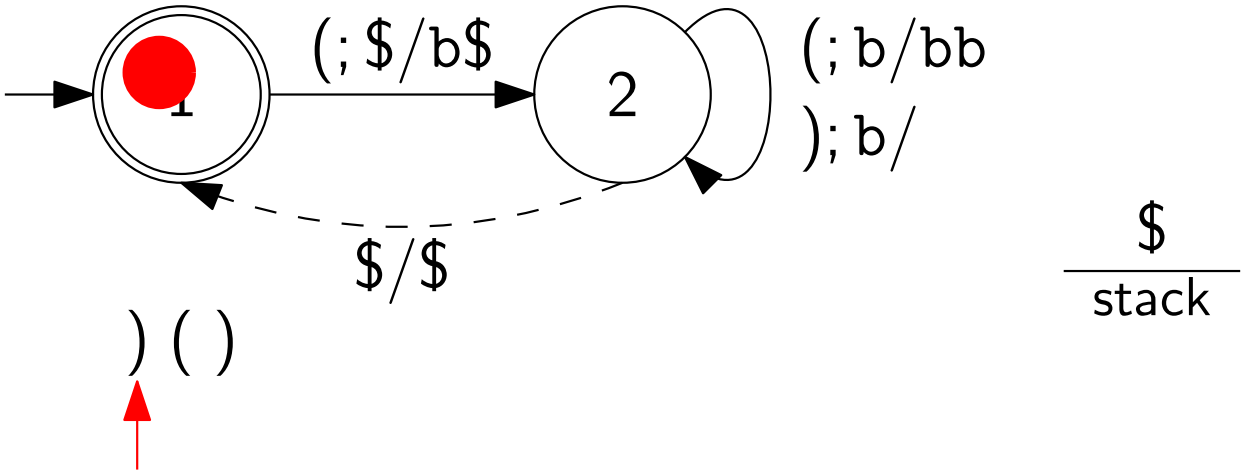
受理する例



受理する例

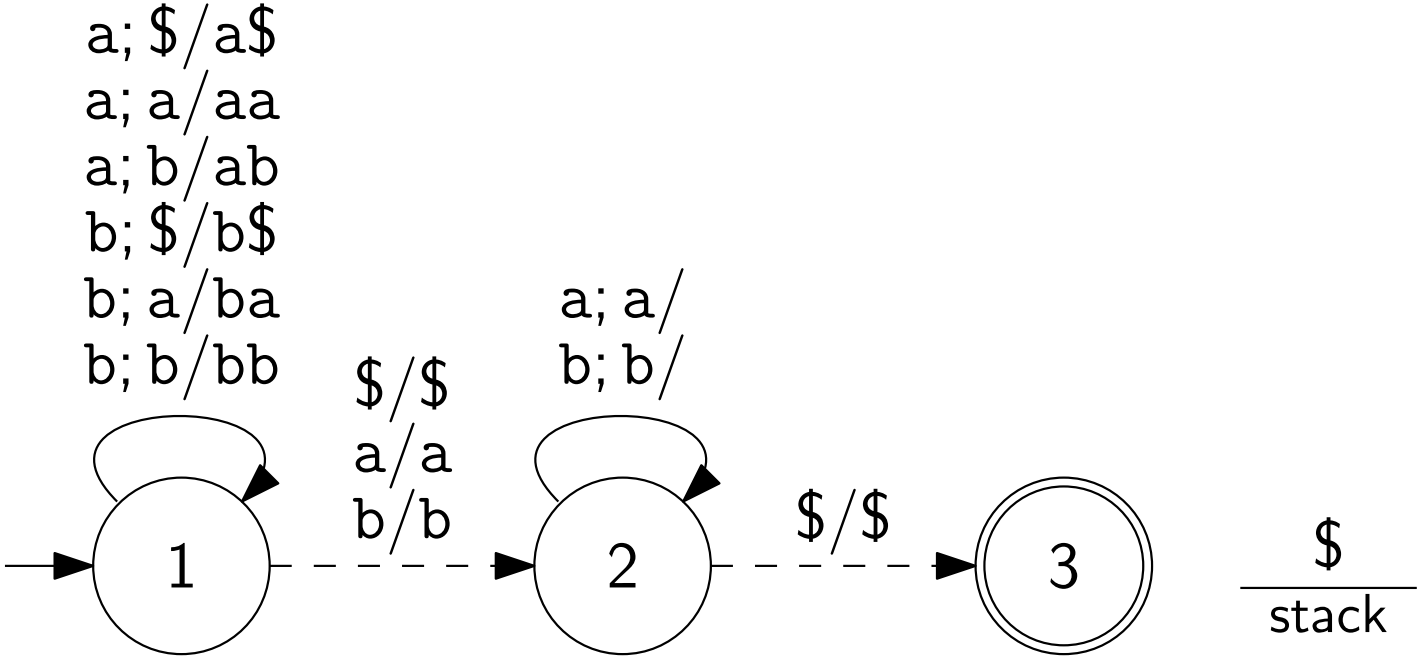


棄却する例

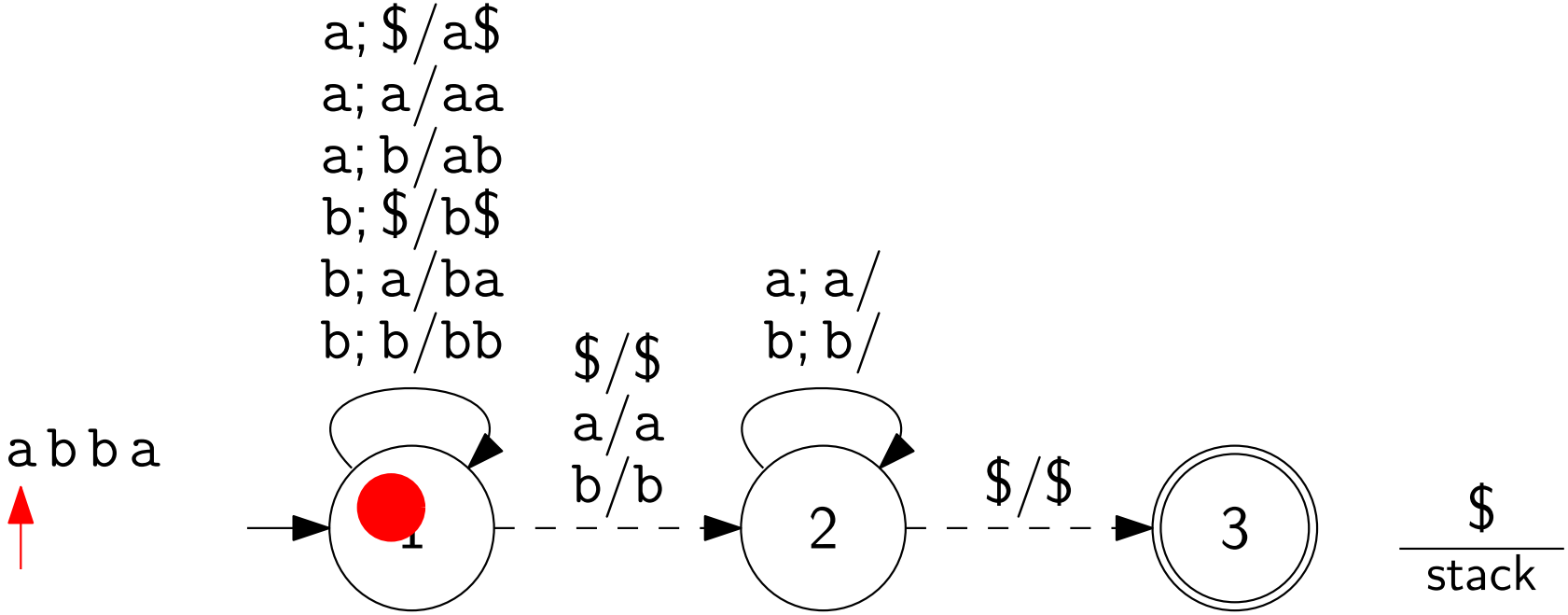


目次

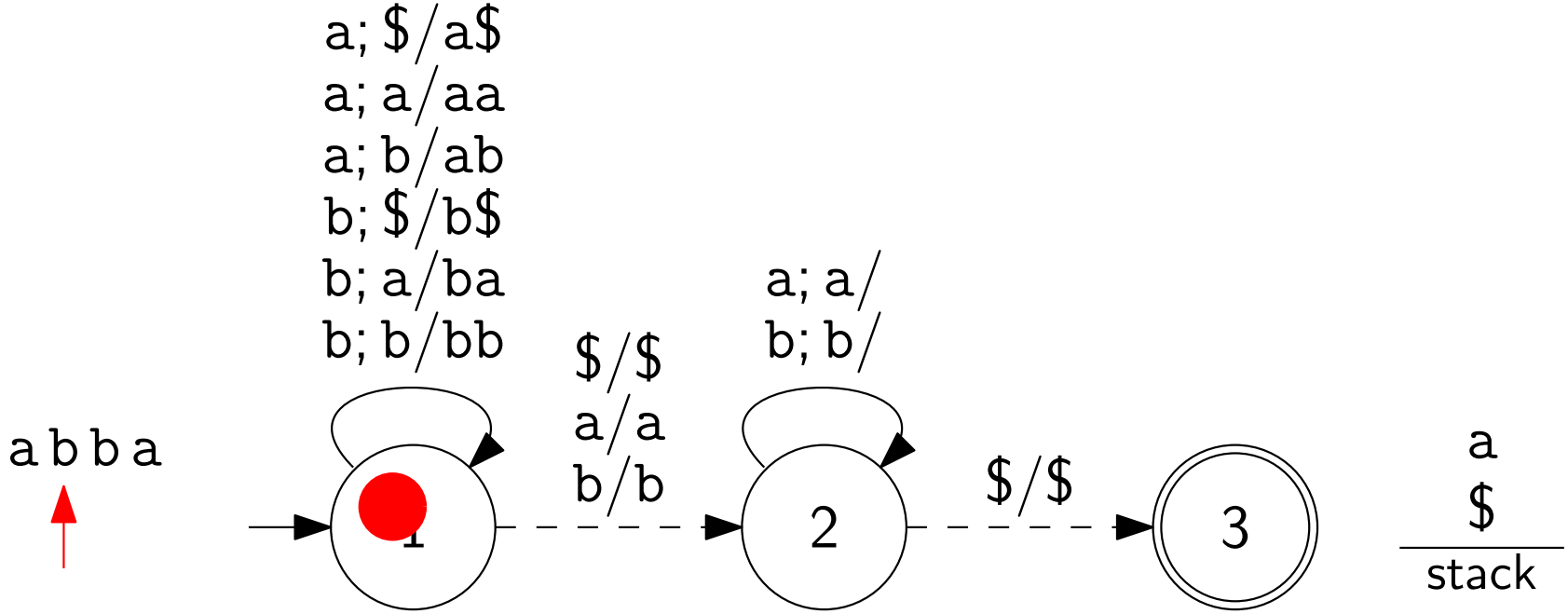
非決定性プッシュダウン・オートマトン



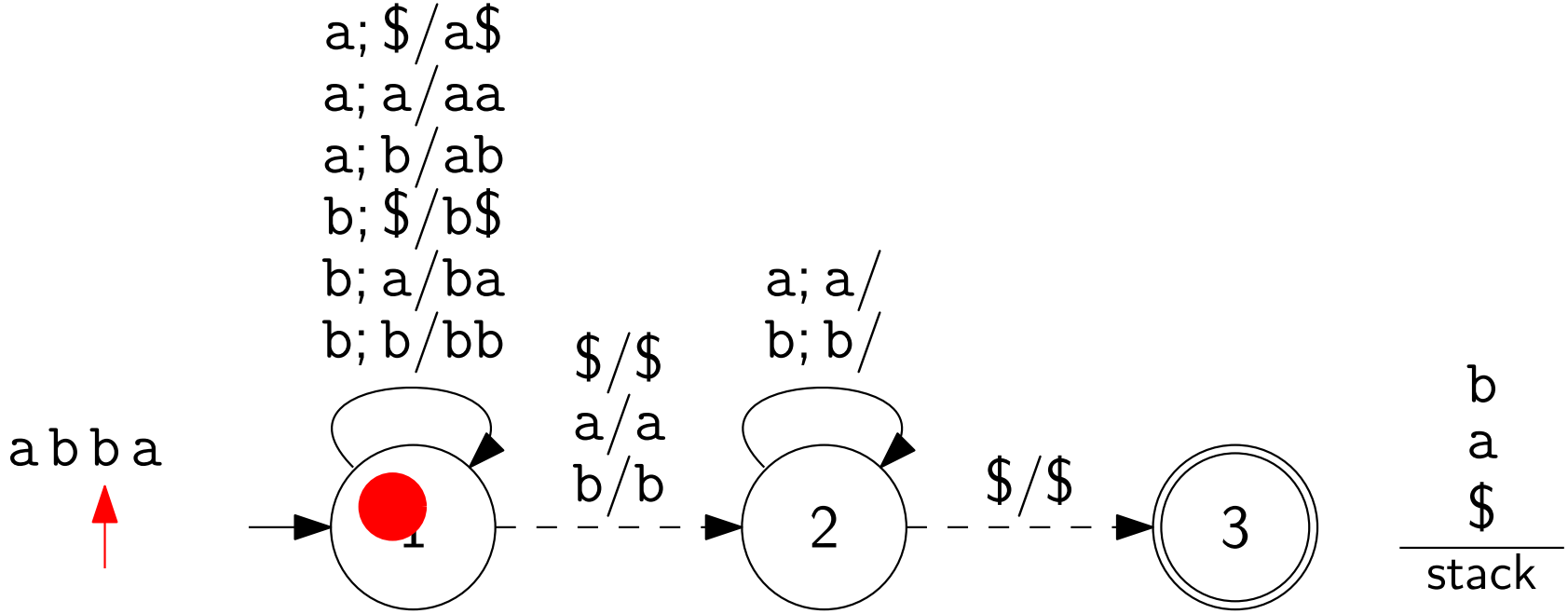
受理する例



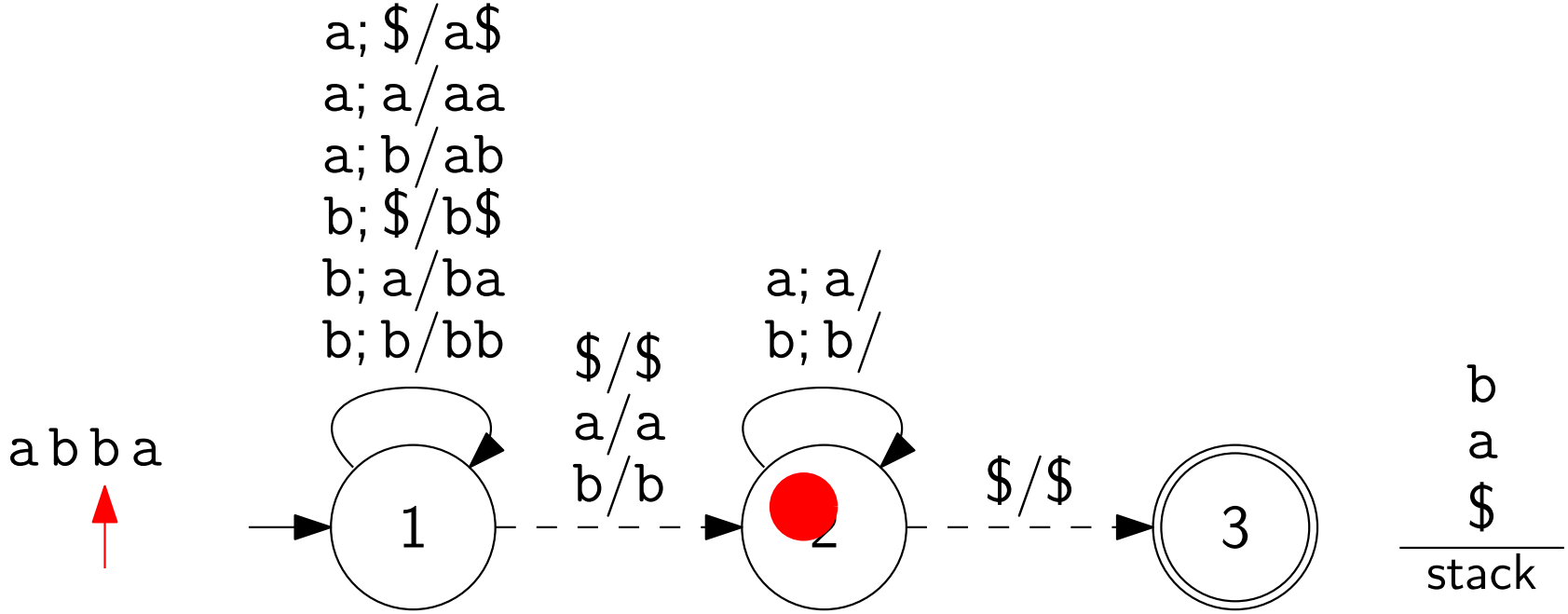
受理する例



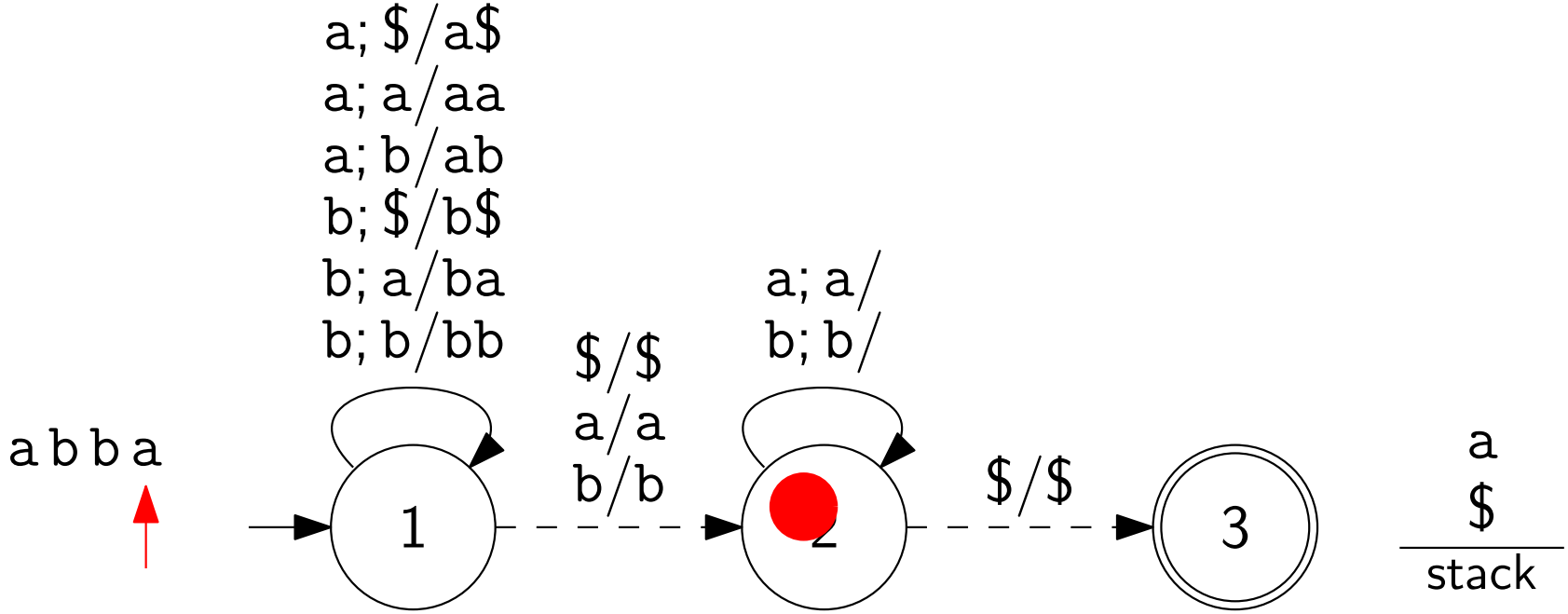
受理する例



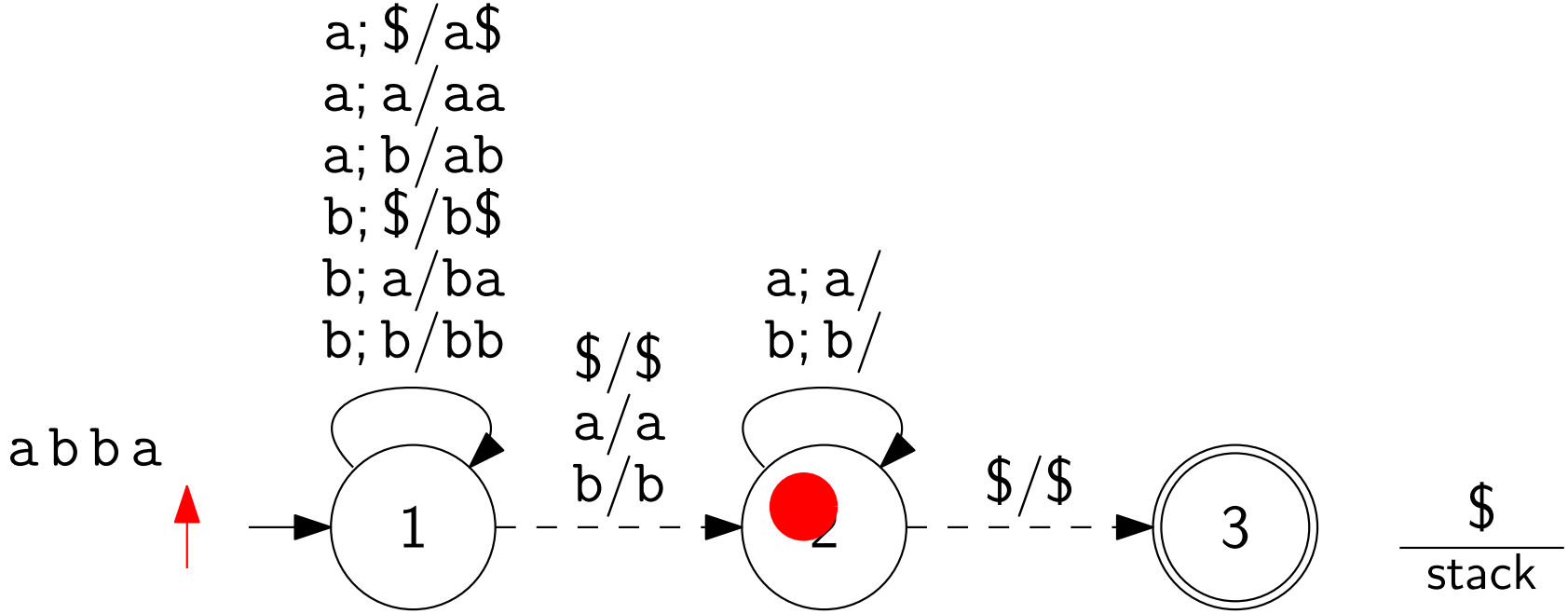
受理する例



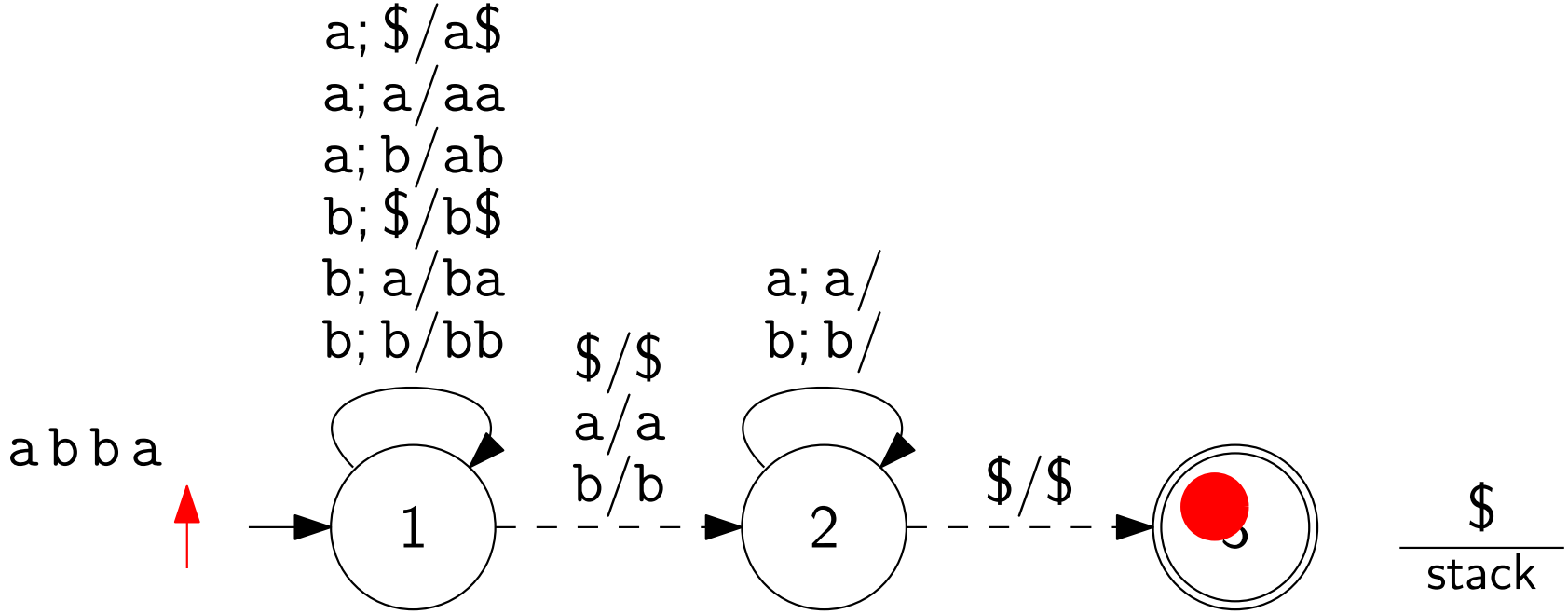
受理する例



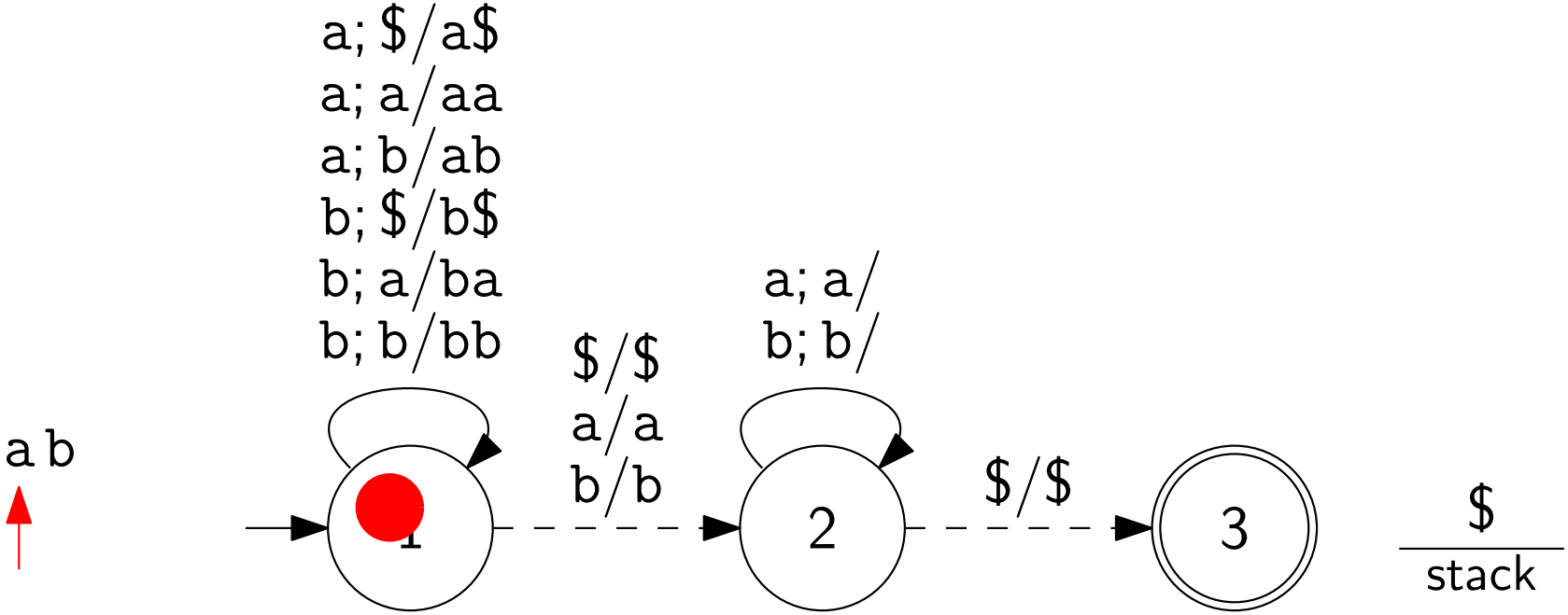
