50行のコードと遷移図付きログで分かるLR1構文解析

h_sakurai

要約

- LR構文解析は理解するのが難しいアルゴリズムです。
- しかしながら、動く短いソースがあれば理解の助けになるはずです。
- そこで、50行弱の小さなLR構文解析のプログラムを読みます。
- その後、構文解析表から状態遷移図生成するプログラムを作ります。
- さらに、パーサにログ機能を付けて(160行強)動きを見ます。

1. とにかくソースを見るぞゴルァ

プログラムは大きく5つに別れます:

1. 文法定義
   - 字句定義の lex
   - 構文定義の grammar
2. 構文解析表 table
3. 字句解析器 lexer
4. 構文解析器 parser
5. メイン処理

1.1. 文法定義

```
(* usage : ocaml -w -8-40 str.cma parse.ml *)
type i = I of int | E
let lex =   [ "ws", "[ \\r\\n\\t]+", (fun i->E);
              "N",  "[1-9][0-9]*",   (fun i-> I(int_of_string i));
              "+",  "+",             (fun i->E);
              "*",  "*",             (fun i->E);                  ]
let grammar=[|"E",["E";"+";"T"],(fun[I a;_;I b]->I(a+b));(*s0*)
              "E",["T"],        (fun[a]    ->a);         (*s1*)
              "T",["T";"*";"N"],(fun[I a;_;I b]->I(a*b));(*s2*)
              "T",["N"],        (fun[a]    ->a);         (*s3*)|]
```

- type i はスタック内に異なるデータを含めるためのデータです。
- 字句の定義と、文法の定義をコールバック付きで定義できます。

1.2. 構文解析表

```
type op = Accept | Shift of int | Reduce of int | Goto of int
let table = [|
(*0*)["N",Shift 2;                                    "E",Goto 1;"T",Goto 3];
(*1*)[           "+",Shift  4;             "$",Accept                        ];
(*2*)[           "+",Reduce 3;"*",Reduce 3;"$",Reduce 3                      ];
(*3*)[           "+",Reduce 1;"*",Shift  5;"$",Reduce 1                      ];
(*4*)["N",Shift 2;                                                "T",Goto 6];
(*5*)["N",Shift 7                                                           ];
(*6*)[           "+",Reduce 0;"*",Shift  5;"$",Reduce 0                      ];
(*7*)[           "+",Reduce 2;"*",Reduce 2;"$",Reduce 2                      ]|]
```

- LR構文解析には構文解析表が別途必要になります。
- 通常構文解析表はYaccなどのコンパイラコンパイラで自動生成します。
- 今回は、自前で用意しました。
- op は構文解析器の命令、完了、シフト、還元、Gotoをそれぞれ表します。

1.3. 字句解析器

```
let rec lexer = function "" -> ["$",E] | input ->
  let rule,_,callback = List.find(fun(_,p,f)->
    Str.string_match(Str.regexp p) input 0
  ) lex in
  match Str.string_after input (Str.match_end()),rule with
  | next,"ws" -> lexer next
  | next,rule -> (rule, callback (Str.matched_string input))::lexer next

let rec pop = function | 0,acc,s -> acc,s | n,acc,a::s -> pop (n-1,a::acc,s)
```

- 字句解析器は文字列を受け取り、字句リストを返します
- 文字列を受け取ってlexにstring_matchでマッチするものを見つけます
- match_endでマッチ文字列以降を取り出し
- ルール名が"ws"なら空白なので飛ばして再帰呼び出し
- それ以外はmatched_stringでマッチ文字列を取得後コールバック関数を読んでリストに加えます。
- pop はリストをn番目で分割し手前と後ろのリストのペアを返します

1.4. 構文解析器

```
let rec parser((t,v)::inputs,s::status,results) =
  match List.assoc t table.(s),(t,v)::inputs,s::status,results with
  | Accept  ,            _,status,r::_   -> r
  | Shift  s,(_,v)::inputs,status,results-> parser(inputs,s::status,v::results)
  | Goto   s,(_,_)::inputs,status,results-> parser(inputs,s::status,   results)
  | Reduce g,       inputs,status,results->
    let t,args,callback = grammar.(g) in let len = List.length args in
    let _,status = pop(len,[],status) in let rs,results=pop(len,[],results) in
    parser((t,E)::inputs,status,callback rs::results)
```

- parserは入力リスト、状態スタック、結果スタックを受け取り、構文解析表から命令を取り出して実行します。ReduceとGotoはペアの命令です。
- Accept は解析完了を意味し、結果スタックから値を返します。
- Shift は入力を結果スタックに積み引数Sを状態スタックに積みます。
- GotoはReduceの次に現れ入力を捨てて引数sを状態スタックに積みます。
- Reduce は引数gの文法を取得しパターン長分Pop、コールバック結果を結果に積み、次の命令Gotoを呼ぶために文法名を入力にpush backします。

1.5. メインプログラム

```
let parse input = parser(lexer input,[0],[])
let show = function
  | I i -> Printf.sprintf "%d" i
  | E -> Printf.sprintf "error"
let _ = assert(parse"1+2"=I 3); assert(parse"1+2*3"=I 7);
        assert(parse"2*3+4"=I 10); assert(parse "1 + 2*3 + 4"=I 11);
        Printf.printf "%s\n" (show (parse "1 + 2*3 + 4"))
```

- parse は文字列を受け取ってparserに文法、構文解析表、字句解析結果と初期のステータススタック、結果スタックを渡して呼び出す関数です。
- 使い方は assert(parse”2*3+4”=I 10); のように使います。
- showは結果の印字処理で、最終行では結果を印字しています。

1.6 まとめ

- LR構文解析のプログラムを見てみました。
- 文法定義は字句定義の lex、構文定義の grammarがありLR構文解析では構文解析表 table が必要なので自前で用意しました。
- 字句解析器 lexer と 構文解析器 parser は短く定義できました。
- メイン処理では様々な入力に対してテストし、結果を表示してみました。
- LR構文解析のプログラム自体はとても簡単に書けました。

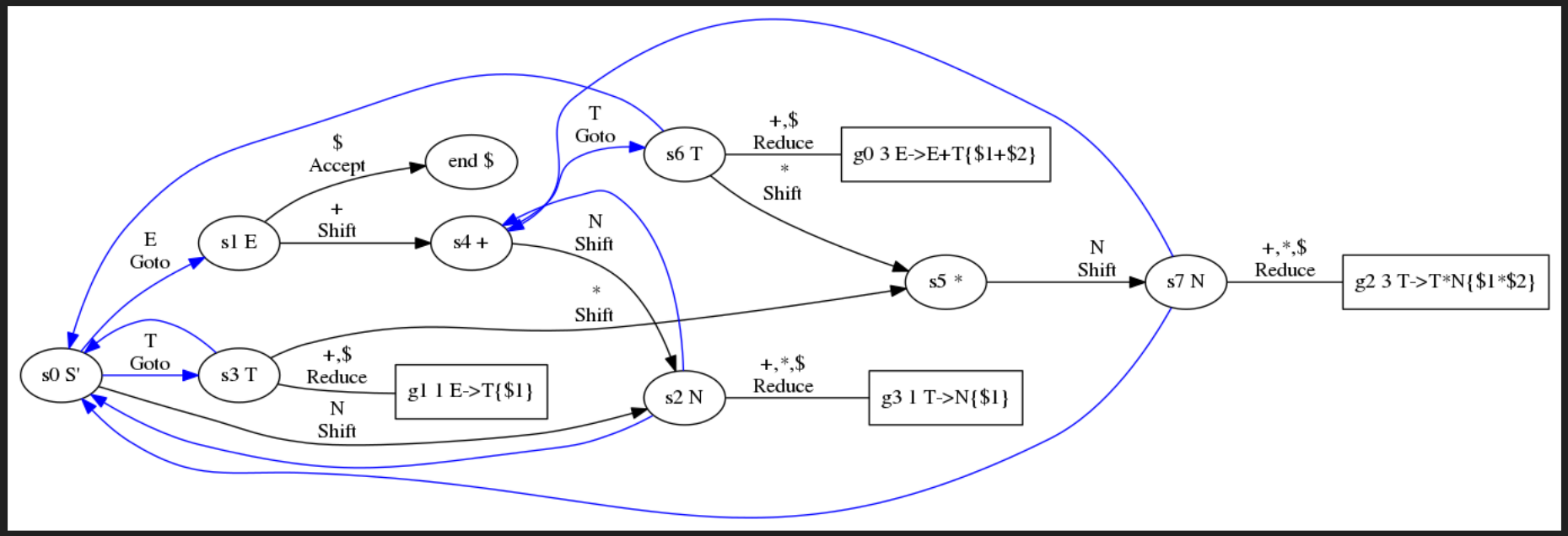
LR構文解析

- LR構文解析は上昇型の表を使って構文解析する手法です
- 仕組み自体はいまはわかりませんが、表さえあれば構文解析プログラム自体は簡単に記述できます

2. 遷移図付きログ

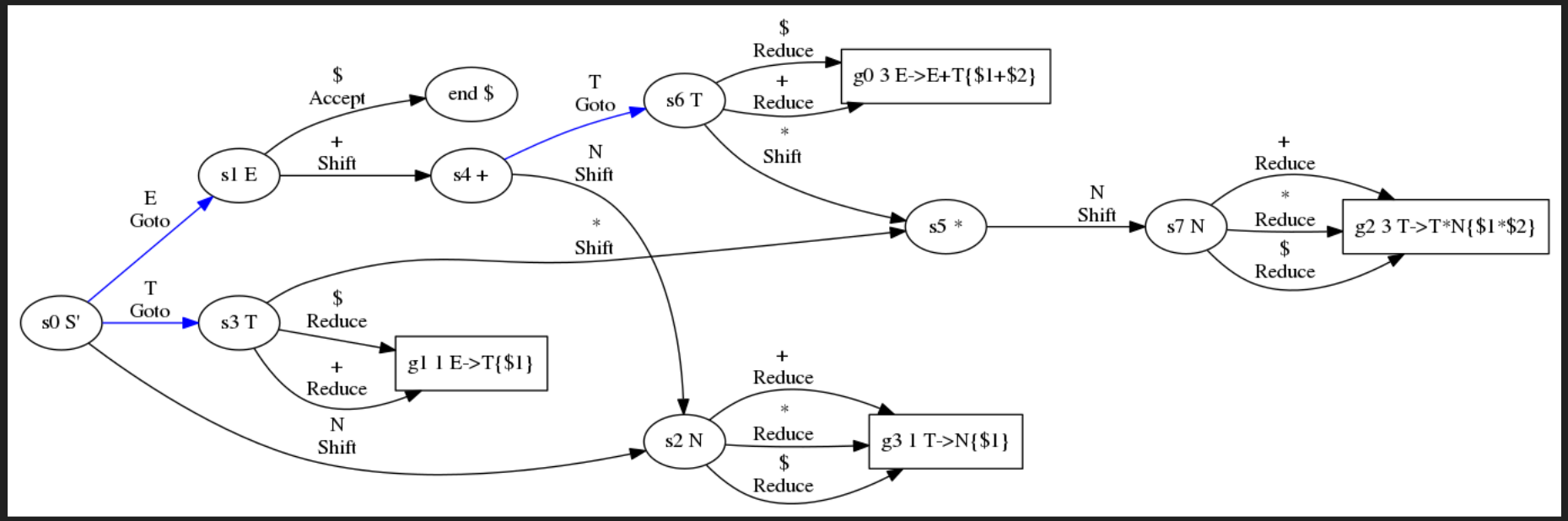
- 1章ではプログラムをみましたが、LR構文解析の説明は足りていないと思います。
- 2章ではLR構文解析の動きを状態遷移図付きのログを見ます。

2.1. 状態遷移図



- 遷移図は granphviz という dot コマンドにテキストファイルを渡せば遷移図を描いてくれるツールを用いて描きました。
- 1,1*2,1*2*3,1+1,1+1+1,1*2+1,1+2*3 の動きを考えてみるとよいはずです。
- Shift は移動し、Reduce文法を見て戻り、Gotoは文法で移動します。
- これを肴にすごろくゲームなどをしながらお酒を一杯飲めそうです。
- しかしながら、これだけ見て理解することはちょっと難しいでしょう。

2.2. 状態遷移図の自動生成



- 次に、50行程のプログラムを書いて自動生成してみました。
- このプログラム制作は、構文解析表の理解を深めるために役立ちました。

2.3. ロギング機能

- このプレゼン自体がMarkdownで書かれていますが、ログもmarkdownで作ればプログラムのログをプレゼンに使えます。
- 遷移図はなかなか分かりやすいので、構文解析プログラムを改造しロギング機能を付けてプレゼン資料を作ってみました。

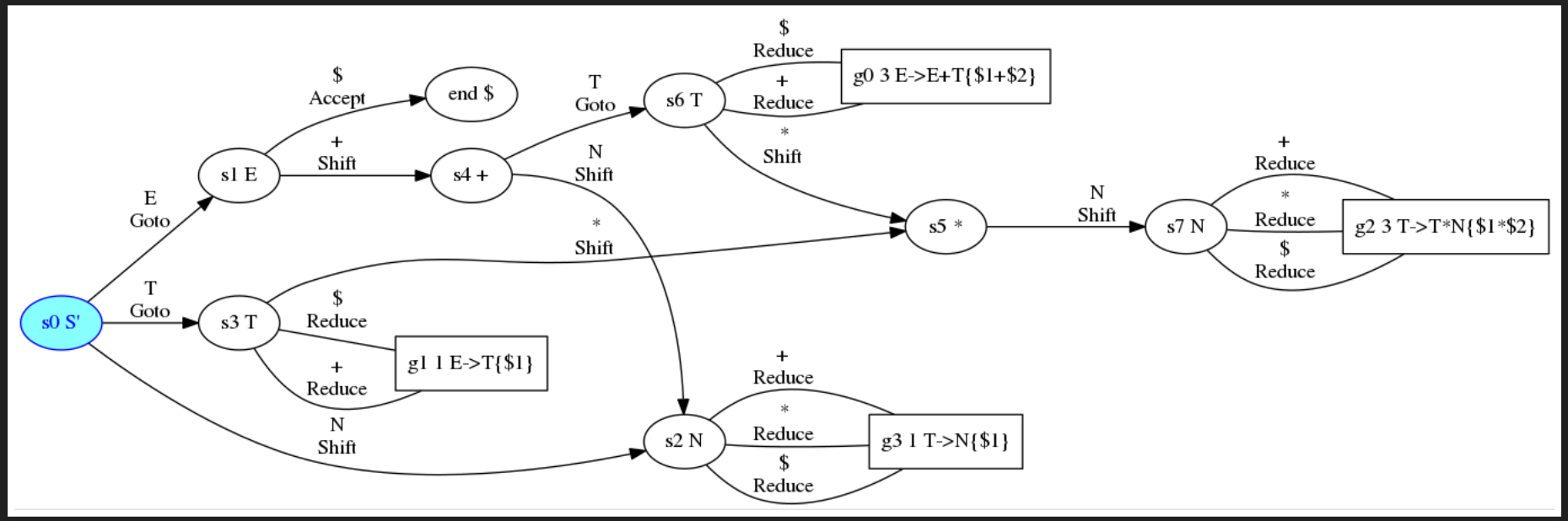
3. LR構文解析の動き

- それでは、ログを実際に見てみましょう。

1

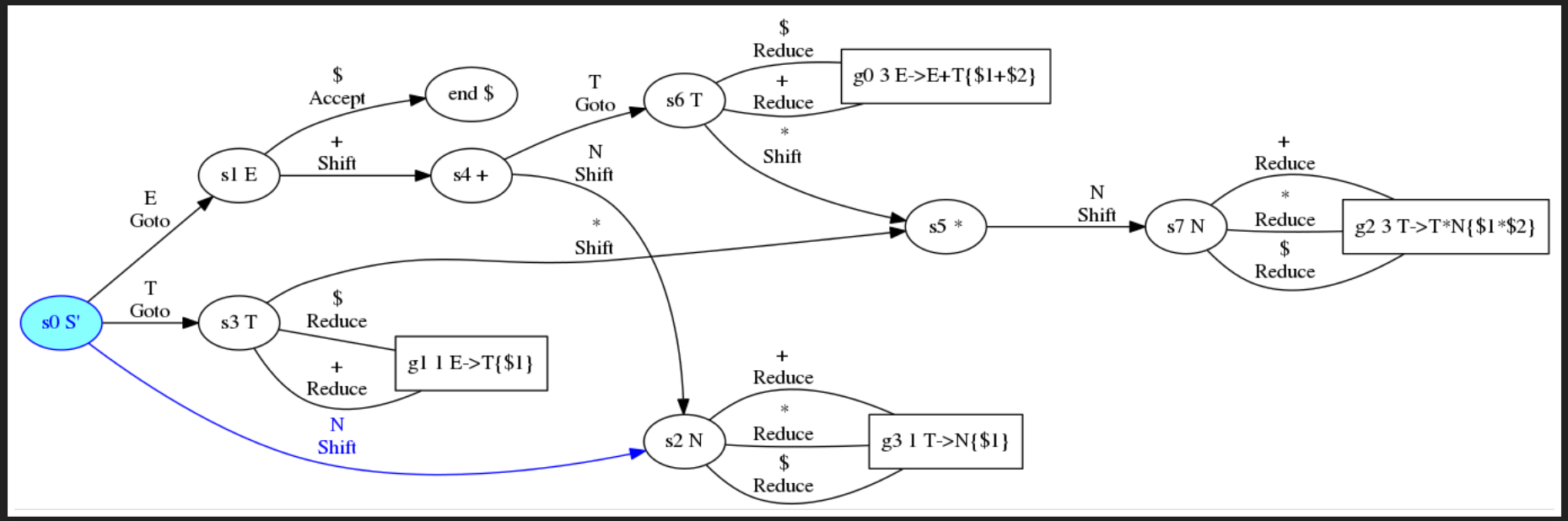
SHIFT S2

```
inputs 1 $
status 0
results
```



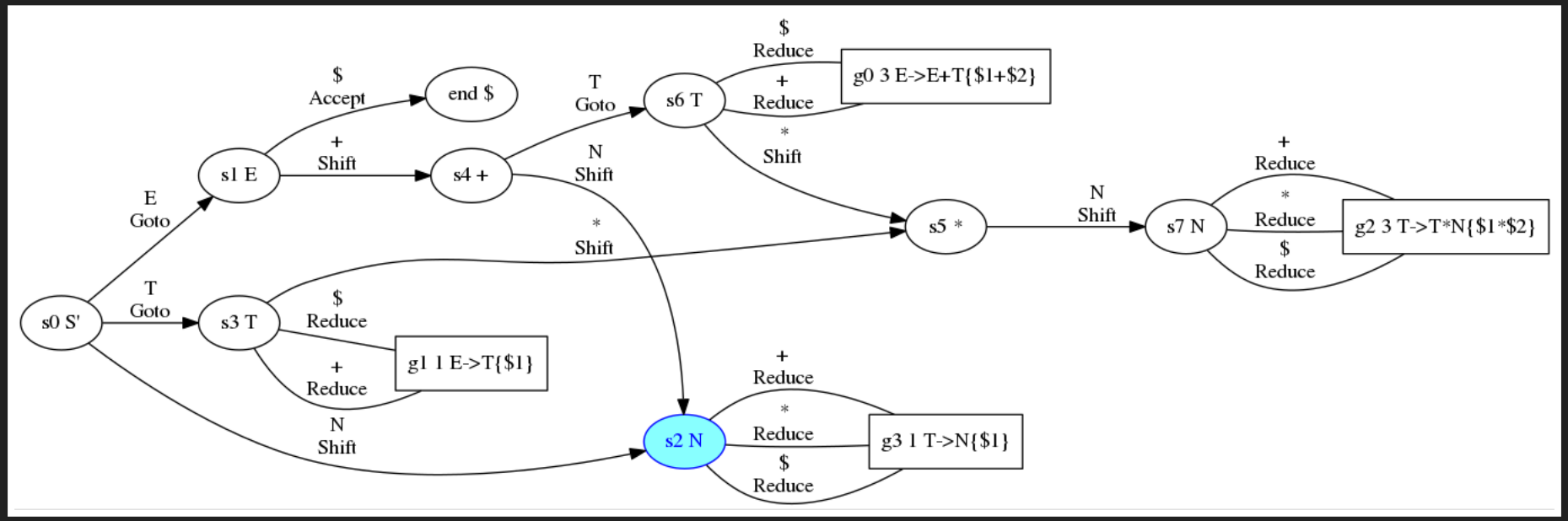
SHIFT S2

```
inputs 1 $ ステータスに2をpush 入力1を結果に移動
status 0
results
```



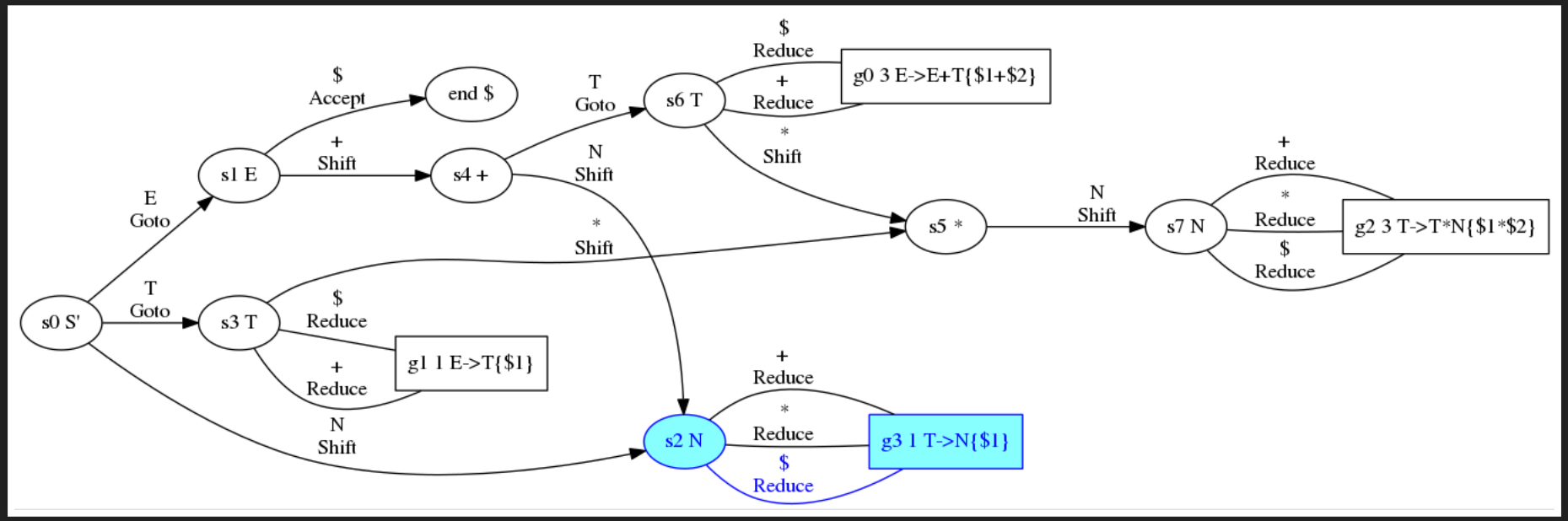
REDUCE G3

```
inputs $
status 2 0
results 1
```



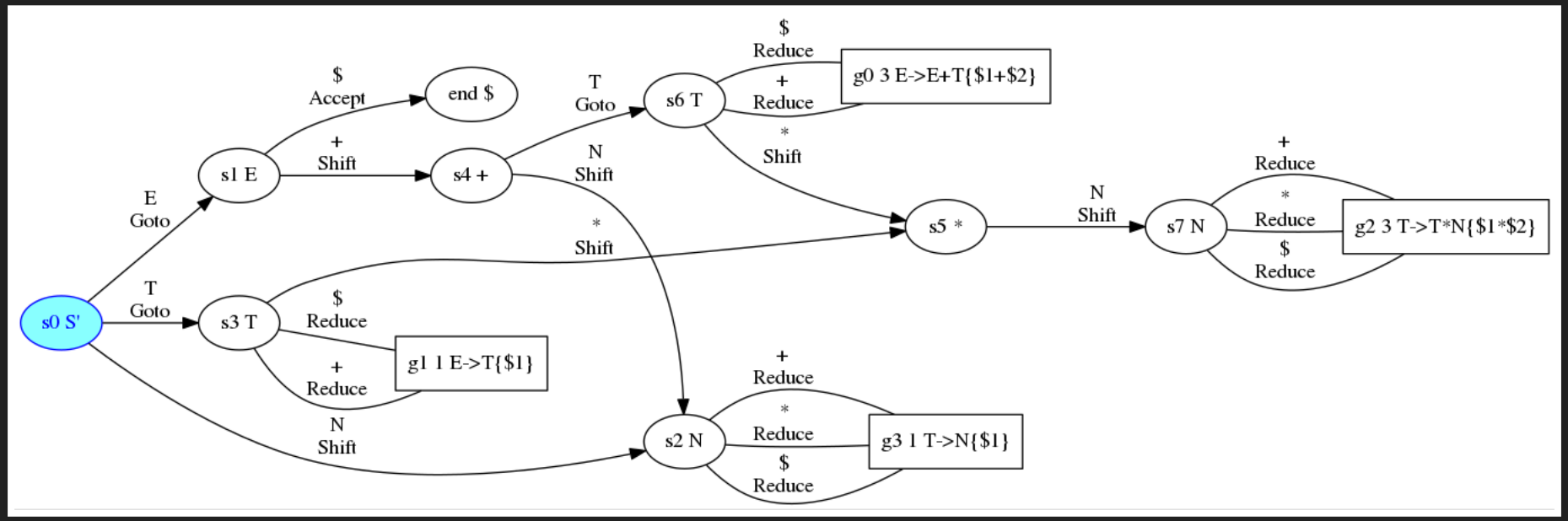
REDUCE G3

```
inputs $ 文法g3を見て、1個Pop、{$1}を結果にpush、文法名Tを入力にpush
status 2 0
results 1
```



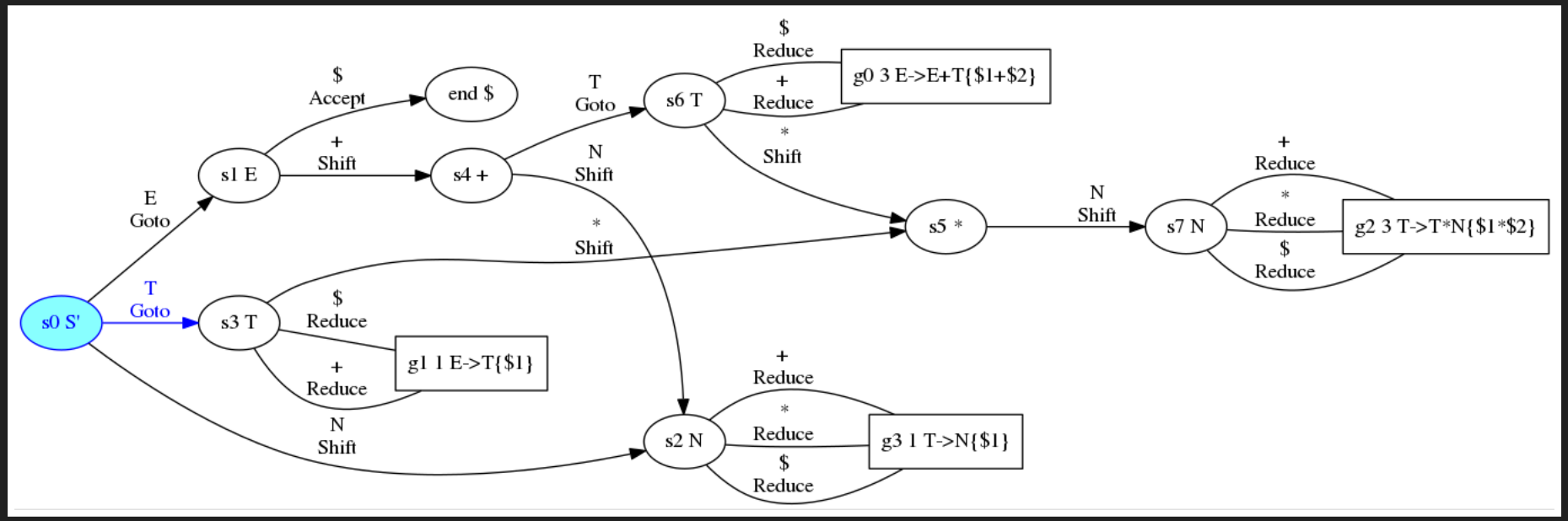
GOTO S3

```
inputs T $
status 0
results 1
```



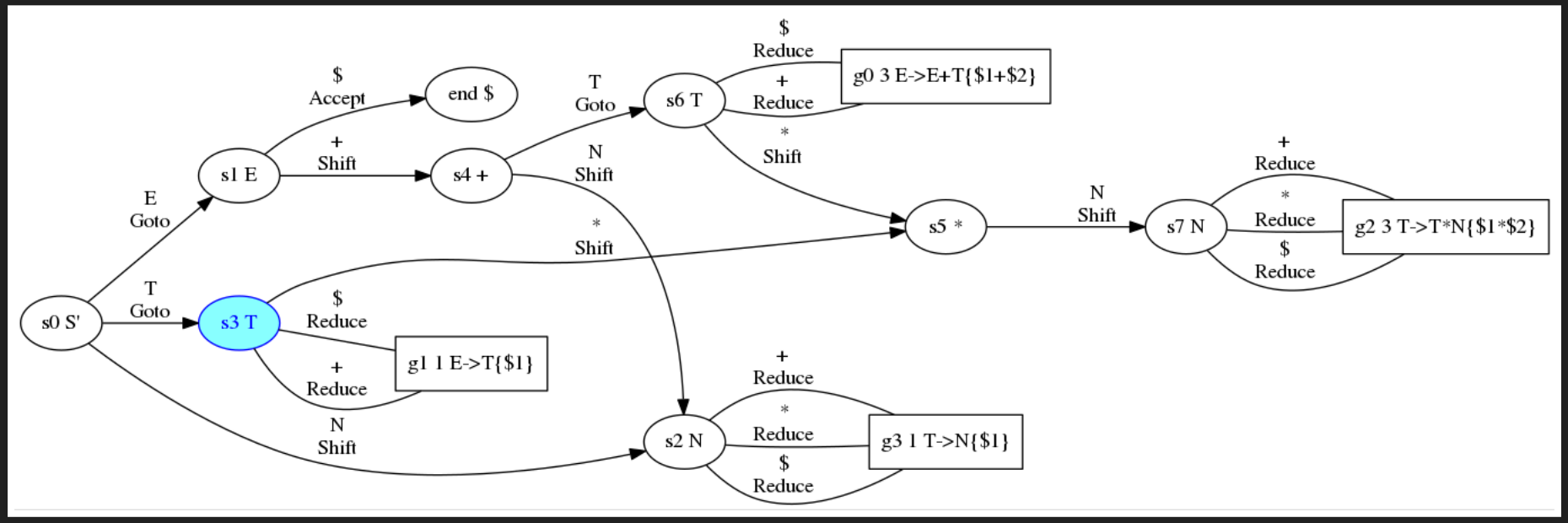
GOTO S3

```
inputs T $ ステータスに3をpush,入力Tを捨てる
status 0
results 1
```



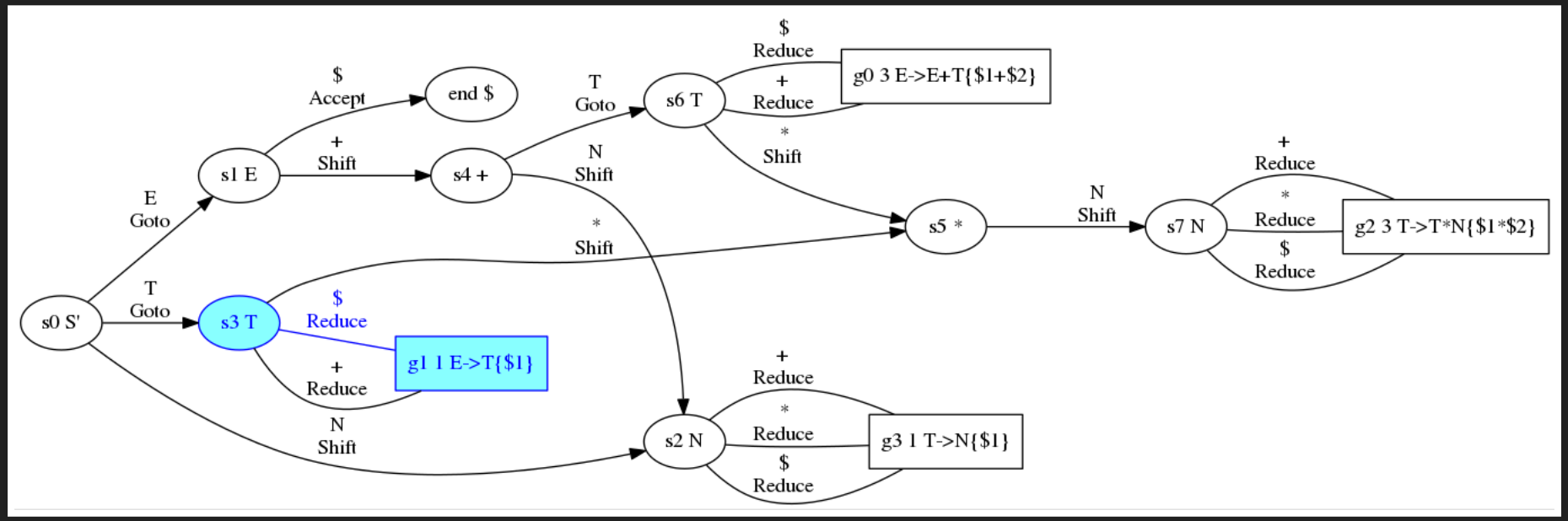
REDUCE G1

```
inputs $
status 3 0
results 1
```



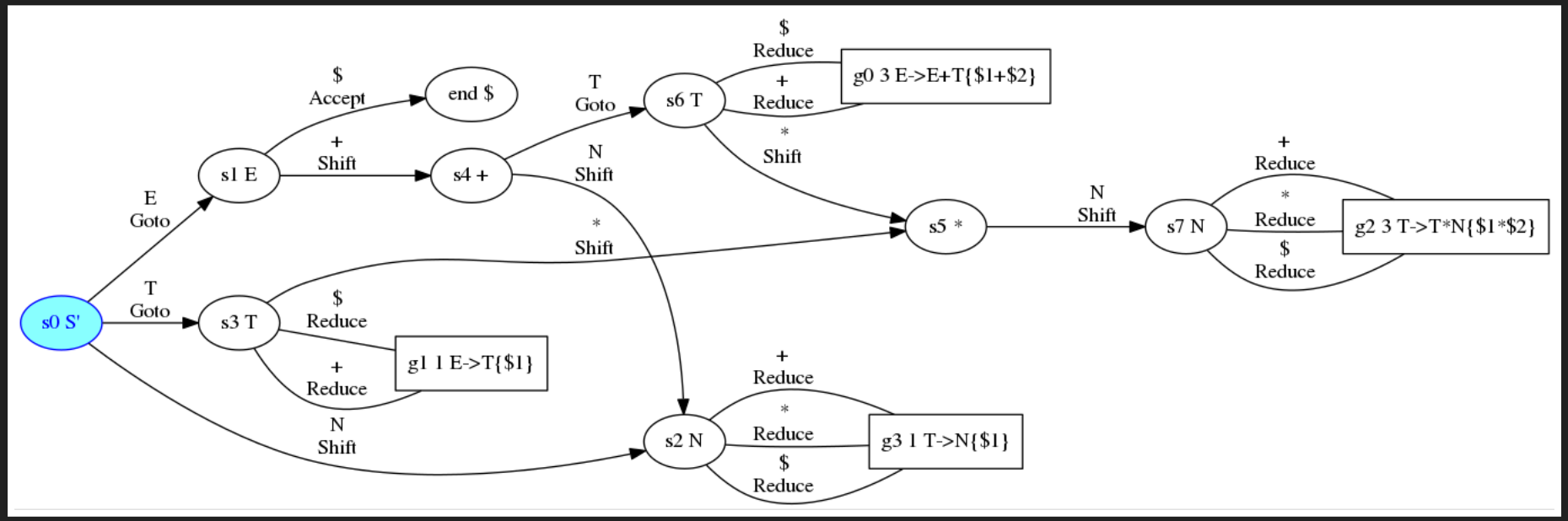
REDUCE G1

```
inputs $ 文法g1を見て、1個Pop、{$1}を結果にpush、文法名Eを入力にpush
status 3 0
results 1
```



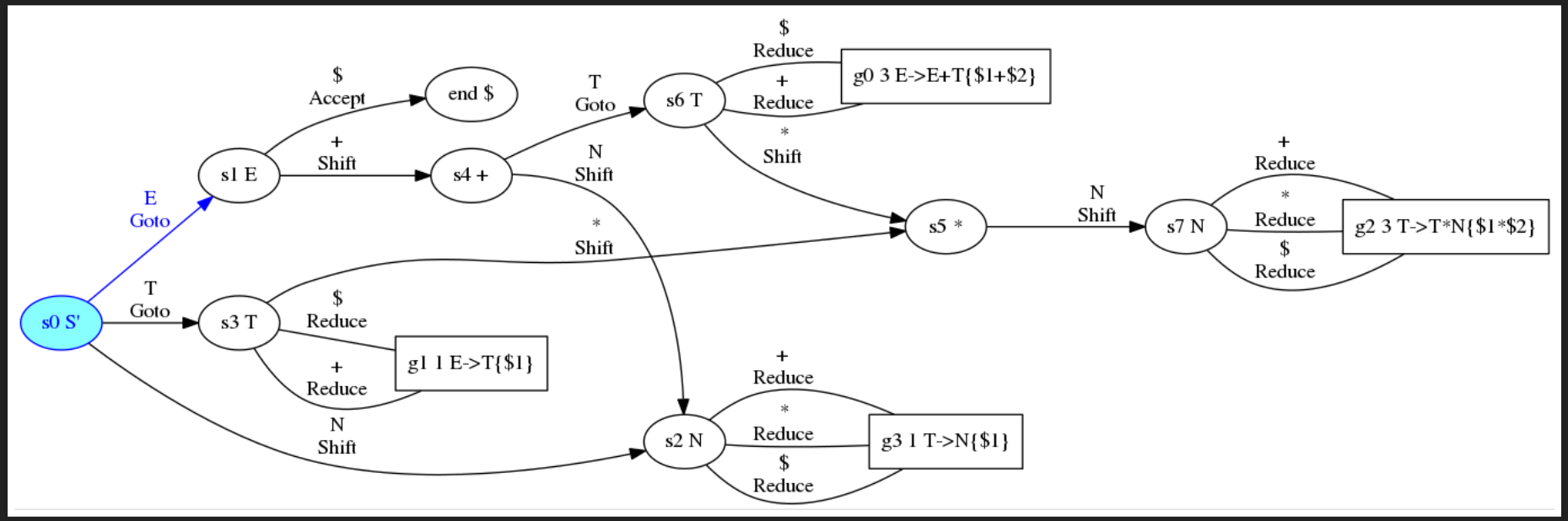
GOTO S1

```
inputs E $
status 0
results 1
```



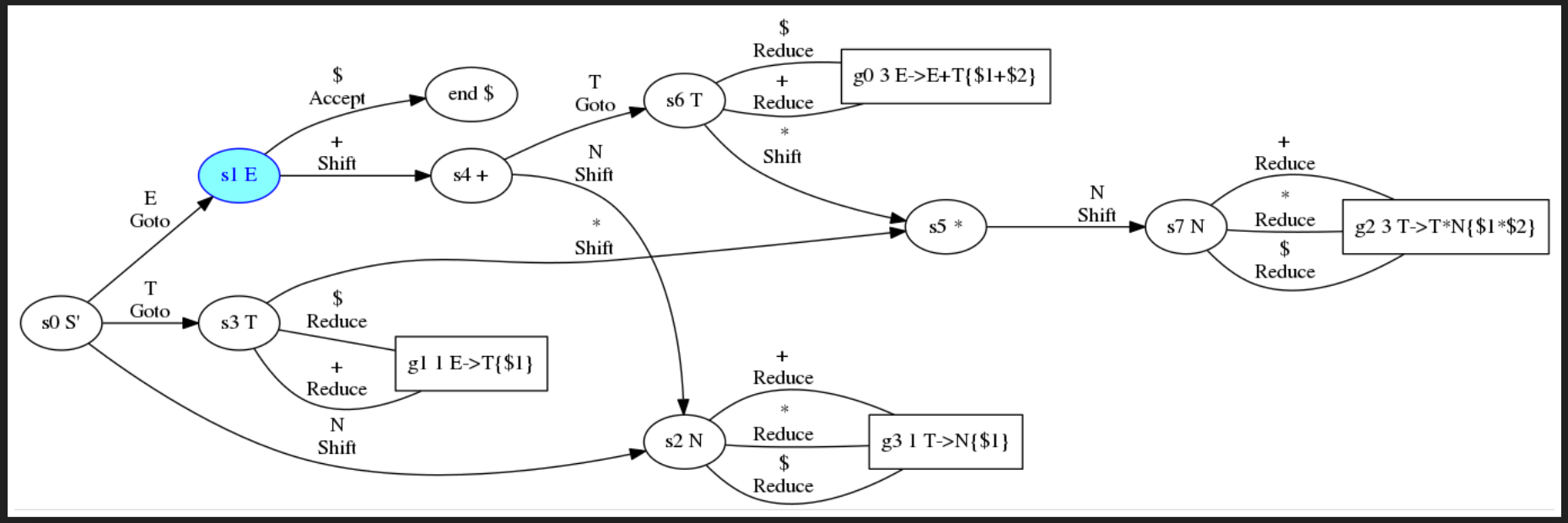
GOTO S1

```
inputs E $ ステータスに1をpush,入力Eを捨てる
status 0
results 1
```



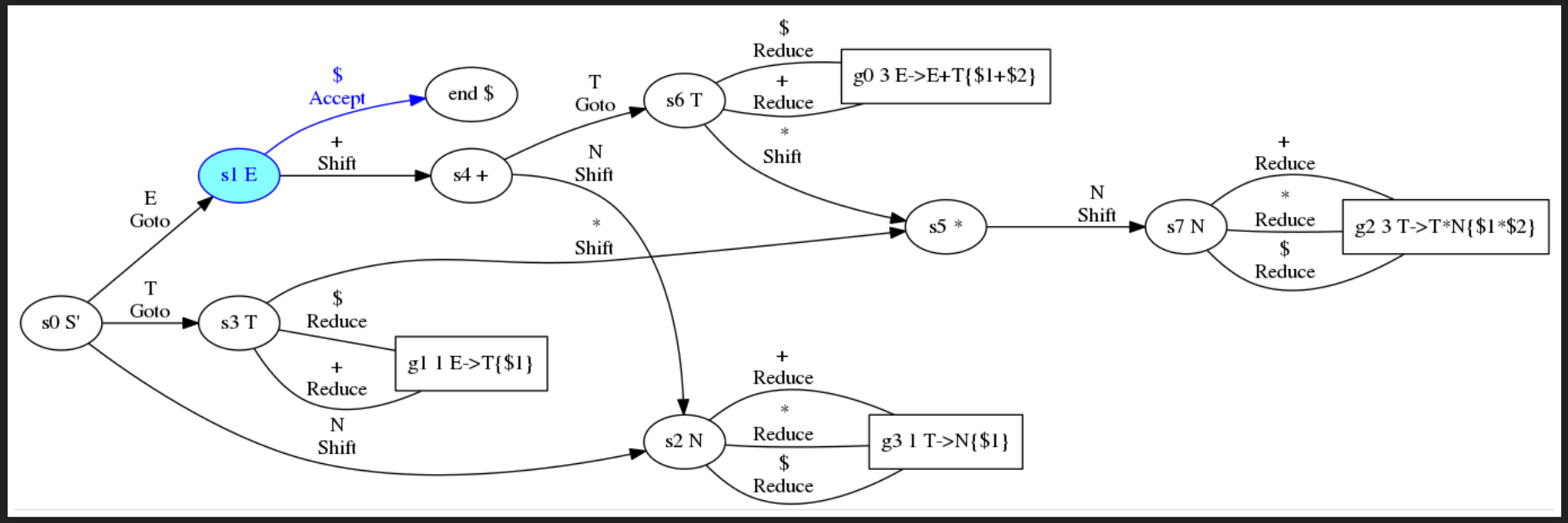
ACCEPT

```
inputs $
status 1 0
results 1
```



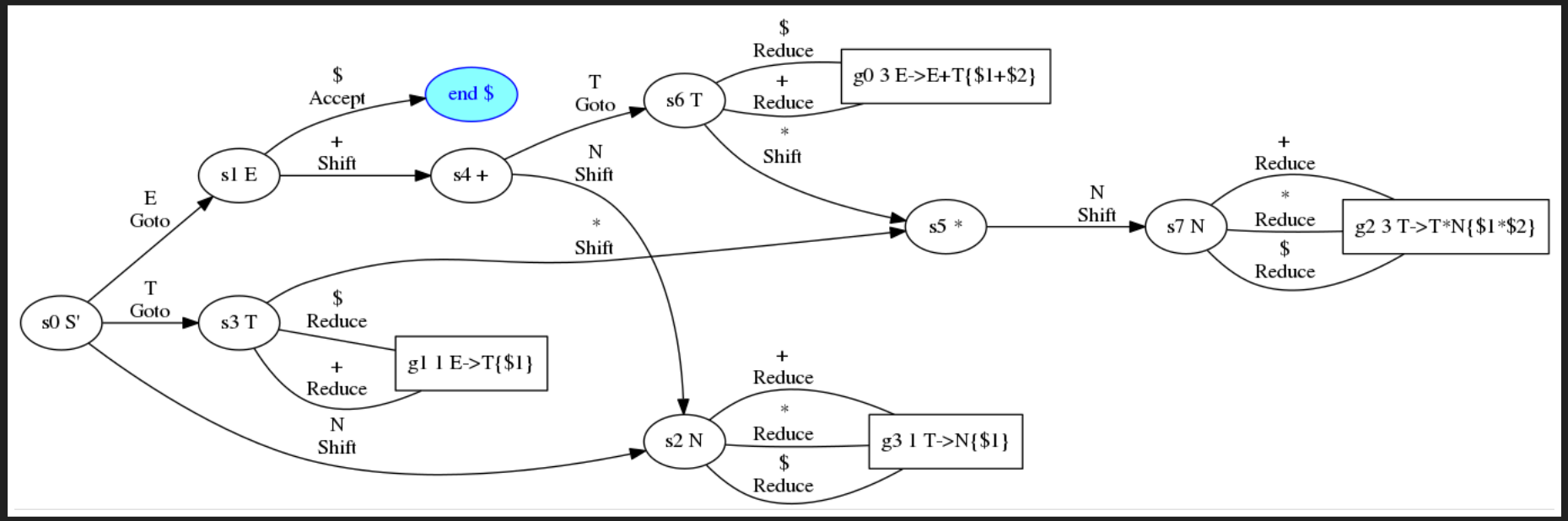
ACCEPT

```
inputs $ アクセプト
status 1 0
results 1
```



ACCEPT

```
inputs $ 結果 1 です
status 1 0
results 1
```

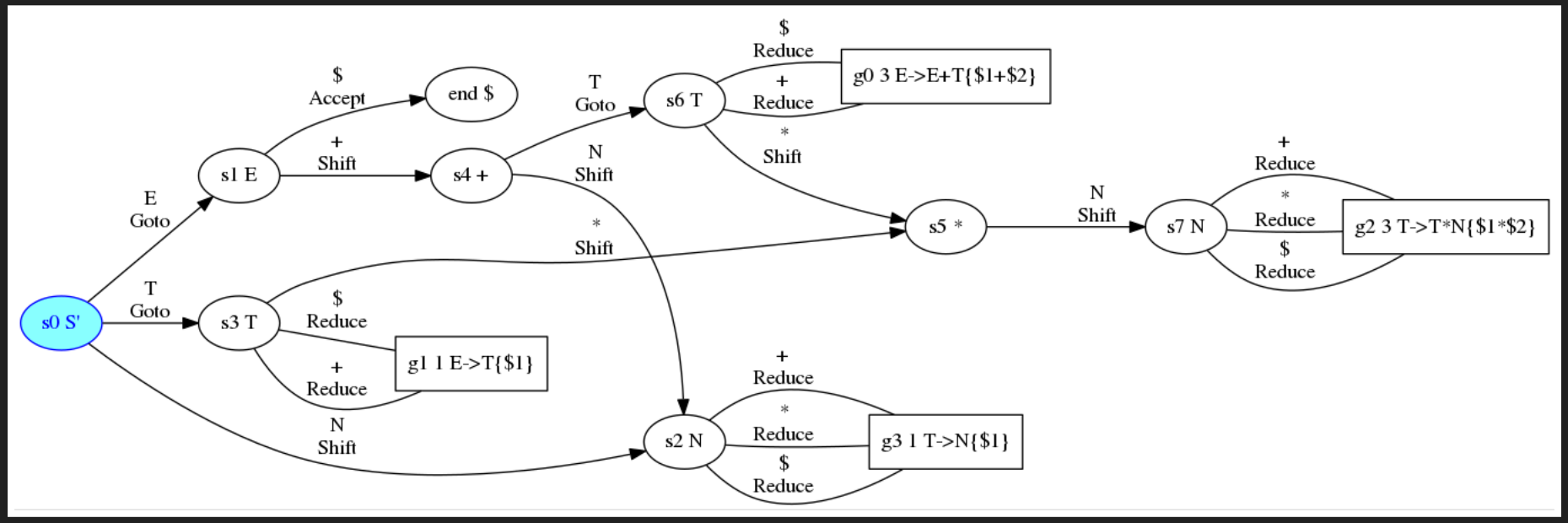


RESULT 1 = 1

2*3

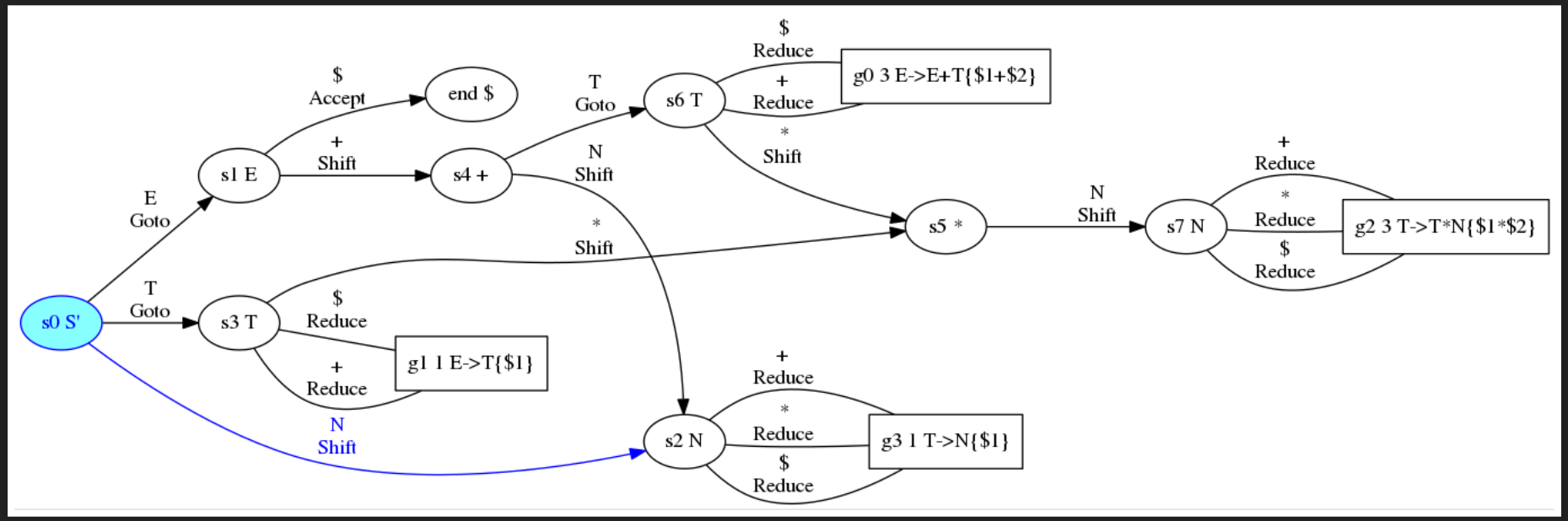
SHIFT S2

```
inputs 2 * 3 $
status 0
results
```



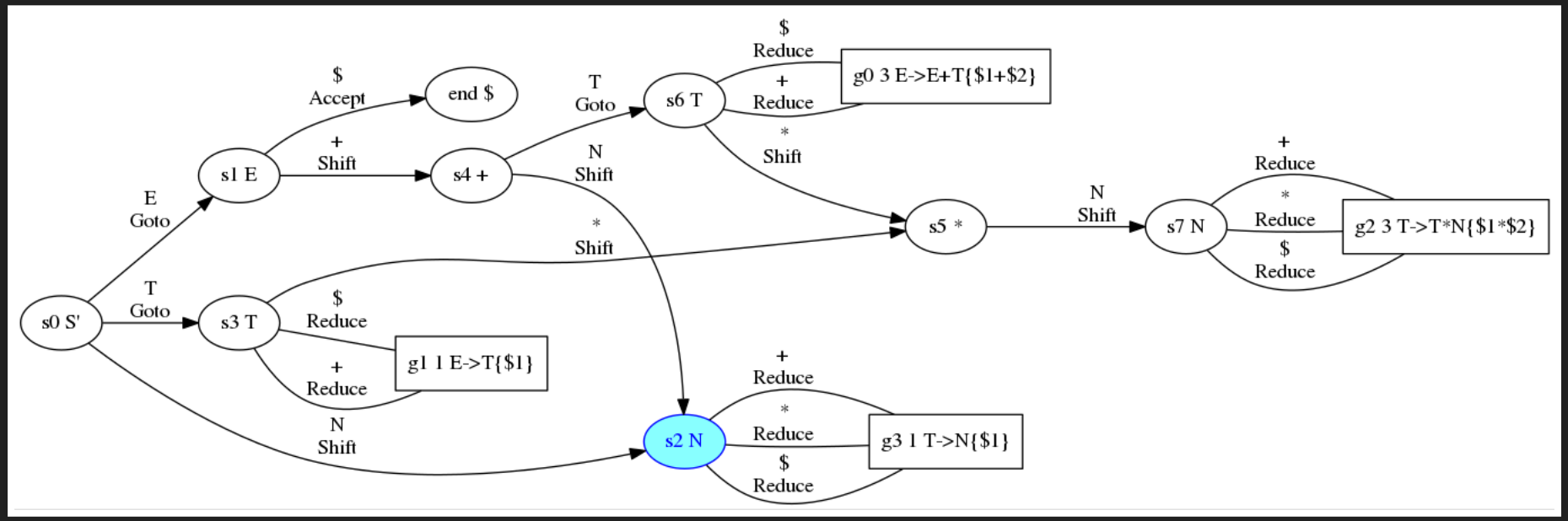
SHIFT S2

```
inputs 2 * 3 $ ステータスに2をpush 入力2を結果に移動
status 0
results
```



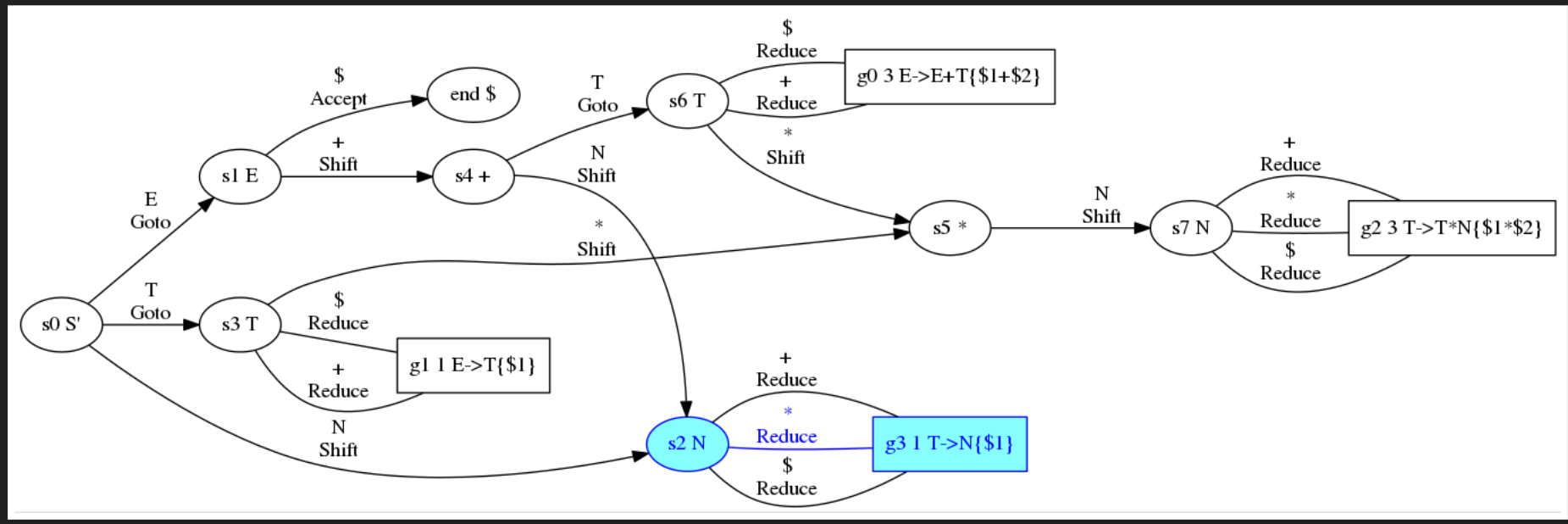
REDUCE G3

```
inputs * 3 $
status 2 0
results 2
```



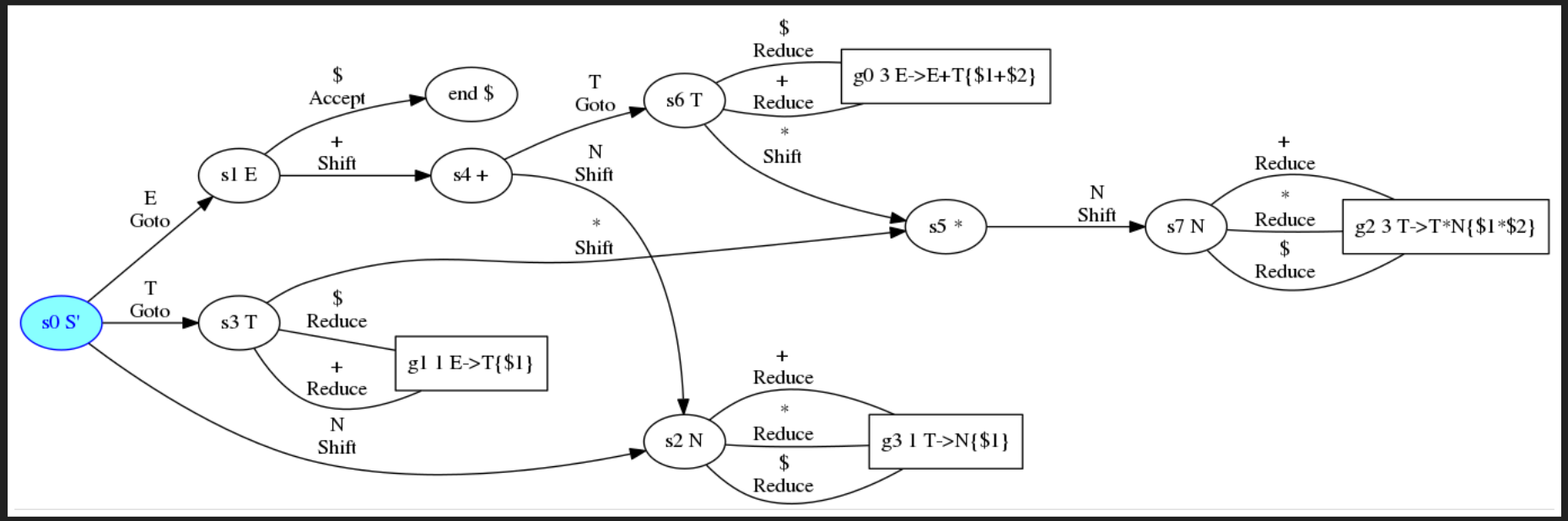
REDUCE G3

```
inputs * 3 $ 文法g3を見て、1個Pop、{$1}を結果にpush、文法名Tを入力にpush
status 2 0
results 2
```



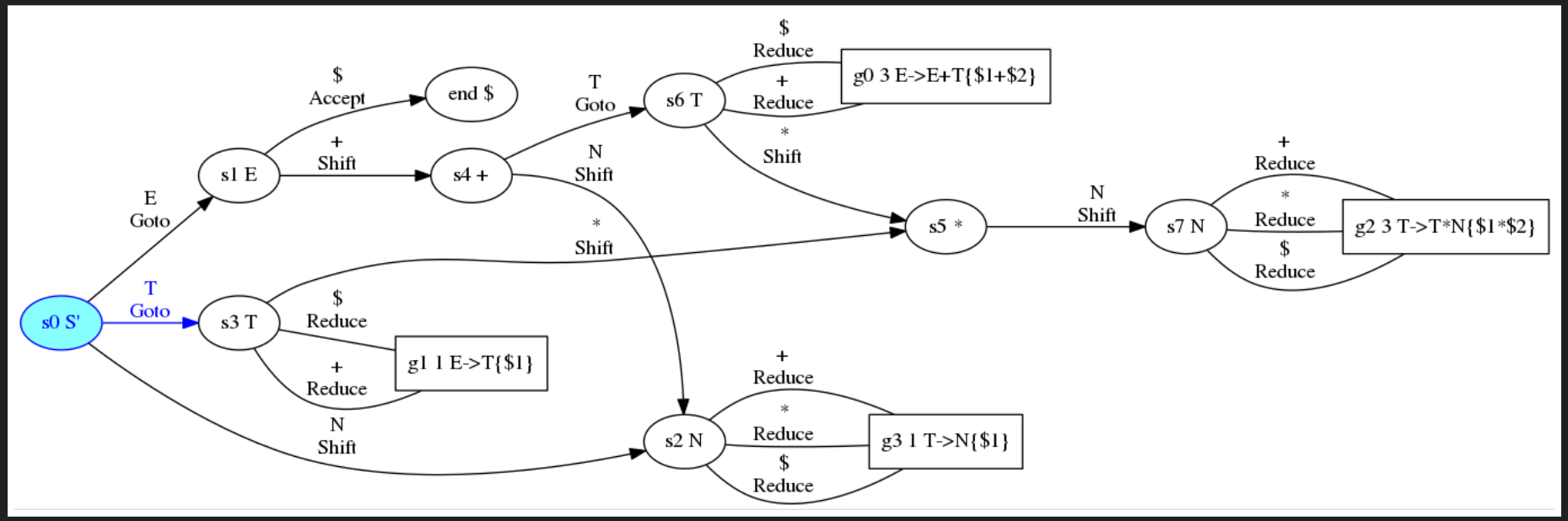
GOTO S3

```
inputs T * 3 $
status 0
results 2
```



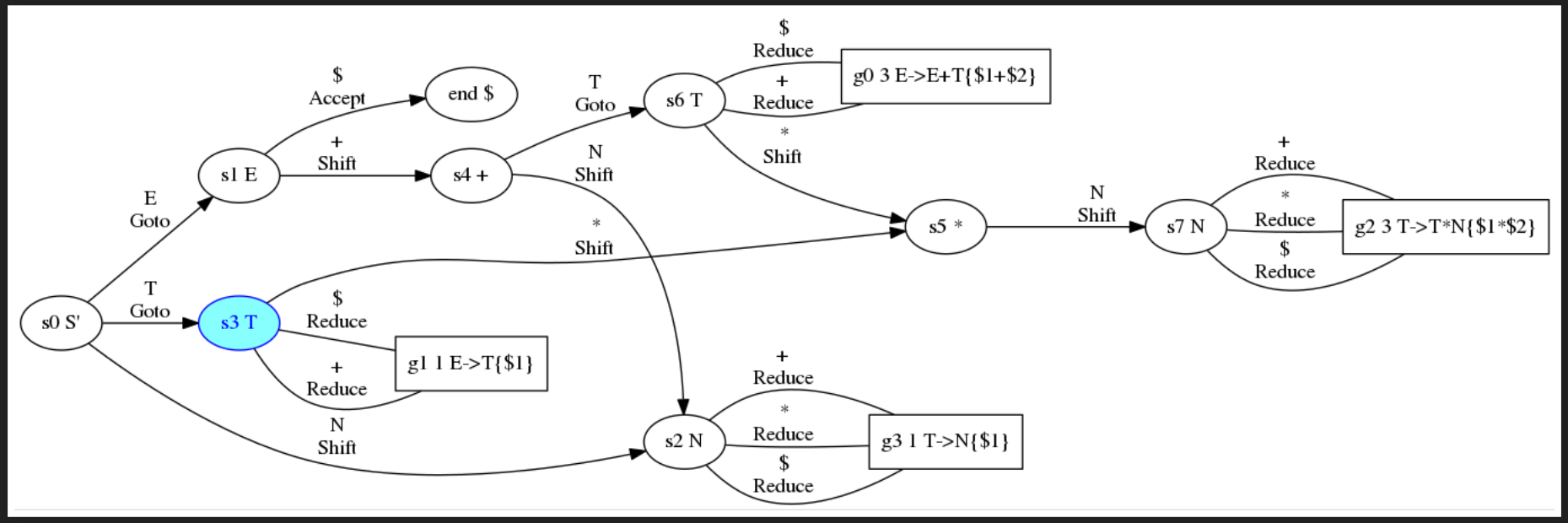
GOTO S3

```
inputs T * 3 $ ステータスに3をpush,入力Tを捨てる
status 0
results 2
```



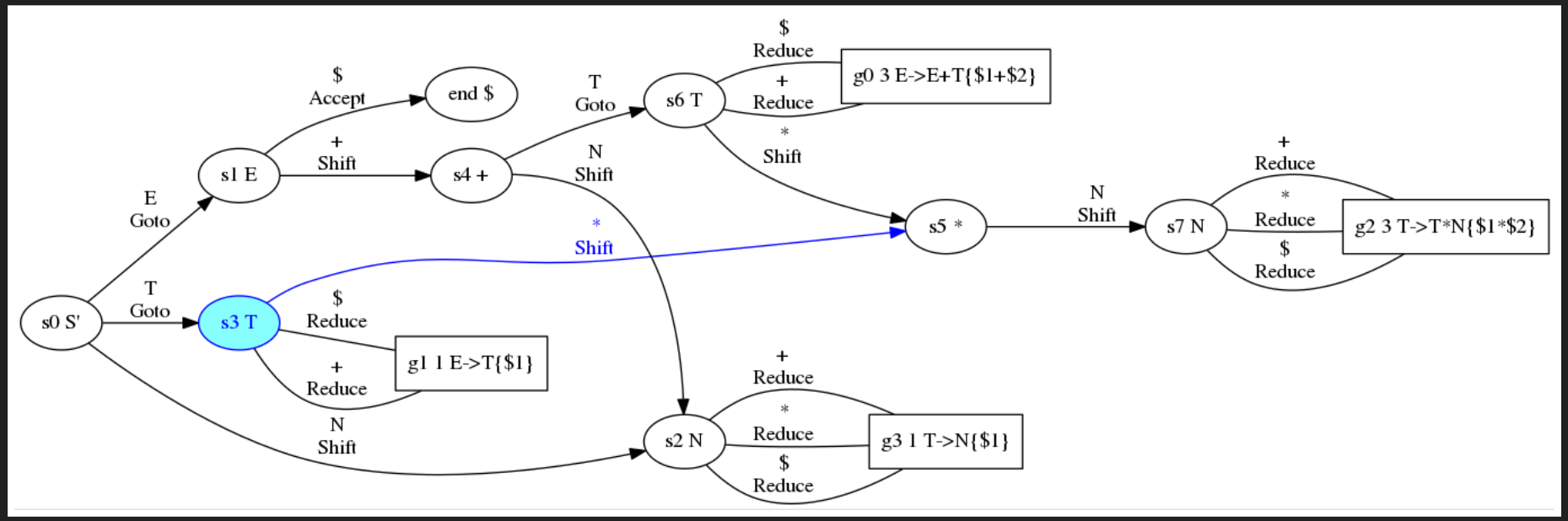
SHIFT S5

```
inputs * 3 $
status 3 0
results 2
```



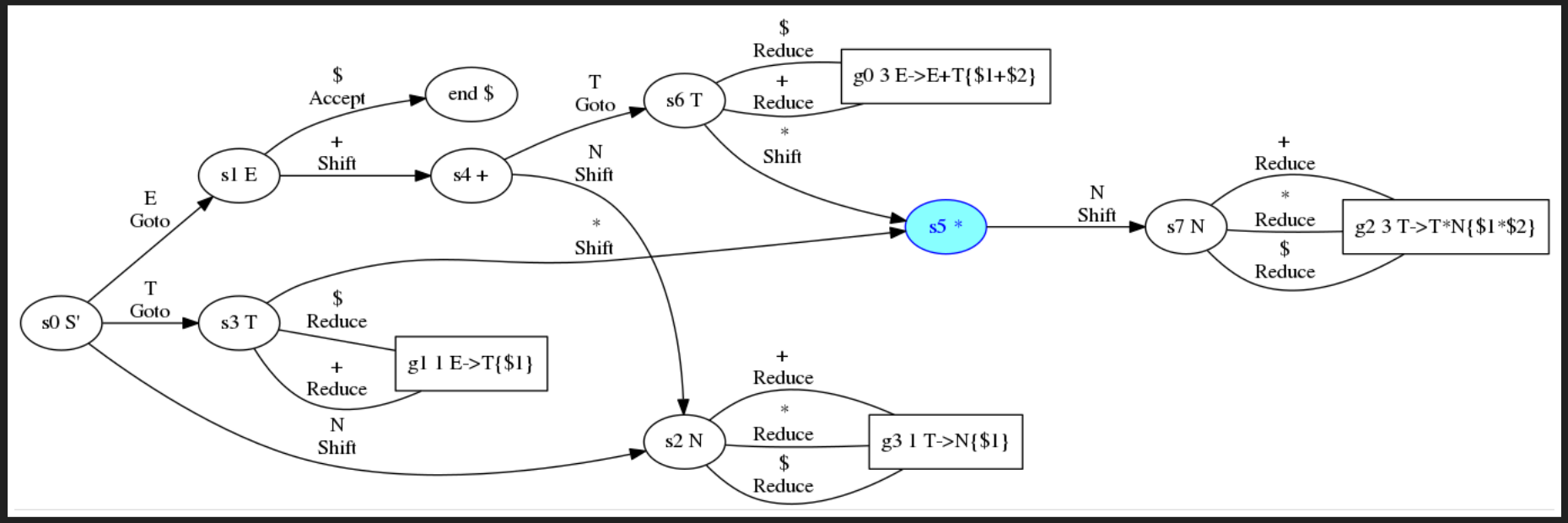
SHIFT S5

```
inputs * 3 $ ステータスに5をpush 入力*を結果に移動
status 3 0
results 2
```



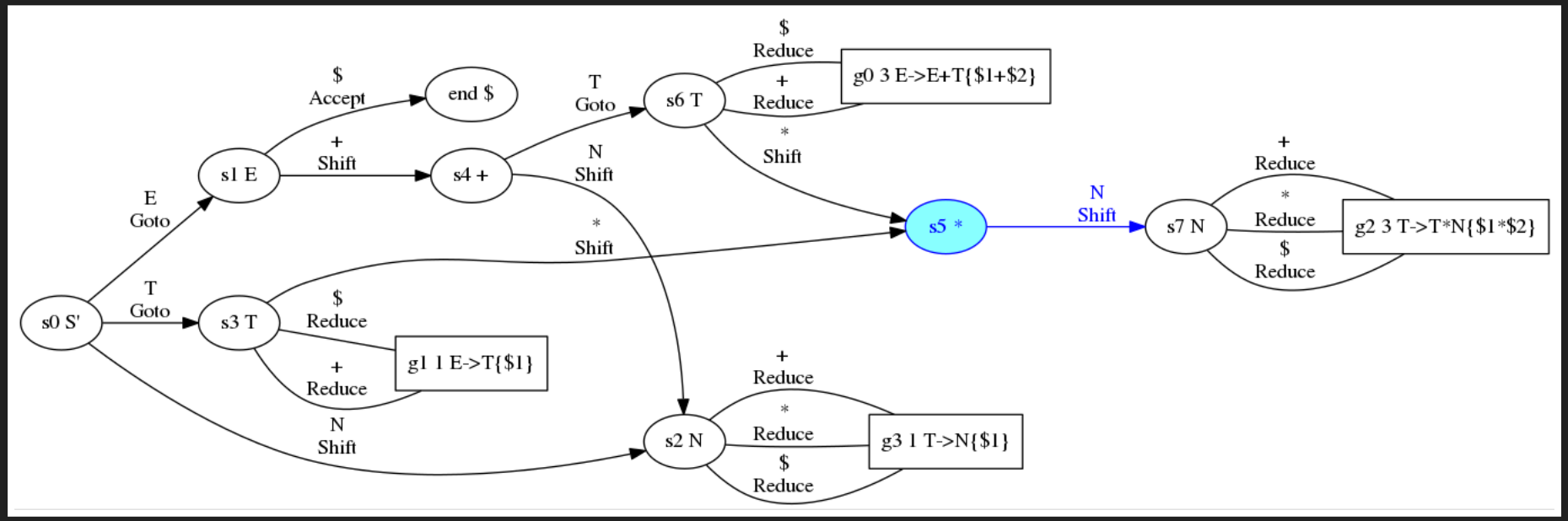
SHIFT S7

```
inputs 3 $
status 5 3 0
results * 2
```



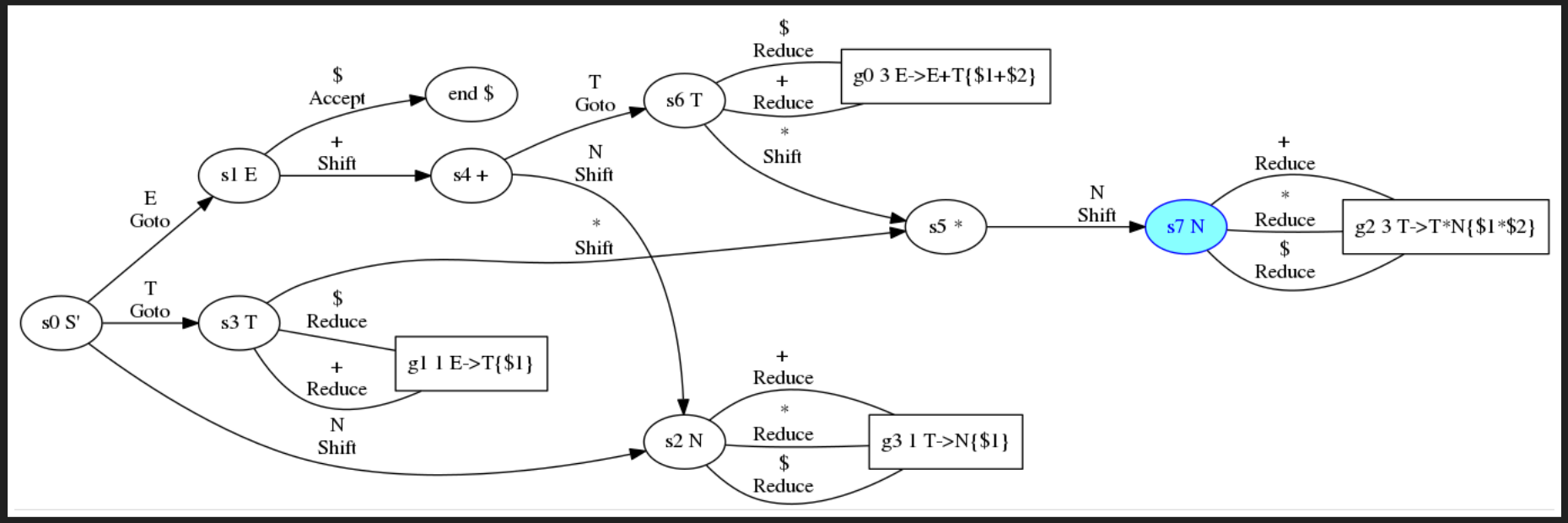
SHIFT S7

```
inputs 3 $ ステータスに7をpush 入力3を結果に移動
status 5 3 0
results * 2
```



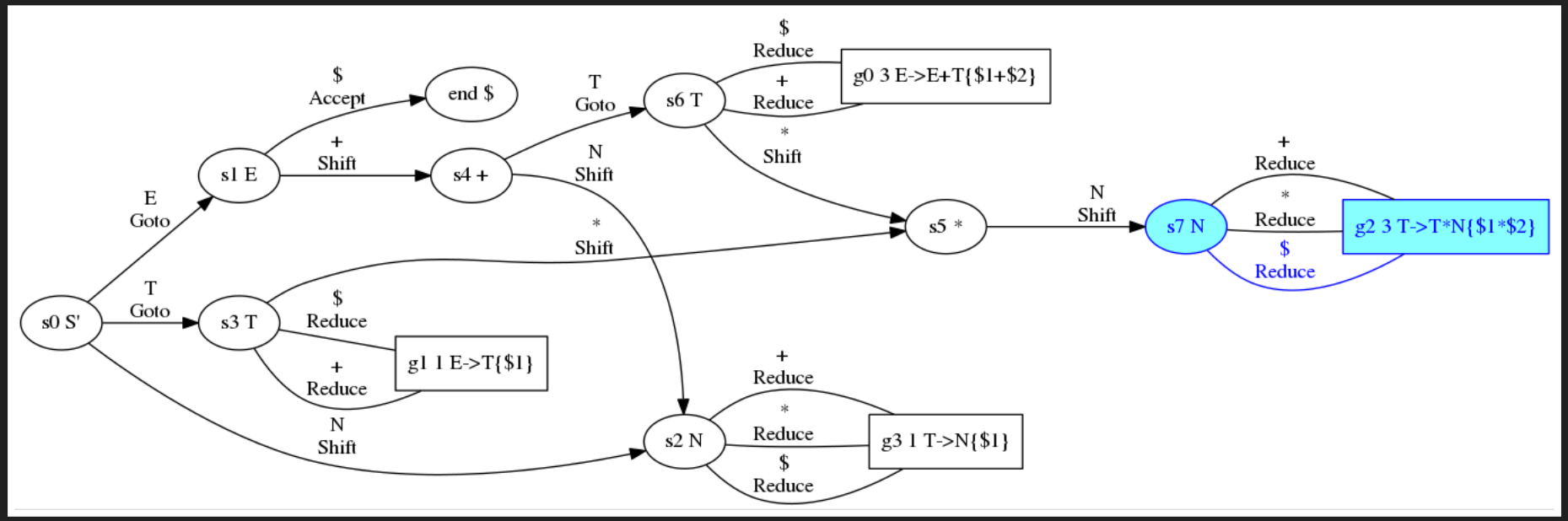
REDUCE G2

```
inputs $
status 7 5 3 0
results 3 * 2
```



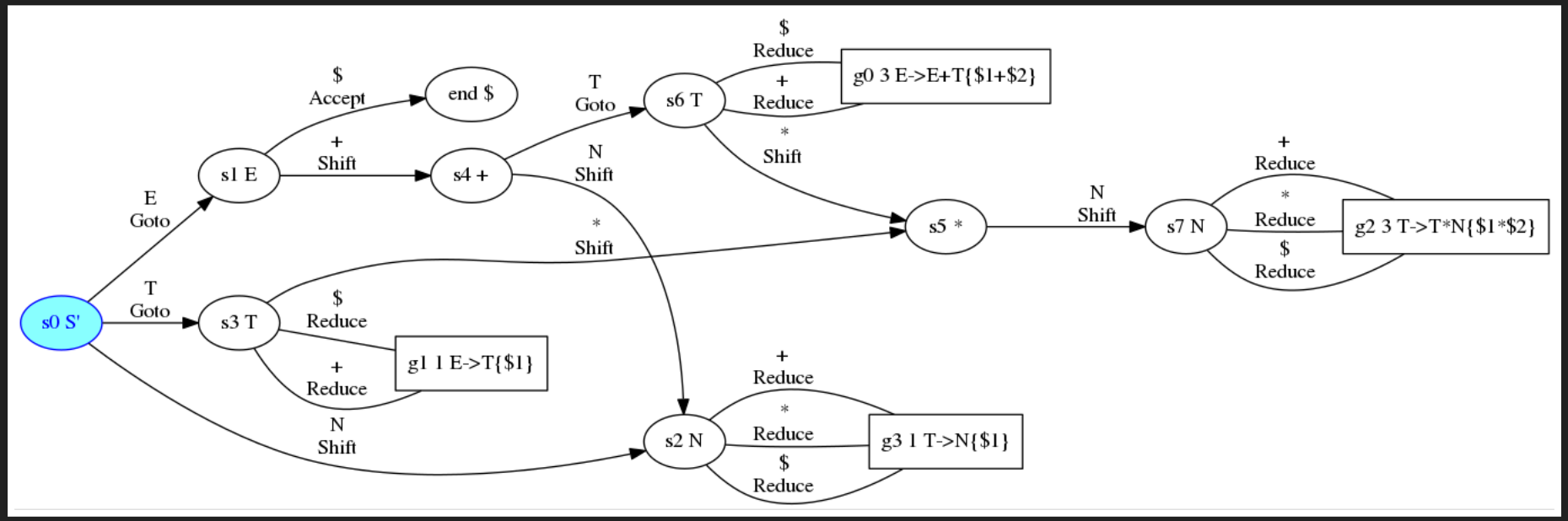
REDUCE G2

```
inputs $ 文法g2を見て、3個Pop、{$1*$2}を結果にpush、文法名Tを入力にpush
status 7 5 3 0
results 3 * 2
```



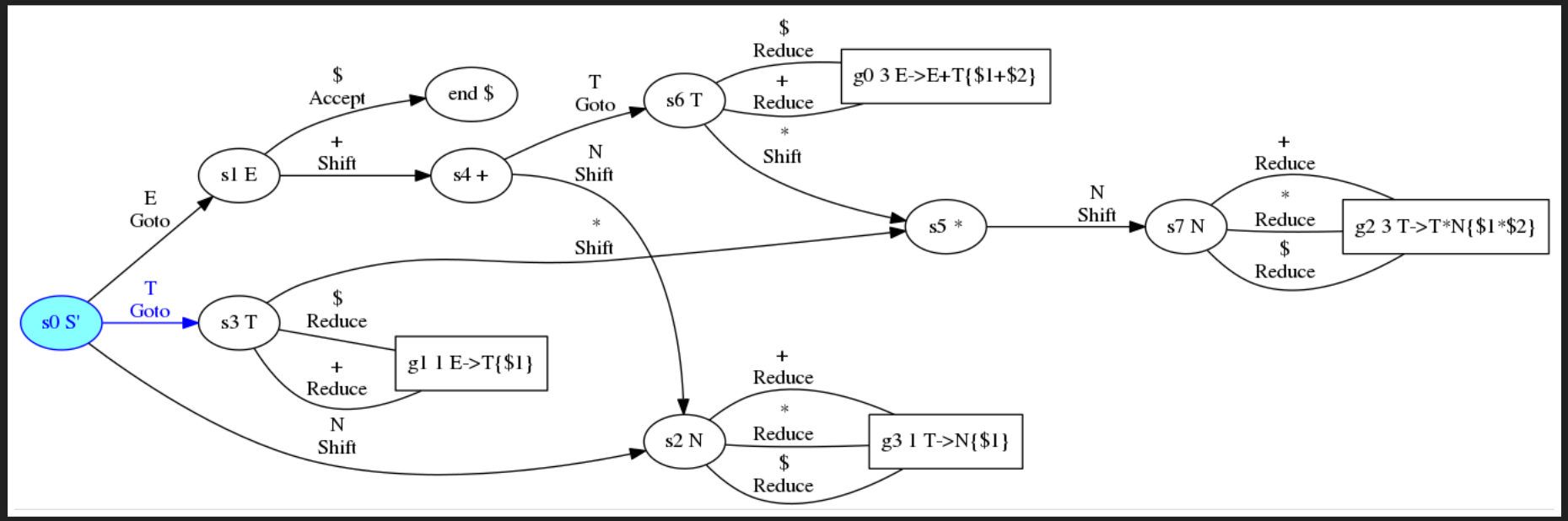
GOTO S3

```
inputs T $
status 0
results 6
```



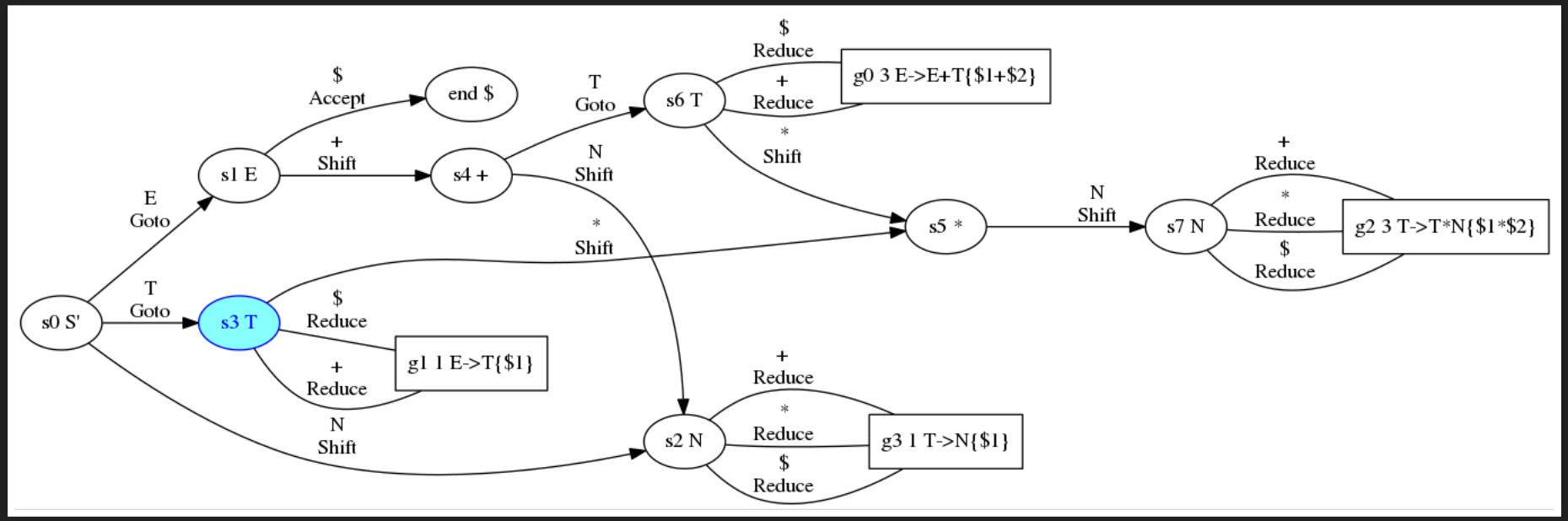
GOTO S3

```
inputs T $ ステータスに3をpush,入力Tを捨てる
status 0
results 6
```



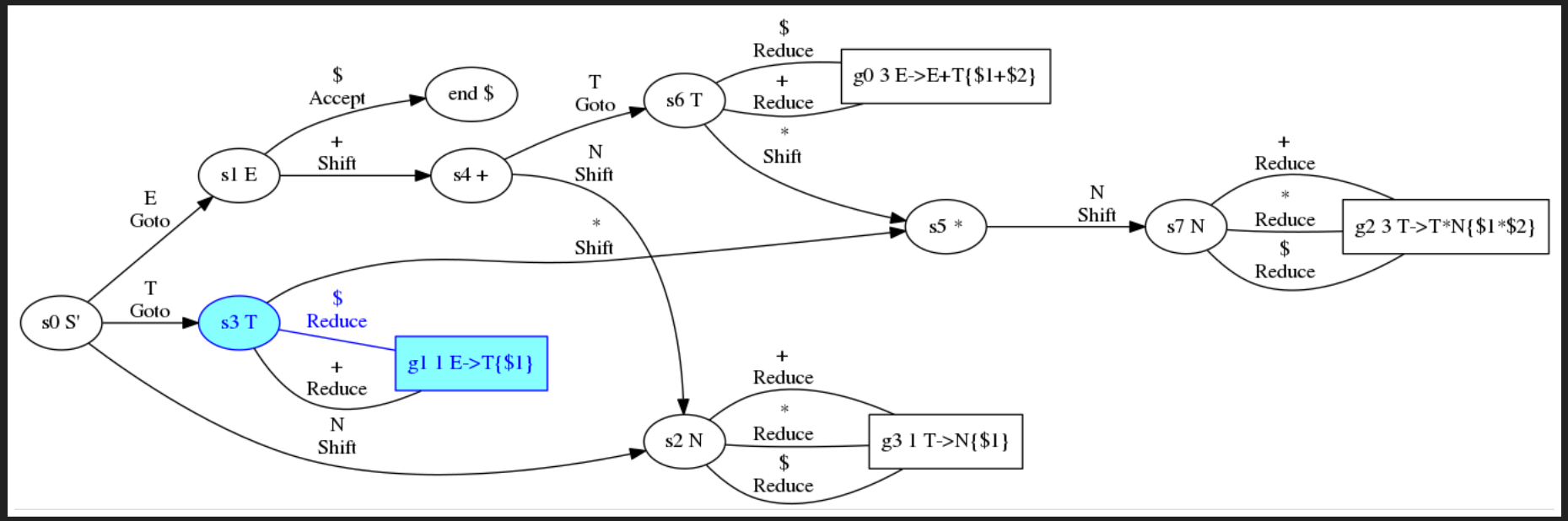
REDUCE G1

```
inputs $
status 3 0
results 6
```



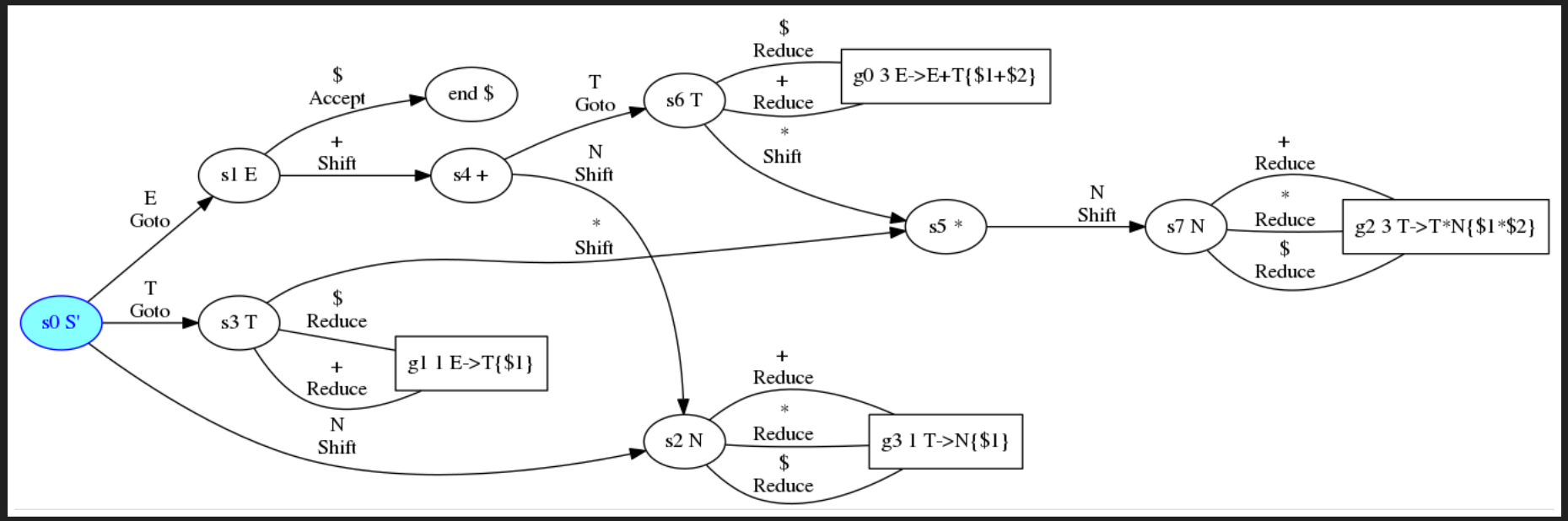
REDUCE G1

```
inputs $ 文法g1を見て、1個Pop、{$1}を結果にpush、文法名Eを入力にpush
status 3 0
results 6
```



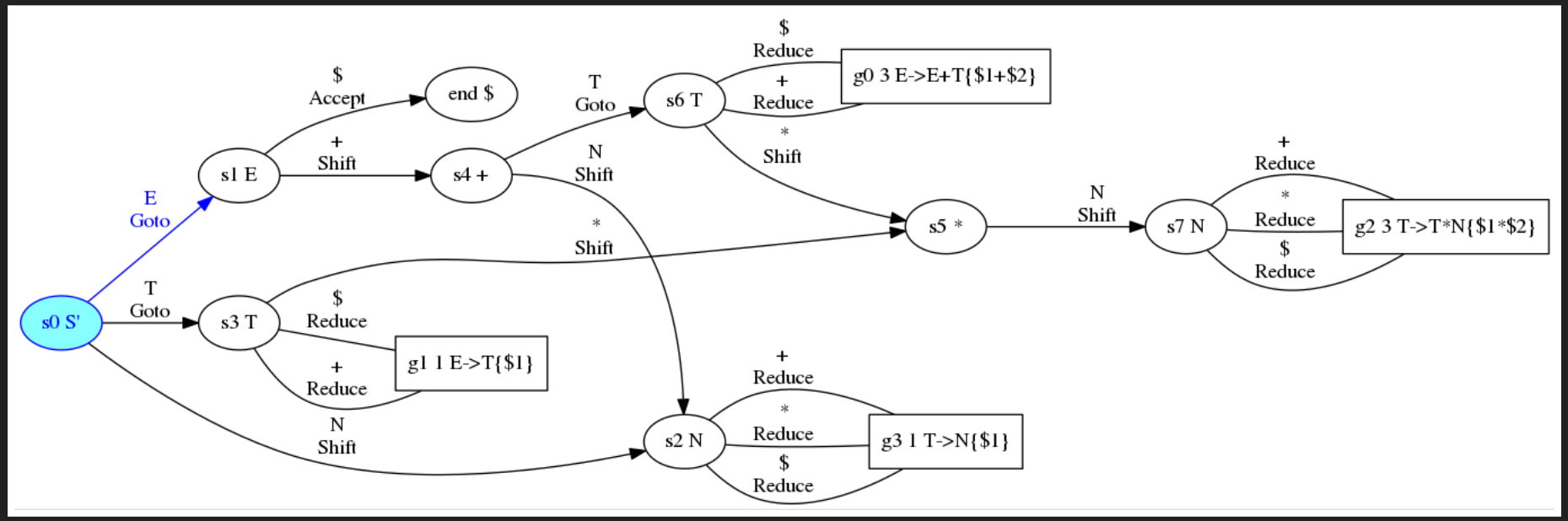
GOTO S1

```
inputs E $
status 0
results 6
```



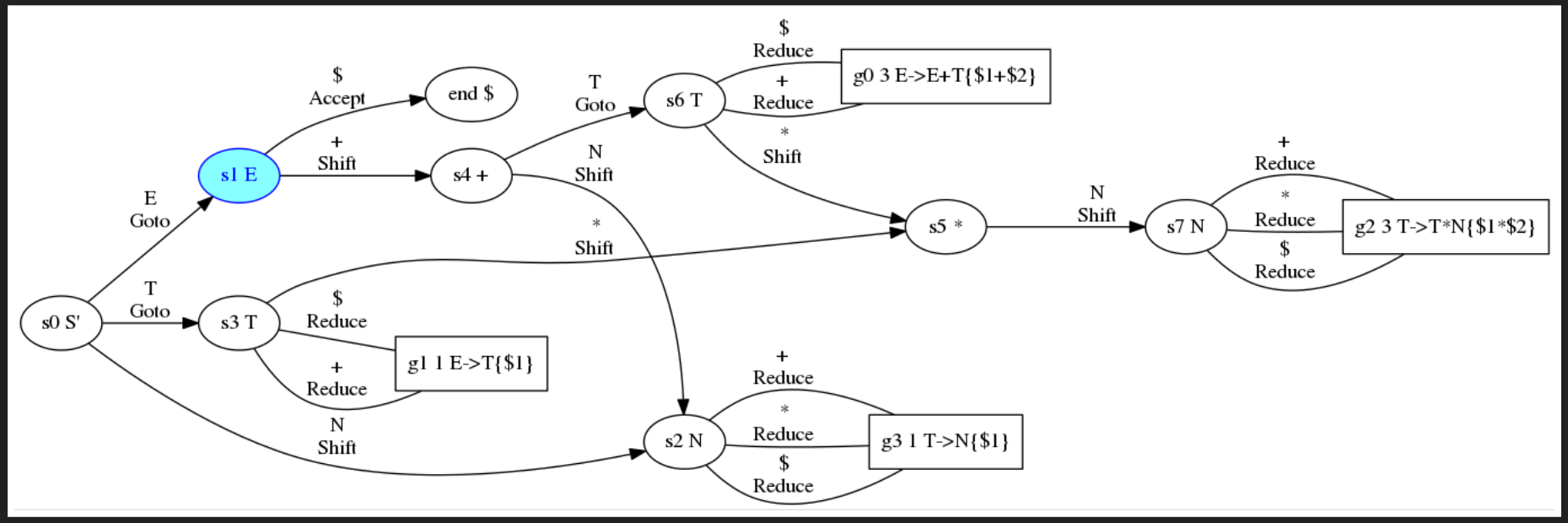
GOTO S1

```
inputs E $ ステータスに1をpush,入力Eを捨てる
status 0
results 6
```



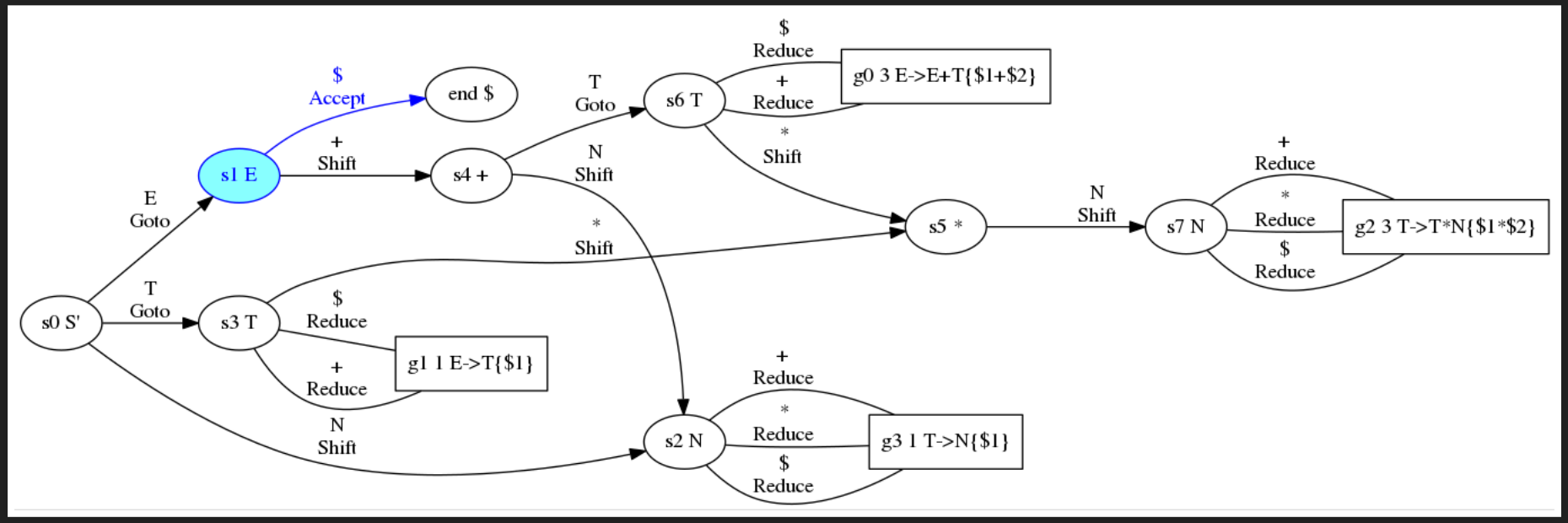
ACCEPT

```
inputs $
status 1 0
results 6
```