受理する例



受理する例



棄却する例



$A \subseteq B$　「A でできることは B でもできる」
$A \neq B$　「A でできることと B でできることは異なる」

計算能力の比較

$$\mathrm{DFA} = \mathrm{NFA} \subsetneq \mathrm{DPDA} \subsetneq \mathrm{NPDA}$$

- DPDA $\setminus$ NFA の例：バランスのとれた括弧列
- NPDA $\setminus$ DPDA の例：回文

注意

単に「プッシュダウン・オートマトン」と言ったら
「非決定性プッシュダウン・オートマトン」を指す

目次

2 つの部分に分かれる

字句解析 (lexical analysis)

- 「文字列」を「トークン列」に変換

正規表現を使って (つまり，有限オートマトンを使って) 実装

構文解析 (syntactic analysis)

- 「トークン列」が有効なプログラムを表現しているか判断

文脈自由文法を使って (つまり，PDA を使って) 実装

自然言語との対応

プログラミング言語	自然言語
プログラム	文章
文字列	文
トークン	品詞
有効なプログラム	文法に従った文章

SIMPLE の文法 (の一部)

```
<statement>  ::= <while> | <assign>
<while>      ::= 'w' '(' <expression> ')' '{' <statement> '}'
<assign>     ::= 'v' '=' <expression>
<expression> ::= <less-than>
<less-than>  ::= <multiply> '<' <less-than> | <multiply>
<multiply>   ::= <term> '*' <multiply> | <term>
<term>       ::= 'n' | 'v'
```

字句解析・構文解析の例

```
while (x < 3)  x = x * 4
```

↓字句解析↓

```
w ( v < n )  v = v * n
```

↓構文解析↓