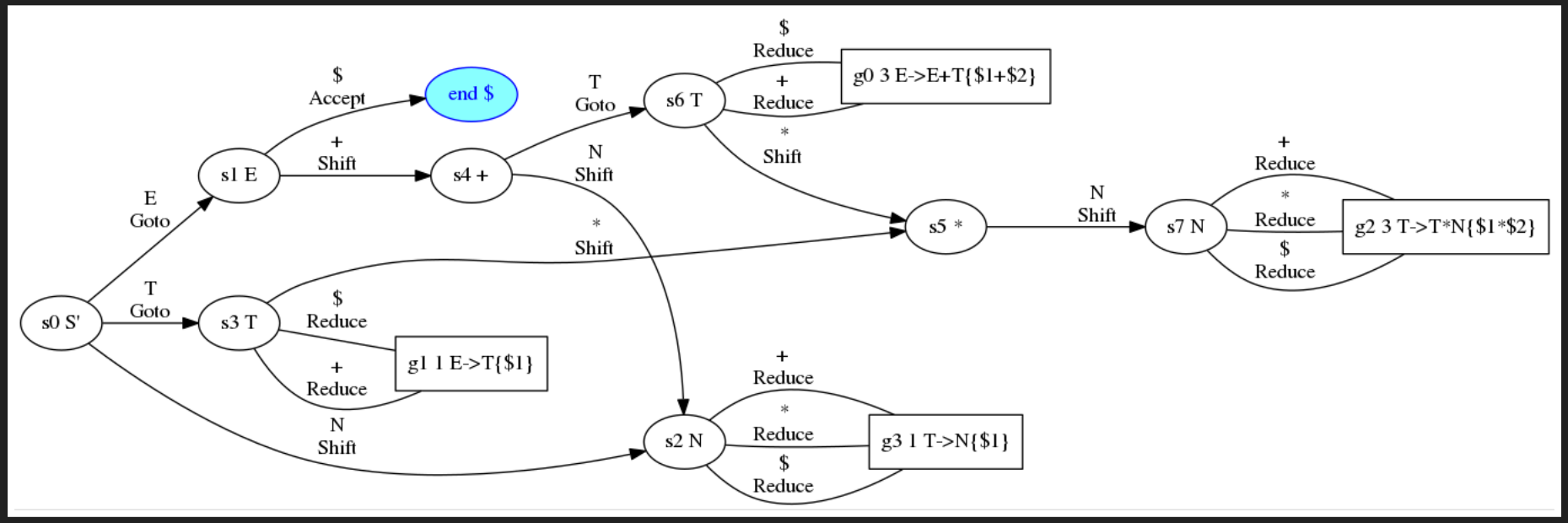
ACCEPT

```
inputs $ アクセプト
status 1 0
results 6
```



ACCEPT

```
inputs $ 結果 6 です
status 1 0
results 6
```

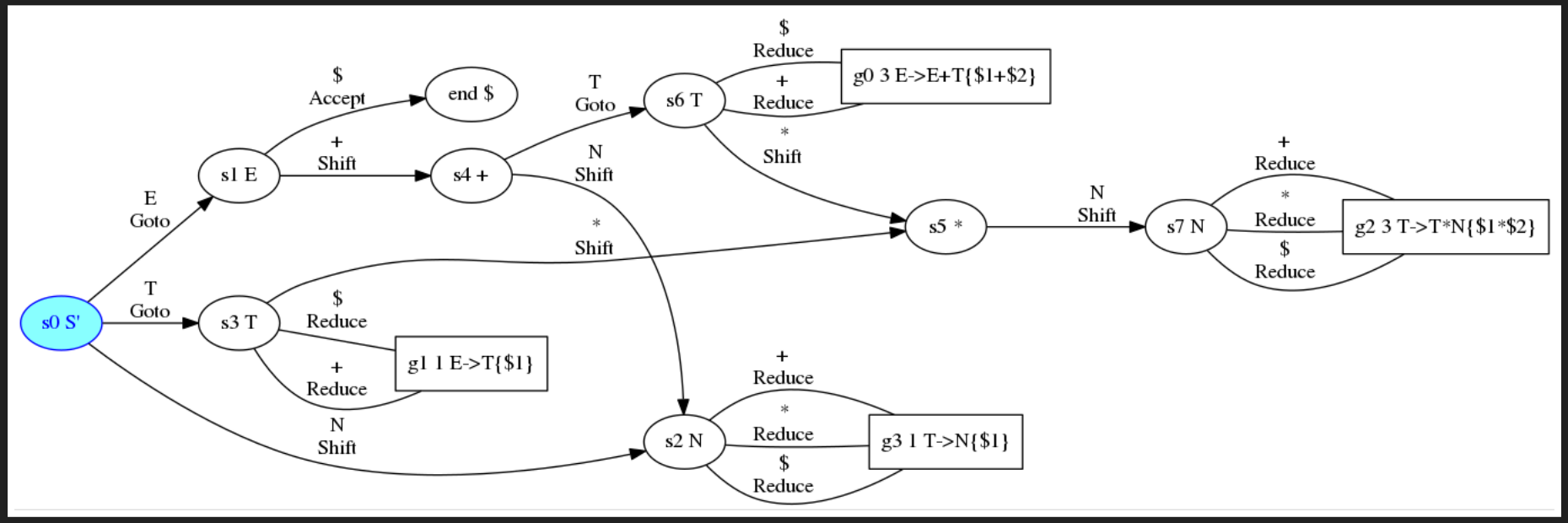


RESULT 2*3 = 6

2*3*4

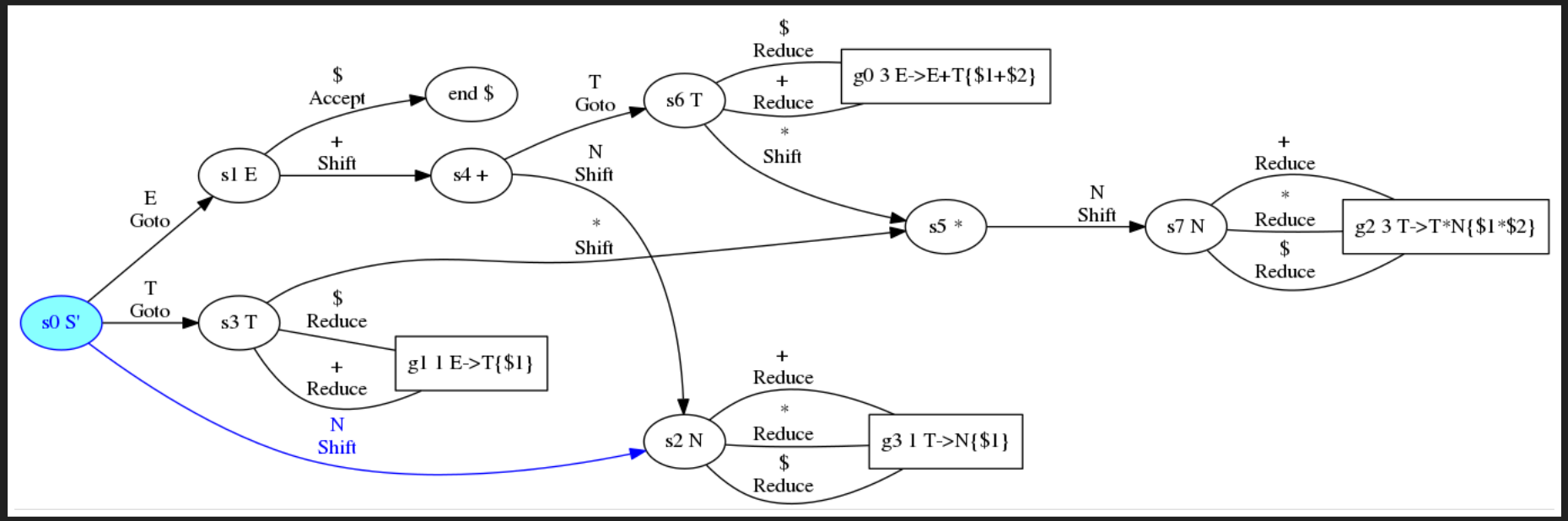
SHIFT S2

```
inputs 2 * 3 * 4 $
status 0
results
```



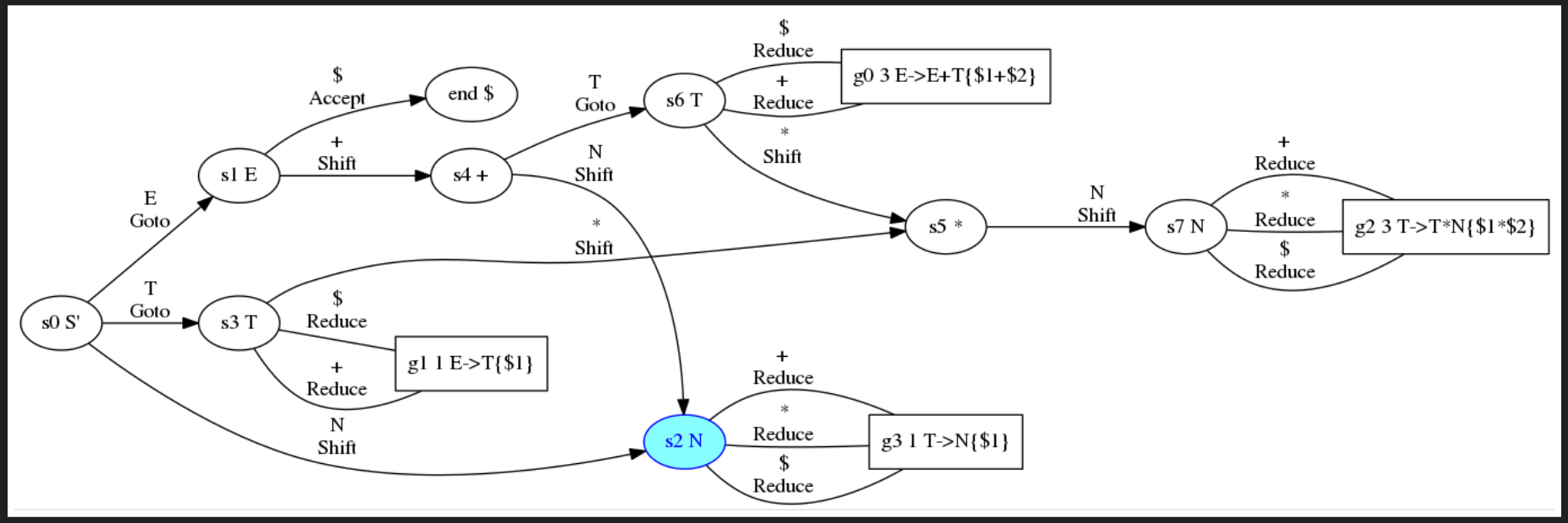
SHIFT S2

```
inputs 2 * 3 * 4 $ ステータスに2をpush 入力2を結果に移動
status 0
results
```



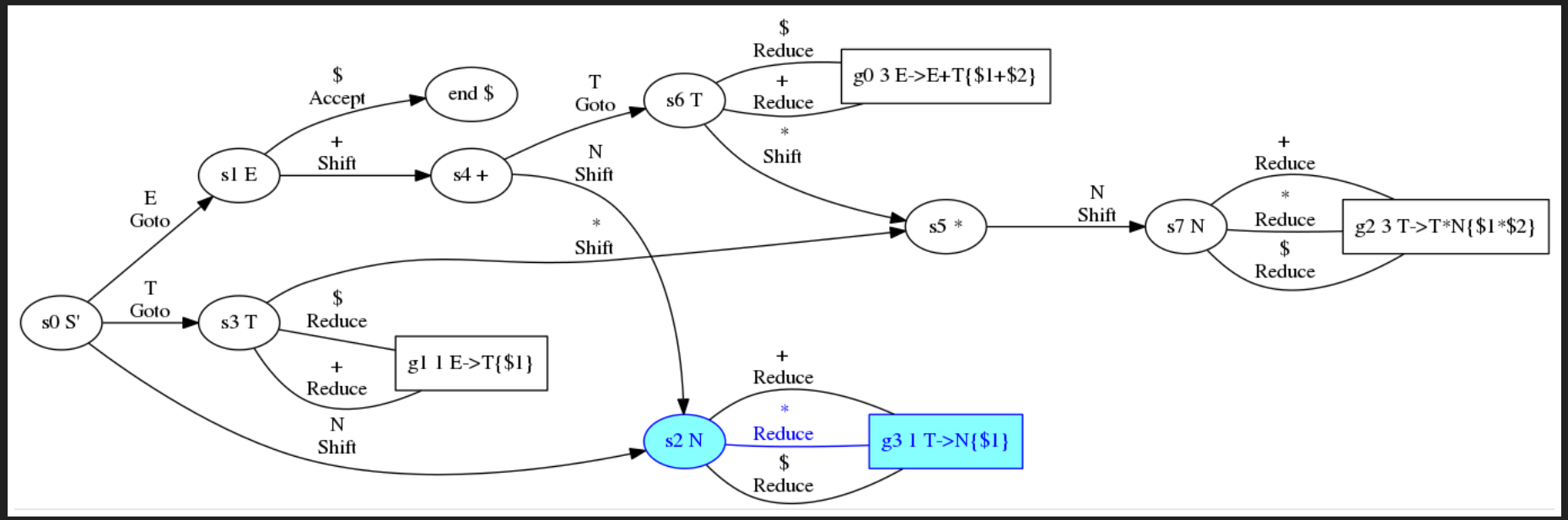
REDUCE G3

```
inputs * 3 * 4 $
status 2 0
results 2
```



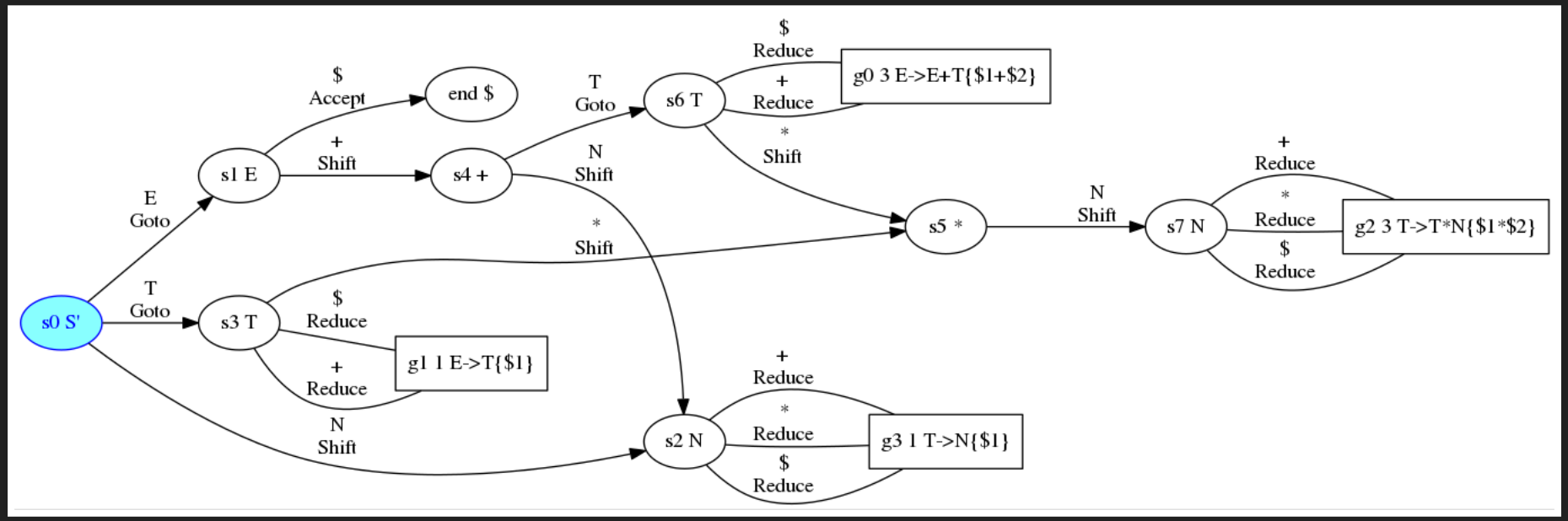
REDUCE G3

```
inputs * 3 * 4 $ 文法g3を見て、1個Pop、{$1}を結果にpush、文法名Tを入力にpush
status 2 0
results 2
```



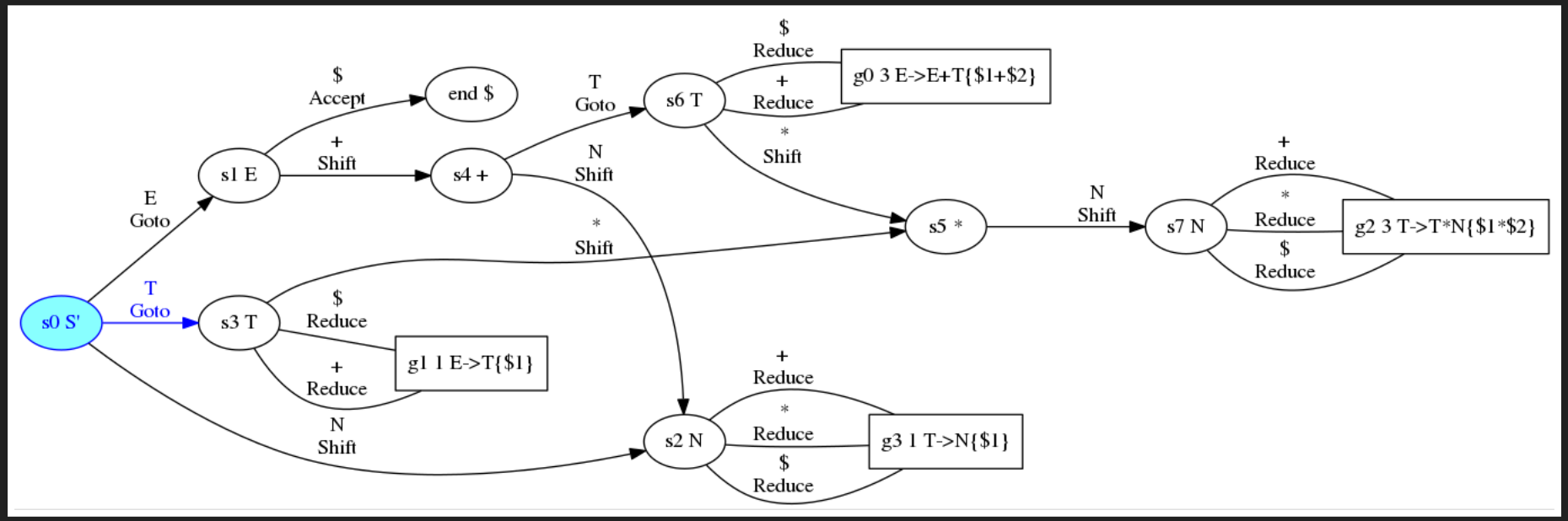
GOTO S3

```
inputs T * 3 * 4 $
status 0
results 2
```



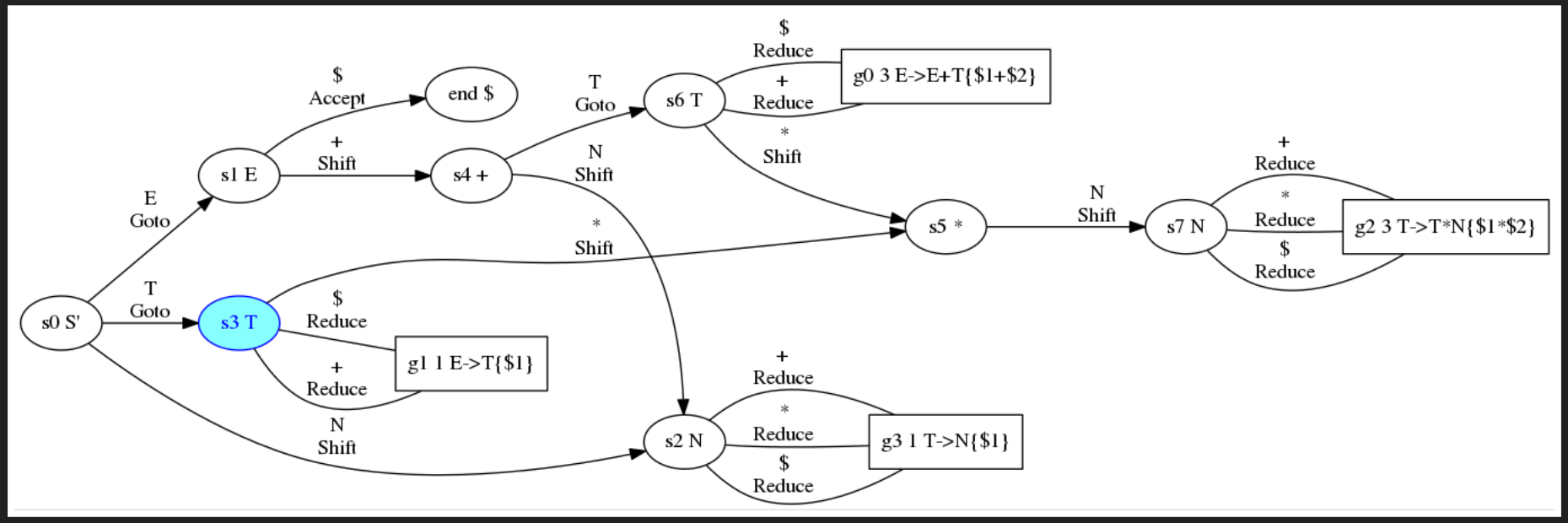
GOTO S3

```
inputs T * 3 * 4 $ ステータスに3をpush,入力Tを捨てる
status 0
results 2
```



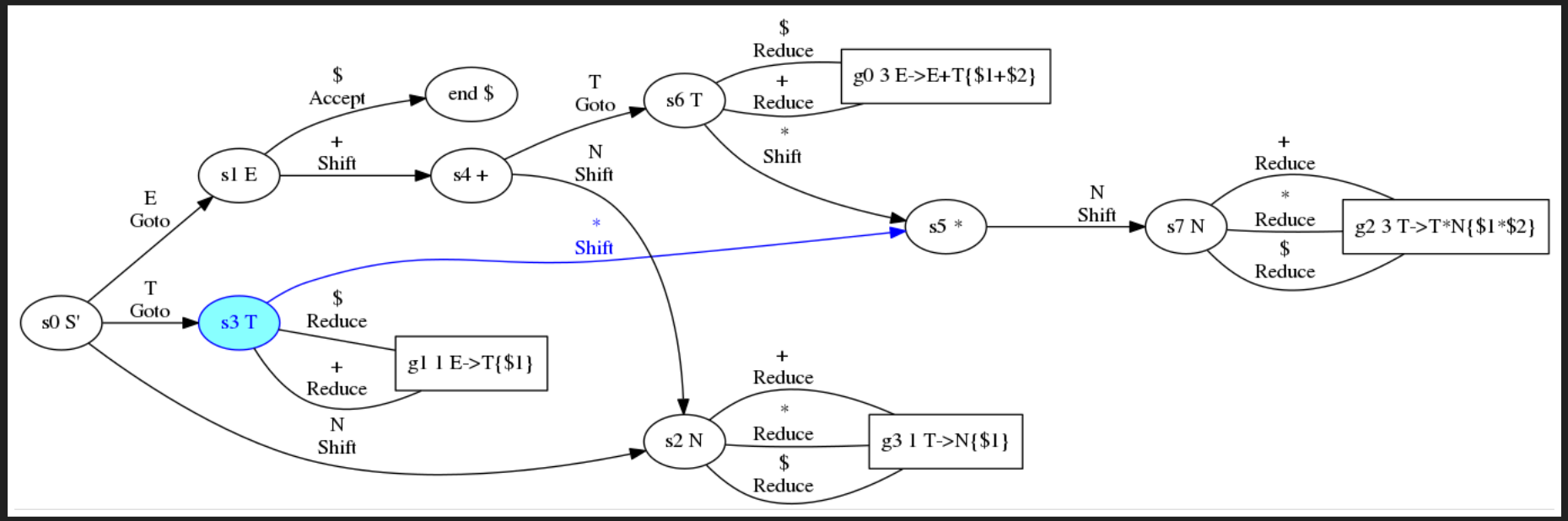
SHIFT S5

```
inputs * 3 * 4 $
status 3 0
results 2
```



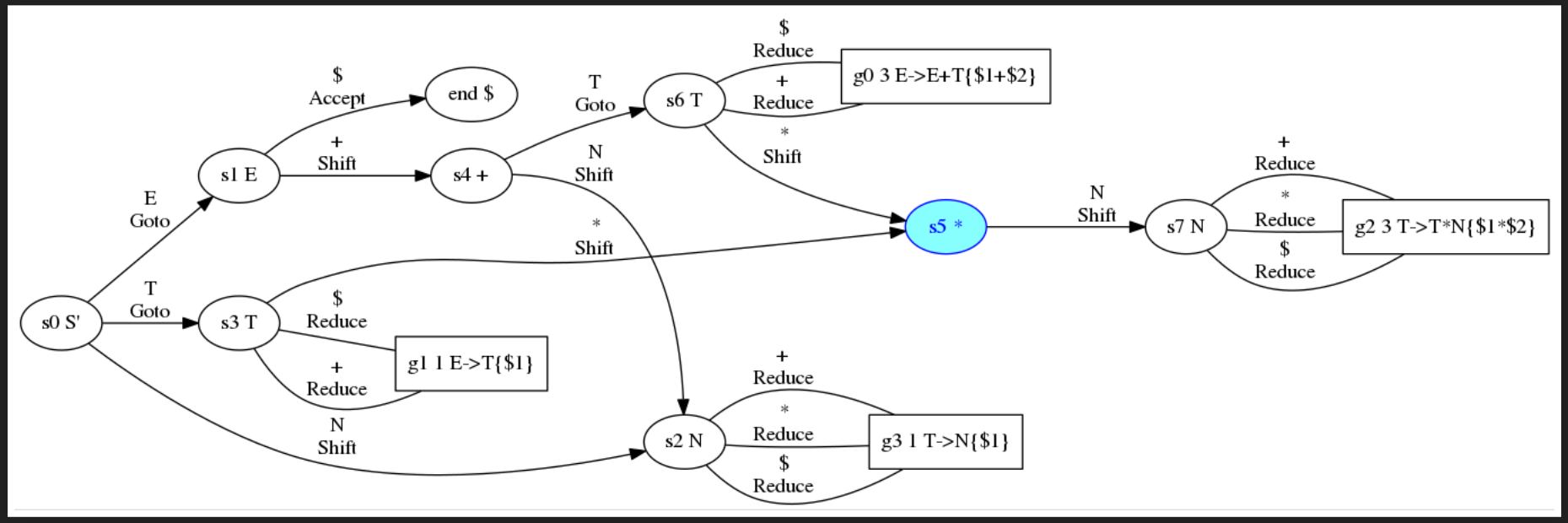
SHIFT S5

```
inputs * 3 * 4 $ ステータスに5をpush 入力*を結果に移動
status 3 0
results 2
```



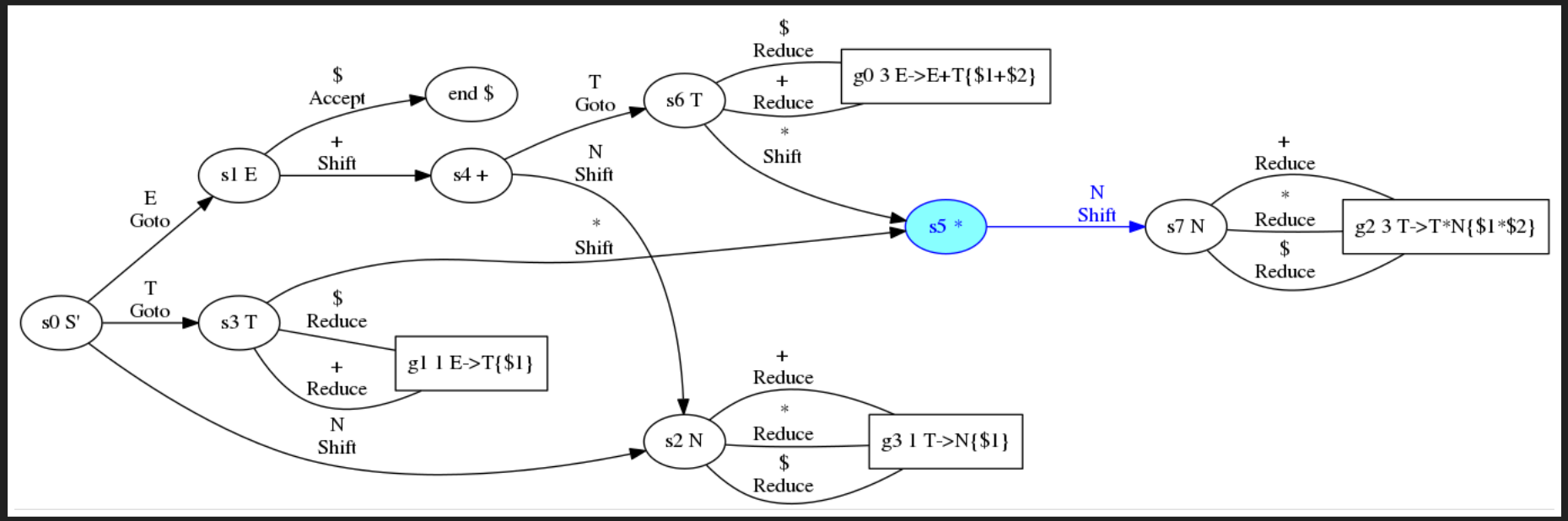
SHIFT S7

```
inputs 3 * 4 $
status 5 3 0
results * 2
```



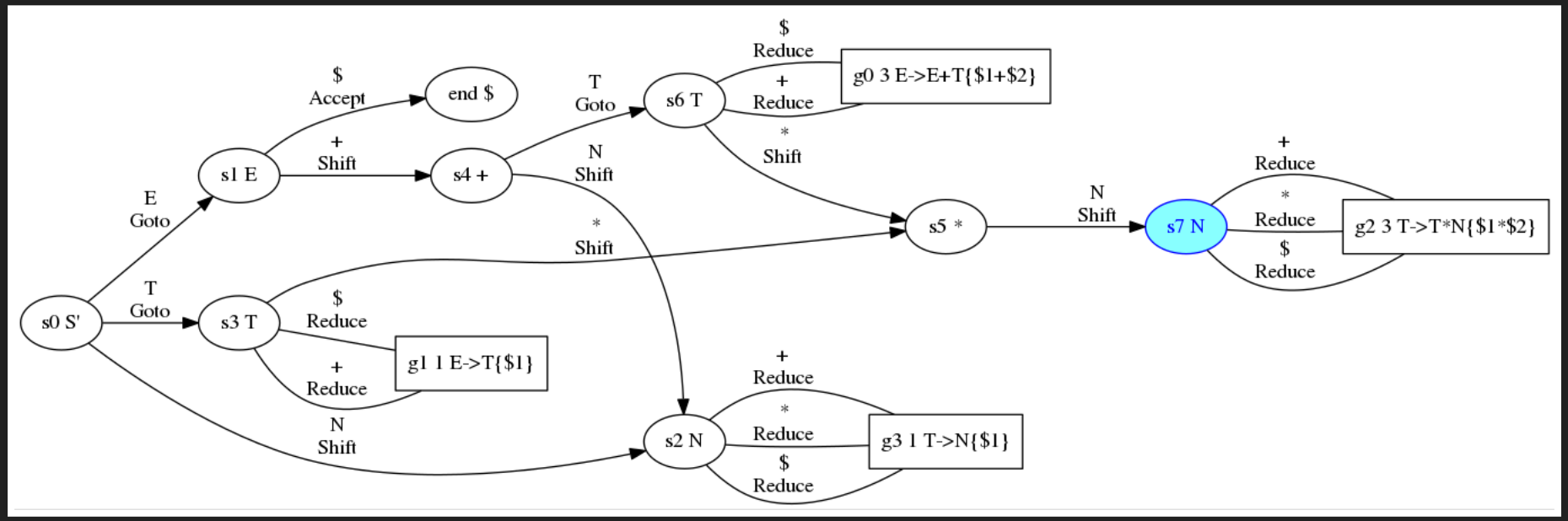
SHIFT S7

```
inputs 3 * 4 $ ステータスに7をpush 入力3を結果に移動
status 5 3 0
results * 2
```



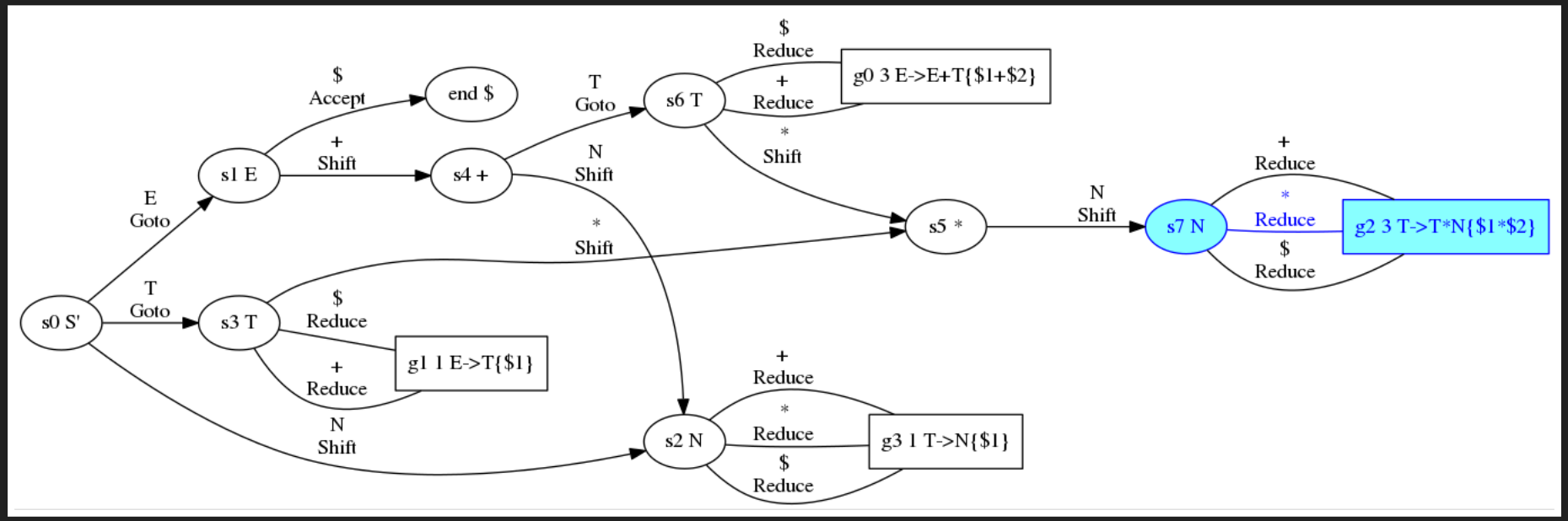
REDUCE G2

```
inputs * 4 $
status 7 5 3 0
results 3 * 2
```



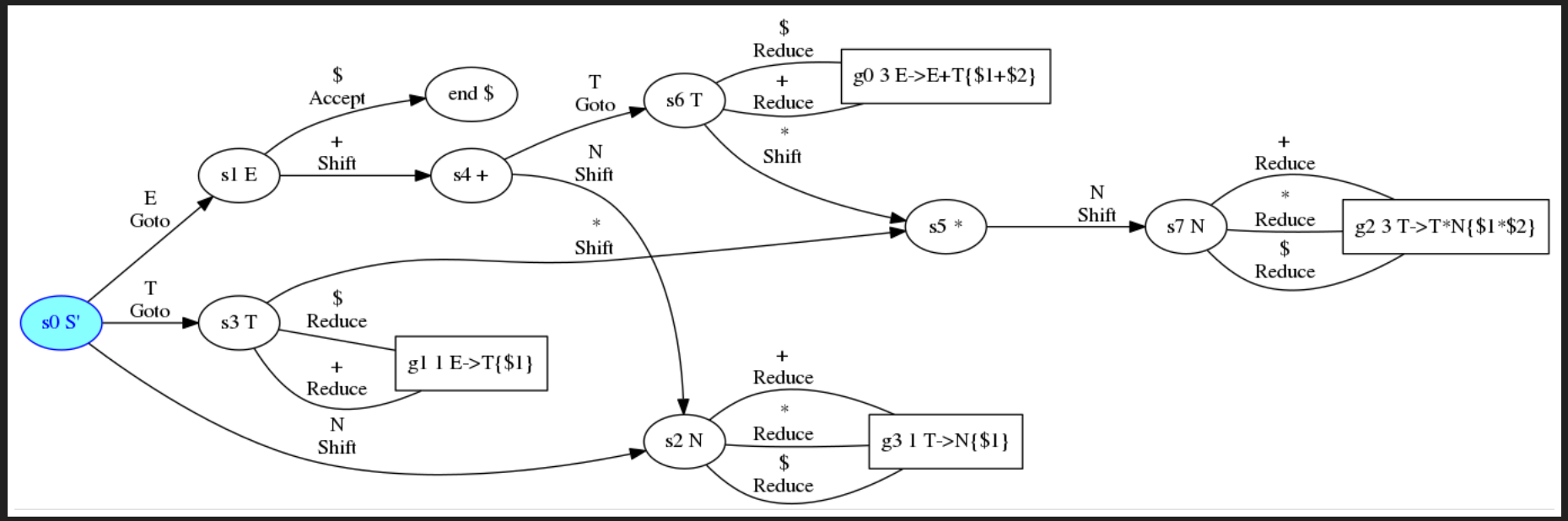
REDUCE G2

```
inputs * 4 $ 文法g2を見て、3個Pop、{$1*$2}を結果にpush、文法名Tを入力にpush
status 7 5 3 0
results 3 * 2
```



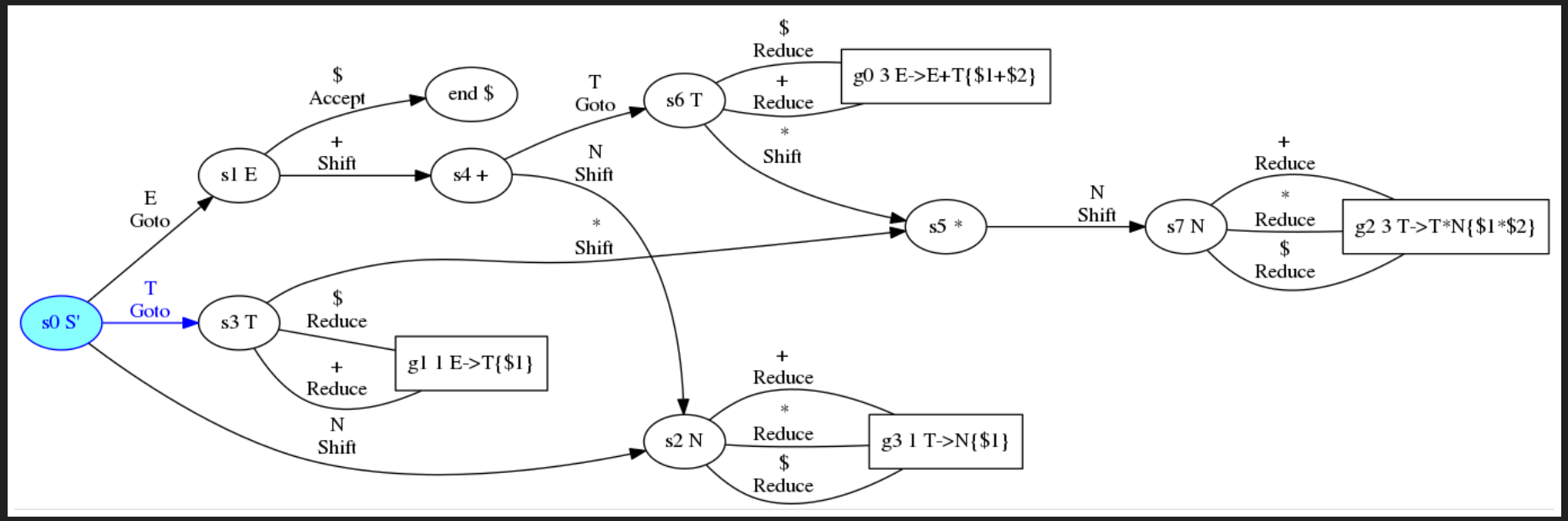
GOTO S3

```
inputs T * 4 $
status 0
results 6
```



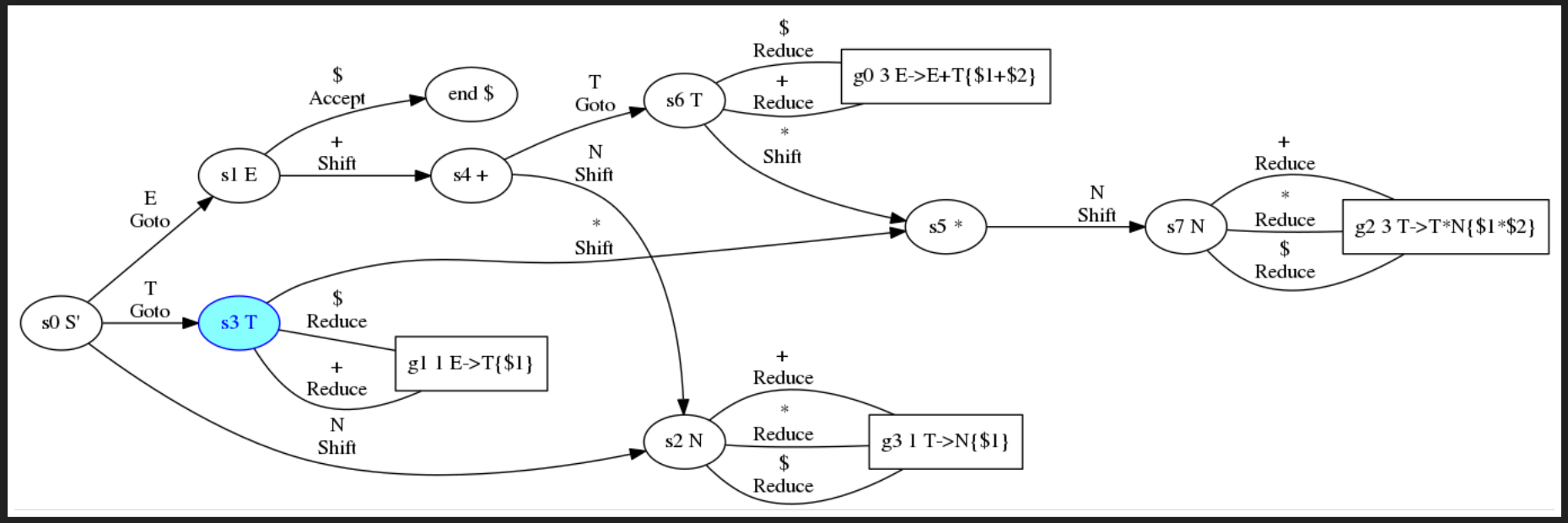
GOTO S3

```
inputs T * 4 $ ステータスに3をpush,入力Tを捨てる
status 0
results 6
```



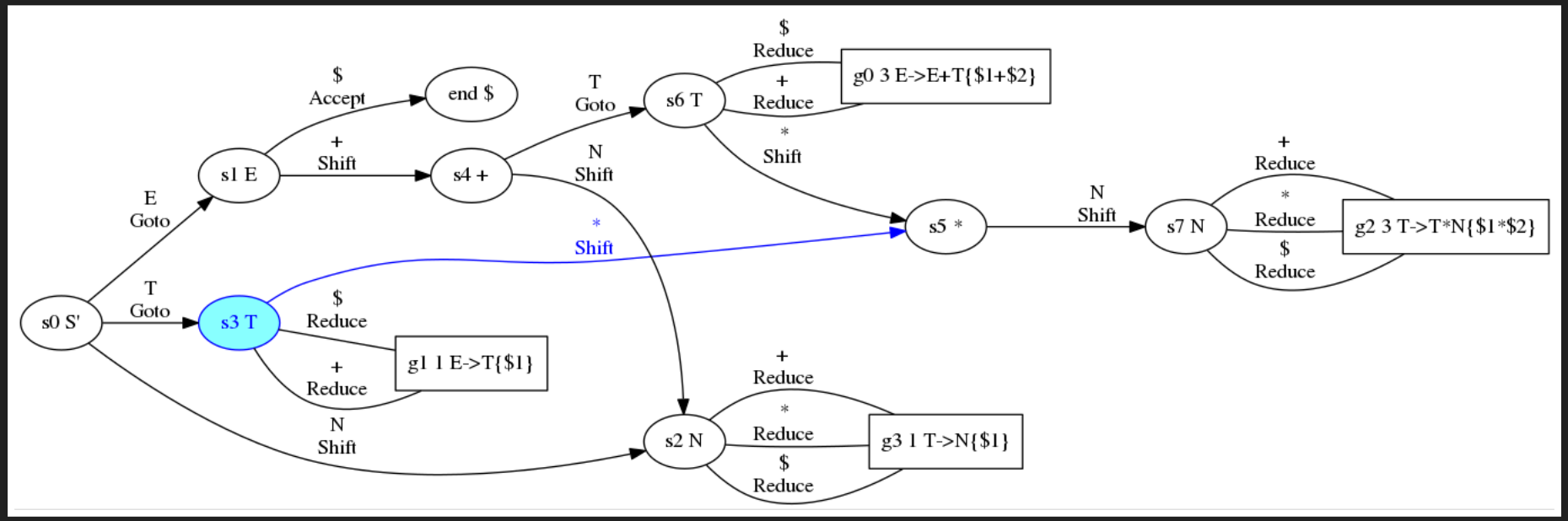
SHIFT S5

```
inputs * 4 $
status 3 0
results 6
```



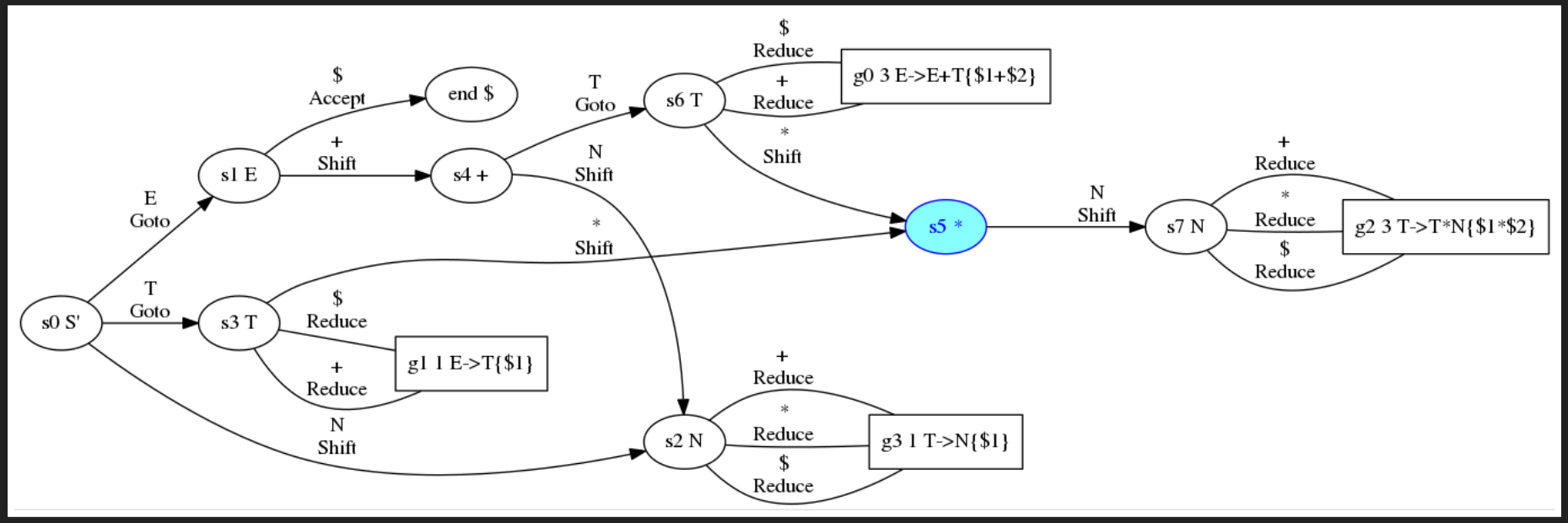
SHIFT S5

```
inputs * 4 $ ステータスに5をpush 入力*を結果に移動
status 3 0
results 6
```



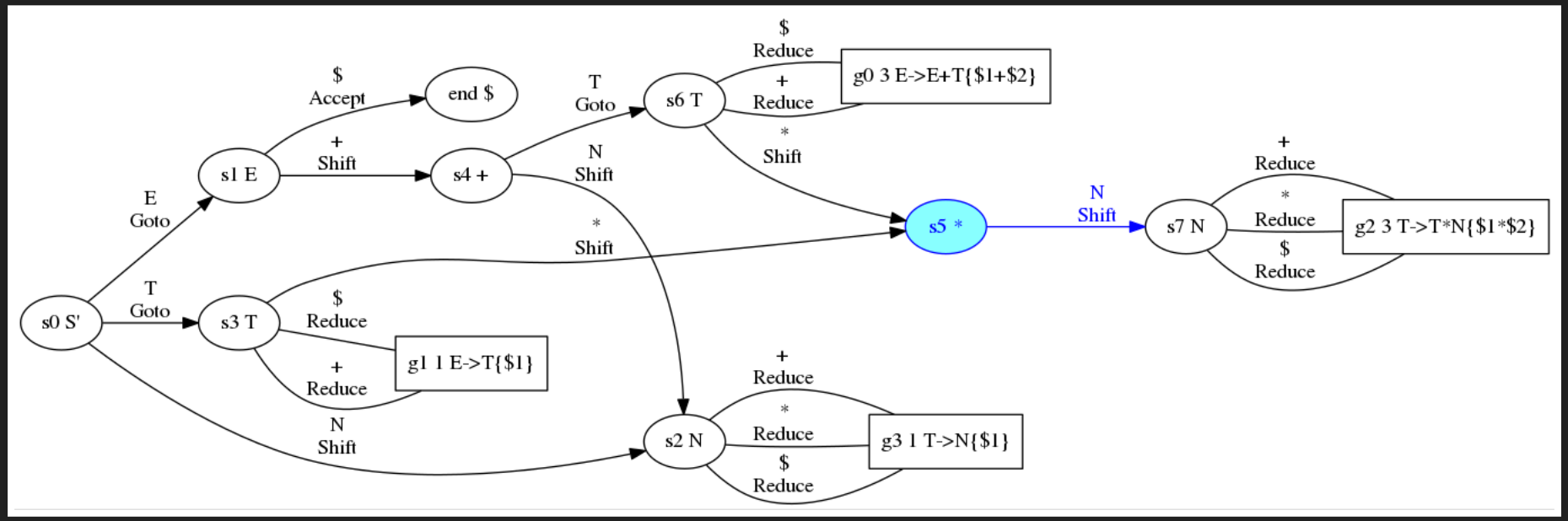
SHIFT S7

```
inputs 4 $
status 5 3 0
results * 6
```



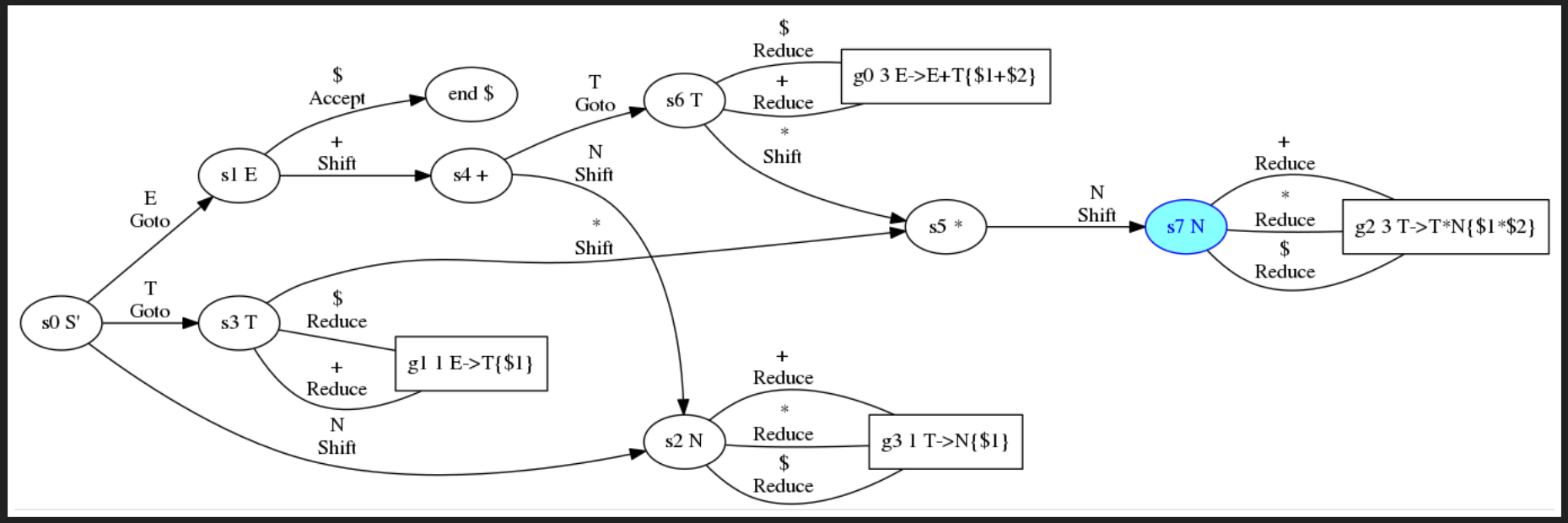
SHIFT S7

```
inputs 4 $ ステータスに7をpush 入力4を結果に移動
status 5 3 0
results * 6
```



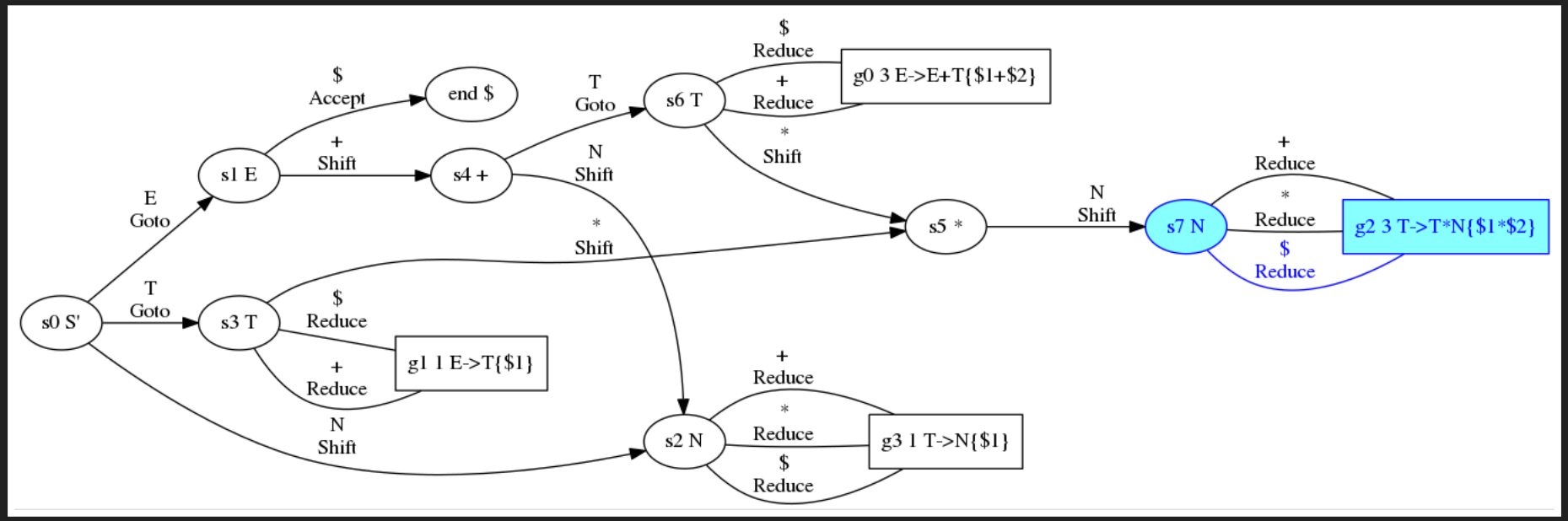
REDUCE G2

```
inputs $
status 7 5 3 0
results 4 * 6
```



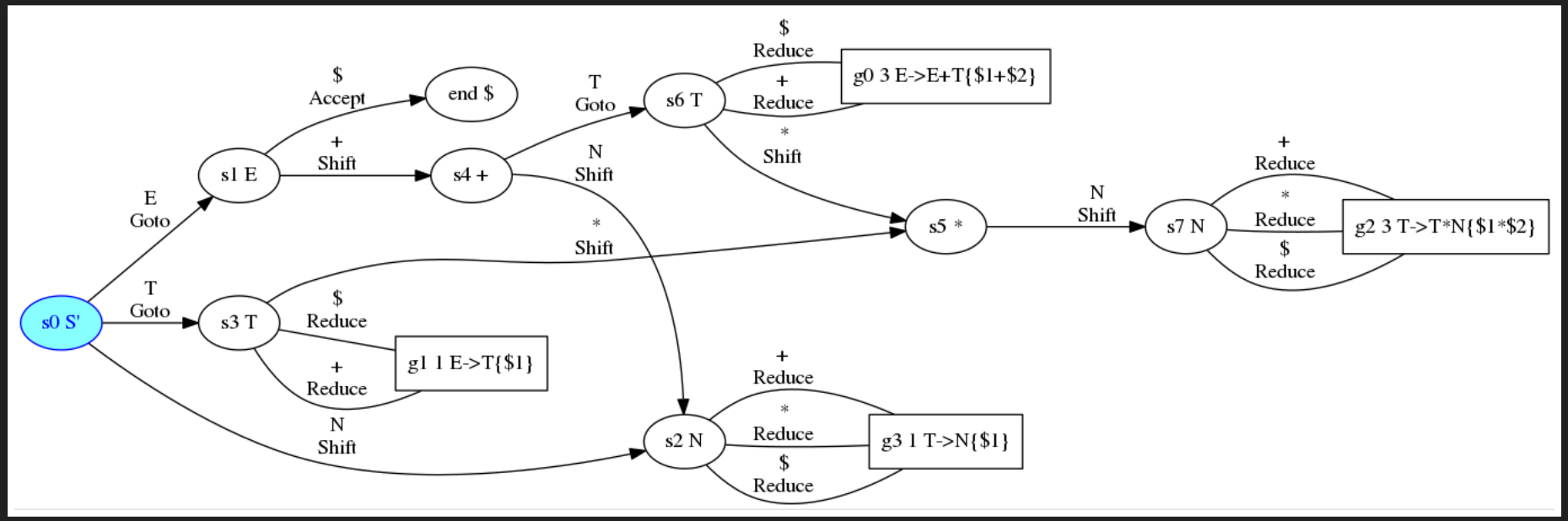
REDUCE G2

```
inputs $ 文法g2を見て、3個Pop、{$1*$2}を結果にpush、文法名Tを入力にpush
status 7 5 3 0
results 4 * 6
```



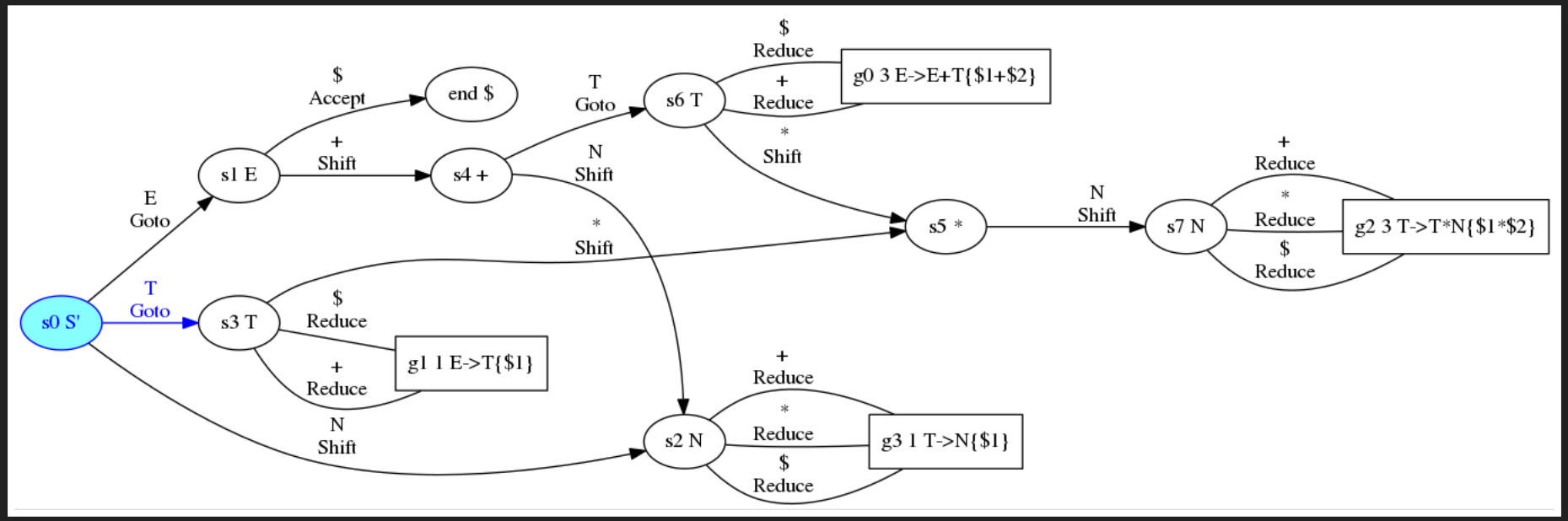
GOTO S3

```
inputs T $
status 0
results 24
```



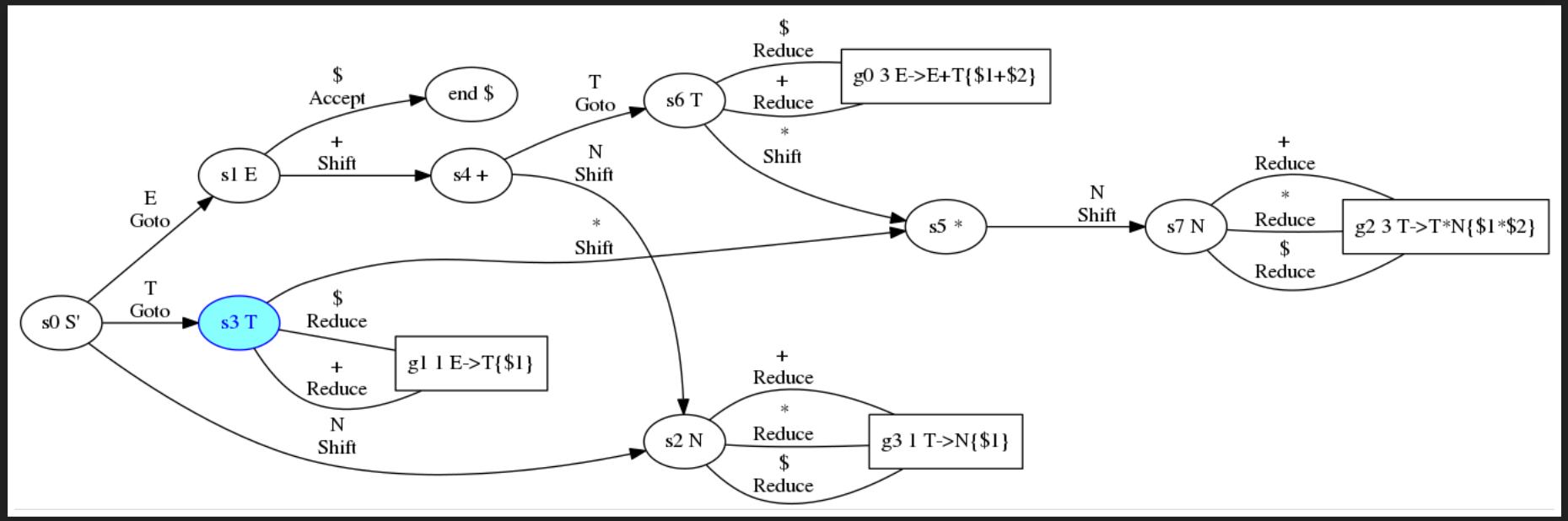
GOTO S3

```
inputs T $ ステータスに3をpush,入力Tを捨てる
status 0
results 24
```



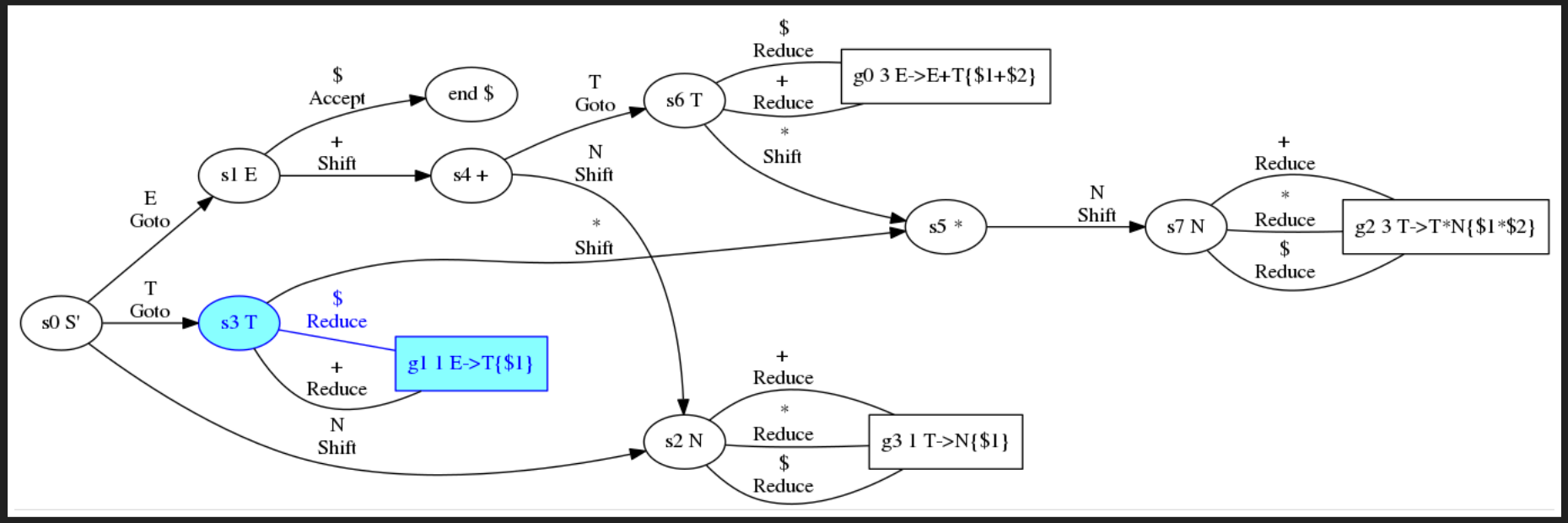
REDUCE G1

```
inputs $
status 3 0
results 24
```



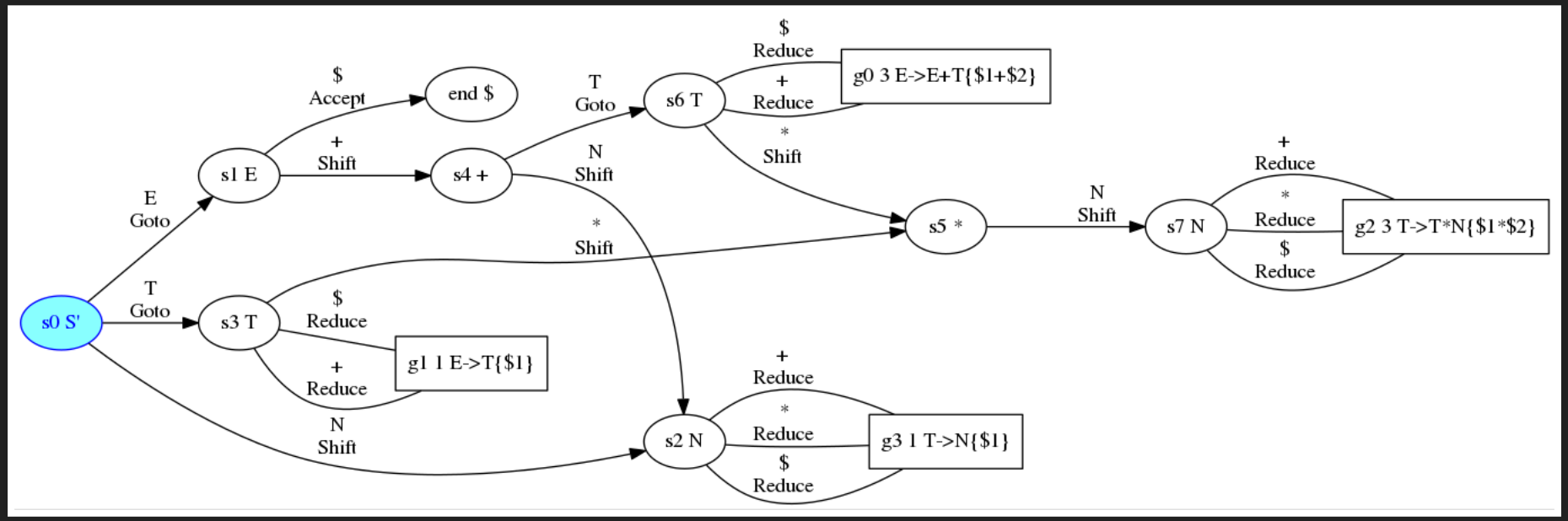
REDUCE G1

```
inputs $ 文法g1を見て、1個Pop、{$1}を結果にpush、文法名Eを入力にpush
status 3 0
results 24
```



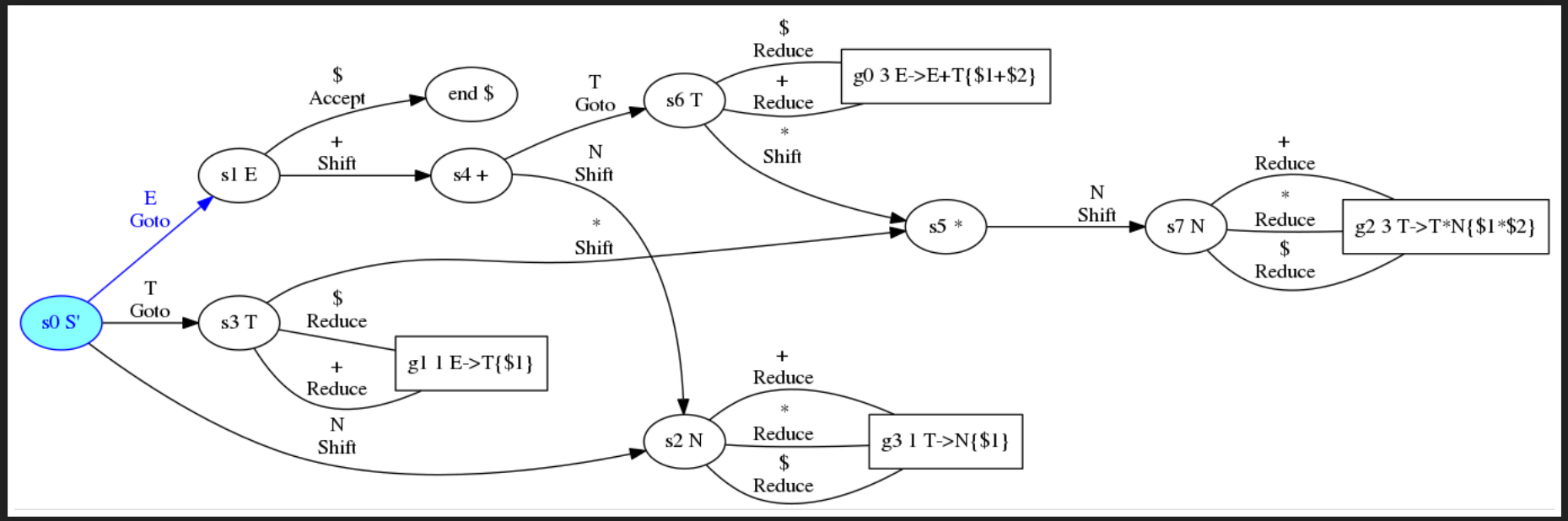
GOTO S1

```
inputs E $
status 0
results 24
```



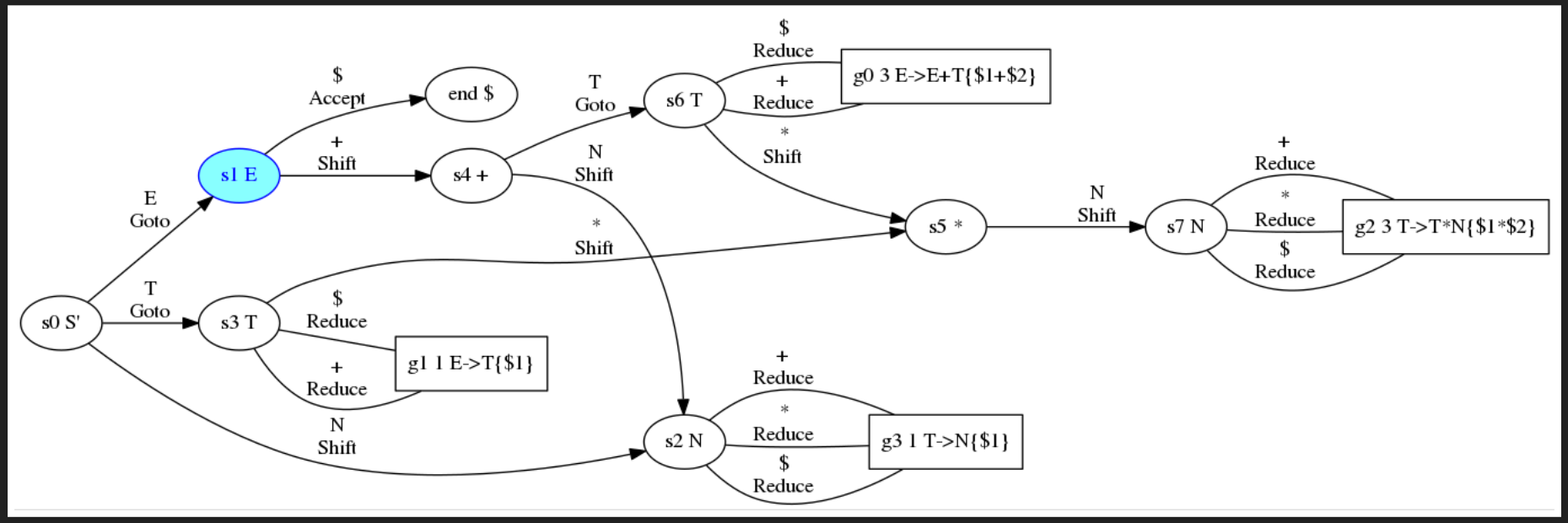
GOTO S1

```
inputs E $ ステータスに1をpush,入力Eを捨てる
status 0
results 24
```



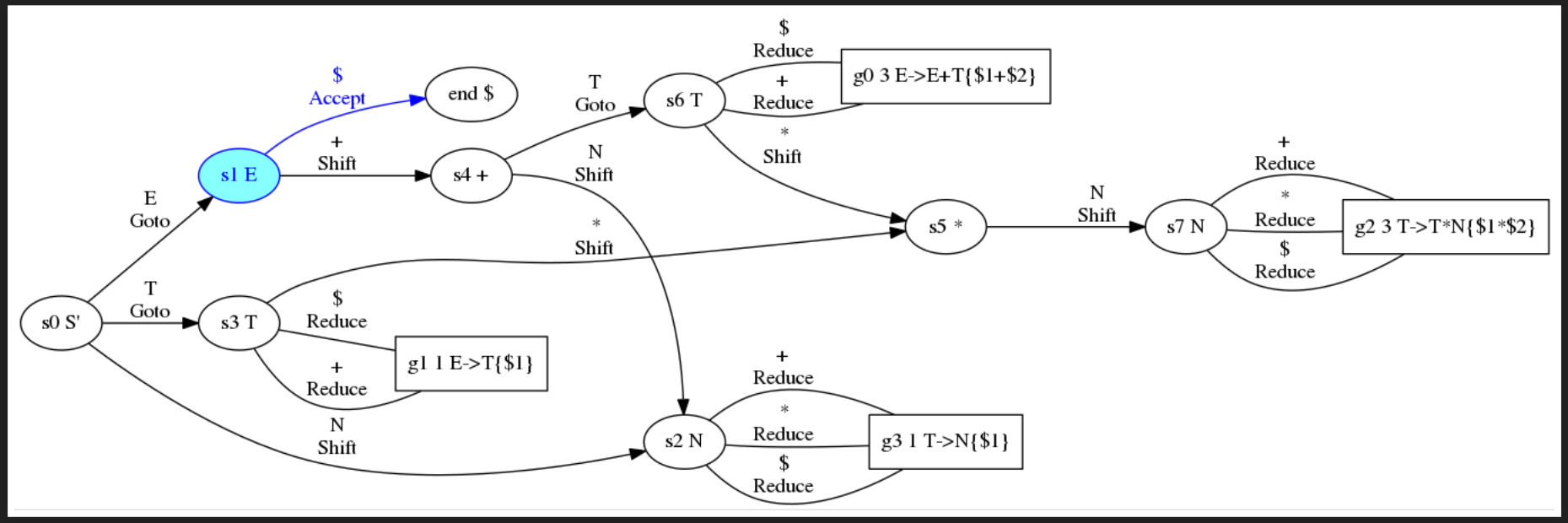
ACCEPT

```
inputs $
status 1 0
results 24
```



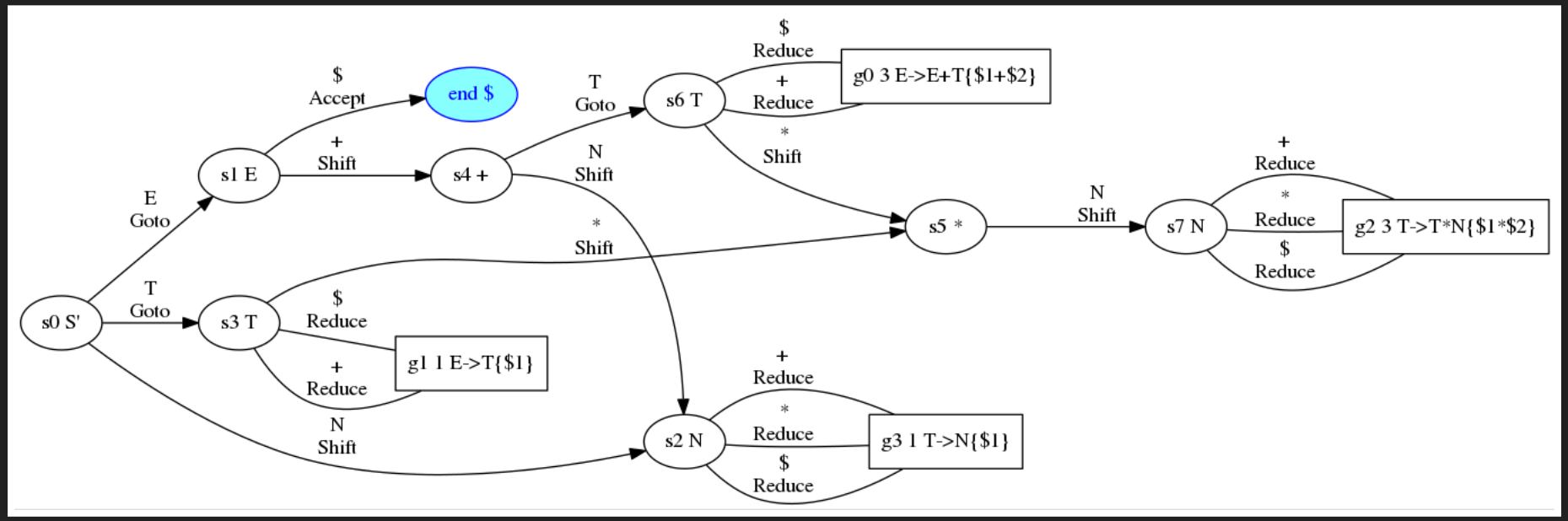
ACCEPT

```
inputs $ アクセプト
status 1 0
results 24
```



ACCEPT

```
inputs $ 結果 24 です
status 1 0
results 24
```

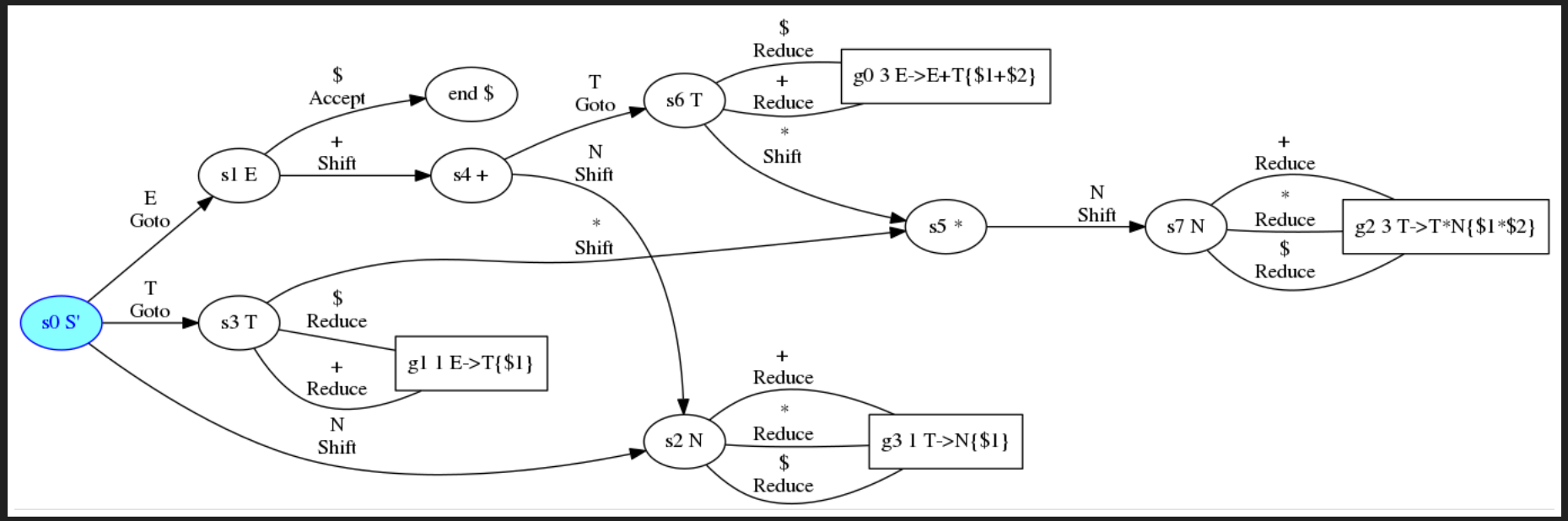


RESULT 2*3*4 = 24

1+2

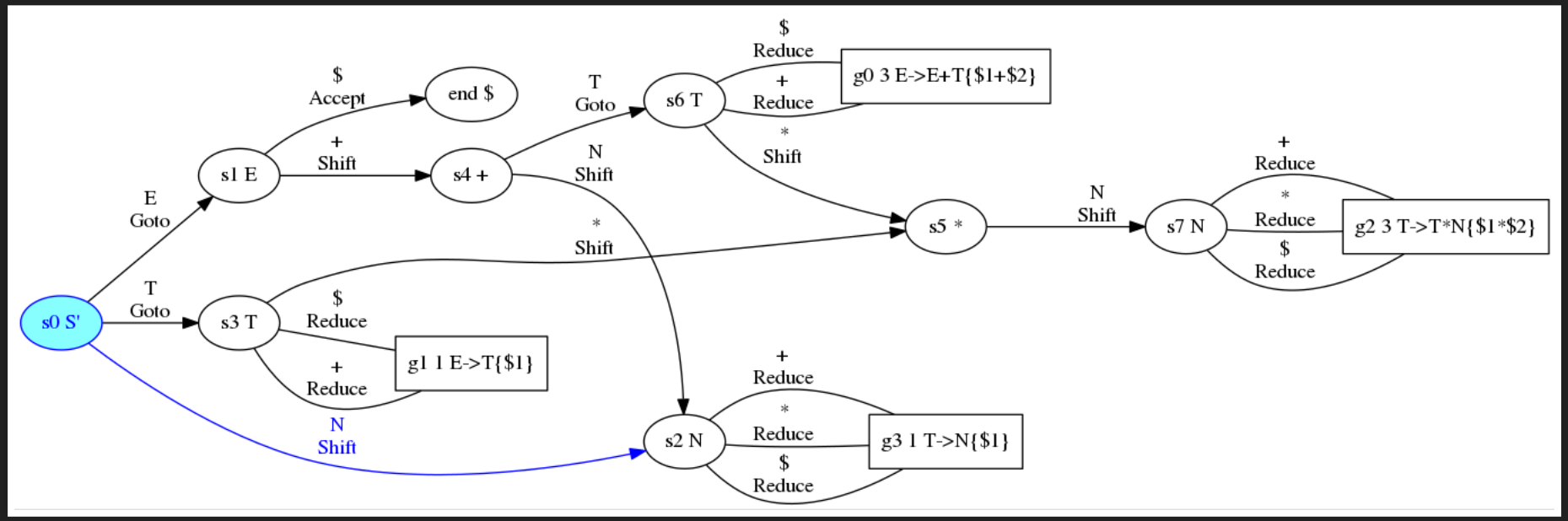
SHIFT S2

```
inputs 1 + 2 $
status 0
results
```



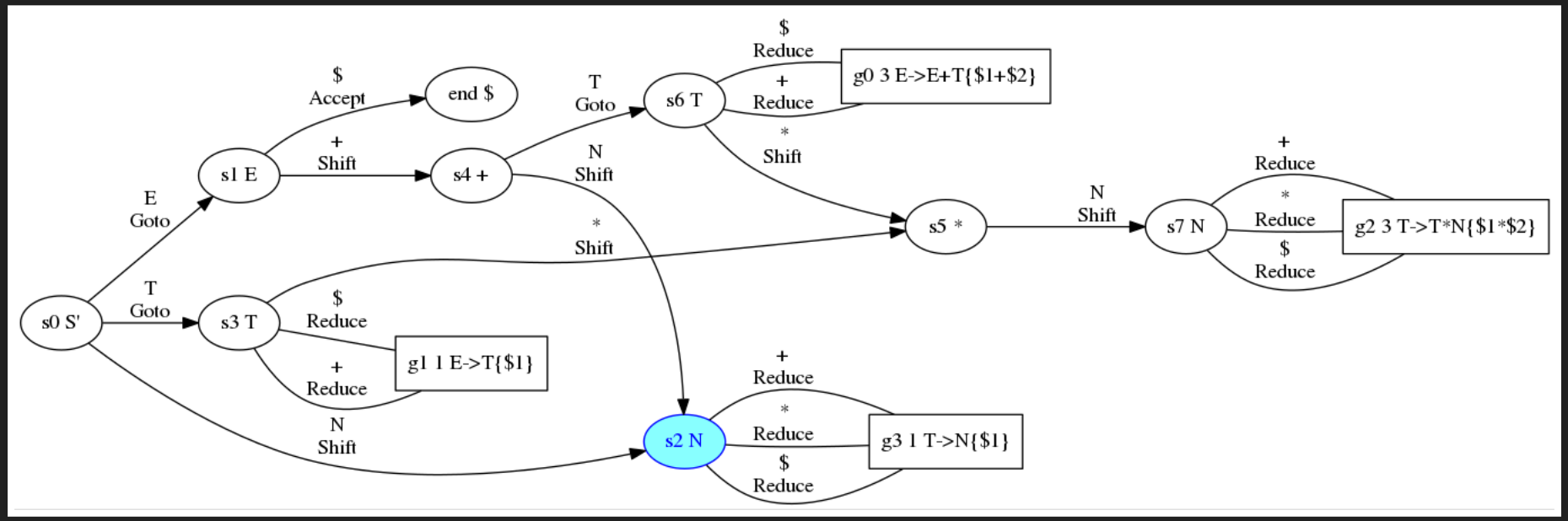
SHIFT S2

```
inputs 1 + 2 $ ステータスに2をpush 入力1を結果に移動
status 0
results
```



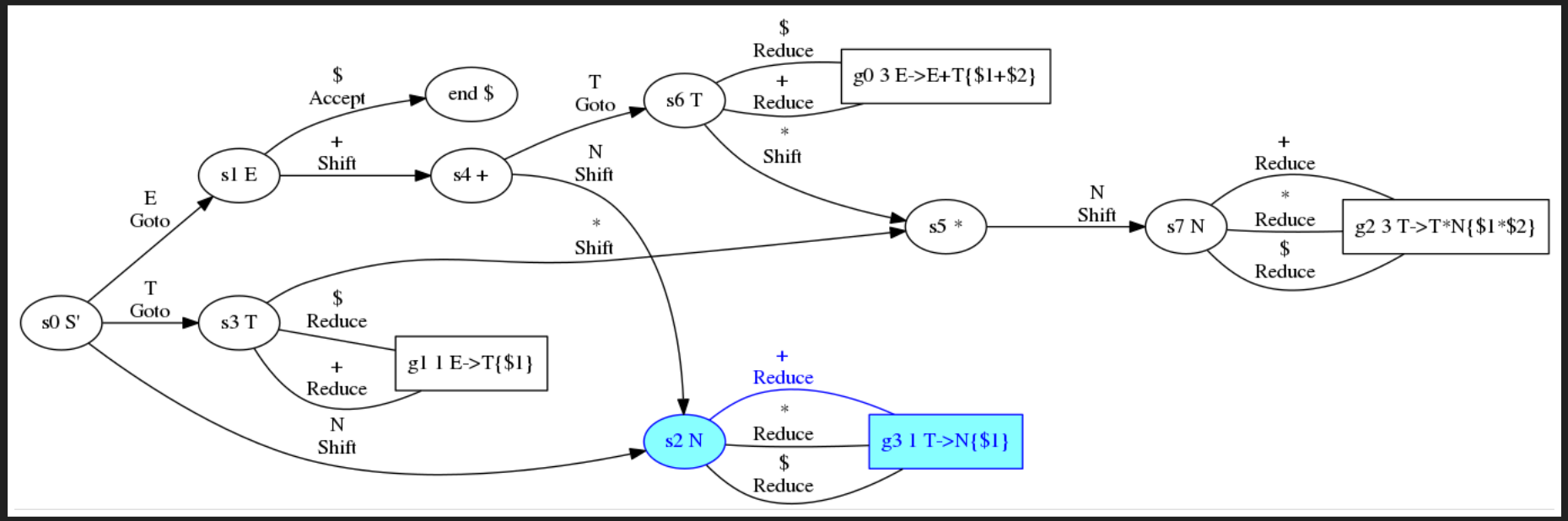
REDUCE G3

```
inputs + 2 $
status 2 0
results 1
```



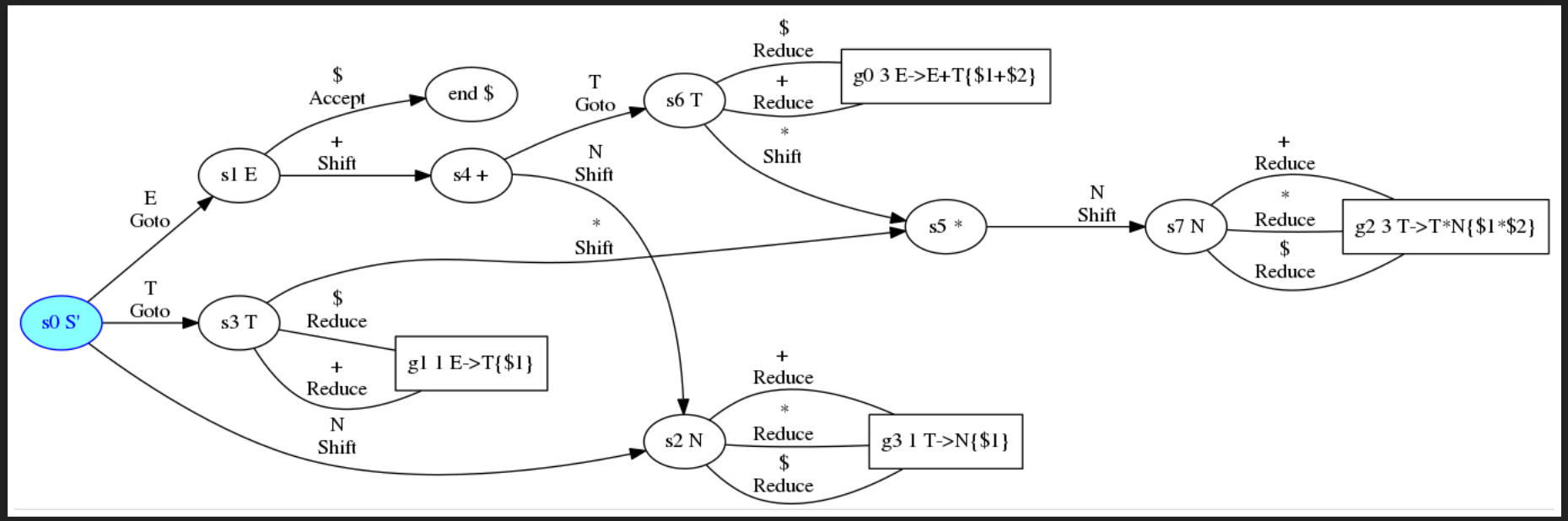
REDUCE G3

```
inputs + 2 $ 文法g3を見て、1個Pop、{$1}を結果にpush、文法名Tを入力にpush
status 2 0
results 1
```



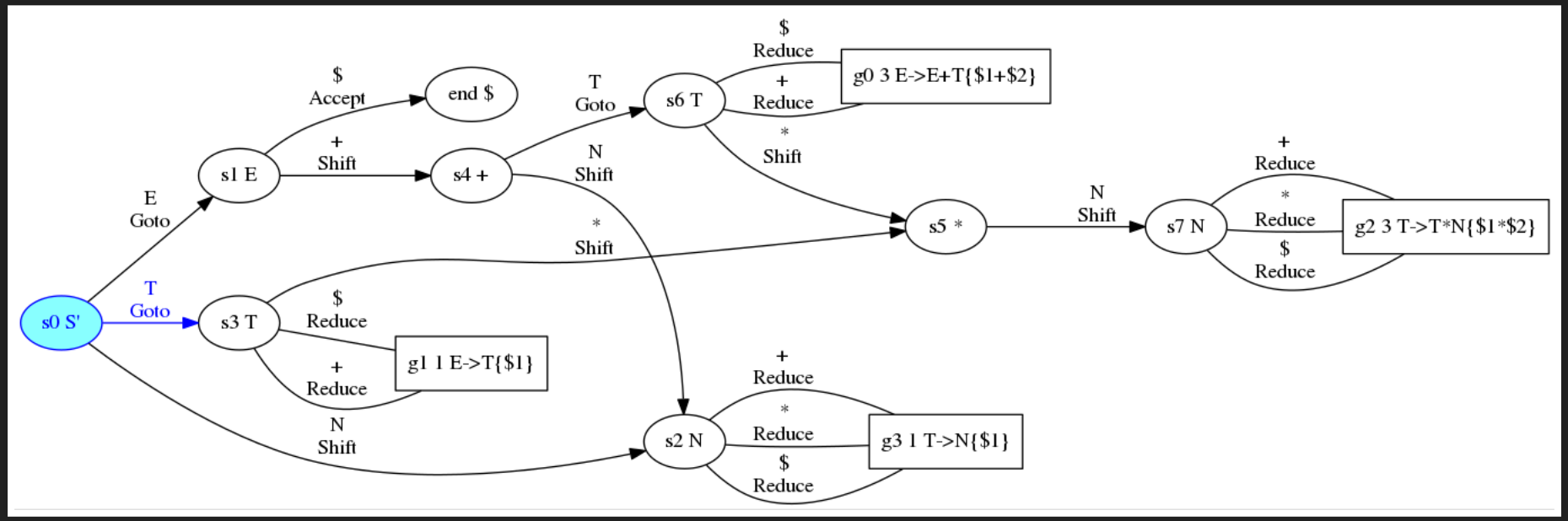
GOTO S3

```
inputs T + 2 $
status 0
results 1
```



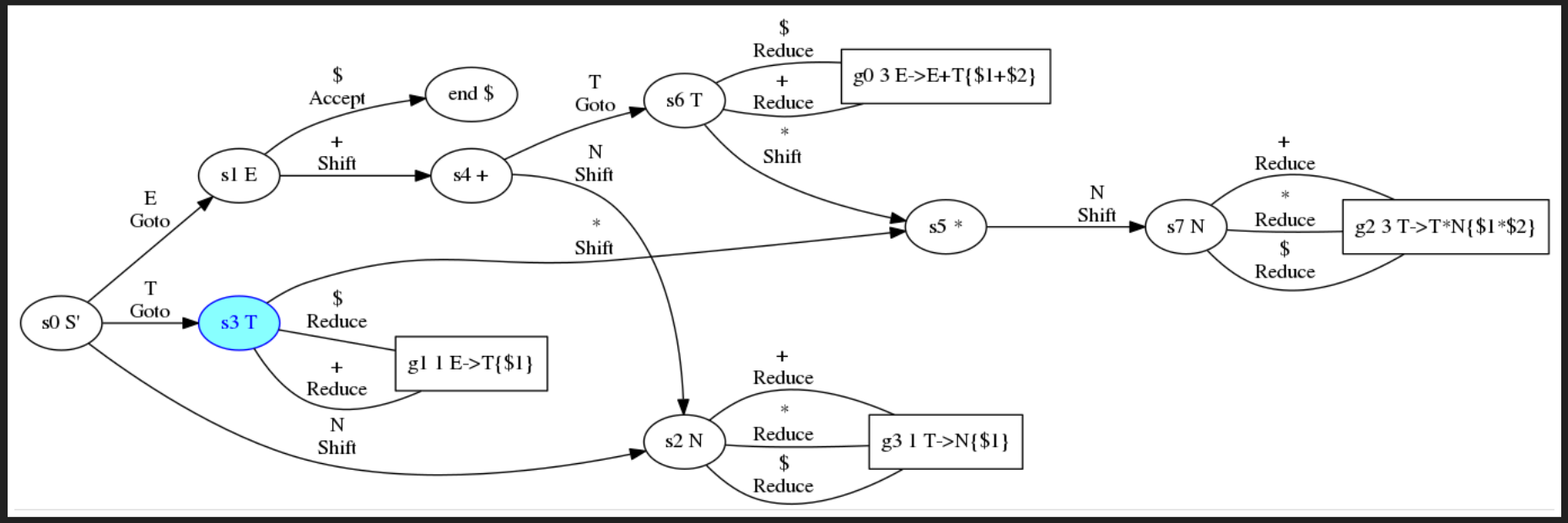
GOTO S3

```
inputs T + 2 $ ステータスに3をpush,入力Tを捨てる
status 0
results 1
```



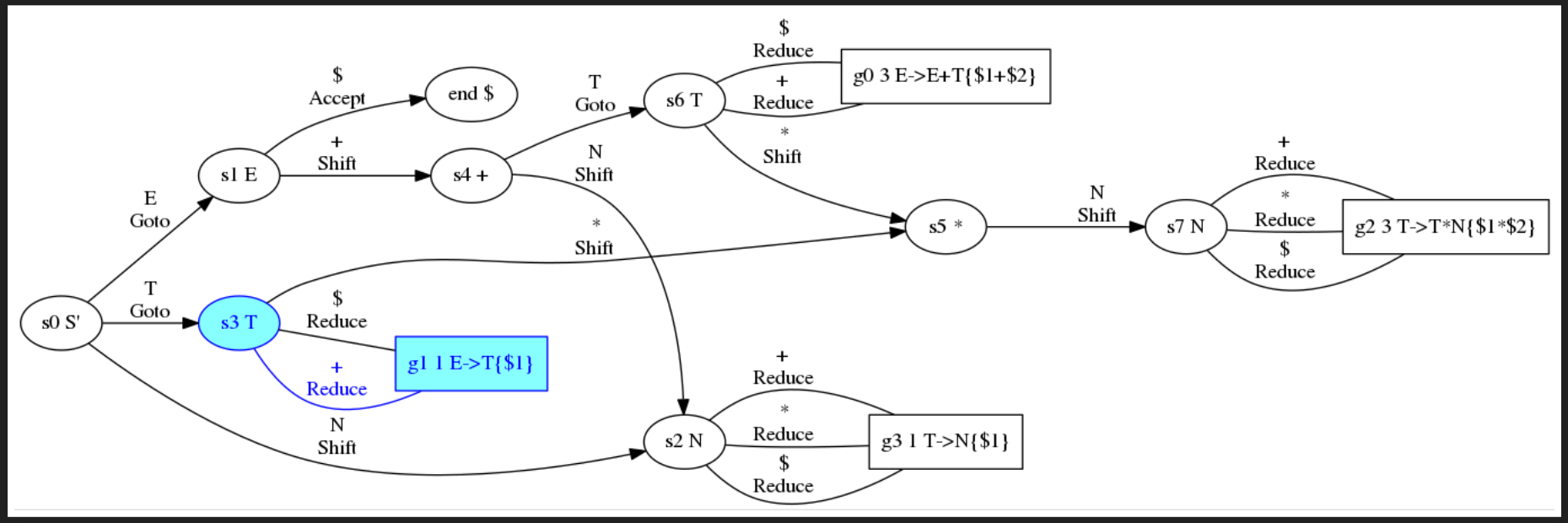
REDUCE G1

```
inputs + 2 $
status 3 0
results 1
```