```
w   (   v   <   n   )   {   v   =   v   *   n   }
```

字句解析・構文解析の例

```
while (x < 3)  x = x * 4
```

↓字句解析↓

```
w ( v < n )  v = v * n
```

↓構文解析↓

```
w   (   v    <   n    )   {   v   =   v    *   n    }
          <term>     <term>          <term>     <term>
```

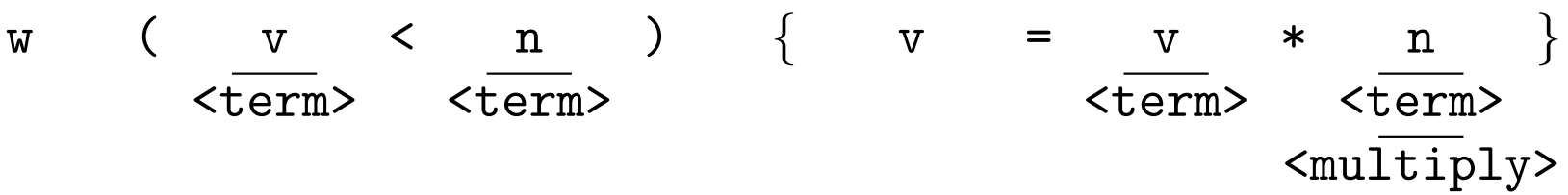
字句解析・構文解析の例

```
while (x < 3)  x = x * 4
```

↓字句解析↓

```
w ( v < n )  v = v * n
```

↓構文解析↓



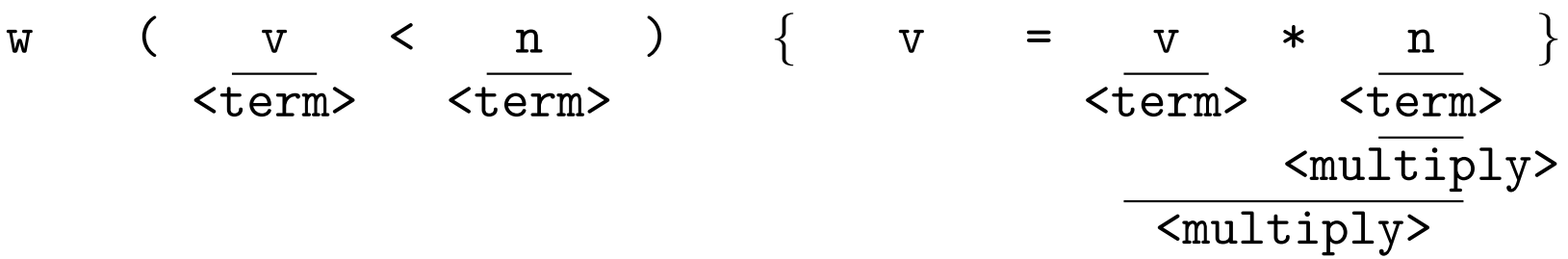
字句解析・構文解析の例

```
while (x < 3)  x = x * 4
```