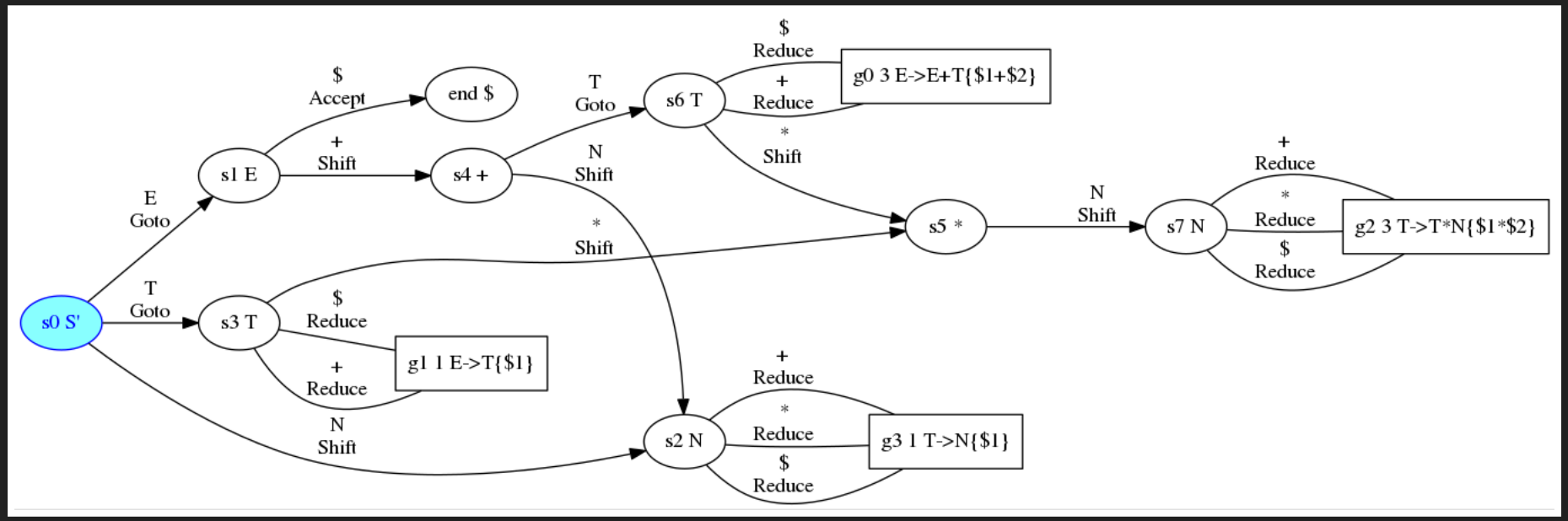
REDUCE G1

```
inputs + 2 $ 文法g1を見て、1個Pop、{$1}を結果にpush、文法名Eを入力にpush
status 3 0
results 1
```



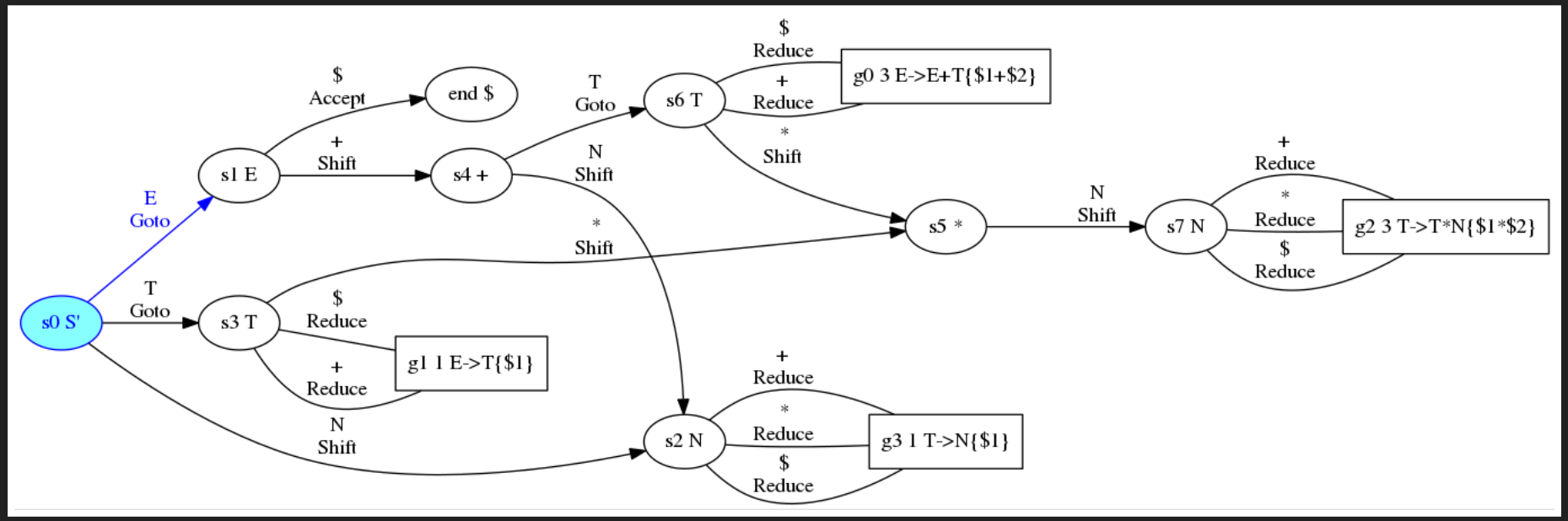
GOTO S1

```
inputs E + 2 $
status 0
results 1
```



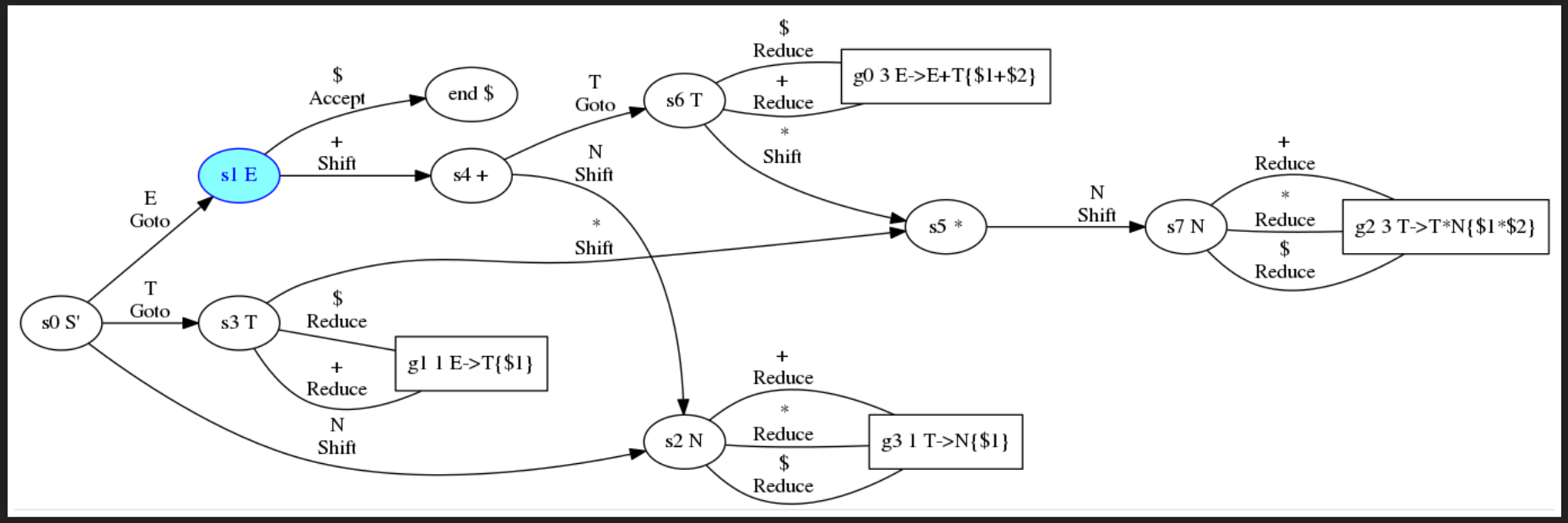
GOTO S1

```
inputs E + 2 $ ステータスに1をpush,入力Eを捨てる
status 0
results 1
```



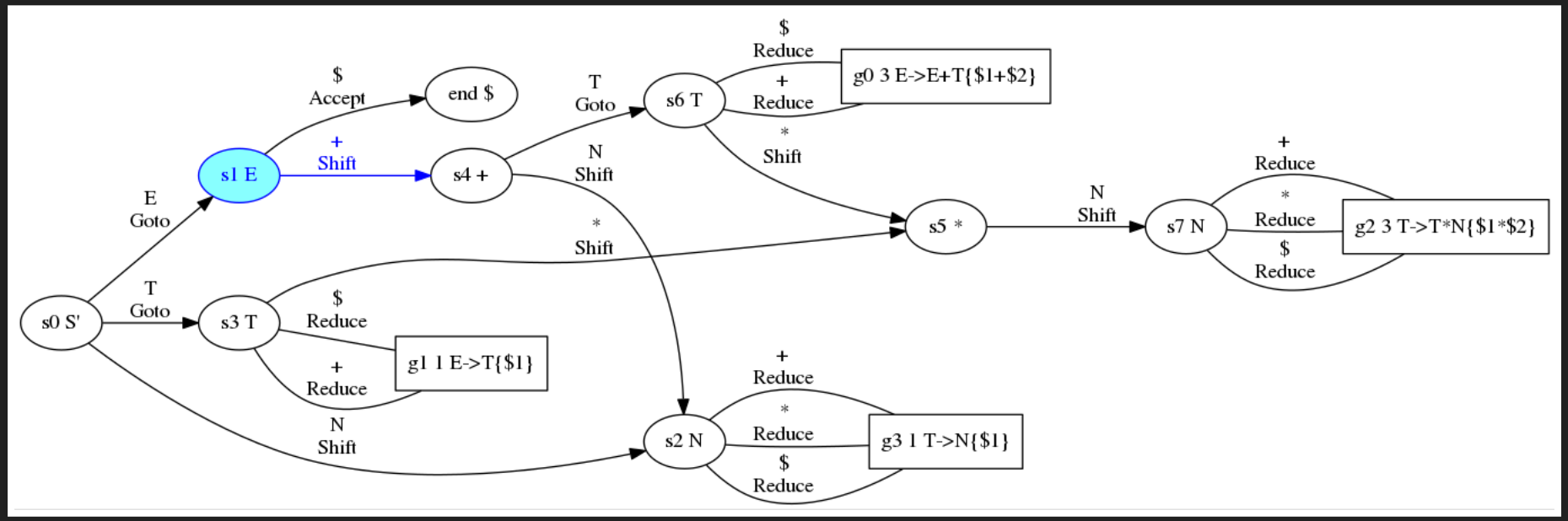
SHIFT S4

```
inputs + 2 $
status 1 0
results 1
```



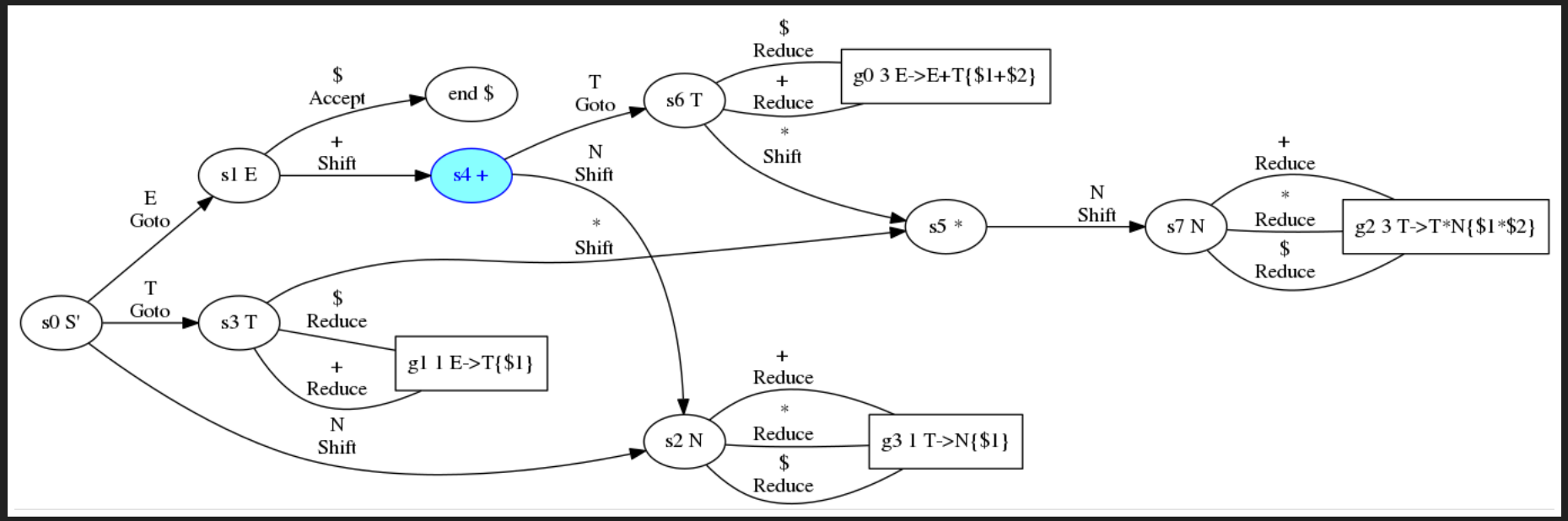
SHIFT S4

```
inputs + 2 $ ステータスに4をpush 入力+を結果に移動
status 1 0
results 1
```



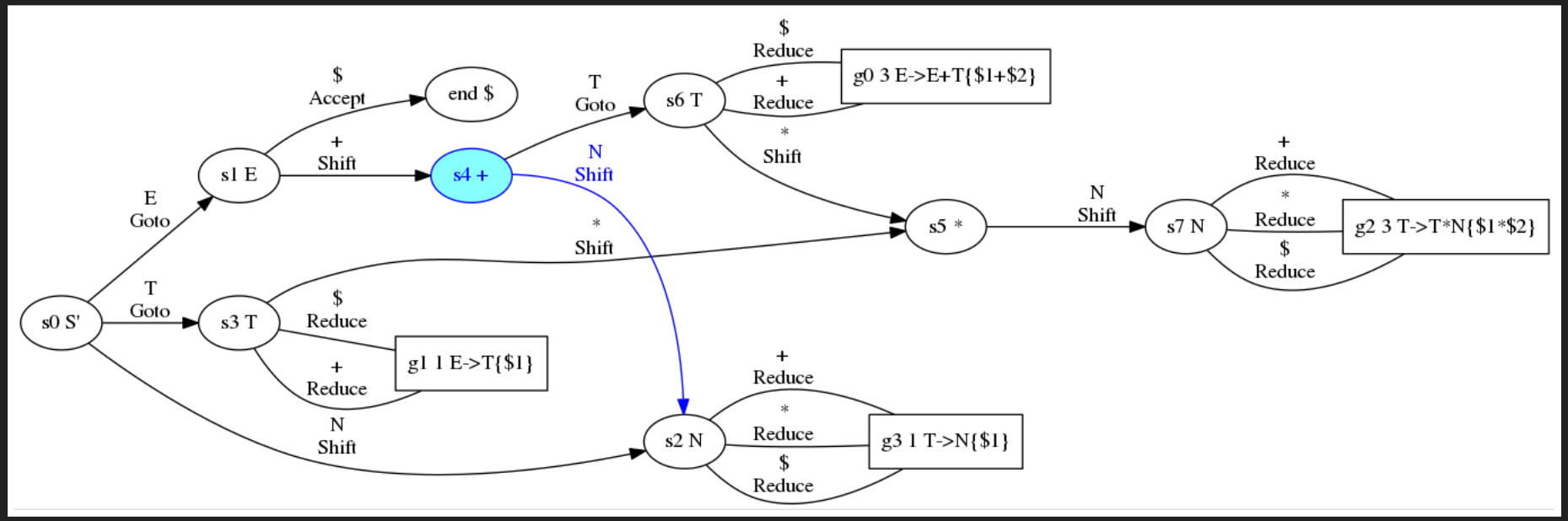
SHIFT S2

```
inputs 2 $
status 4 1 0
results + 1
```



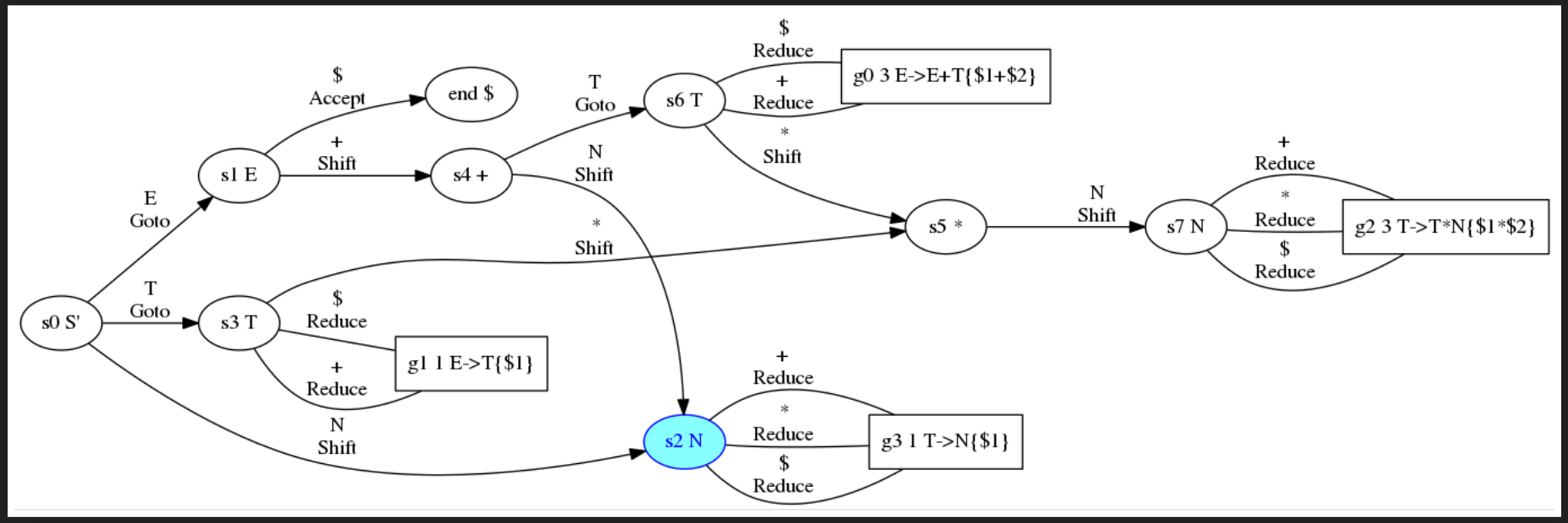
SHIFT S2

```
inputs 2 $ ステータスに2をpush 入力2を結果に移動
status 4 1 0
results + 1
```



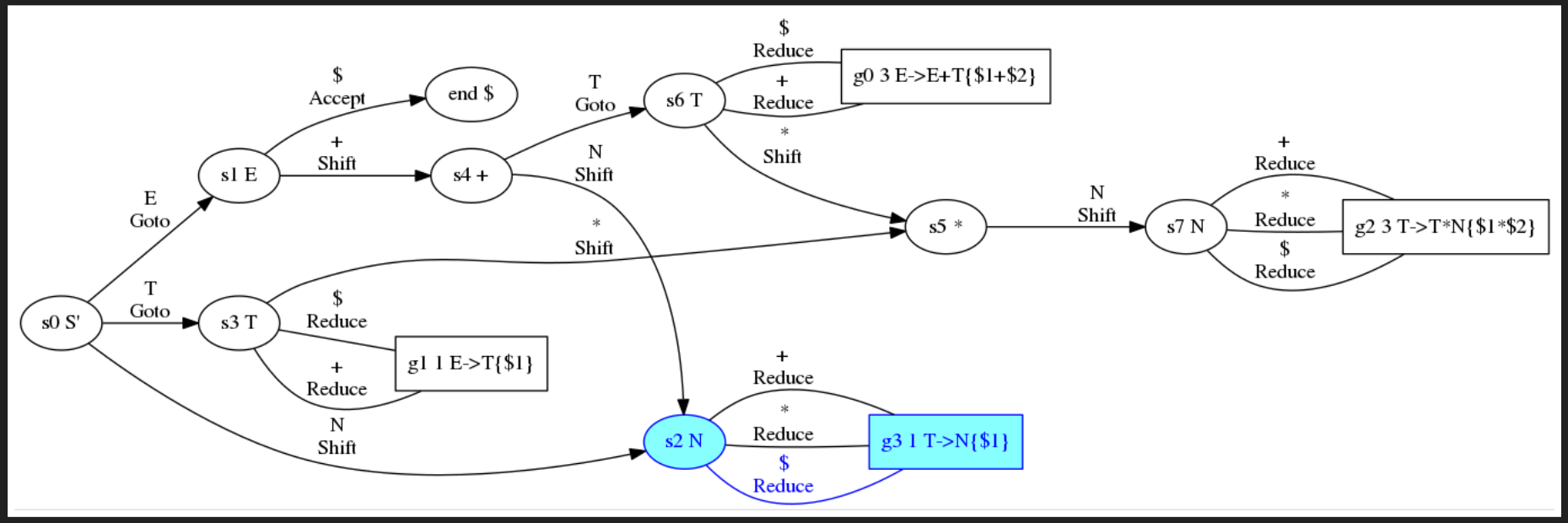
REDUCE G3

```
inputs $
status 2 4 1 0
results 2 + 1
```



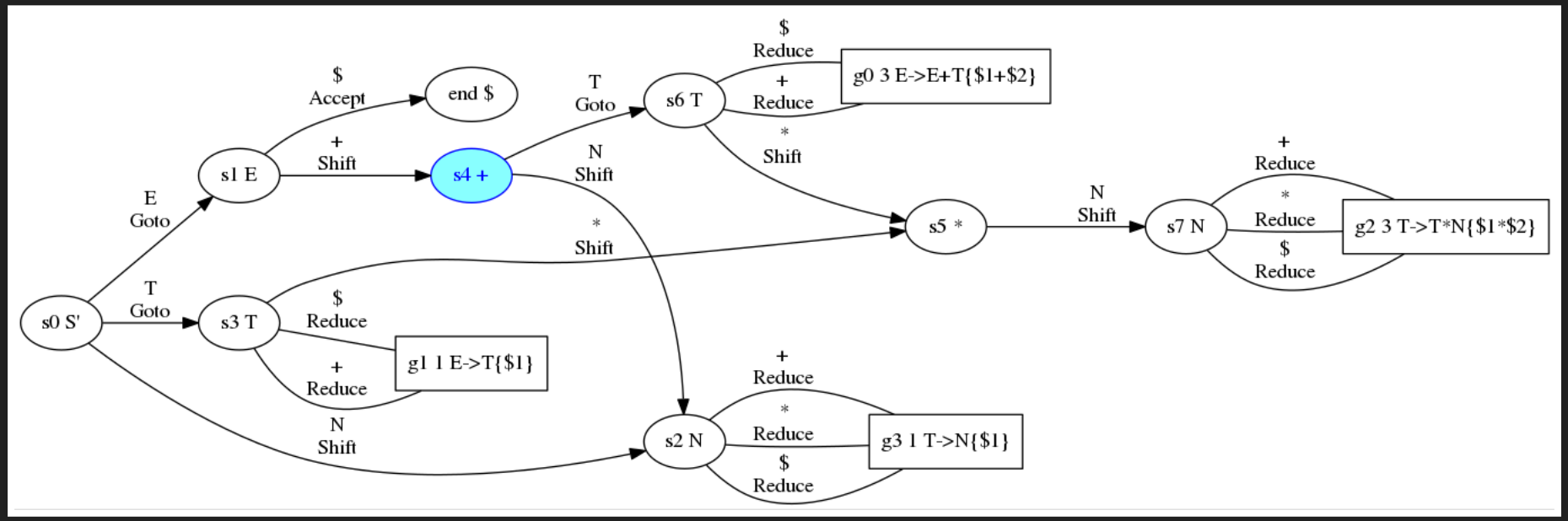
REDUCE G3

```
inputs $ 文法g3を見て、1個Pop、{$1}を結果にpush、文法名Tを入力にpush
status 2 4 1 0
results 2 + 1
```



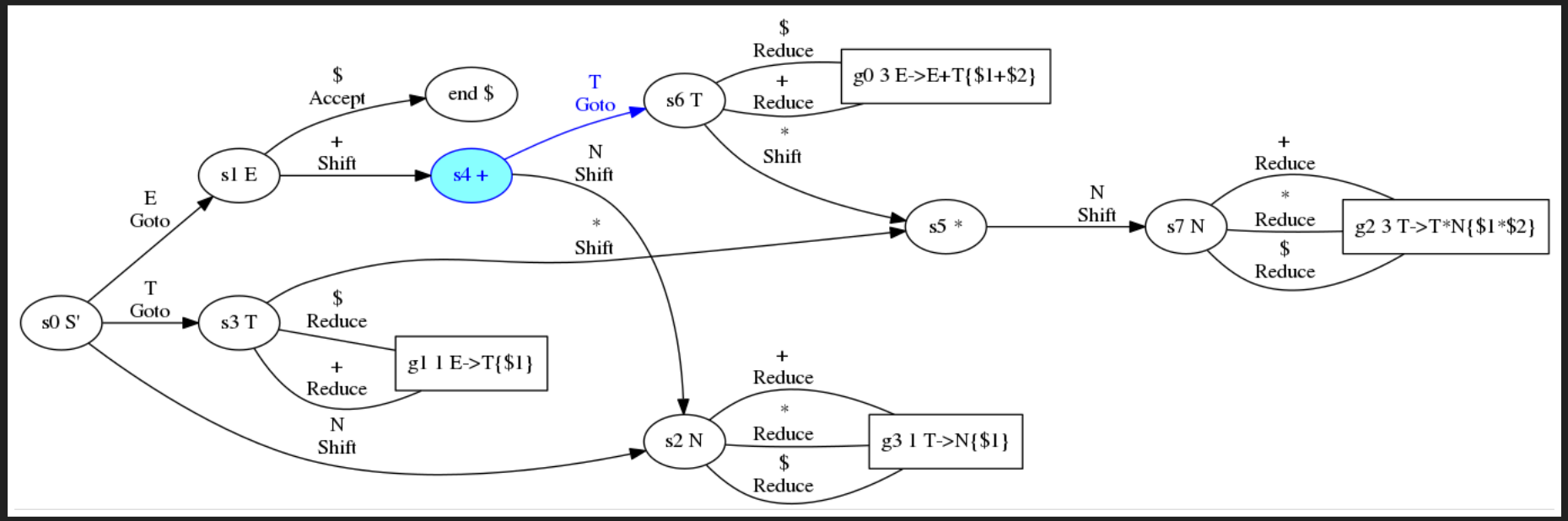
GOTO S6

```
inputs T $
status 4 1 0
results 2 + 1
```



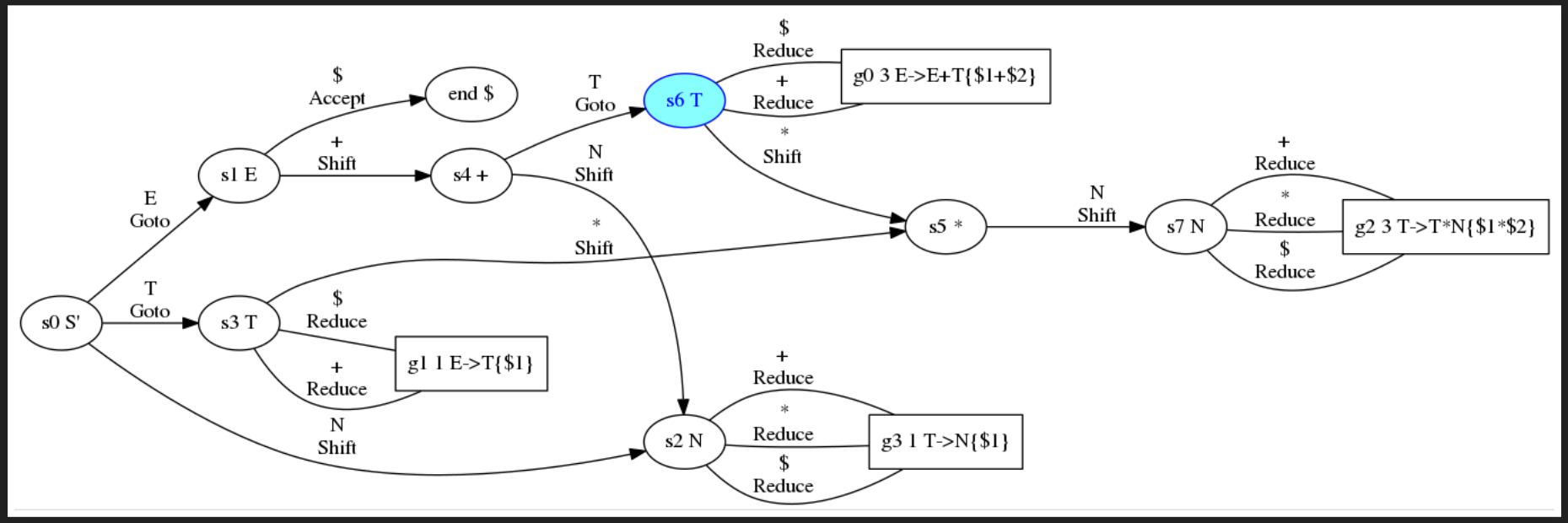
GOTO S6

```
inputs T $ ステータスに6をpush,入力Tを捨てる
status 4 1 0
results 2 + 1
```



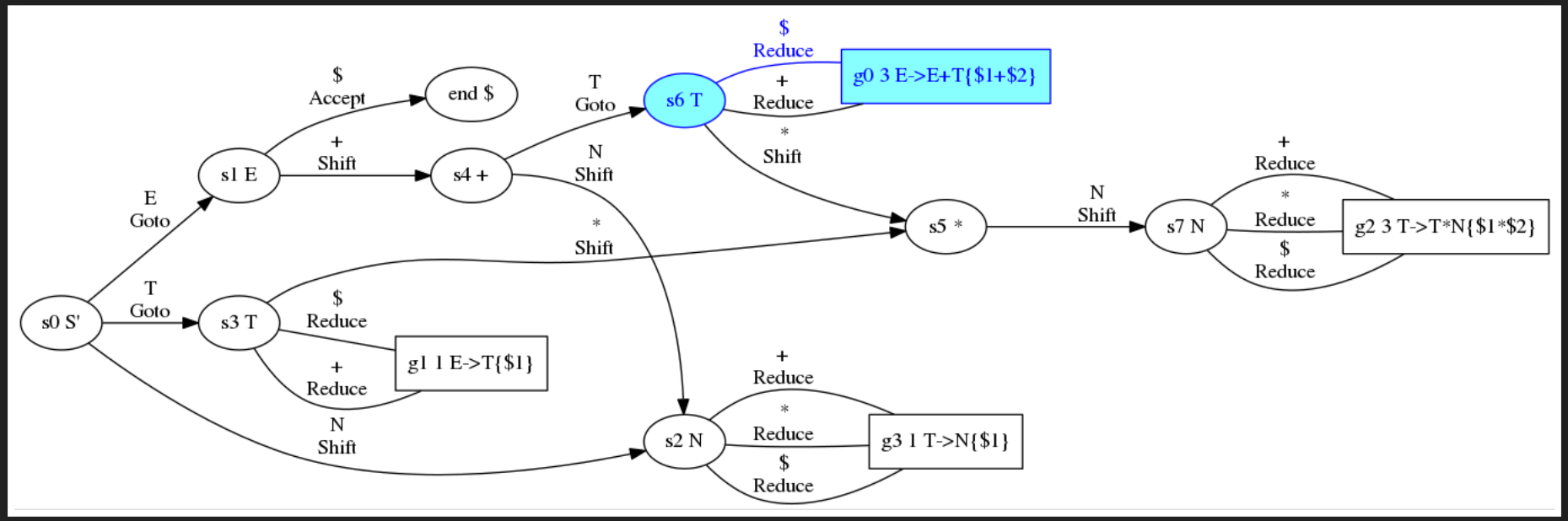
REDUCE G0

```
inputs $
status 6 4 1 0
results 2 + 1
```



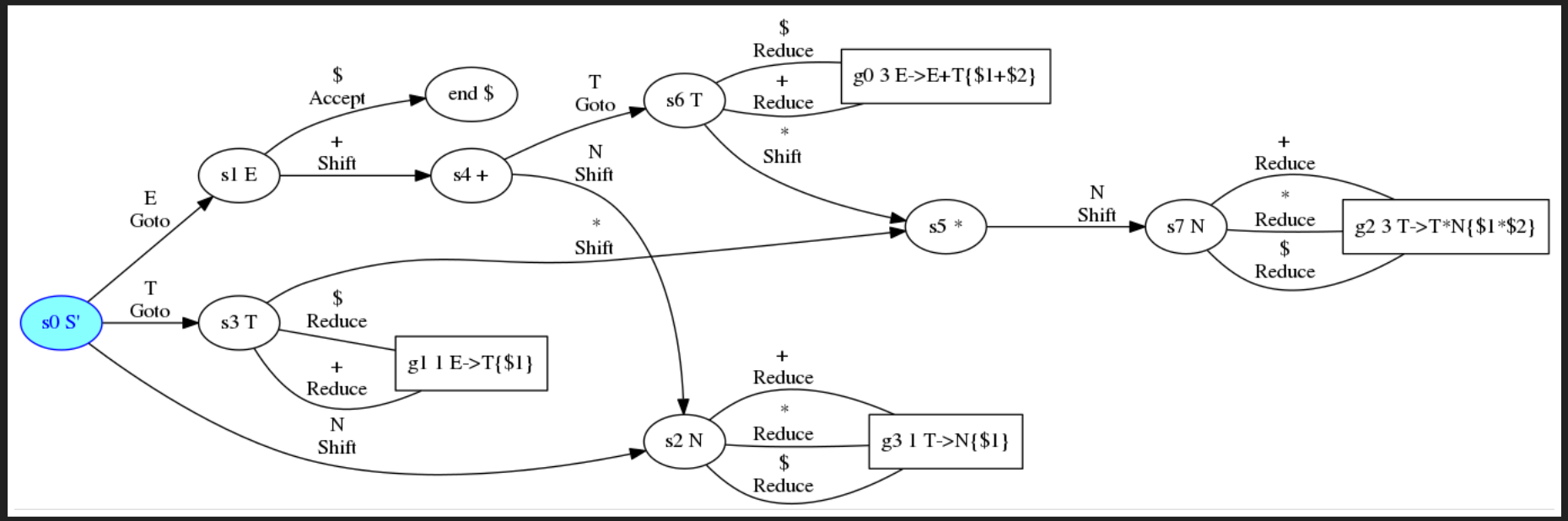
REDUCE G0

```
inputs $ 文法g0を見て、3個Pop、{$1+$2}を結果にpush、文法名Eを入力にpush
status 6 4 1 0
results 2 + 1
```



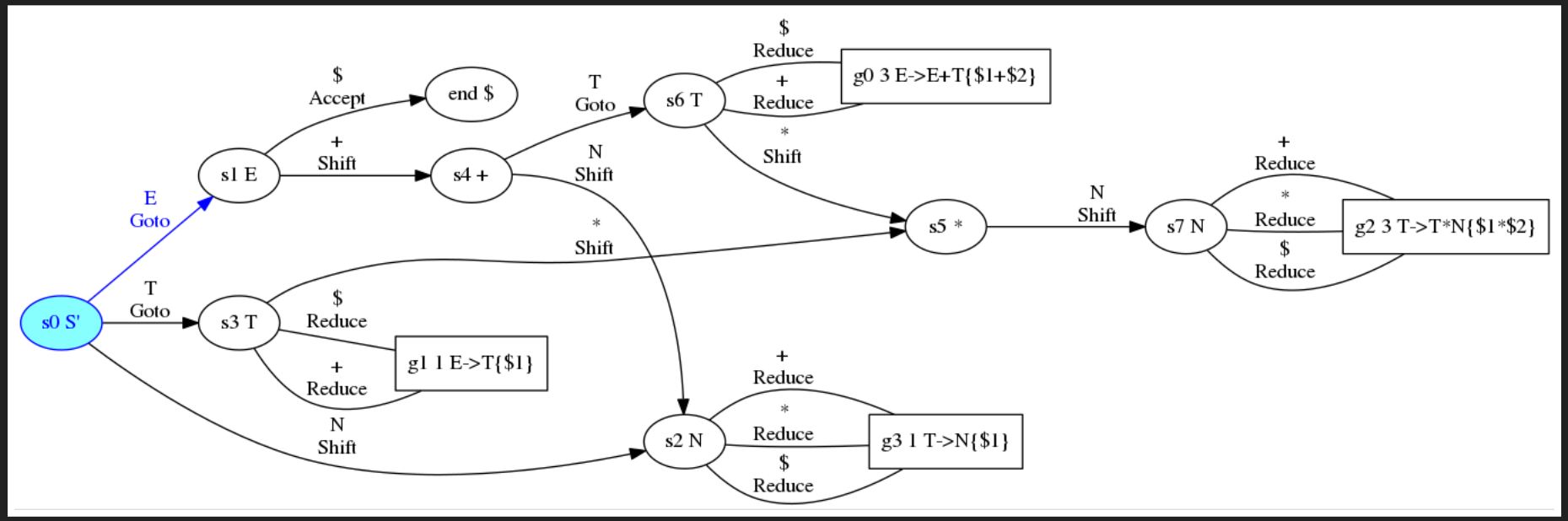
GOTO S1

```
inputs E $
status 0
results 3
```



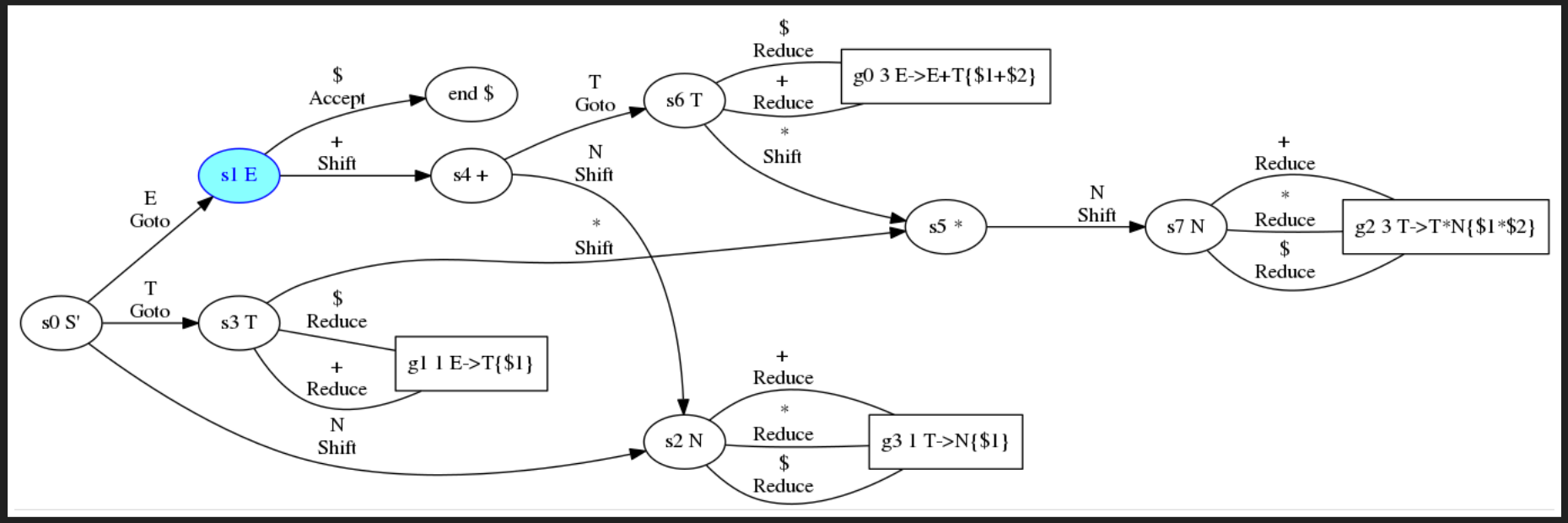
GOTO S1

```
inputs E $ ステータスに1をpush,入力Eを捨てる
status 0
results 3
```



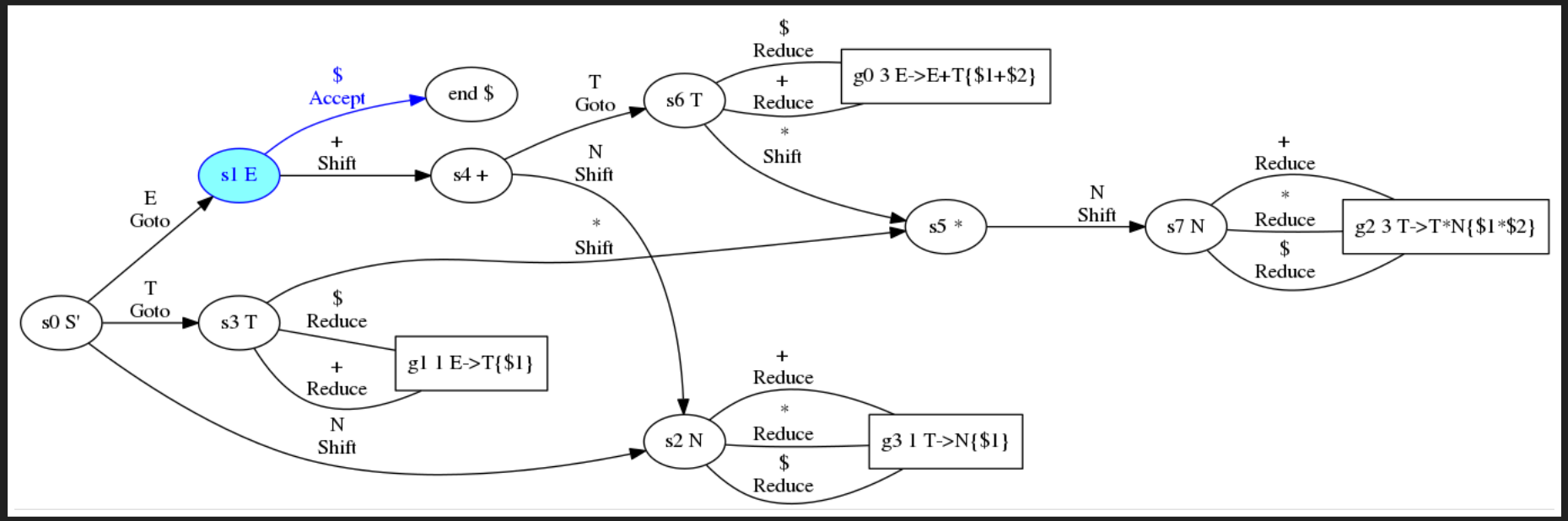
ACCEPT

```
inputs $
status 1 0
results 3
```



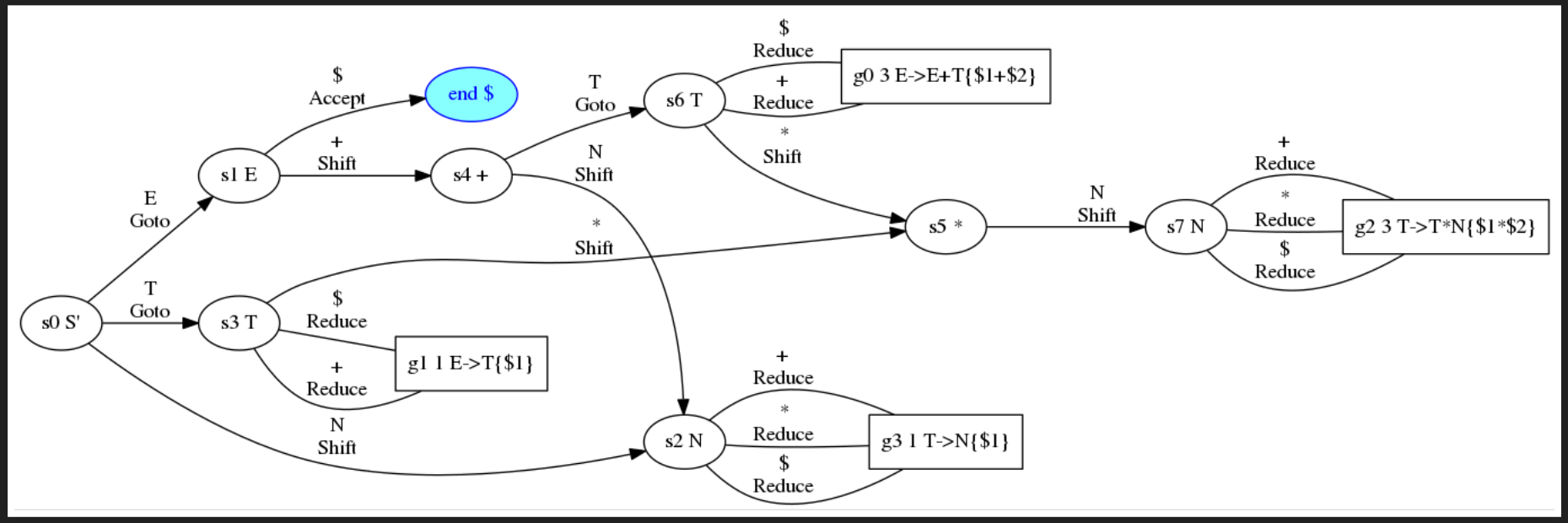
ACCEPT

```
inputs $ アクセプト
status 1 0
results 3
```



ACCEPT

```
inputs $ 結果 3 です
status 1 0
results 3
```

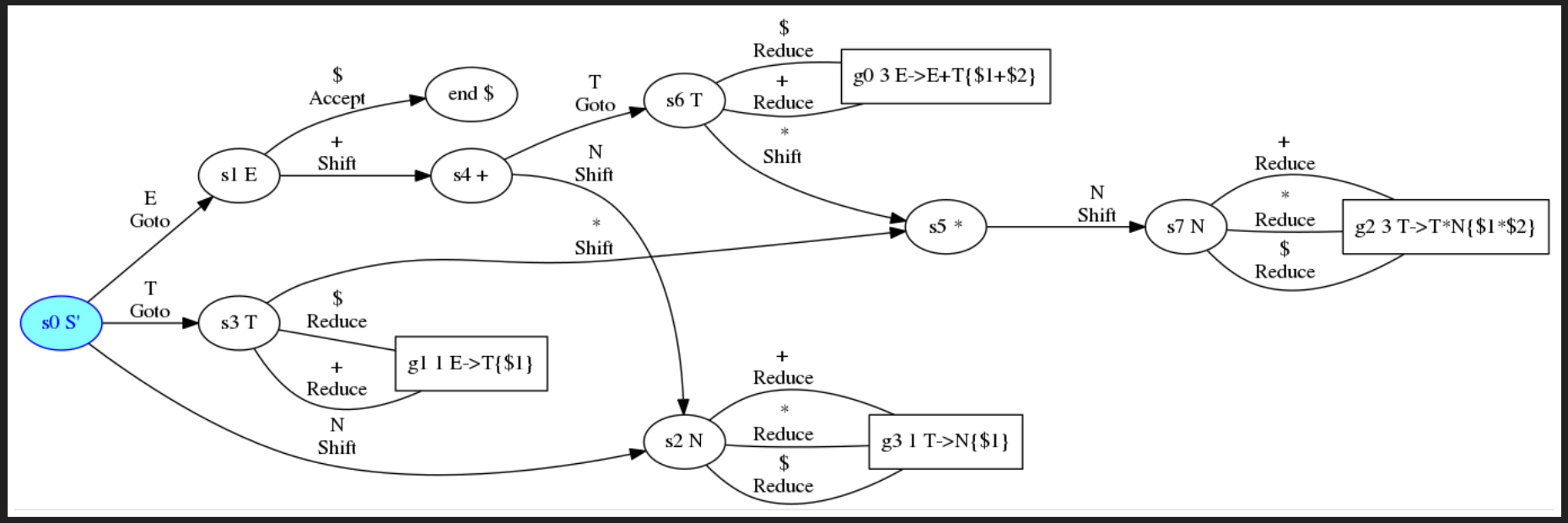


RESULT 1+2 = 3

1+2+3

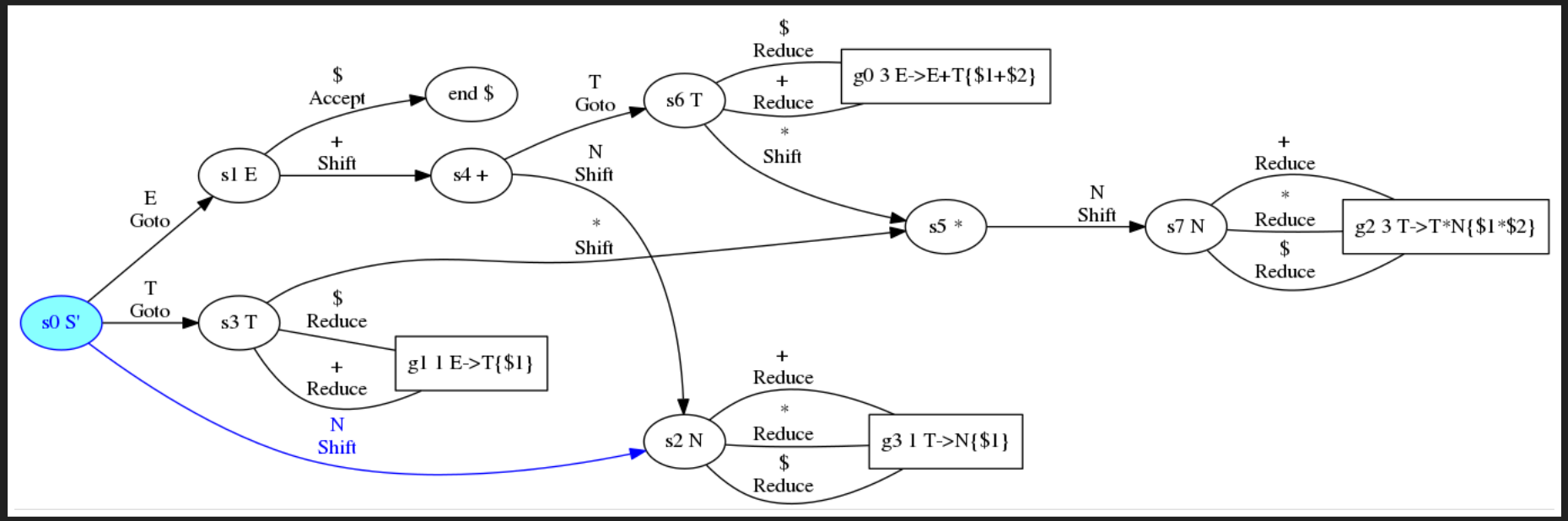
SHIFT S2

```
inputs 1 + 2 + 3 $
status 0
results
```