↓字句解析↓

```
w ( v < n )  v = v * n
```

↓構文解析↓



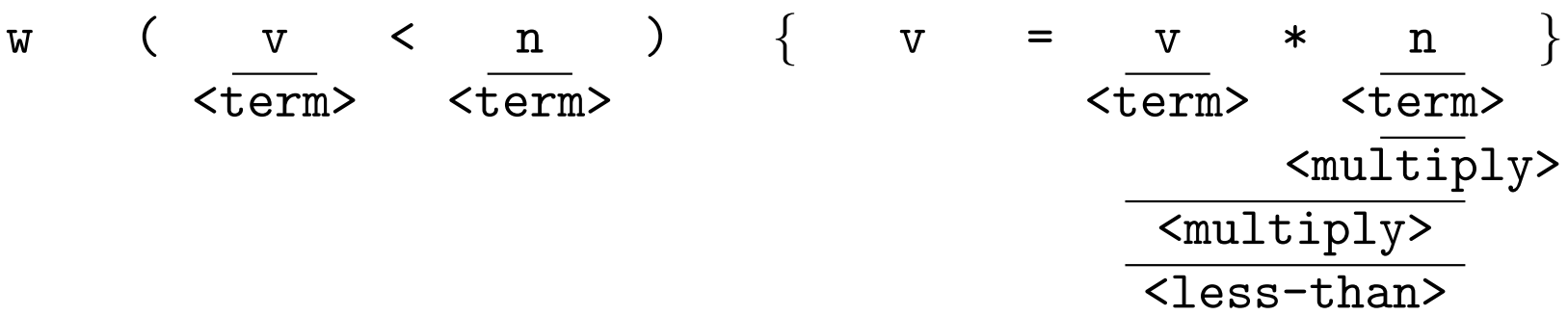
字句解析・構文解析の例

```
while (x < 3)  x = x * 4
```

↓字句解析↓

```
w ( v < n )  v = v * n
```

↓構文解析↓



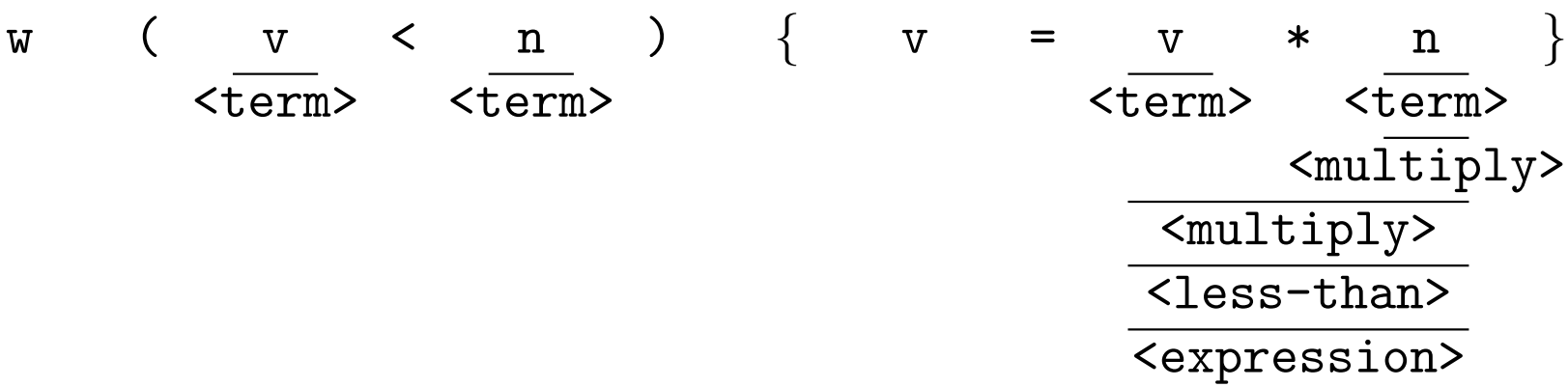
字句解析・構文解析の例

```
while (x < 3)  x = x * 4
```

↓字句解析↓

```
w ( v < n )  v = v * n
```

↓構文解析↓



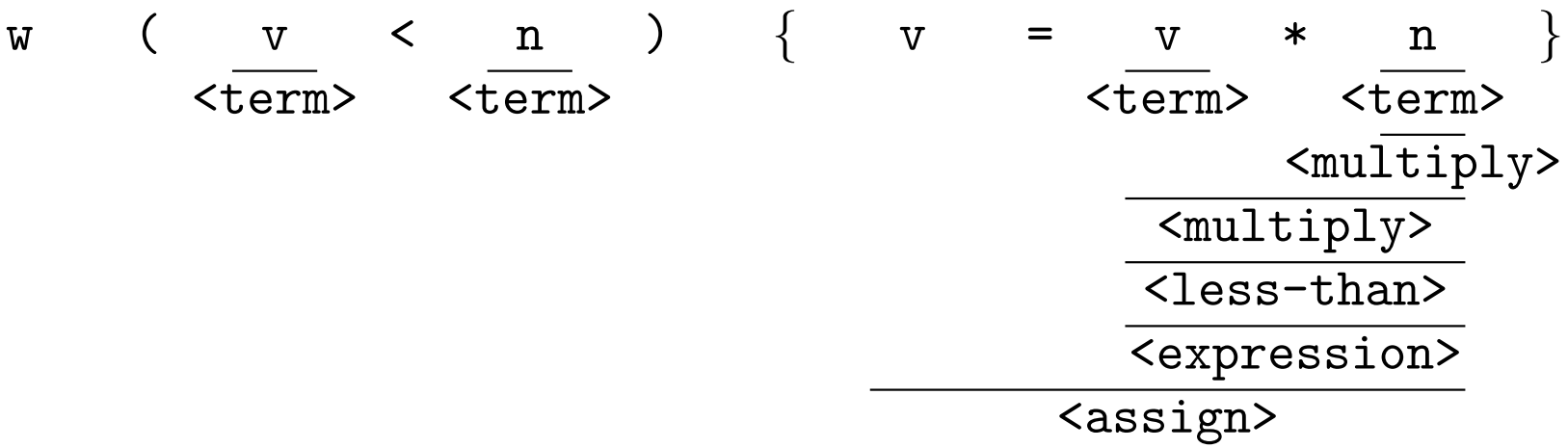
字句解析・構文解析の例

```
while (x < 3)  x = x * 4
```

↓字句解析↓

```
w ( v < n )  v = v * n
```

↓構文解析↓



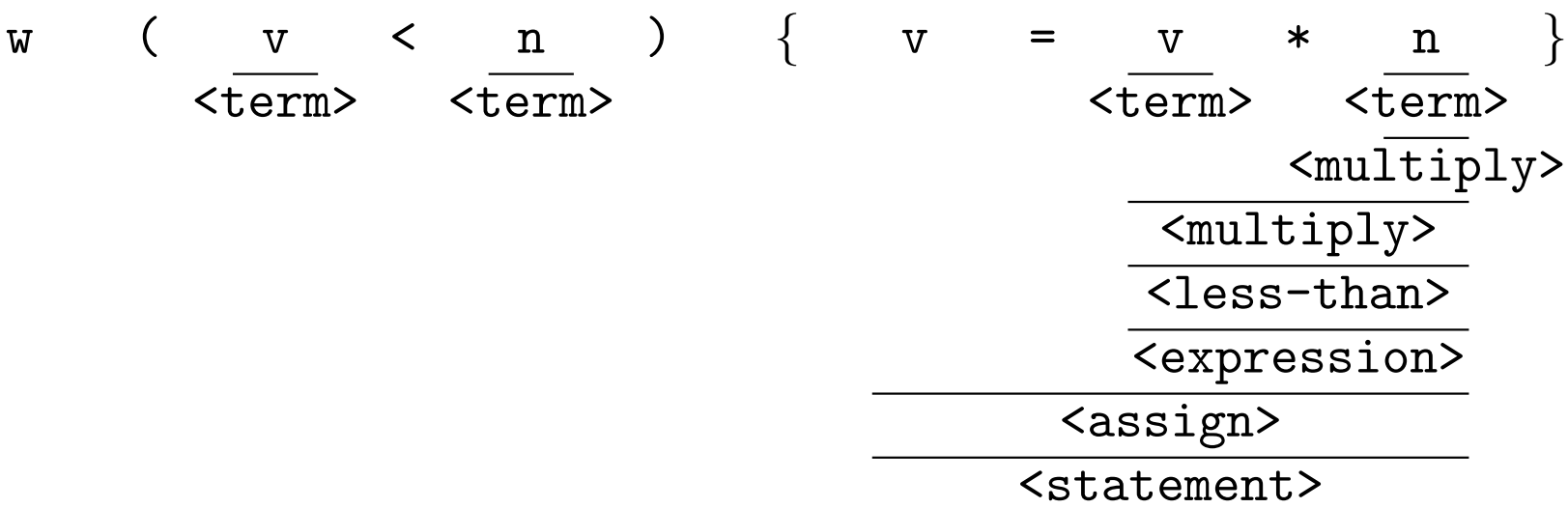
字句解析・構文解析の例

```
while (x < 3)  x = x * 4
```

↓字句解析↓

```
w ( v < n )  v = v * n
```

↓構文解析↓



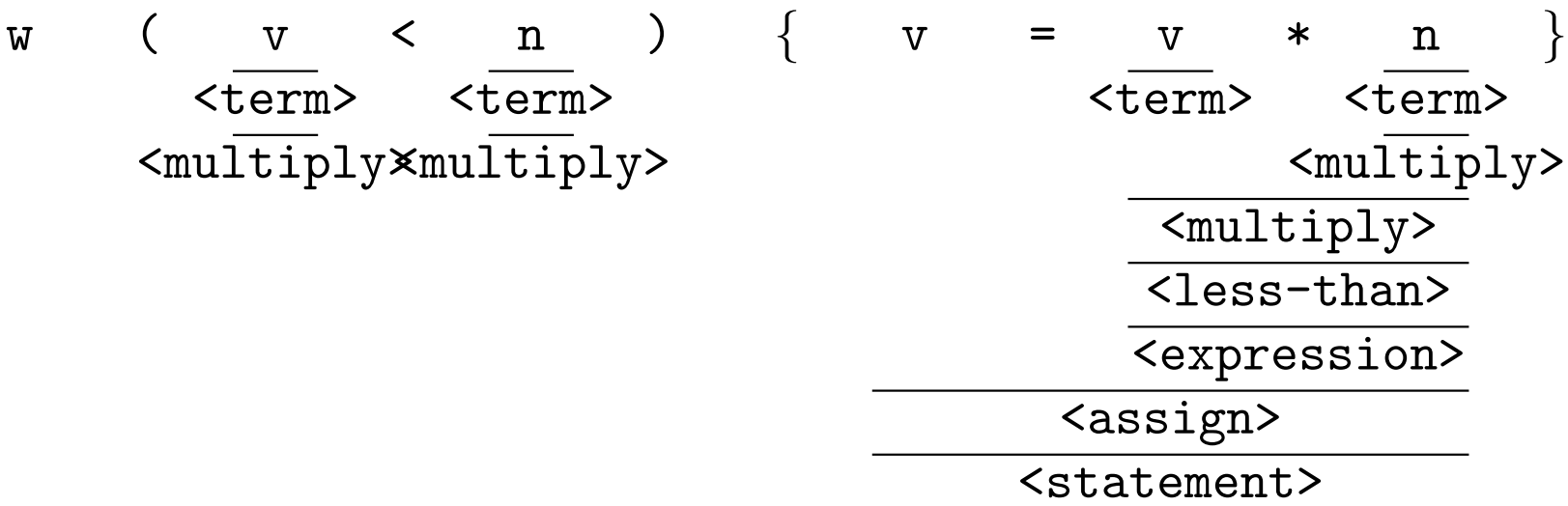
字句解析・構文解析の例

```
while (x < 3)  x = x * 4
```

↓字句解析↓

```
w ( v < n )  v = v * n
```

↓構文解析↓



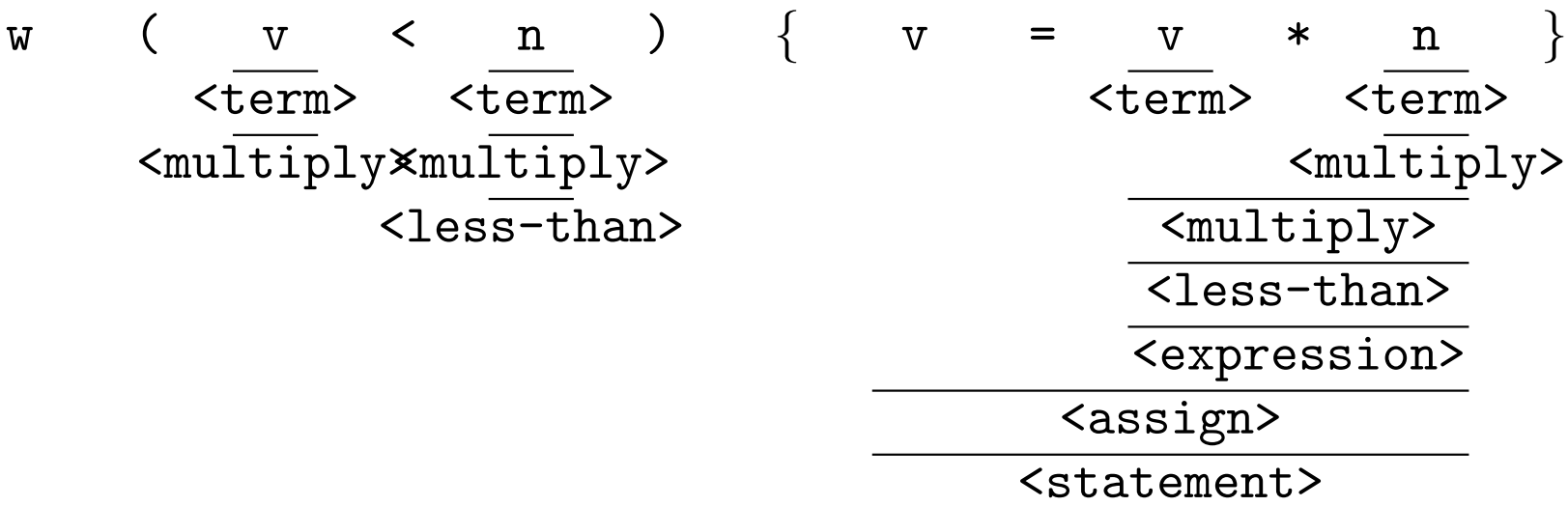
字句解析・構文解析の例

```
while (x < 3)  x = x * 4
```

↓字句解析↓

```
w ( v < n )  v = v * n
```

↓構文解析↓



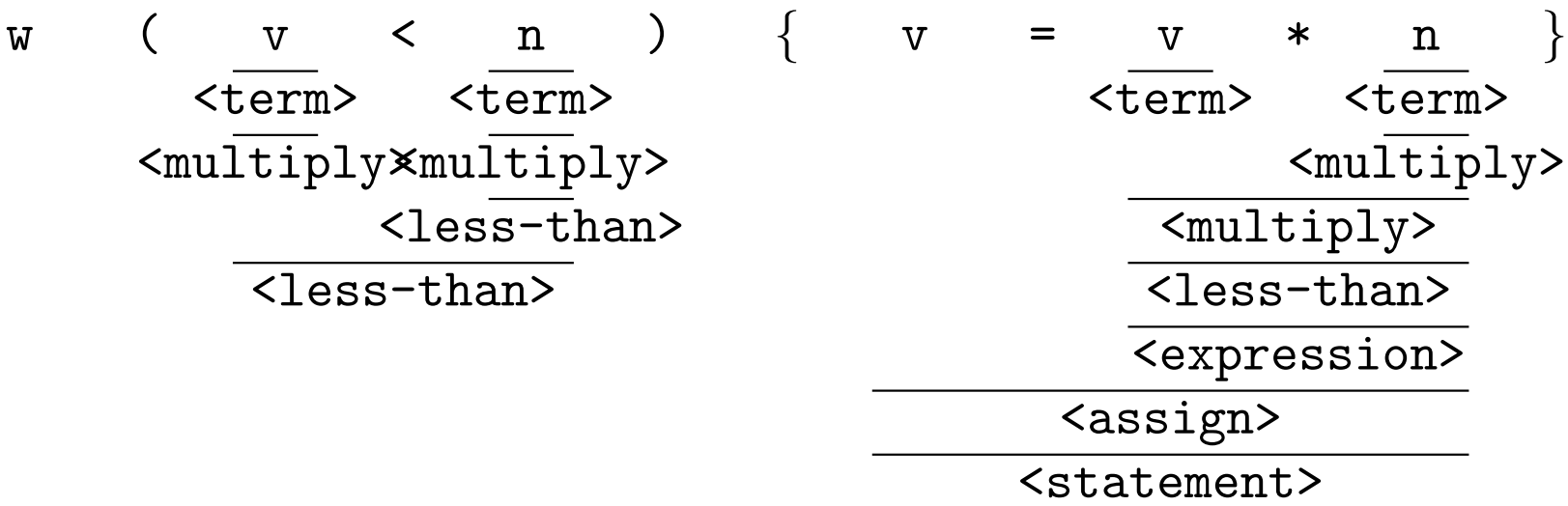
字句解析・構文解析の例

```
while (x < 3)  x = x * 4
```