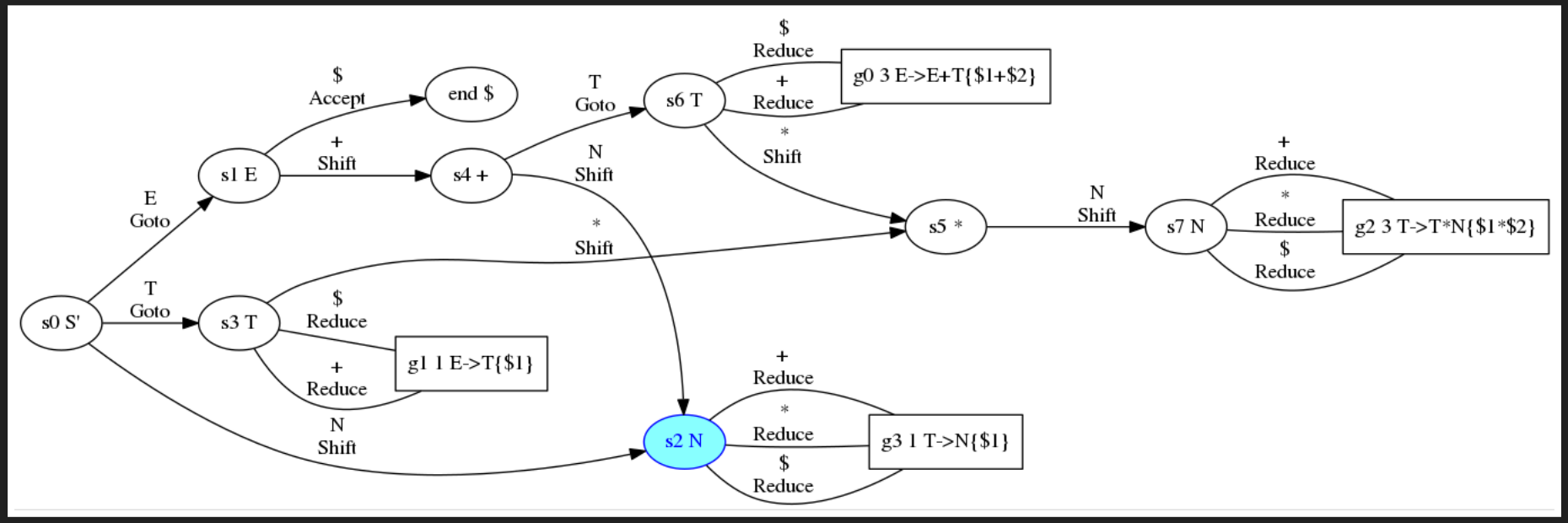
SHIFT S2

```
inputs 1 + 2 + 3 $ ステータスに2をpush 入力1を結果に移動
status 0
results
```



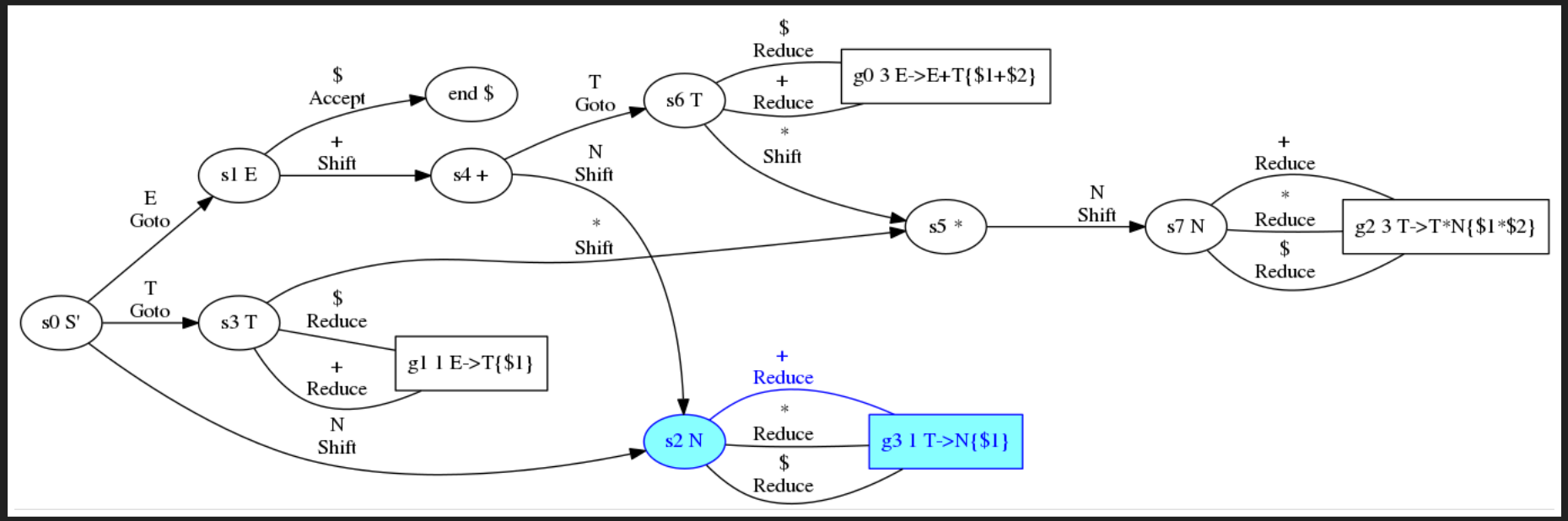
REDUCE G3

```
inputs + 2 + 3 $
status 2 0
results 1
```



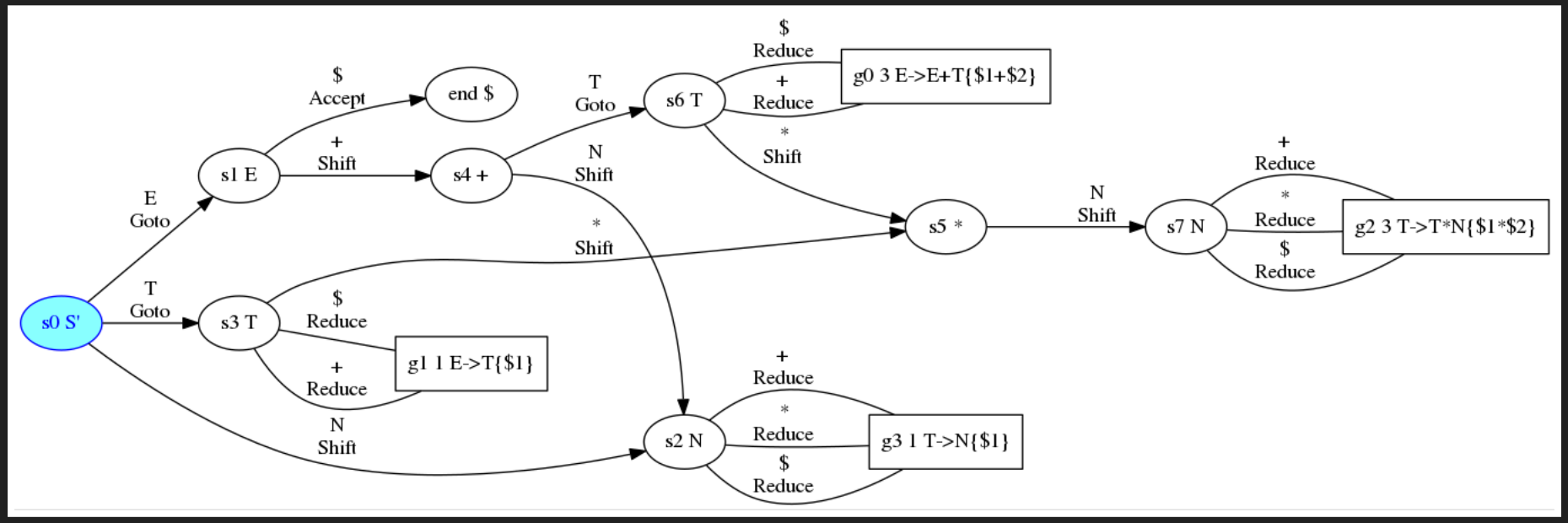
REDUCE G3

```
inputs + 2 + 3 $ 文法g3を見て、1個Pop、{$1}を結果にpush、文法名Tを入力にpush
status 2 0
results 1
```



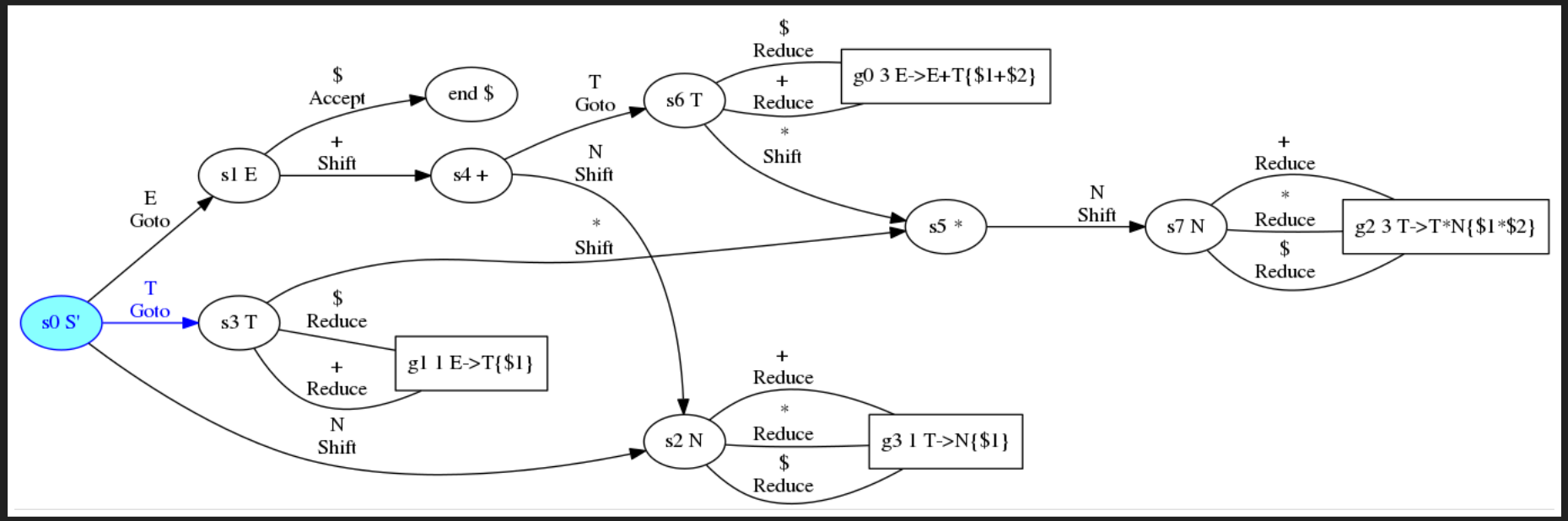
GOTO S3

```
inputs T + 2 + 3 $
status 0
results 1
```



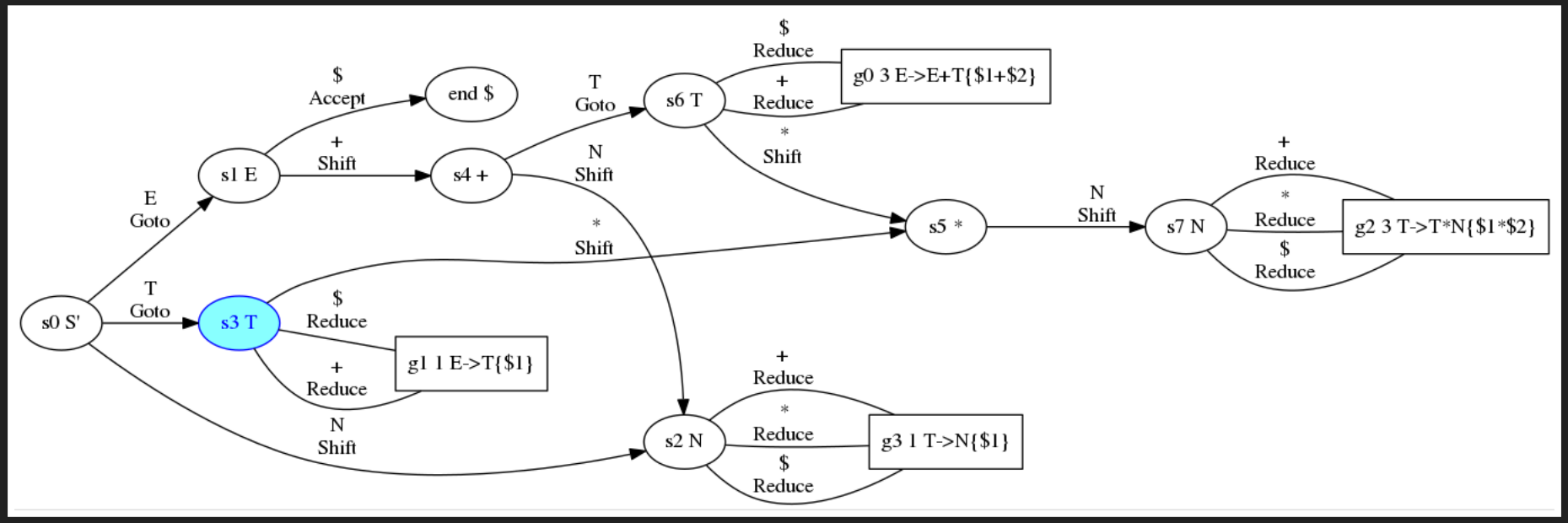
GOTO S3

```
inputs T + 2 + 3 $ ステータスに3をpush,入力Tを捨てる
status 0
results 1
```



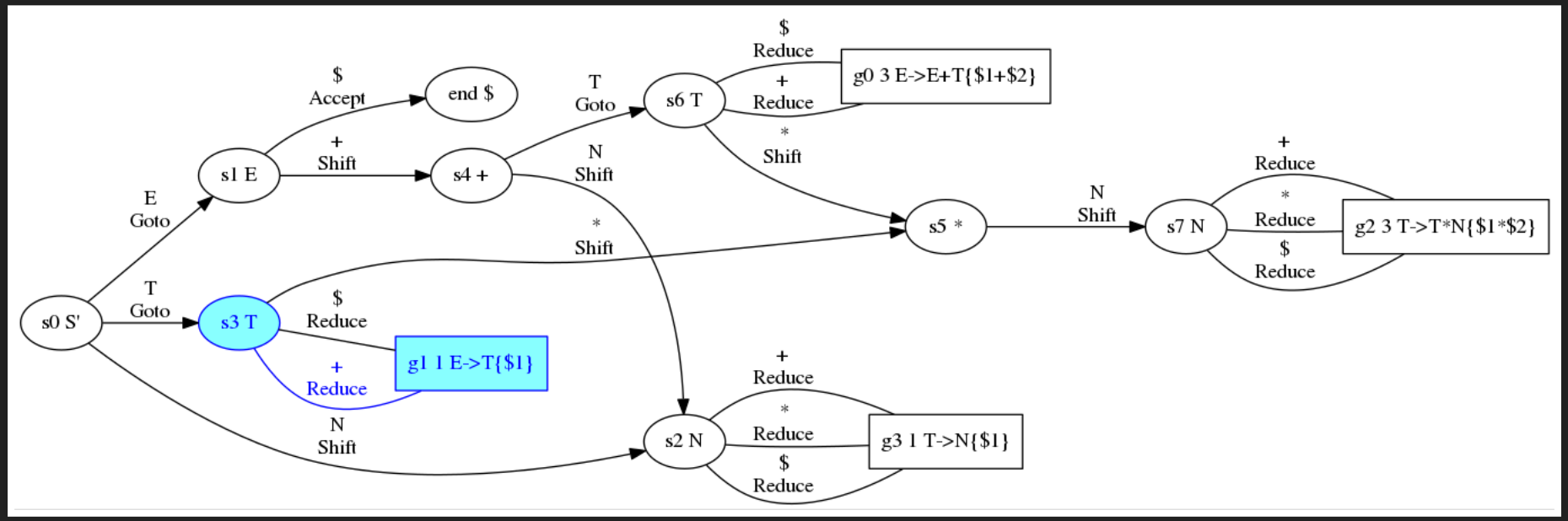
REDUCE G1

```
inputs + 2 + 3 $
status 3 0
results 1
```



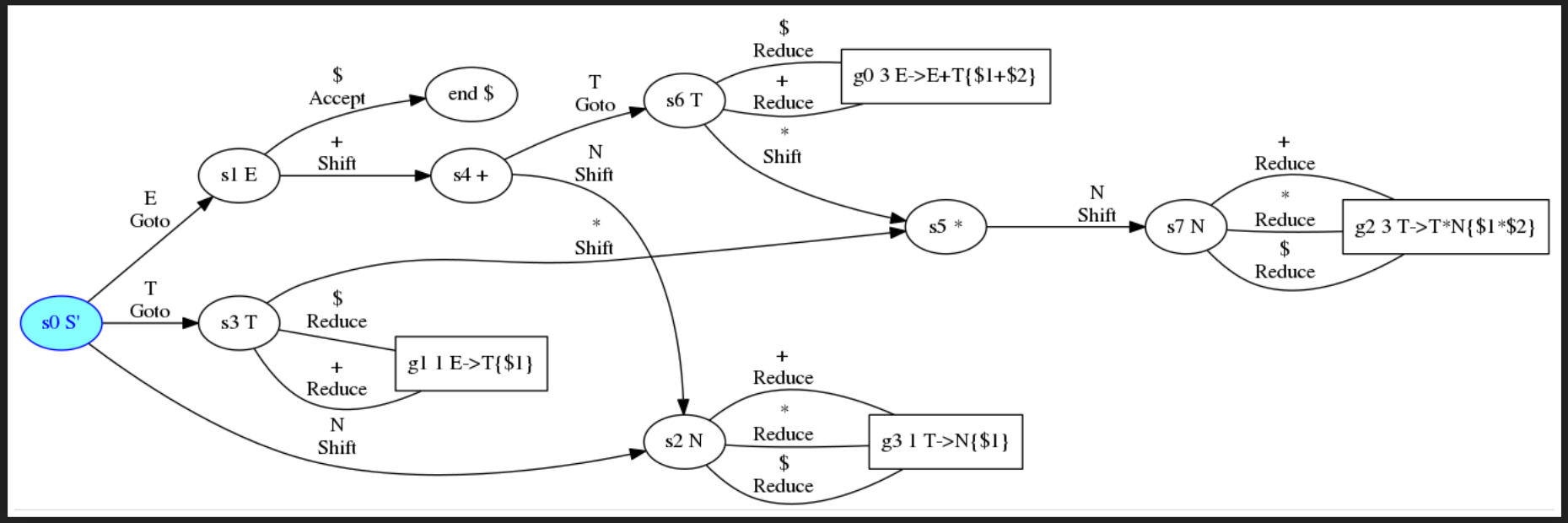
REDUCE G1

```
inputs + 2 + 3 $ 文法g1を見て、1個Pop、{$1}を結果にpush、文法名Eを入力にpush
status 3 0
results 1
```



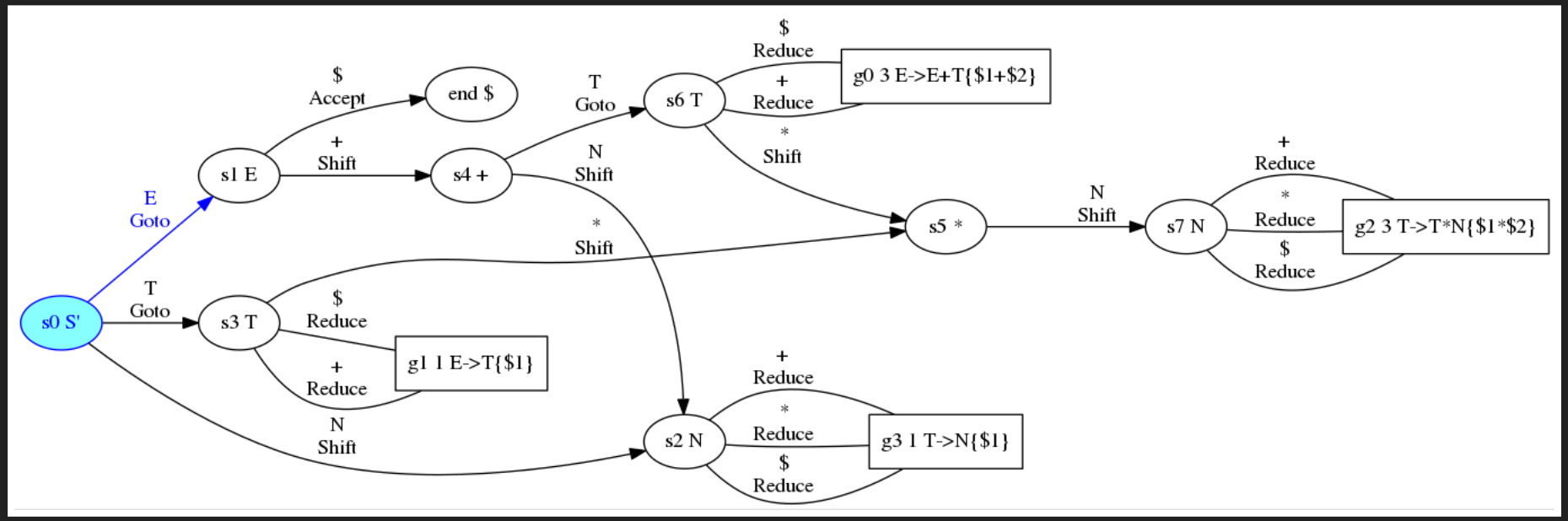
GOTO S1

```
inputs E + 2 + 3 $
status 0
results 1
```



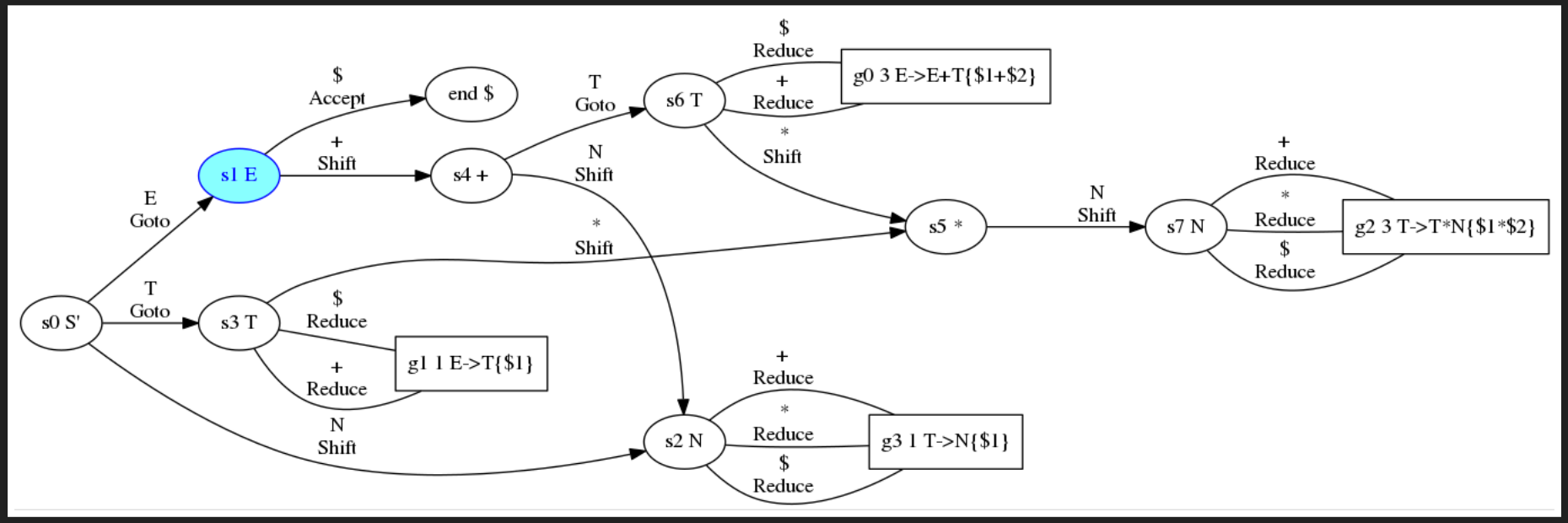
GOTO S1

```
inputs E + 2 + 3 $ ステータスに1をpush,入力Eを捨てる
status 0
results 1
```



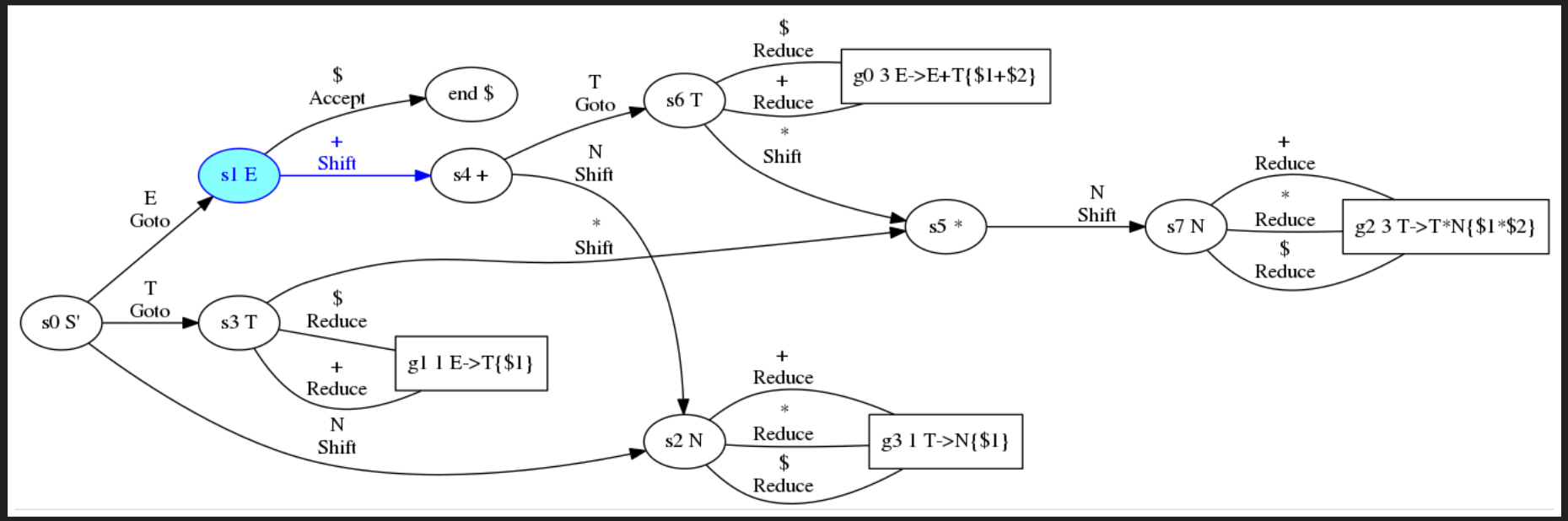
SHIFT S4

```
inputs + 2 + 3 $
status 1 0
results 1
```



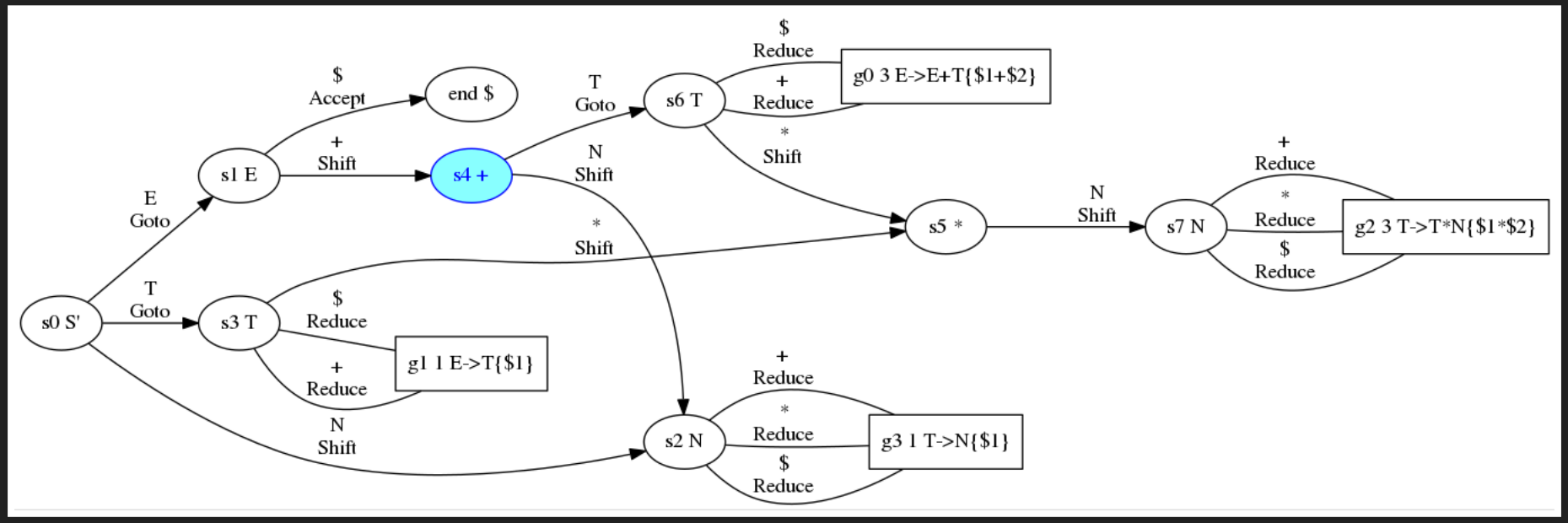
SHIFT S4

```
inputs + 2 + 3 $ ステータスに4をpush 入力+を結果に移動
status 1 0
results 1
```



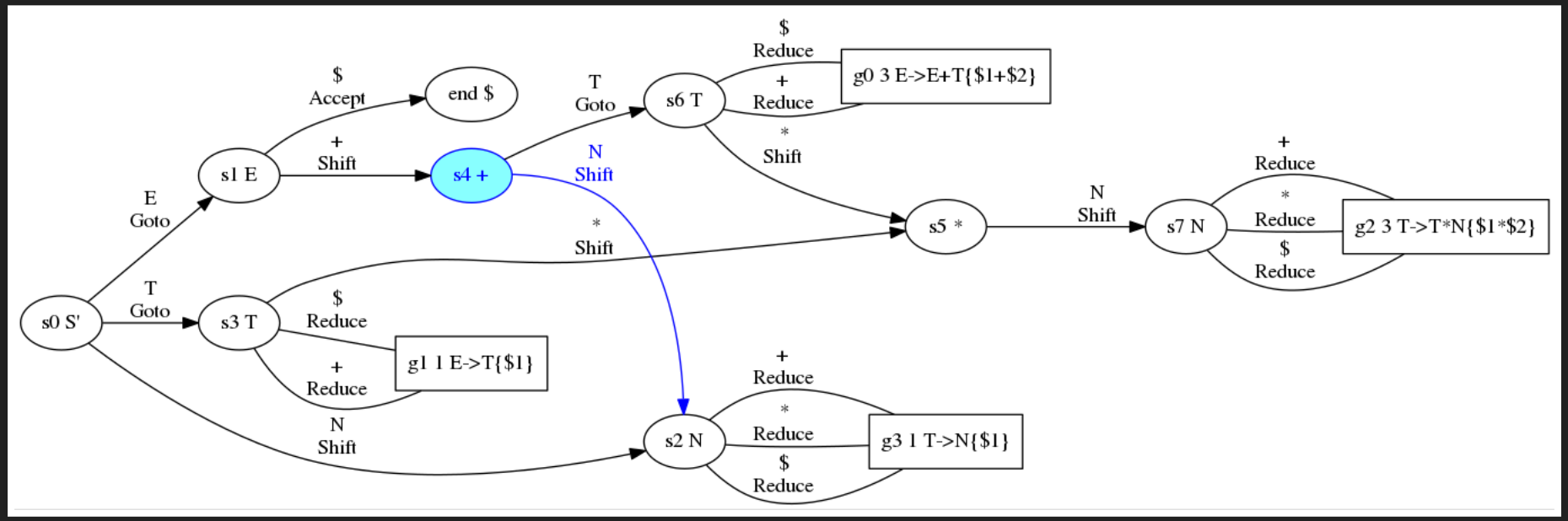
SHIFT S2

```
inputs 2 + 3 $
status 4 1 0
results + 1
```



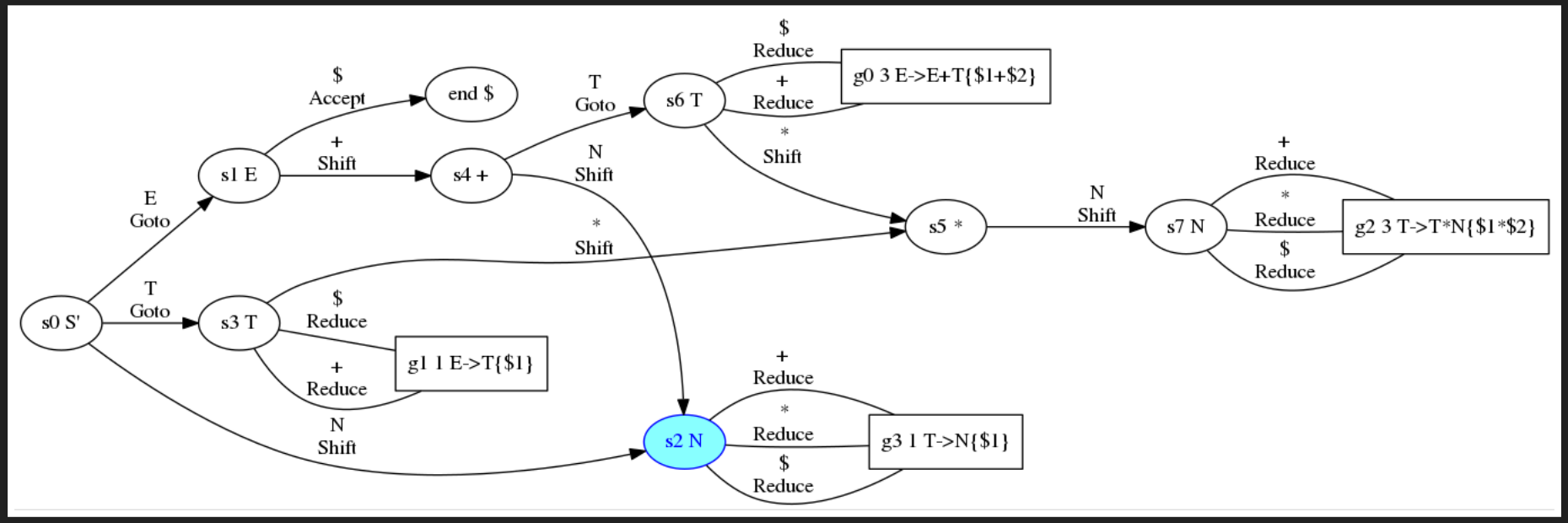
SHIFT S2

```
inputs 2 + 3 $ ステータスに2をpush 入力2を結果に移動
status 4 1 0
results + 1
```



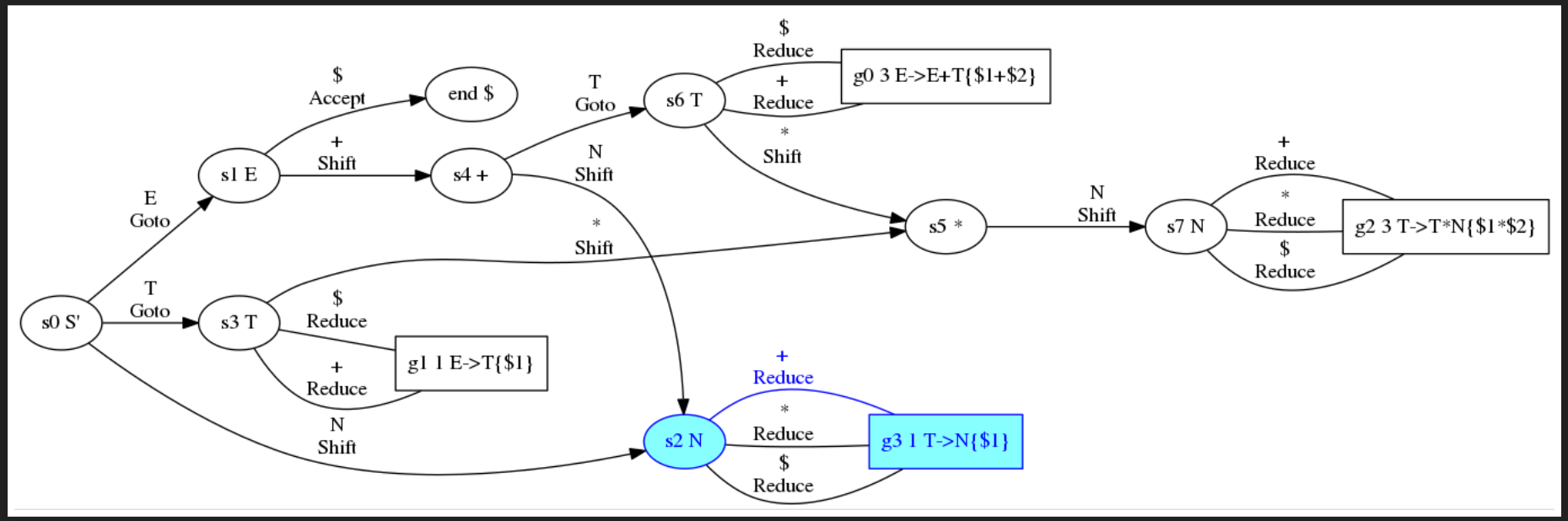
REDUCE G3

```
inputs + 3 $
status 2 4 1 0
results 2 + 1
```



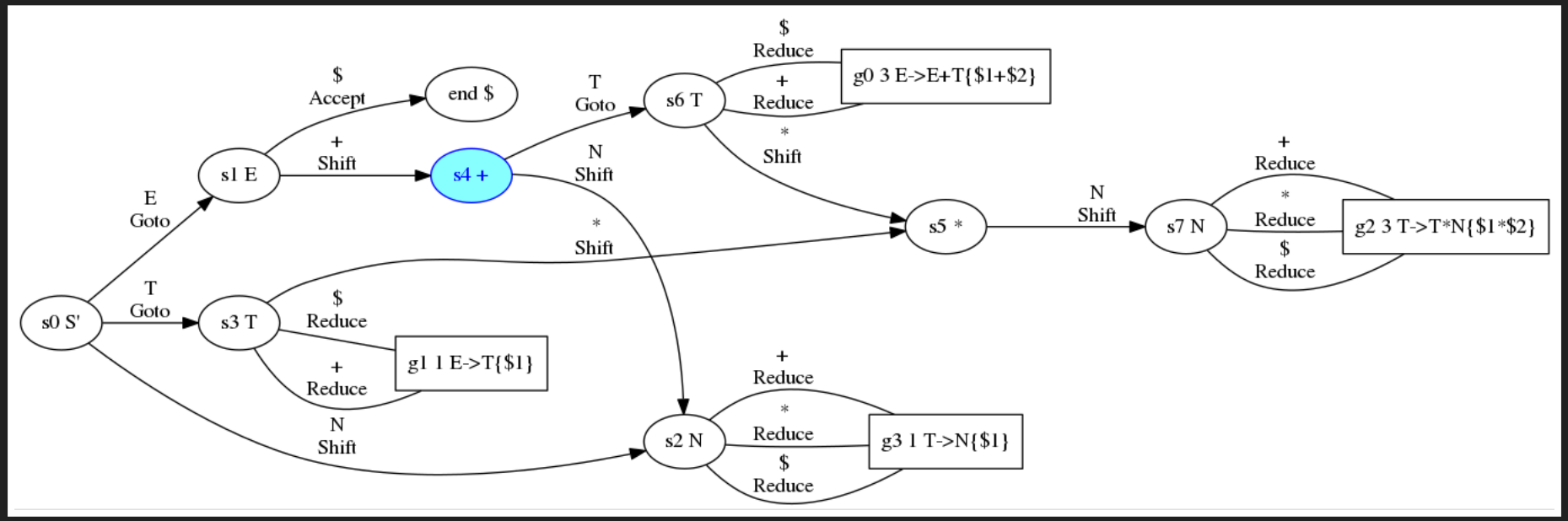
REDUCE G3

```
inputs + 3 $ 文法g3を見て、1個Pop、{$1}を結果にpush、文法名Tを入力にpush
status 2 4 1 0
results 2 + 1
```



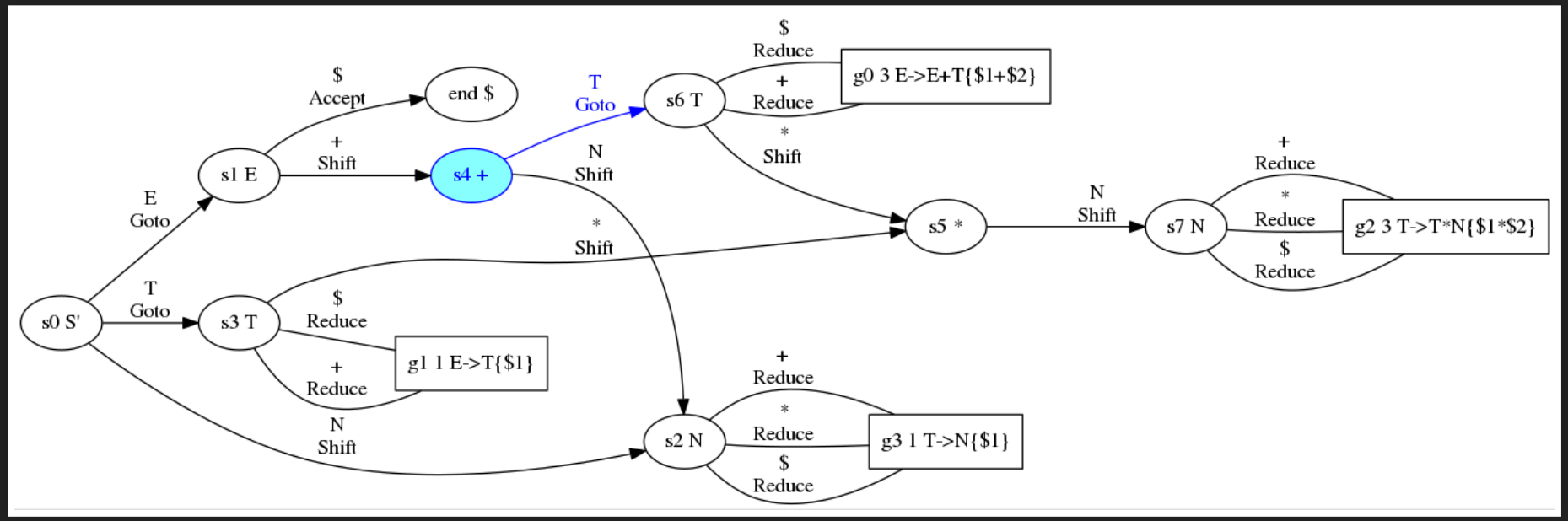
GOTO S6

```
inputs T + 3 $
status 4 1 0
results 2 + 1
```



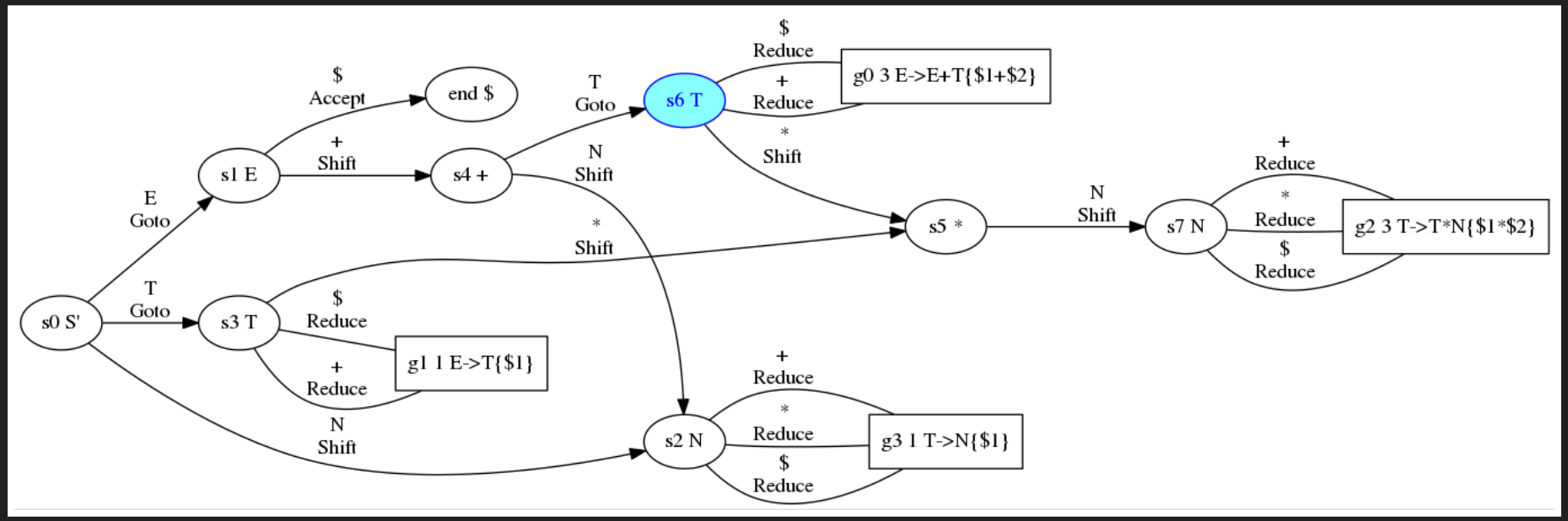
GOTO S6

```
inputs T + 3 $ ステータスに6をpush,入力Tを捨てる
status 4 1 0
results 2 + 1
```



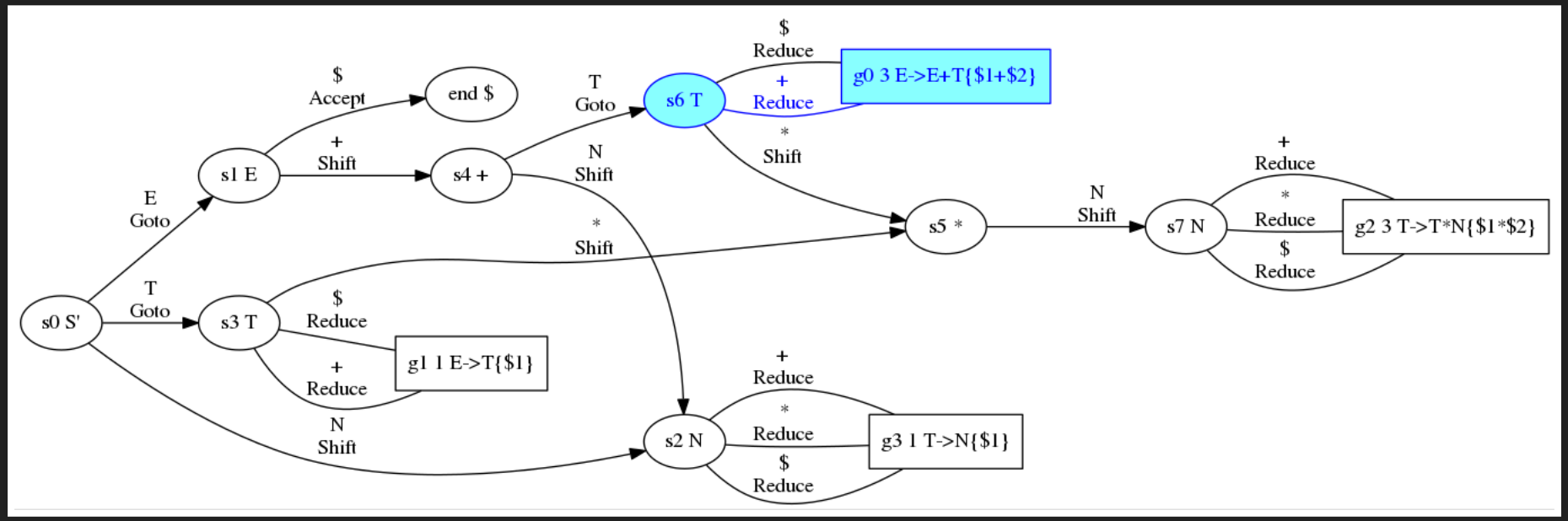
REDUCE G0

```
inputs + 3 $
status 6 4 1 0
results 2 + 1
```