↓字句解析↓

```
w ( v < n )  v = v * n
```

↓構文解析↓



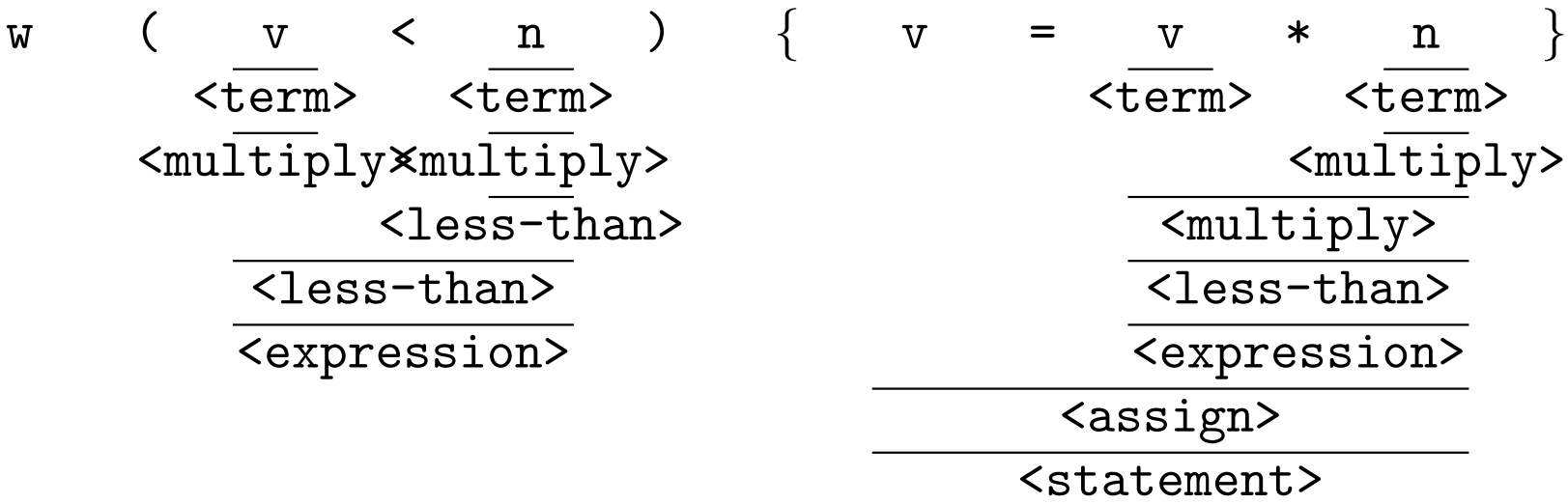
字句解析・構文解析の例

```
while (x < 3)  x = x * 4
```

↓字句解析↓

```
w ( v < n )  v = v * n
```

↓構文解析↓



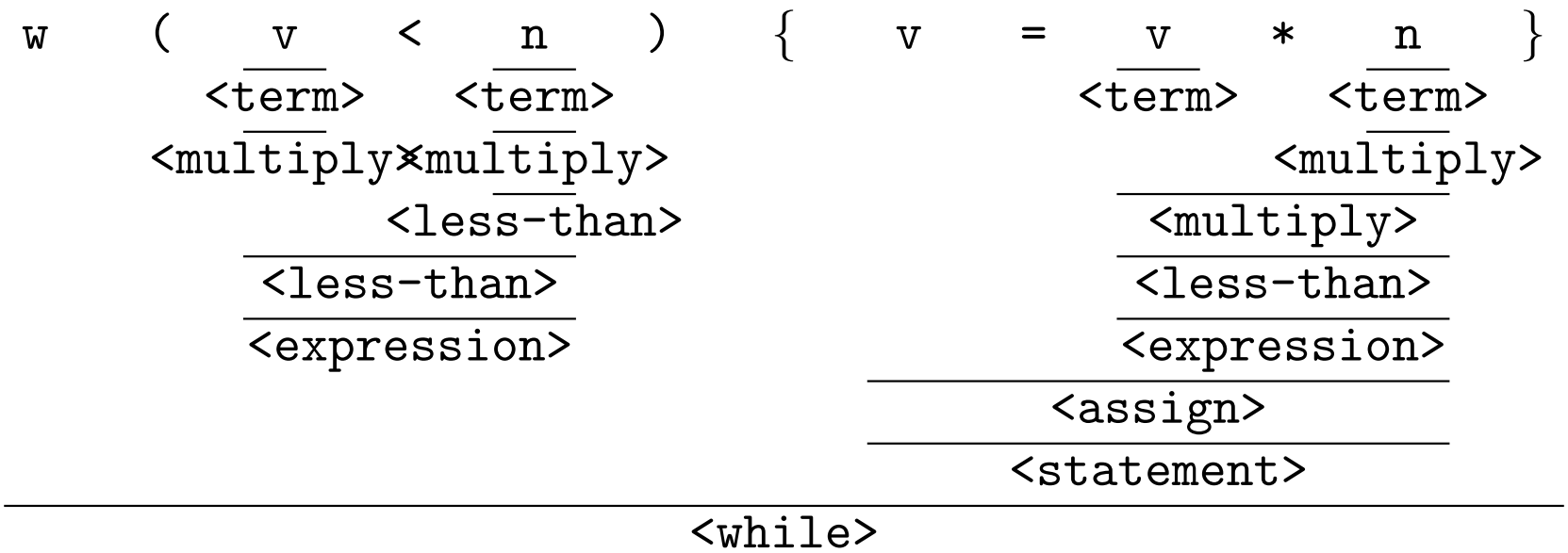
字句解析・構文解析の例

```
while (x < 3)  x = x * 4
```

↓字句解析↓

```
w ( v < n )  v = v * n
```

↓構文解析↓



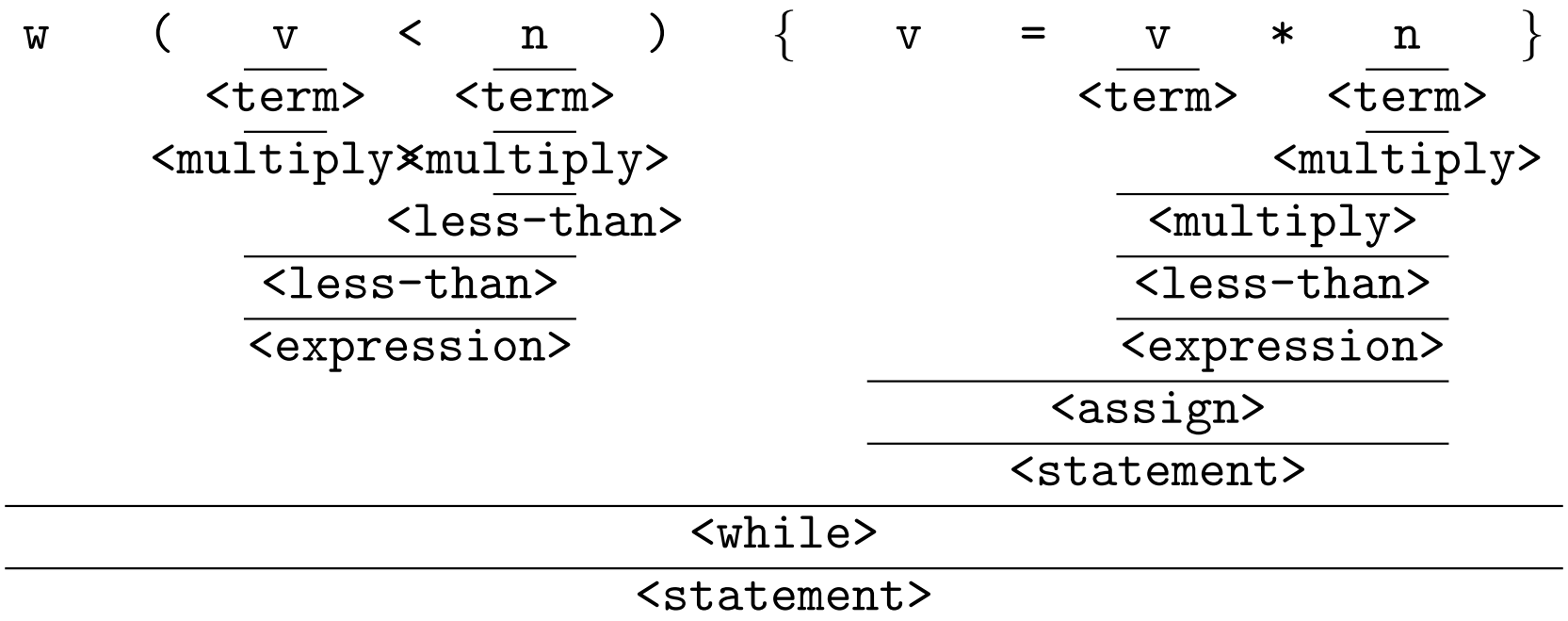
字句解析・構文解析の例

```
while (x < 3)  x = x * 4
```

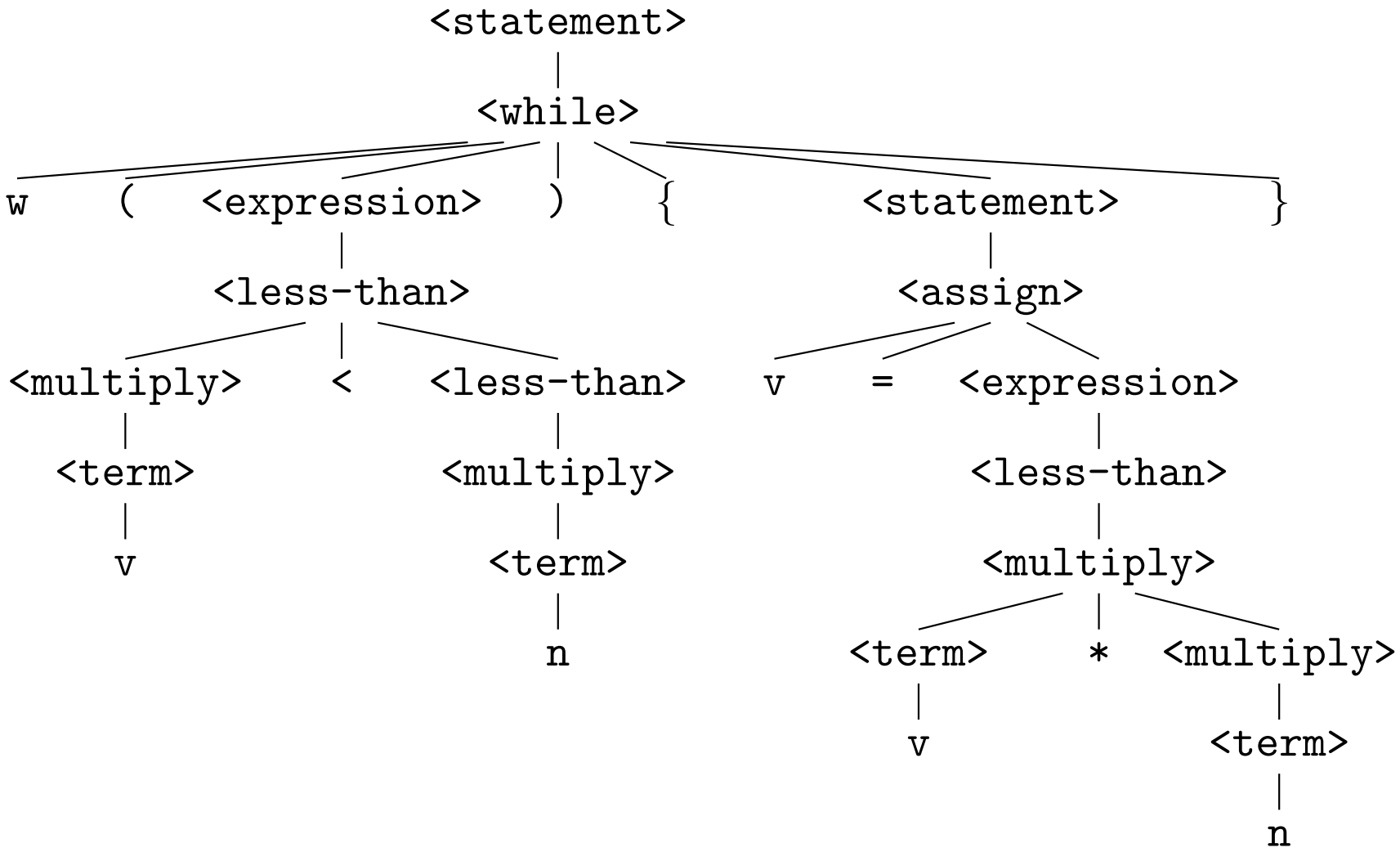
↓字句解析↓

```
w ( v < n )  v = v * n
```

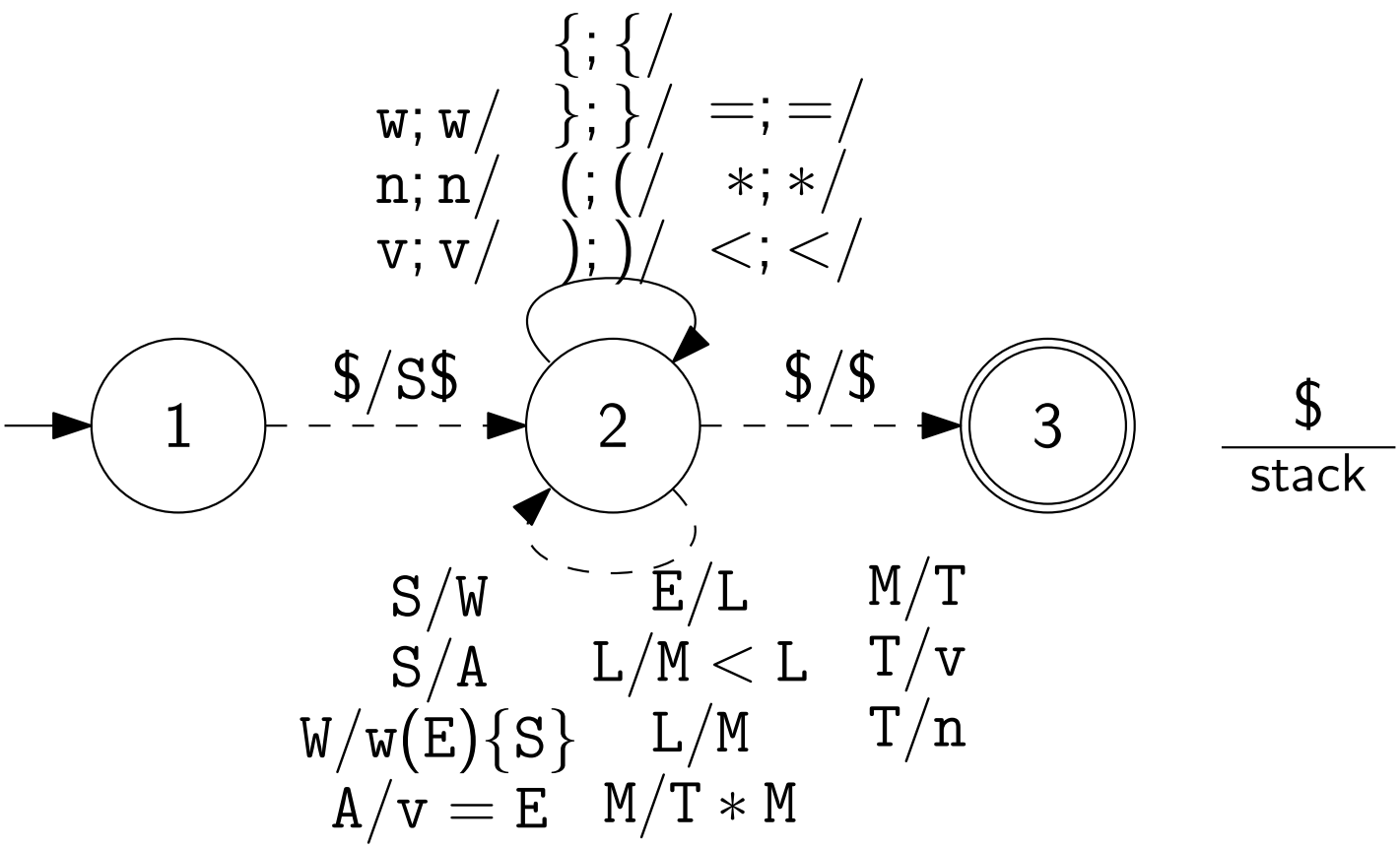
↓構文解析↓



構文解析の結果として得られる木構造



構文解析を非決定性プッシュダウン・オートマトンで行う



「w (v < n) v = v * n 」の構文解析に必要な部分だけ

目次

個人プロジェクト案の例

- クラス DPDA，NPDA にメソッドを追加する
  - 文字列を入力して，状態とスタックの変化を一文字ずつ追い，
    到達した状態とスタックの内容を
    画面に逐一出力するメソッド `trace(string)` を実装する
- 4.3.1 節「字句解析」を
  第 3 章で実装した正規表現を使って構築し直す
- 4.3.2 節「構文解析」を完成させる
- LL 法以外の構文解析アルゴリズムを調べて，実装する
  - 例えば，LR 法
- 「プッシュダウン・オートマトンの類似概念」について調べて，
  実装してみる