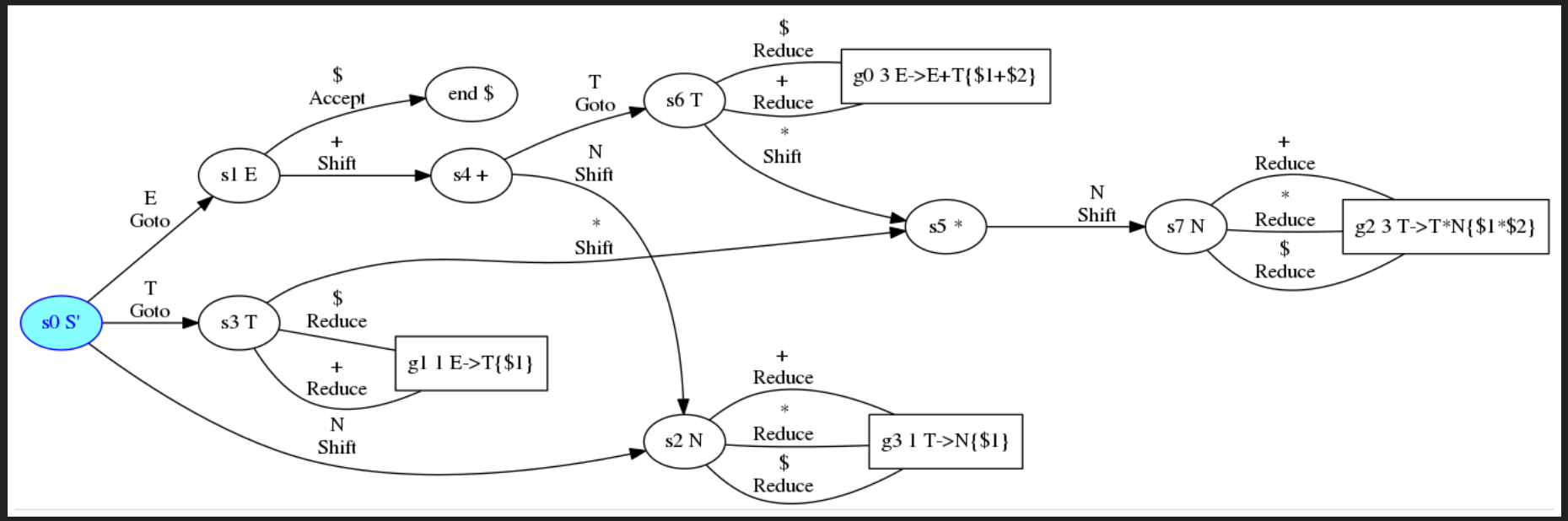
REDUCE G0

```
inputs + 3 $ 文法g0を見て、3個Pop、{$1+$2}を結果にpush、文法名Eを入力にpush
status 6 4 1 0
results 2 + 1
```



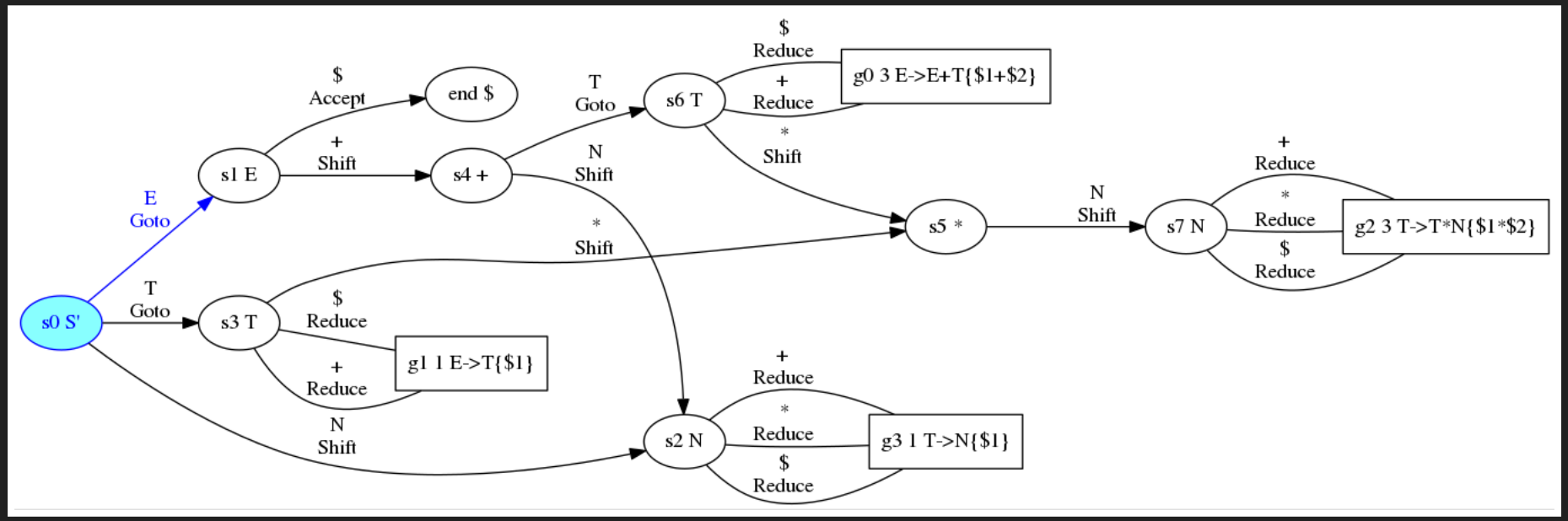
GOTO S1

```
inputs E + 3 $
status 0
results 3
```



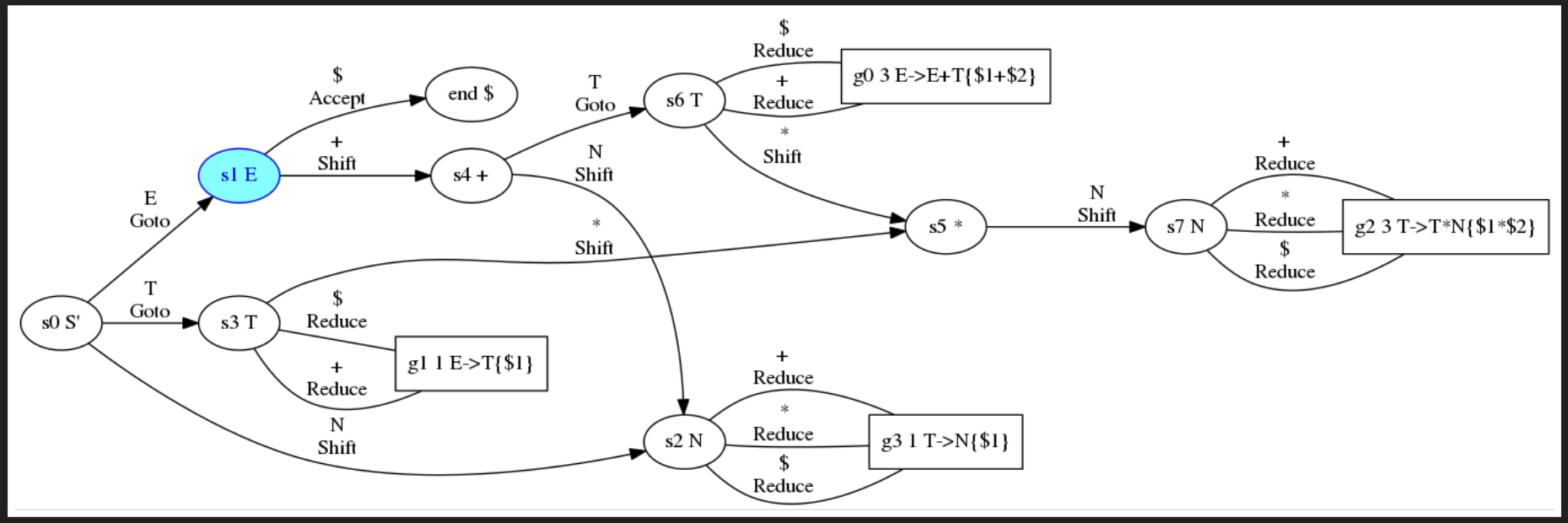
GOTO S1

```
inputs E + 3 $ ステータスに1をpush,入力Eを捨てる
status 0
results 3
```



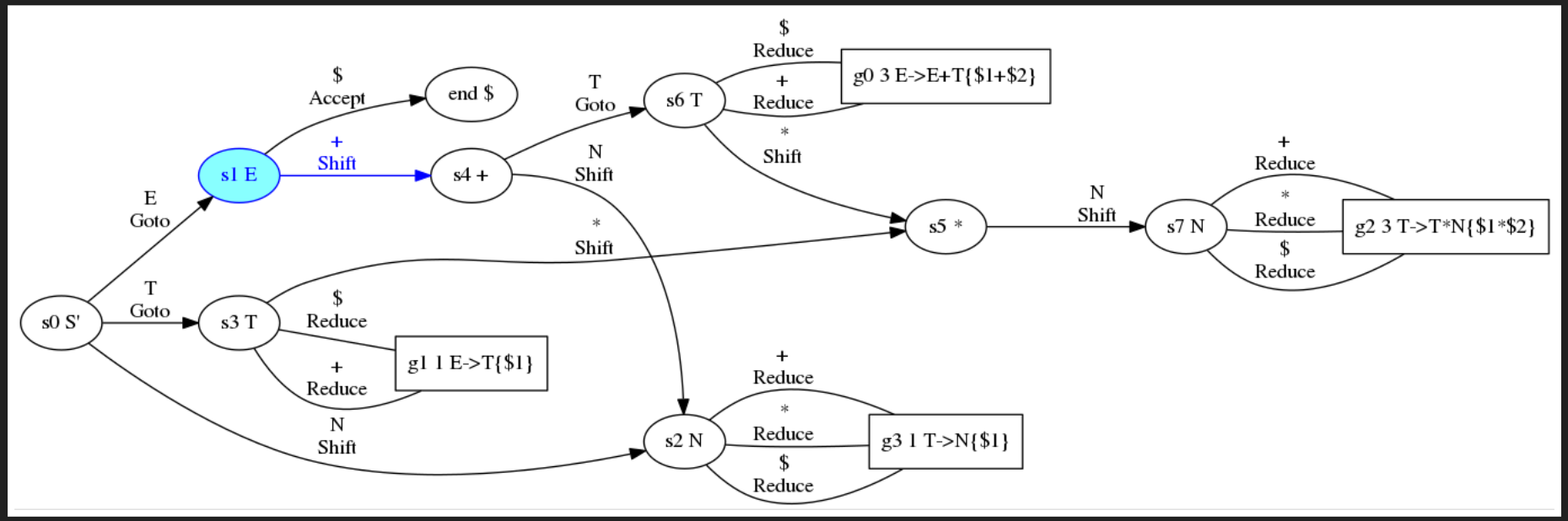
SHIFT S4

```
inputs + 3 $
status 1 0
results 3
```



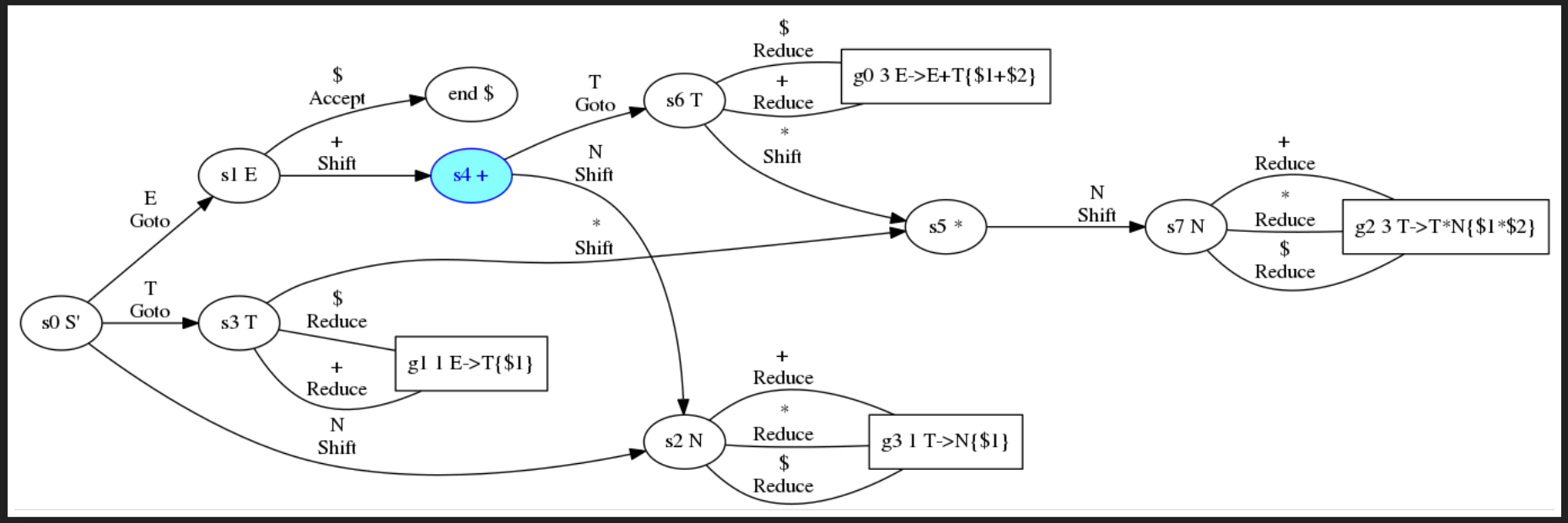
SHIFT S4

```
inputs + 3 $ ステータスに4をpush 入力+を結果に移動
status 1 0
results 3
```



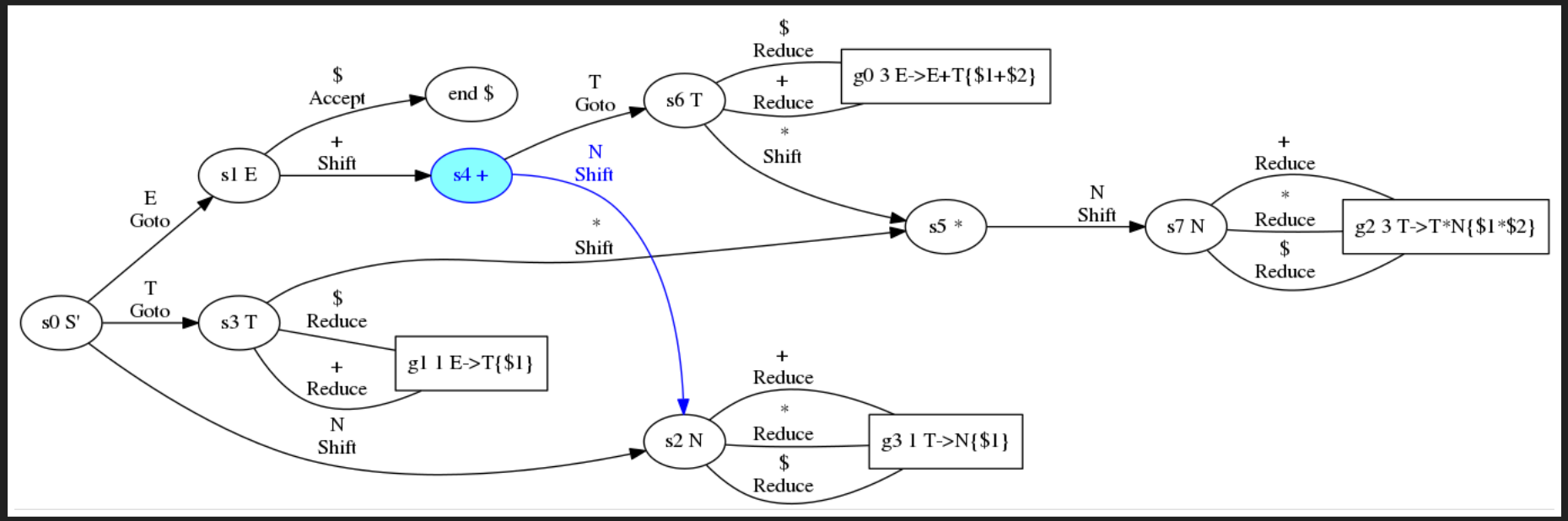
SHIFT S2

```
inputs 3 $
status 4 1 0
results + 3
```



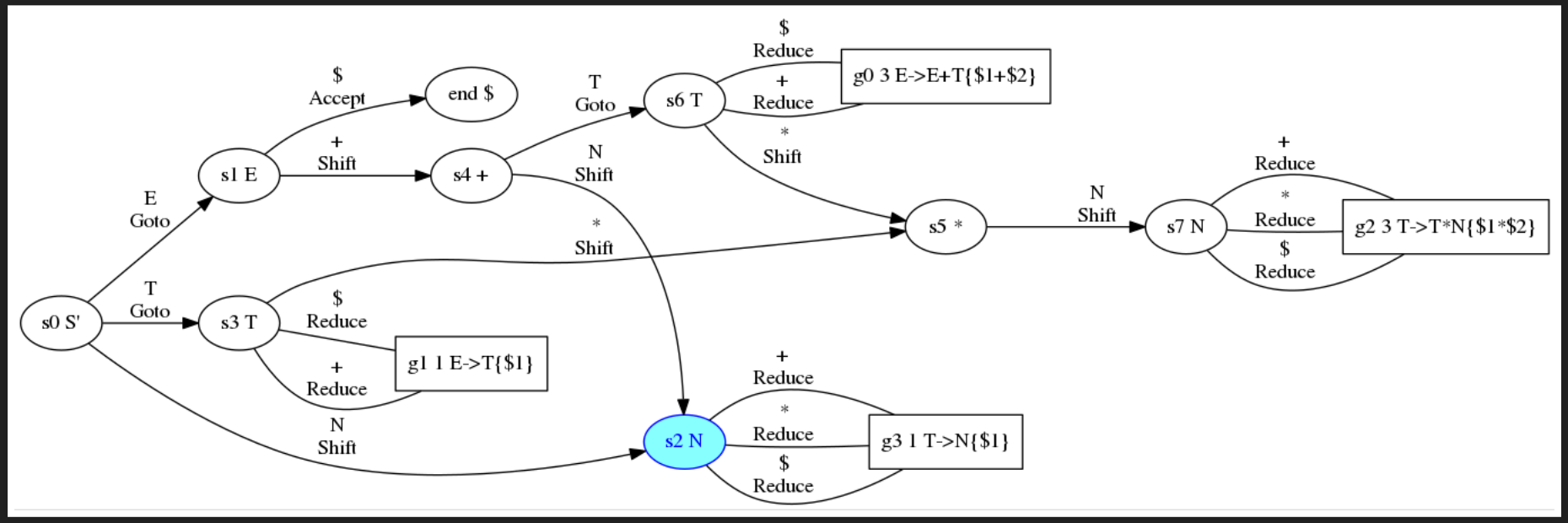
SHIFT S2

```
inputs 3 $ ステータスに2をpush 入力3を結果に移動
status 4 1 0
results + 3
```



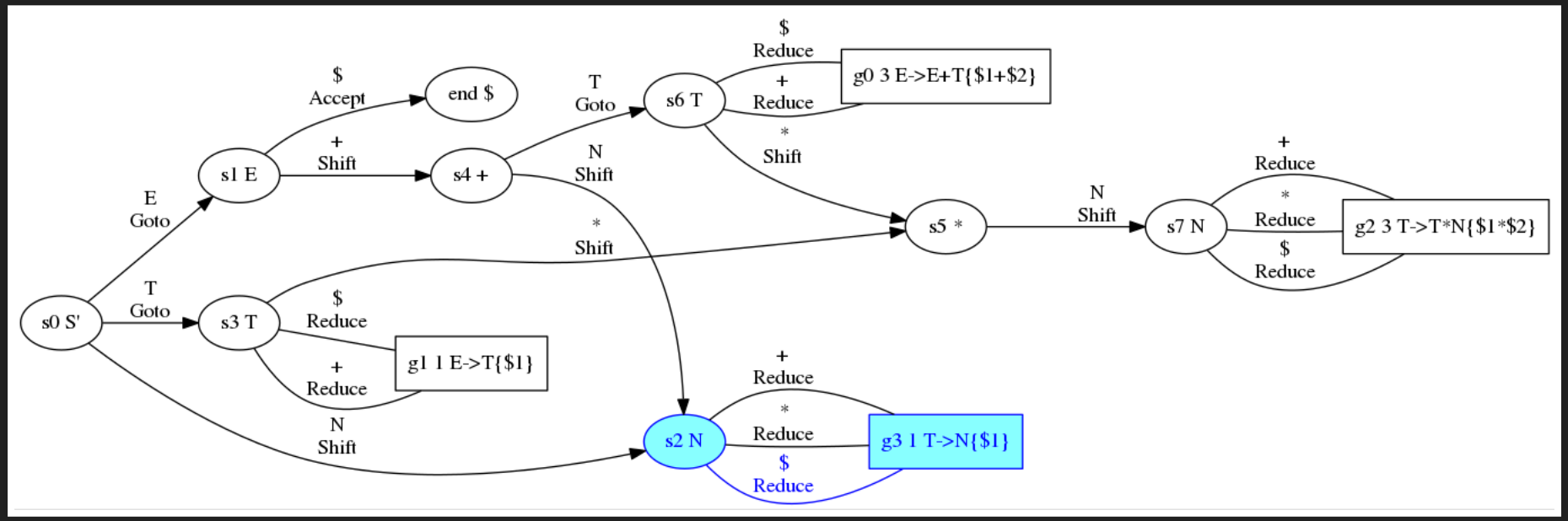
REDUCE G3

```
inputs $
status 2 4 1 0
results 3 + 3
```



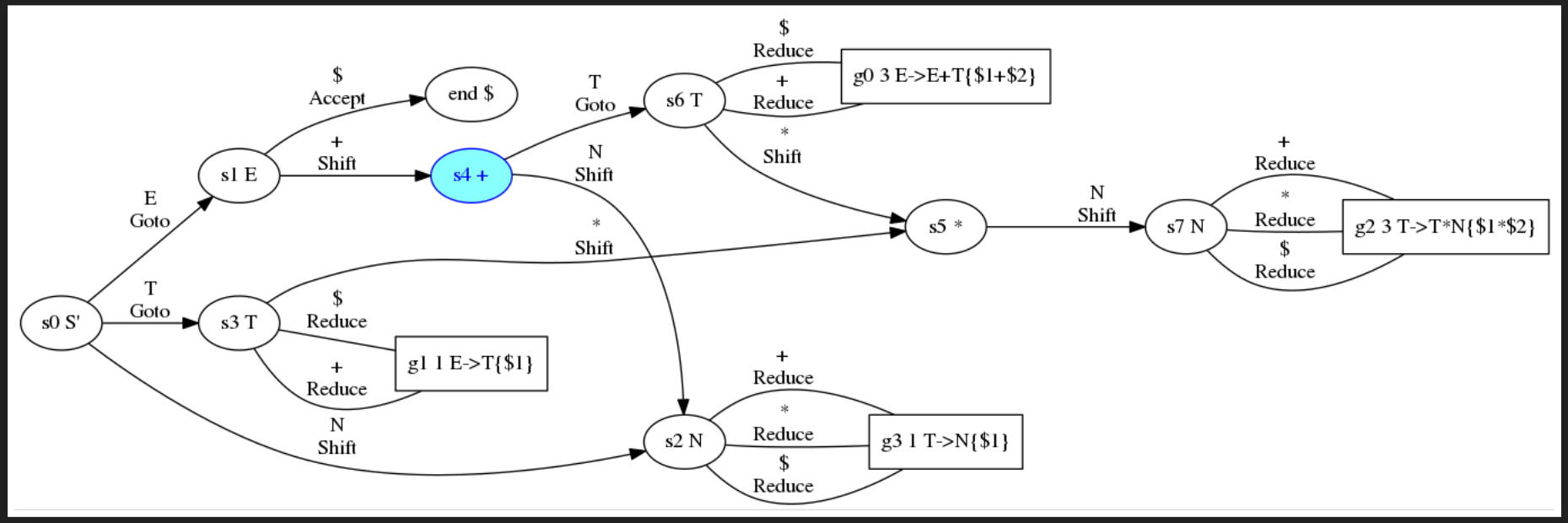
REDUCE G3

```
inputs $ 文法g3を見て、1個Pop、{$1}を結果にpush、文法名Tを入力にpush
status 2 4 1 0
results 3 + 3
```



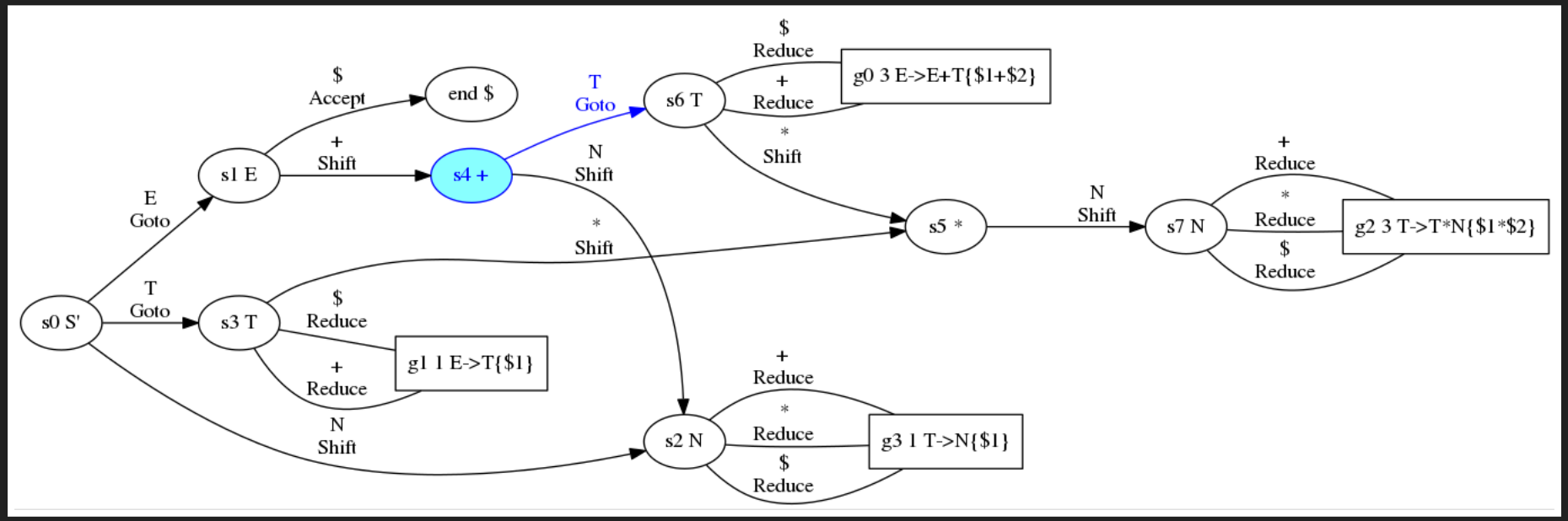
GOTO S6

```
inputs T $
status 4 1 0
results 3 + 3
```



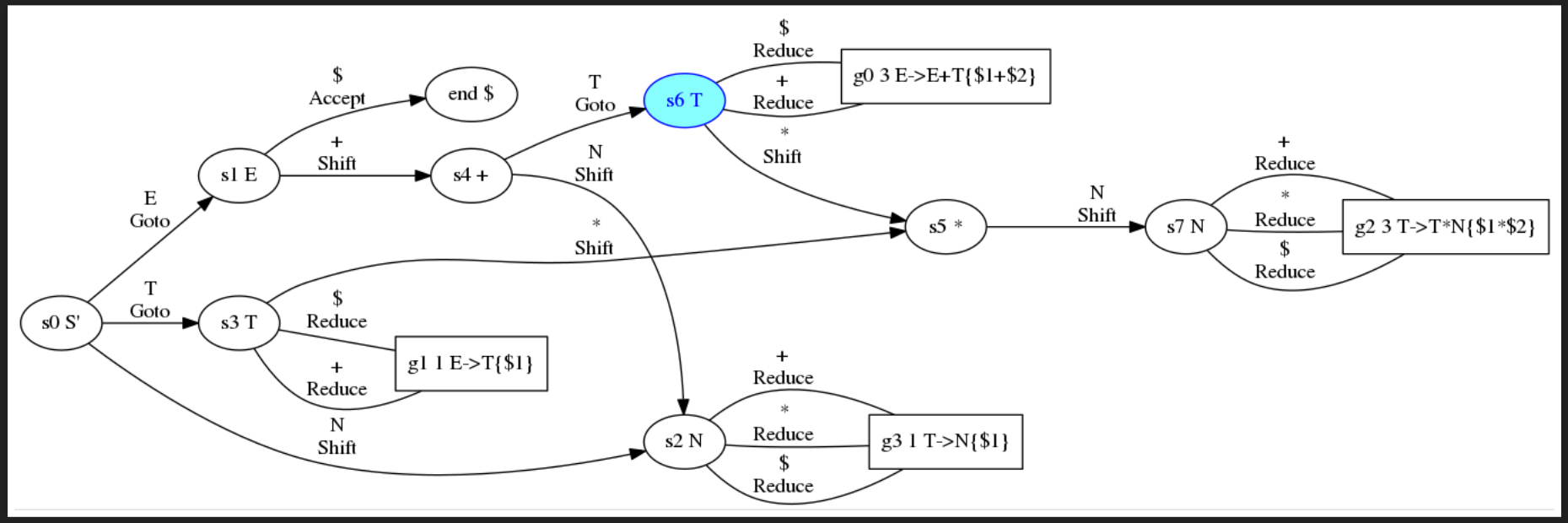
GOTO S6

```
inputs T $ ステータスに6をpush,入力Tを捨てる
status 4 1 0
results 3 + 3
```



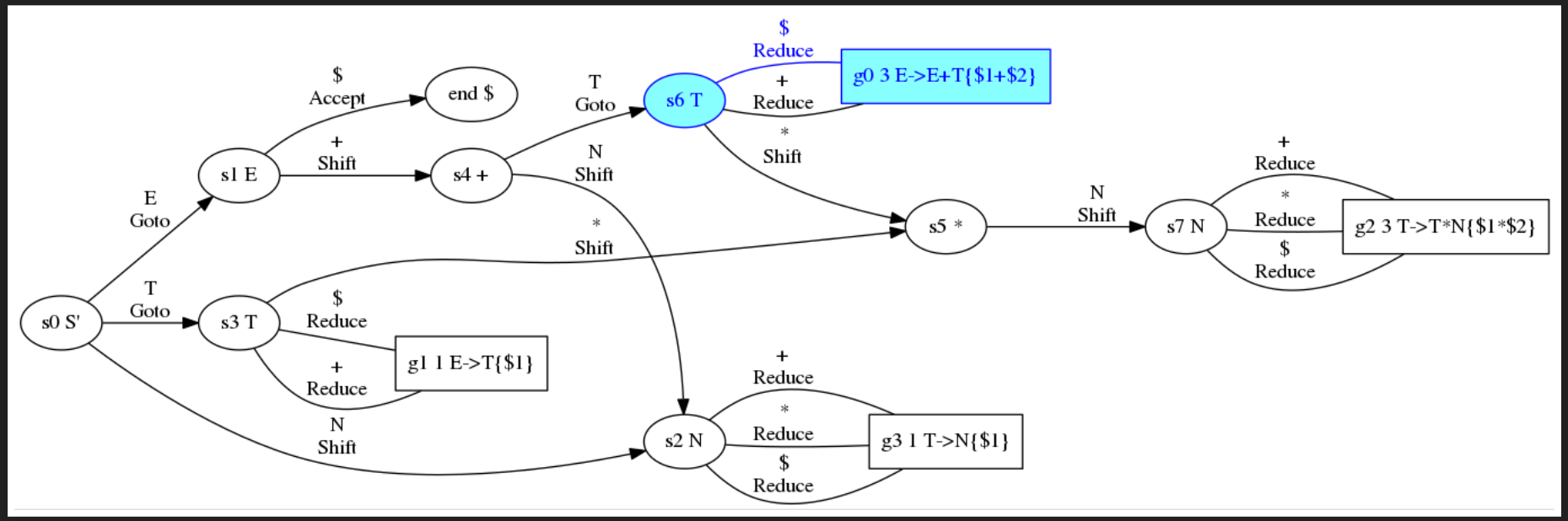
REDUCE G0

```
inputs $
status 6 4 1 0
results 3 + 3
```



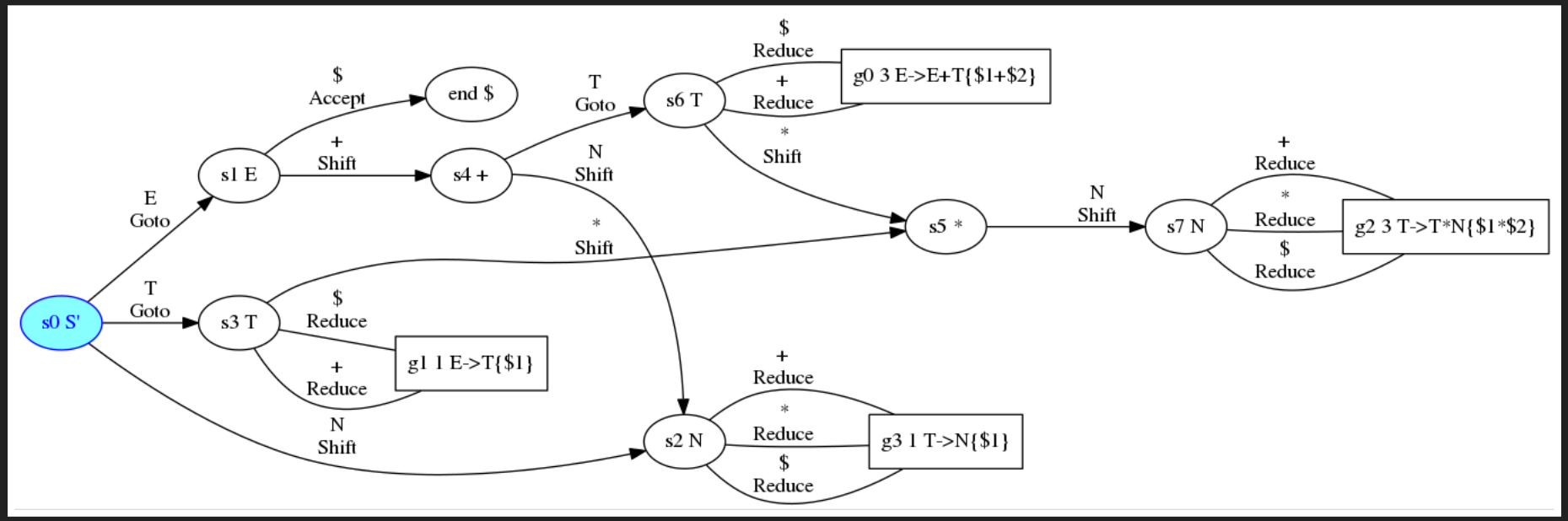
REDUCE G0

```
inputs $ 文法g0を見て、3個Pop、{$1+$2}を結果にpush、文法名Eを入力にpush
status 6 4 1 0
results 3 + 3
```



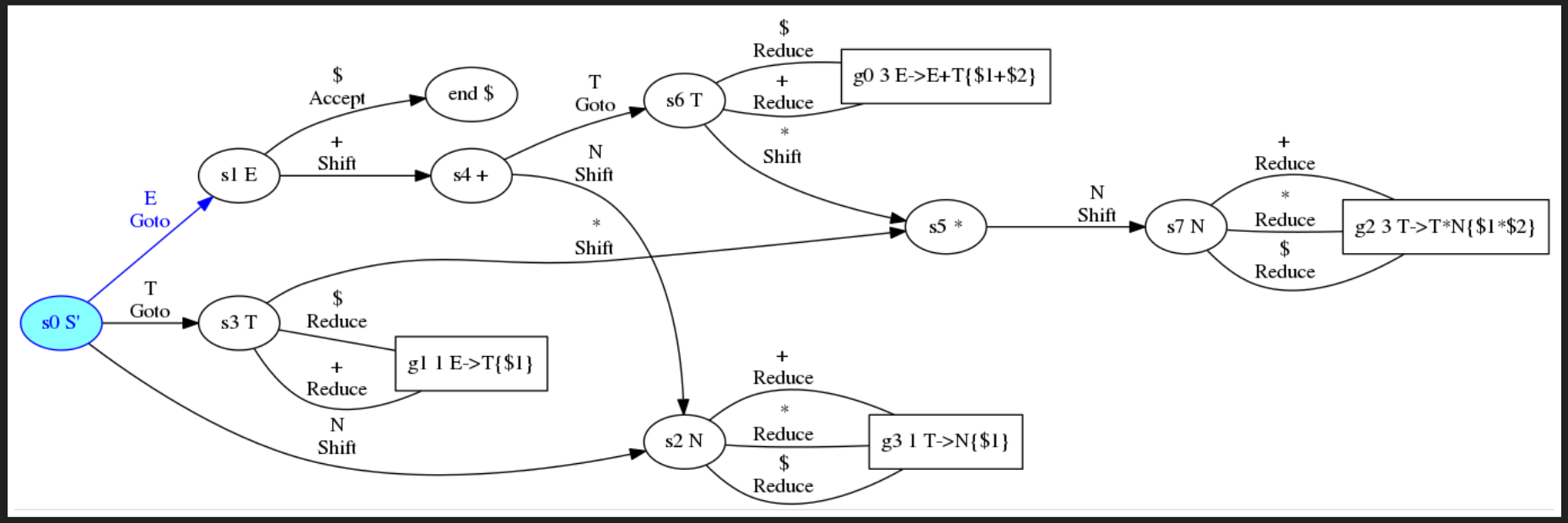
GOTO S1

```
inputs E $
status 0
results 6
```



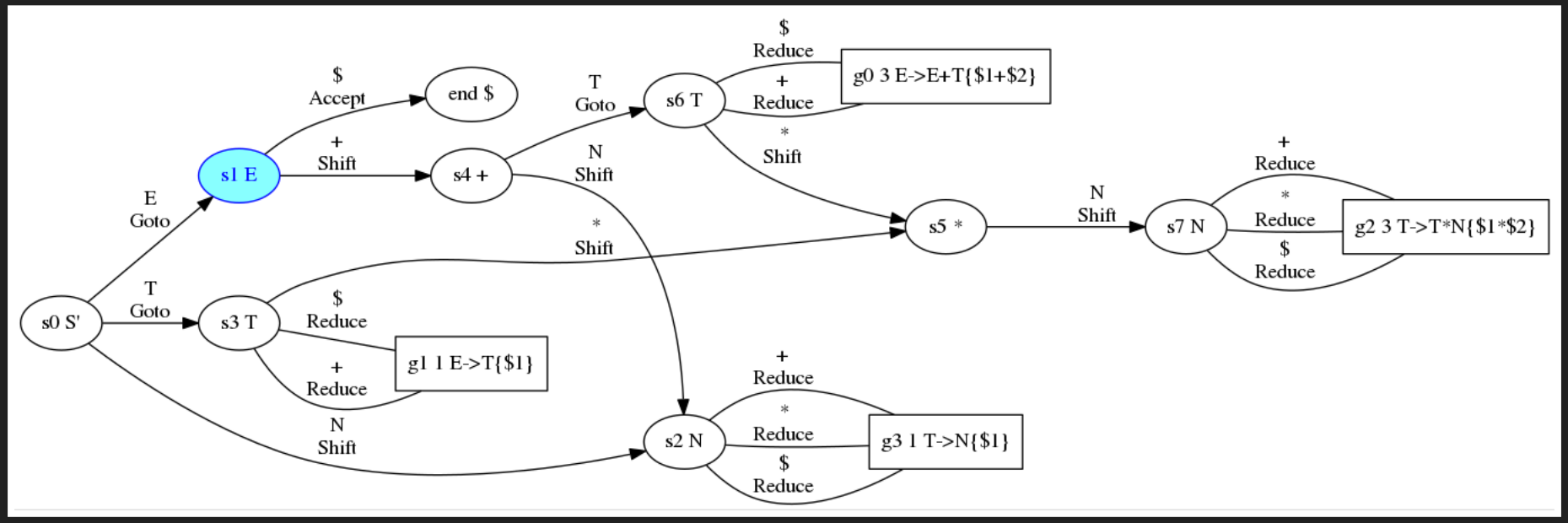
GOTO S1

```
inputs E $ ステータスに1をpush,入力Eを捨てる
status 0
results 6
```



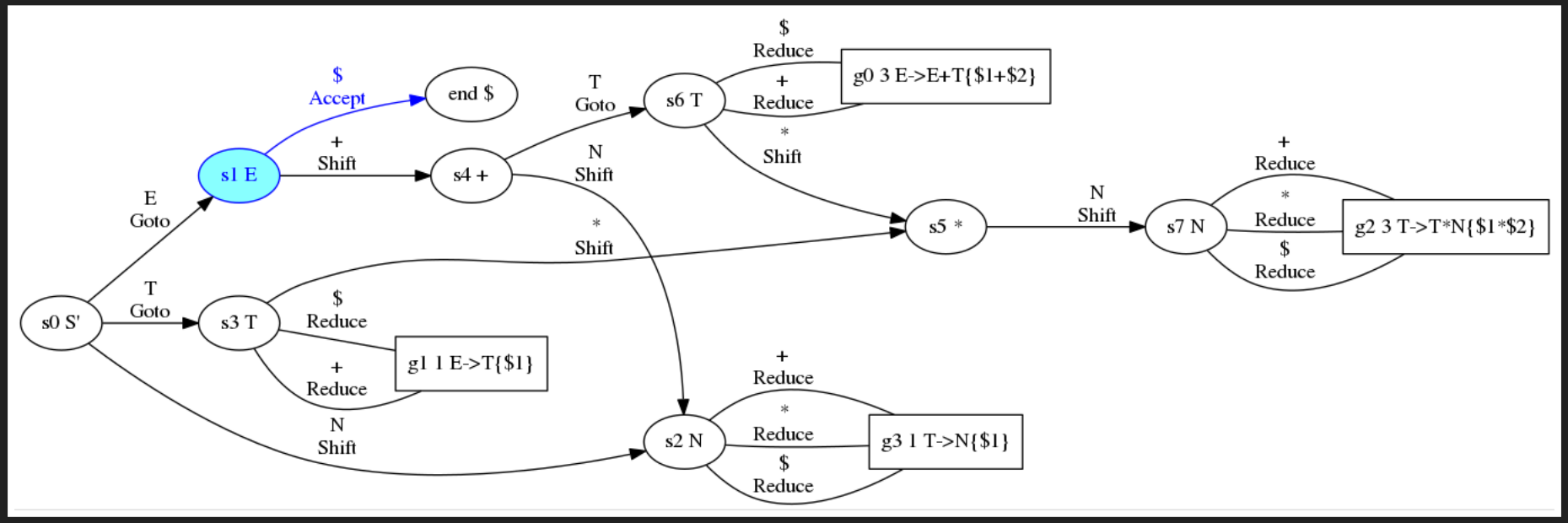
ACCEPT

```
inputs $
status 1 0
results 6
```



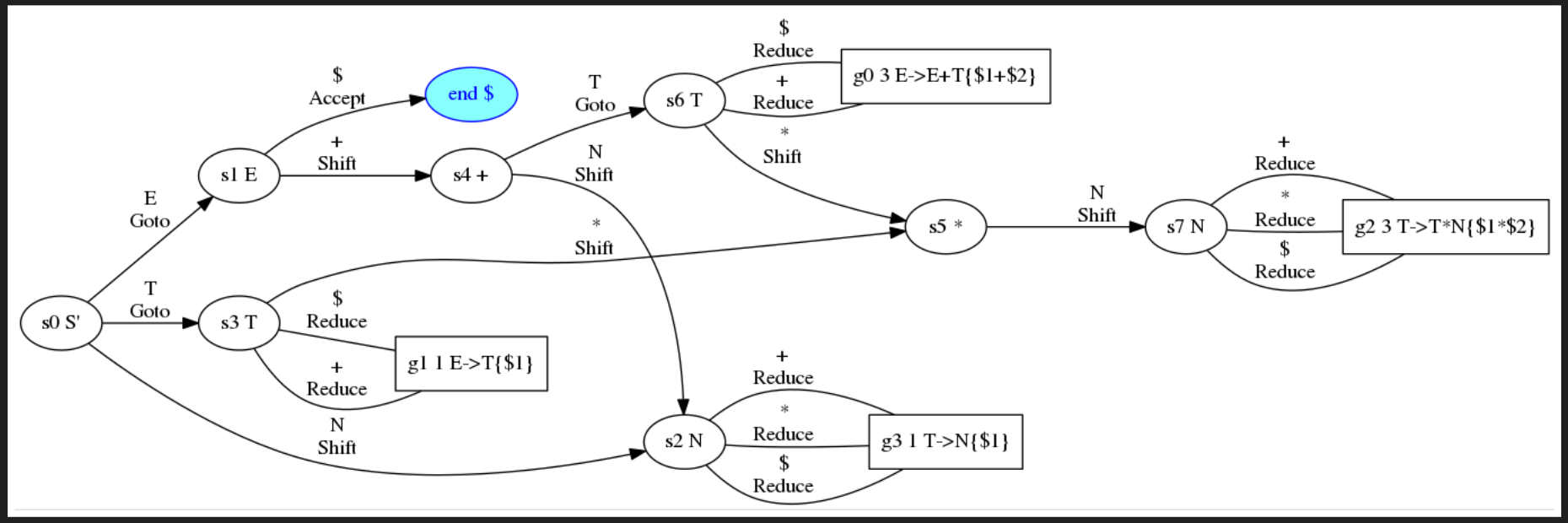
ACCEPT

```
inputs $ アクセプト
status 1 0
results 6
```



ACCEPT

```
inputs $ 結果 6 です
status 1 0
results 6
```

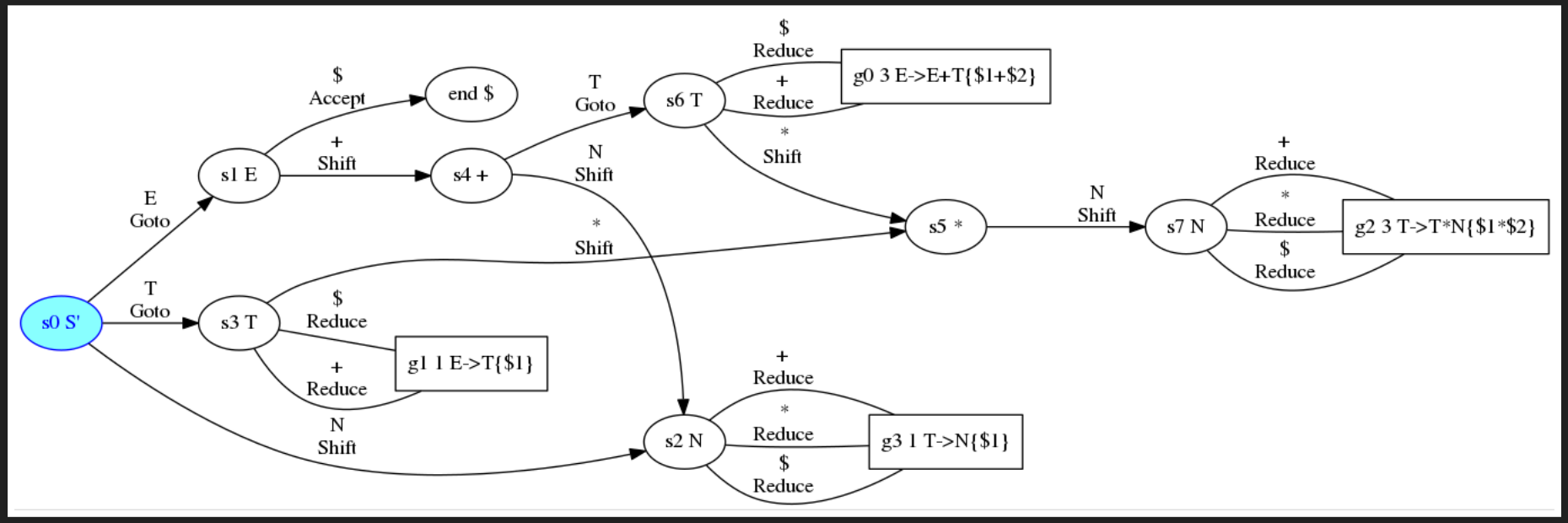


RESULT 1+2+3 = 6

1+2*3

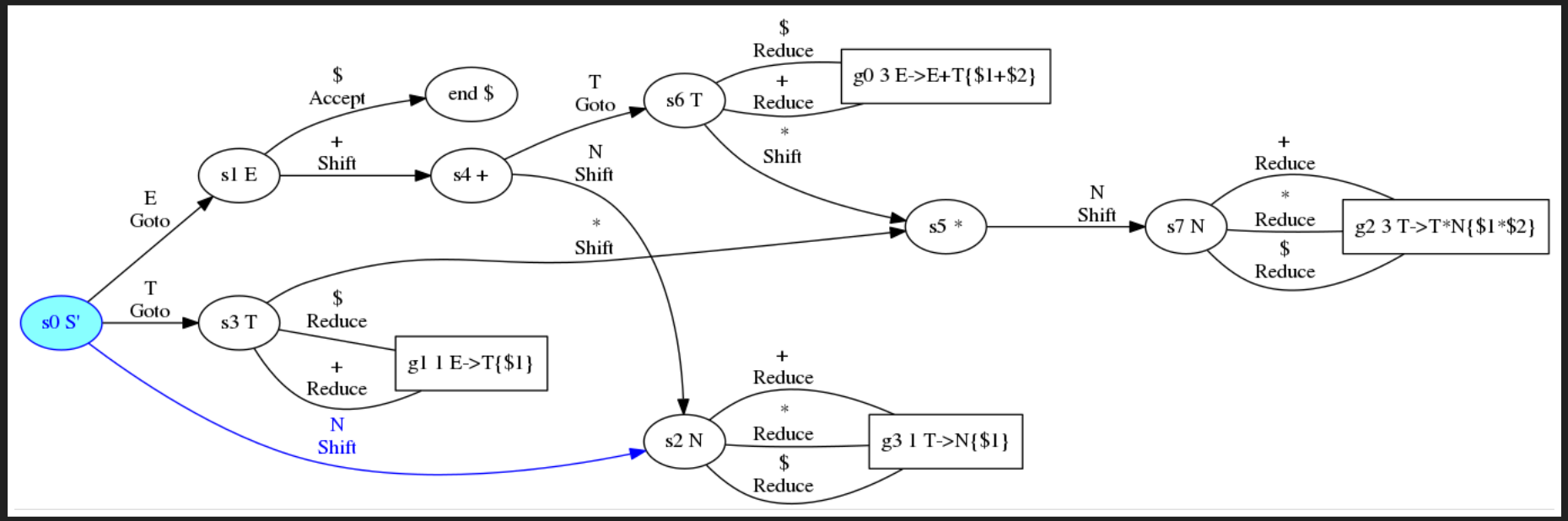
SHIFT S2

```
inputs 1 + 2 * 3 $
status 0
results
```



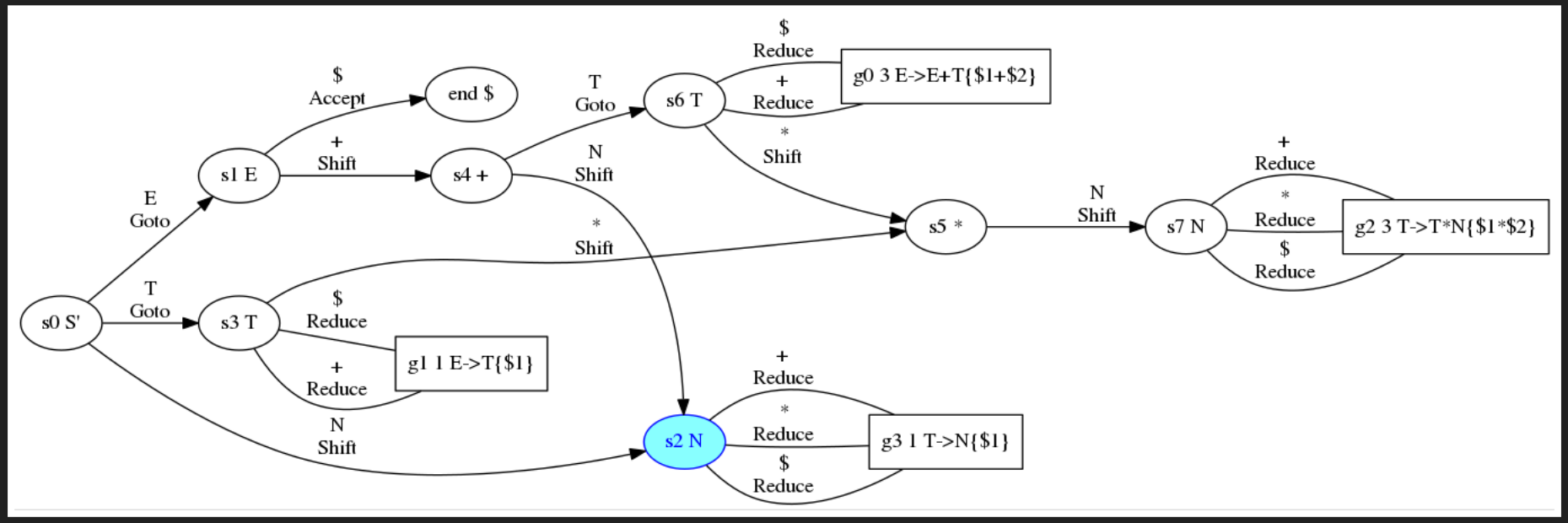
SHIFT S2

```
inputs 1 + 2 * 3 $ ステータスに2をpush 入力1を結果に移動
status 0
results
```



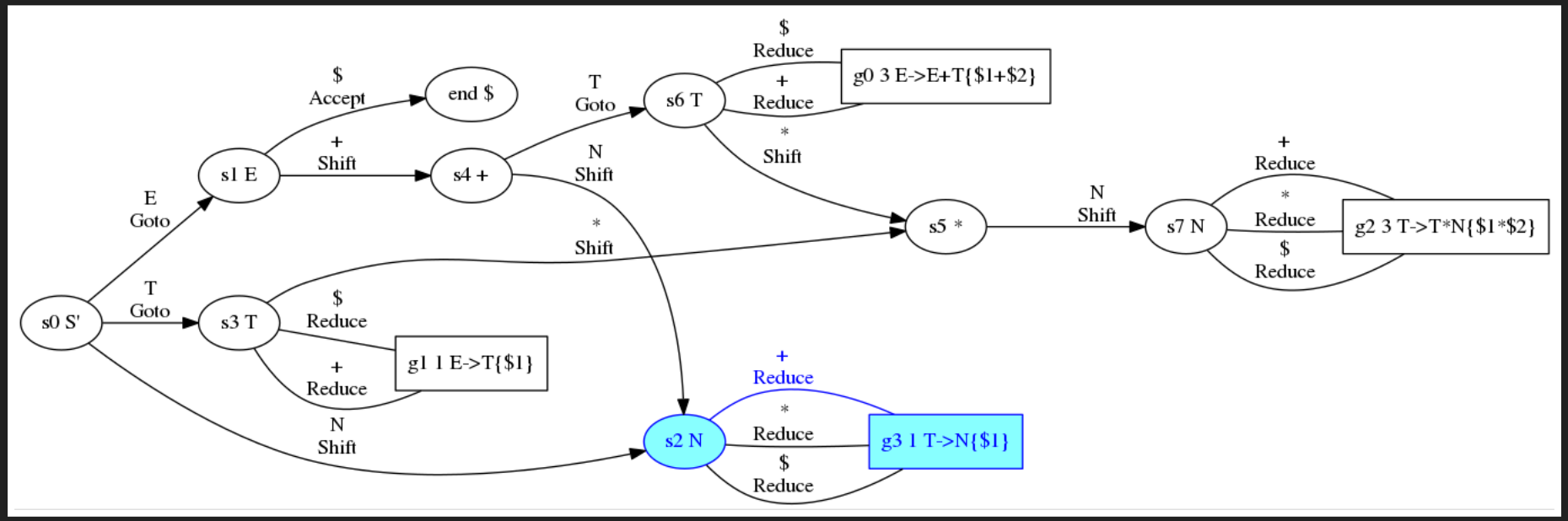
REDUCE G3

```
inputs + 2 * 3 $
status 2 0
results 1
```



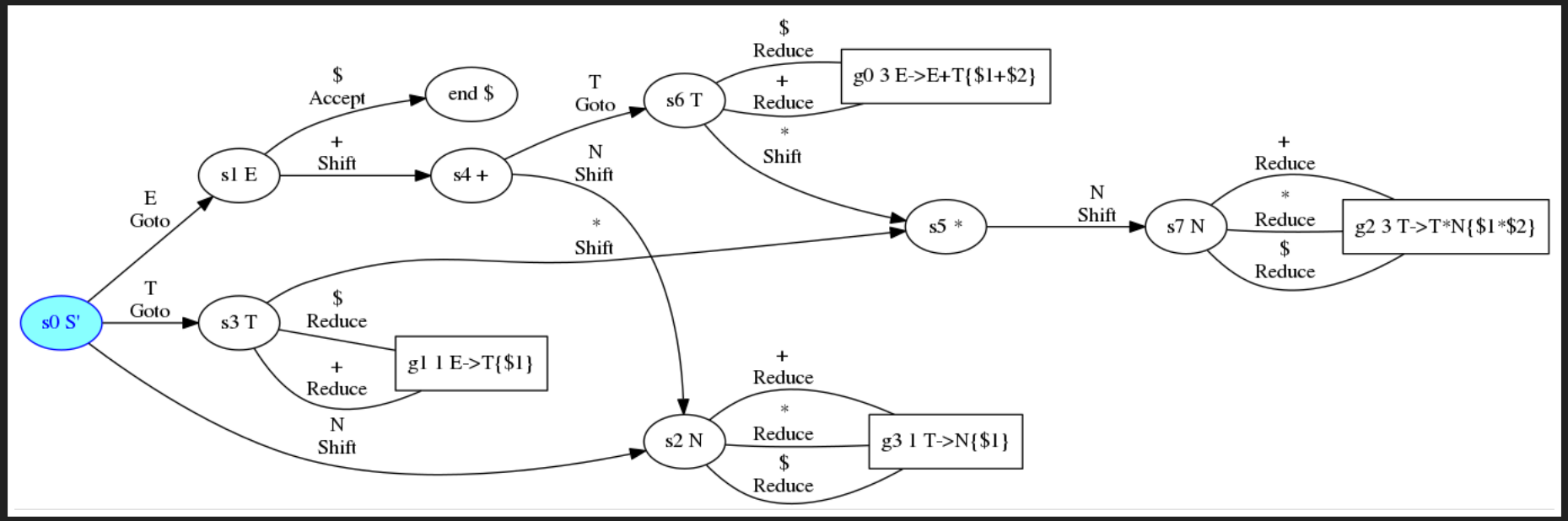
REDUCE G3

```
inputs + 2 * 3 $ 文法g3を見て、1個Pop、{$1}を結果にpush、文法名Tを入力にpush
status 2 0
results 1
```



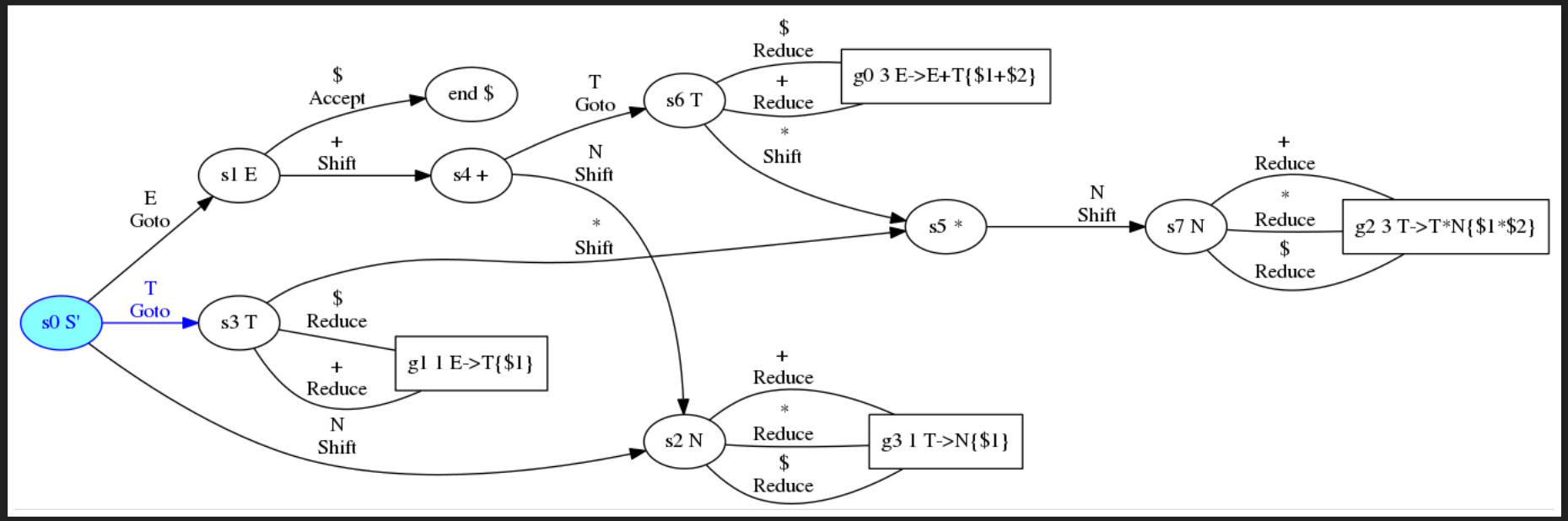
GOTO S3

```
inputs T + 2 * 3 $
status 0
results 1
```



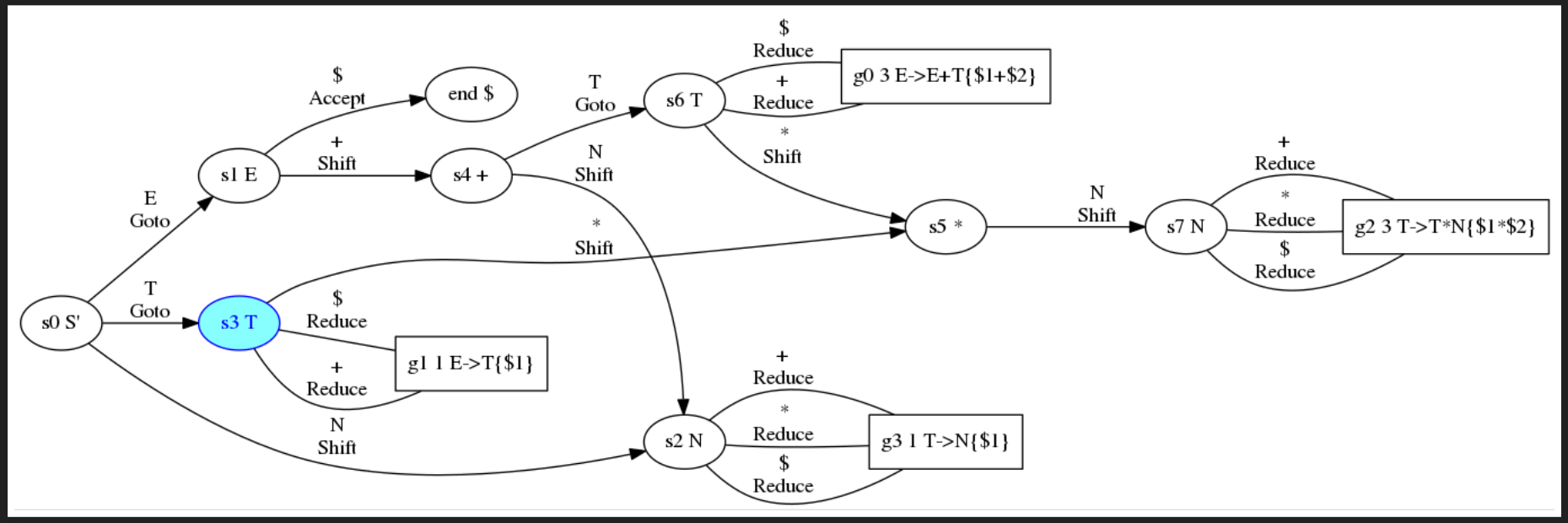
GOTO S3

```
inputs T + 2 * 3 $ ステータスに3をpush,入力Tを捨てる
status 0
results 1
```



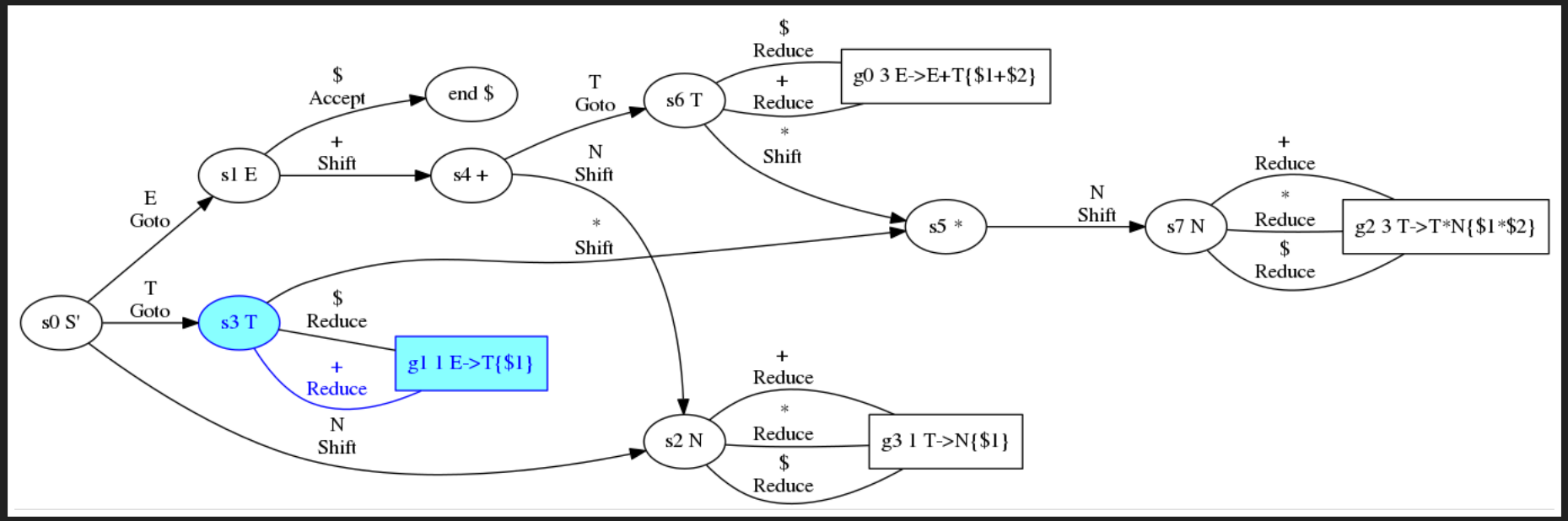
REDUCE G1

```
inputs + 2 * 3 $
status 3 0
results 1
```



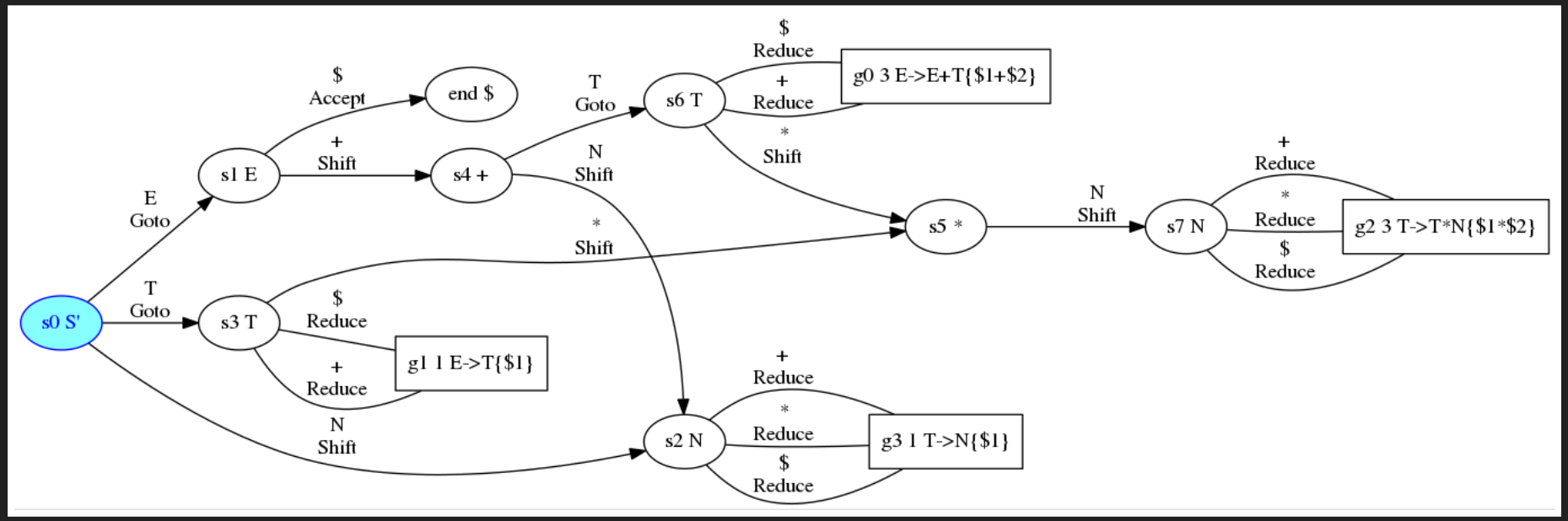
REDUCE G1

```
inputs + 2 * 3 $ 文法g1を見て、1個Pop、{$1}を結果にpush、文法名Eを入力にpush
status 3 0
results 1
```



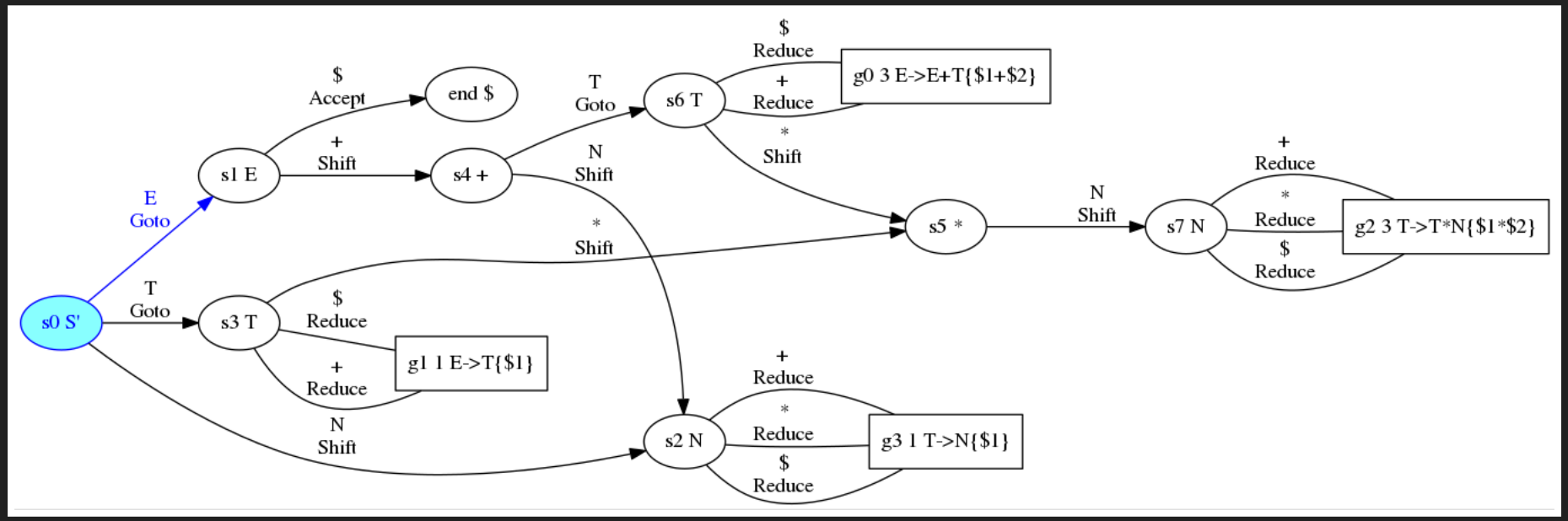
GOTO S1

```
inputs E + 2 * 3 $
status 0
results 1
```



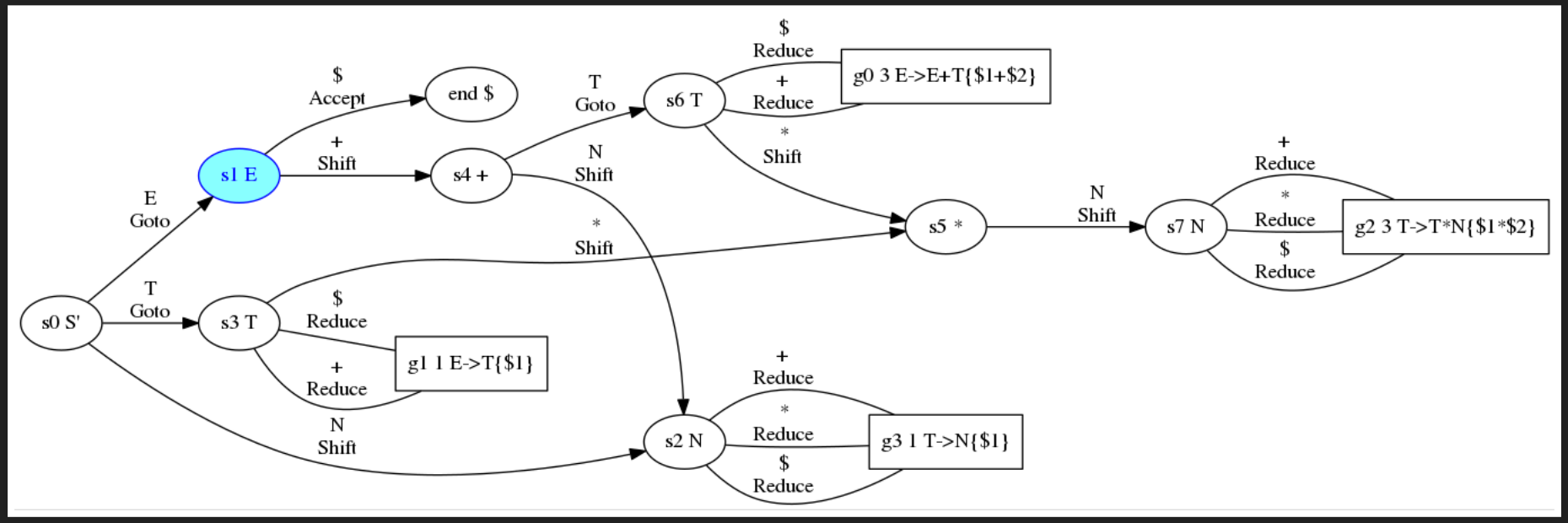
GOTO S1

```
inputs E + 2 * 3 $ ステータスに1をpush,入力Eを捨てる
status 0
results 1
```



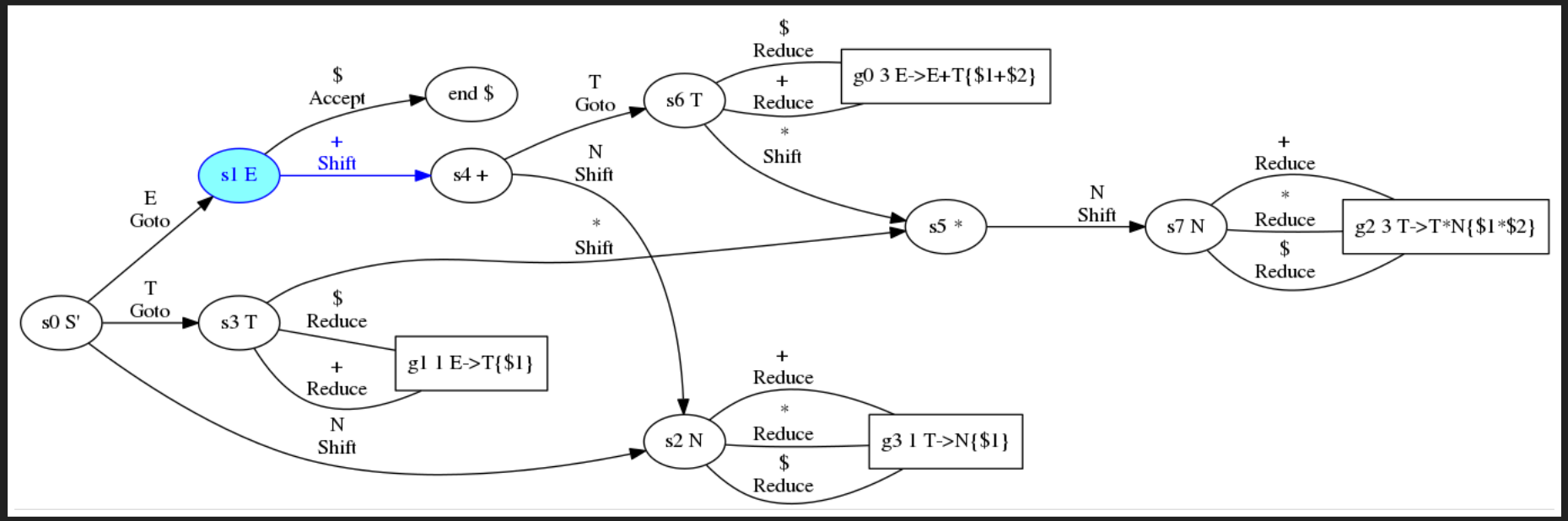
SHIFT S4

```
inputs + 2 * 3 $
status 1 0
results 1
```



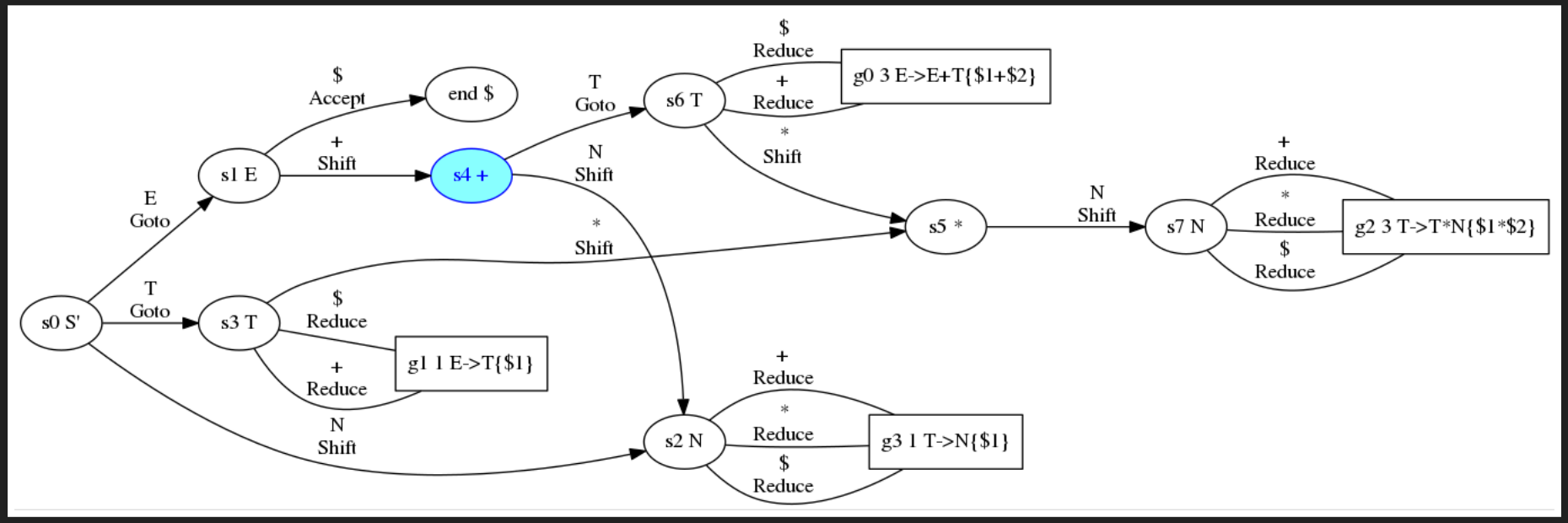
SHIFT S4

```
inputs + 2 * 3 $ ステータスに4をpush 入力+を結果に移動
status 1 0
results 1
```



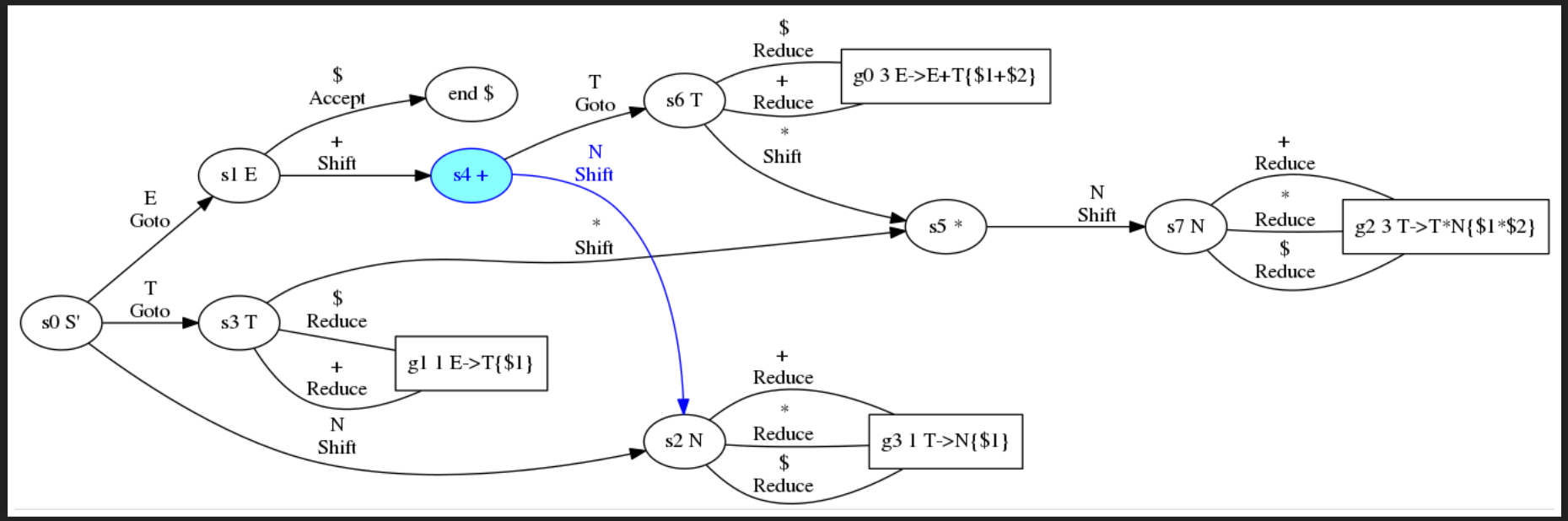
SHIFT S2

```
inputs 2 * 3 $
status 4 1 0
results + 1
```



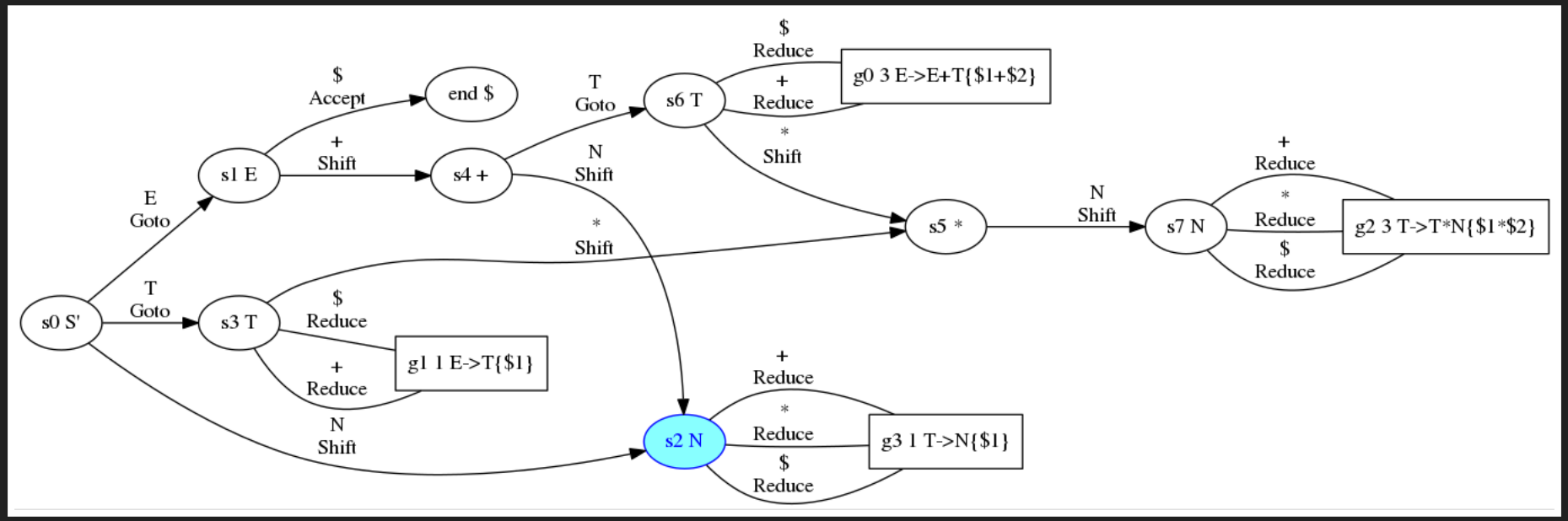
SHIFT S2

```
inputs 2 * 3 $ ステータスに2をpush 入力2を結果に移動
status 4 1 0
results + 1
```



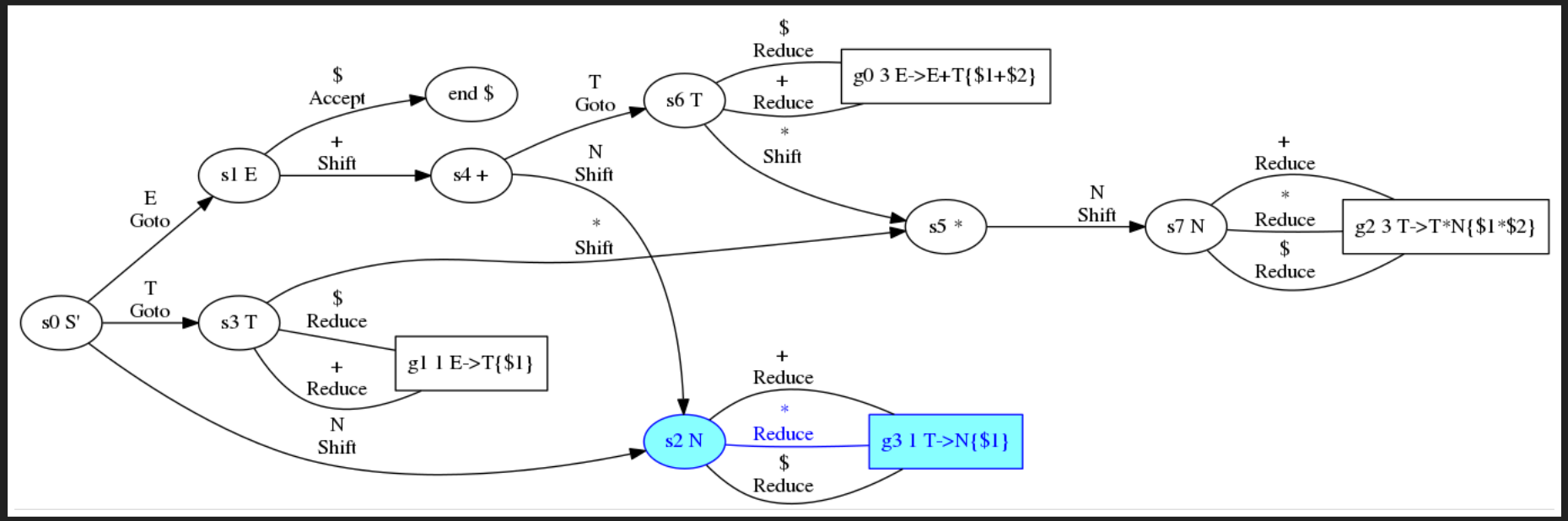
REDUCE G3

```
inputs * 3 $
status 2 4 1 0
results 2 + 1
```



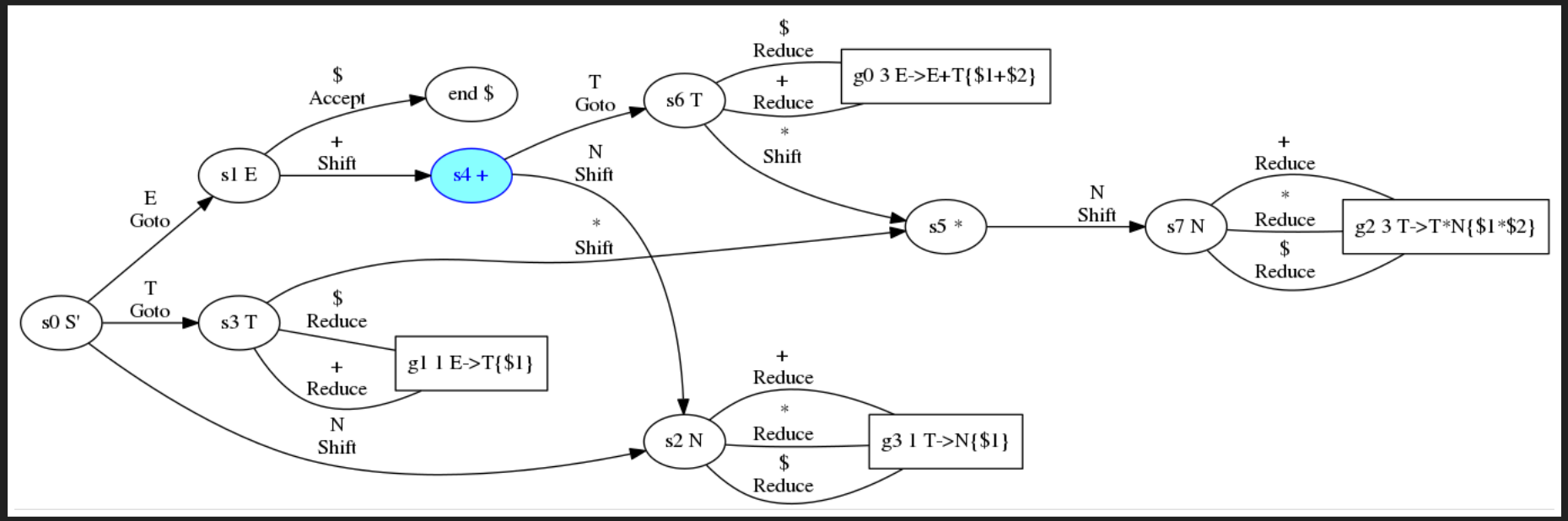
REDUCE G3

```
inputs * 3 $ 文法g3を見て、1個Pop、{$1}を結果にpush、文法名Tを入力にpush
status 2 4 1 0
results 2 + 1
```



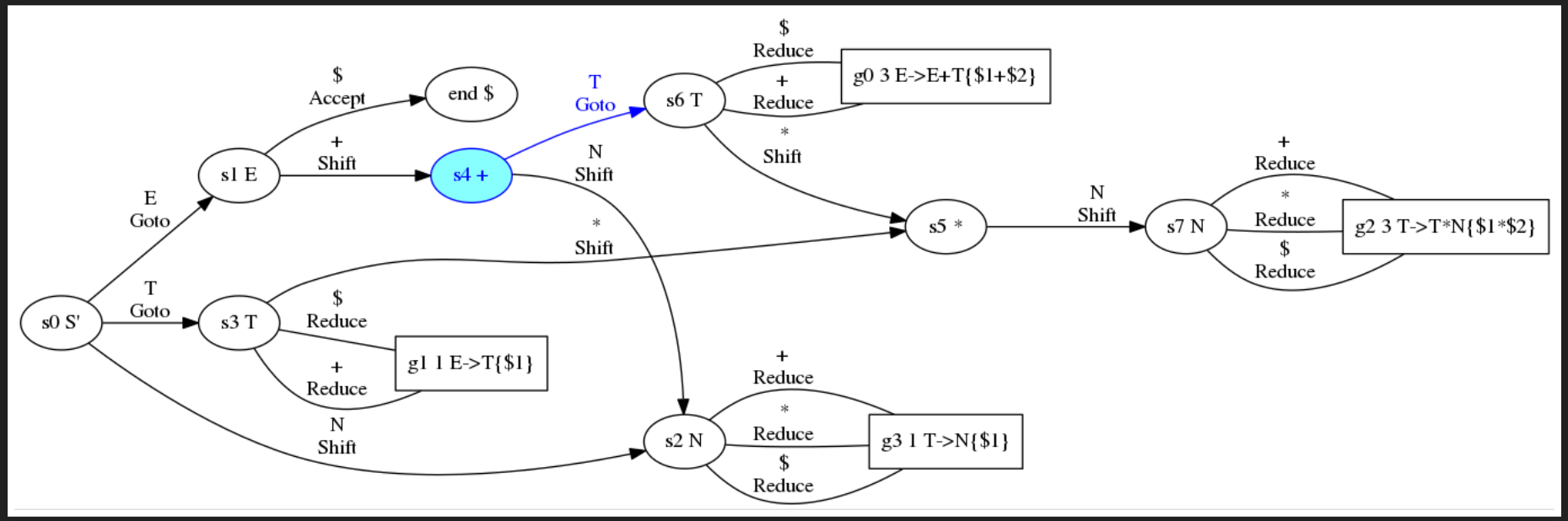
GOTO S6

```
inputs T * 3 $
status 4 1 0
results 2 + 1
```



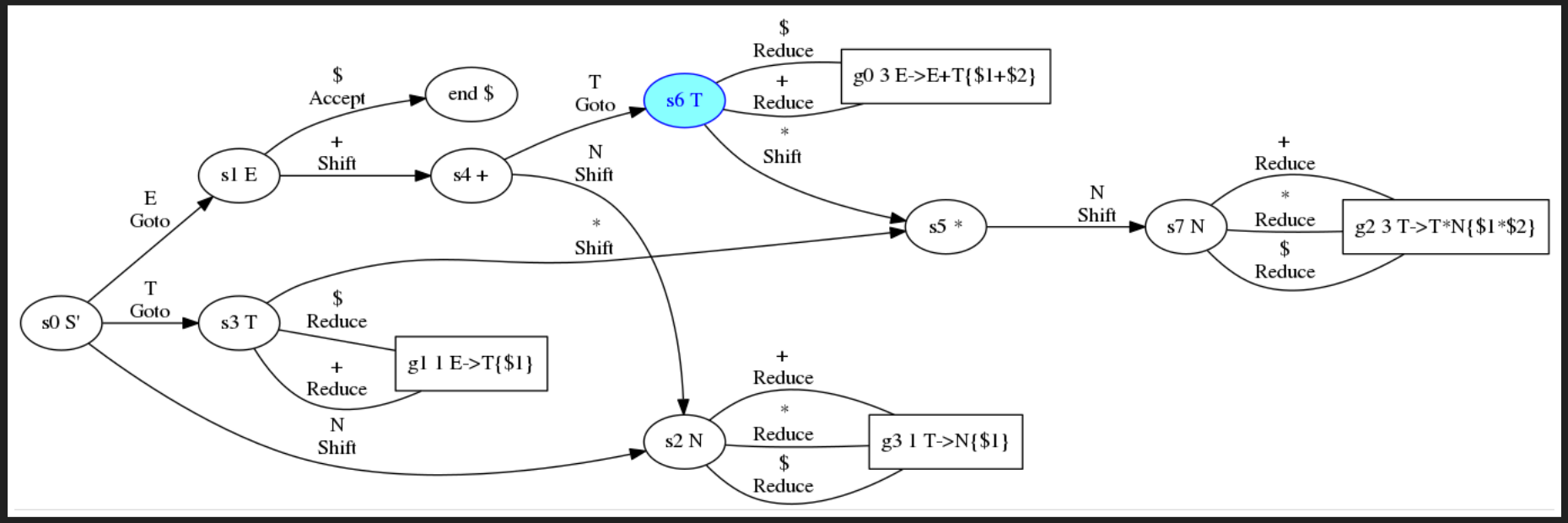
GOTO S6

```
inputs T * 3 $ ステータスに6をpush,入力Tを捨てる
status 4 1 0
results 2 + 1
```



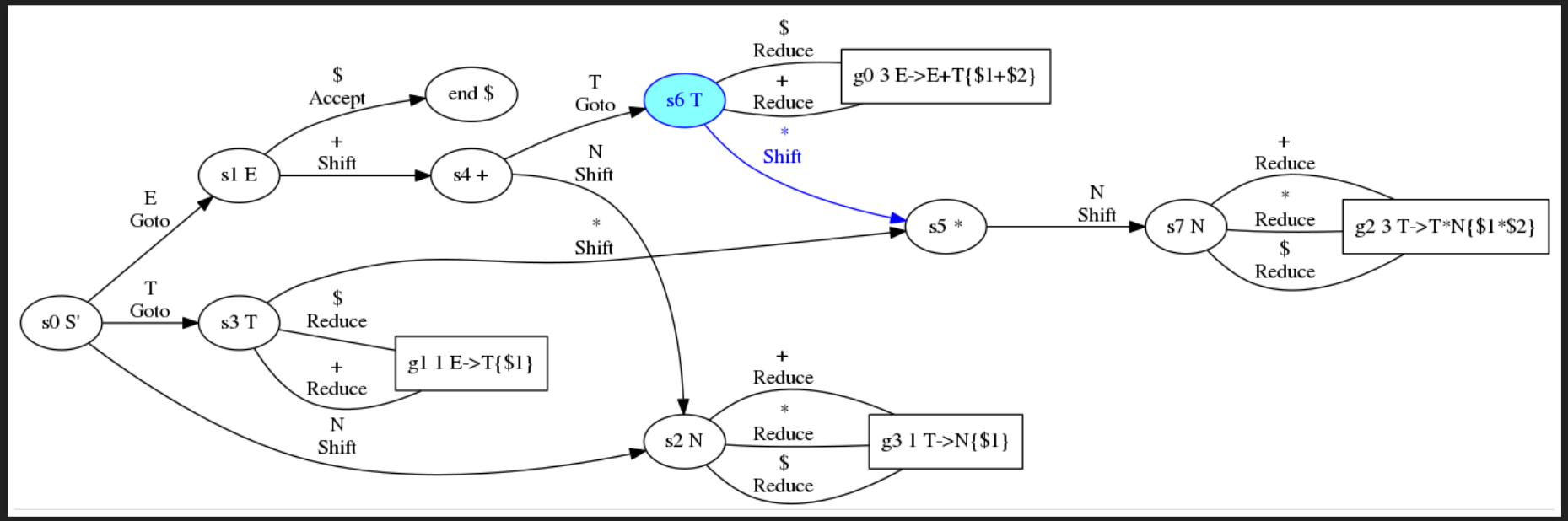
SHIFT S5

```
inputs * 3 $
status 6 4 1 0
results 2 + 1
```



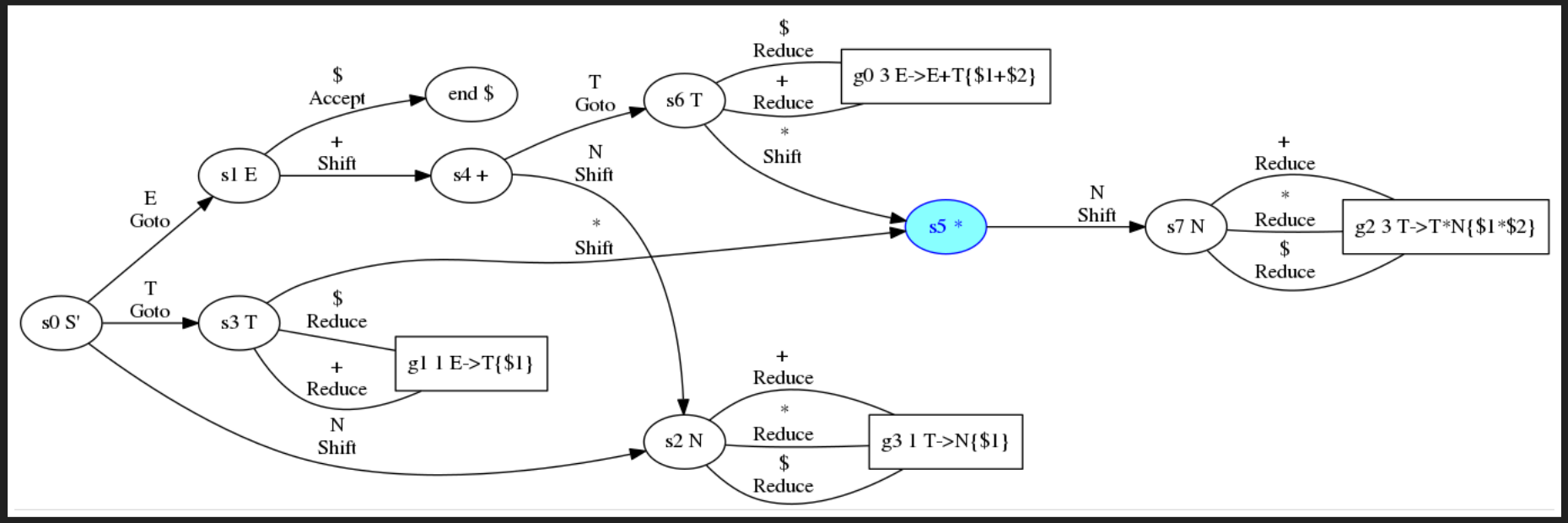
SHIFT S5

```
inputs * 3 $ ステータスに5をpush 入力*を結果に移動
status 6 4 1 0
results 2 + 1
```



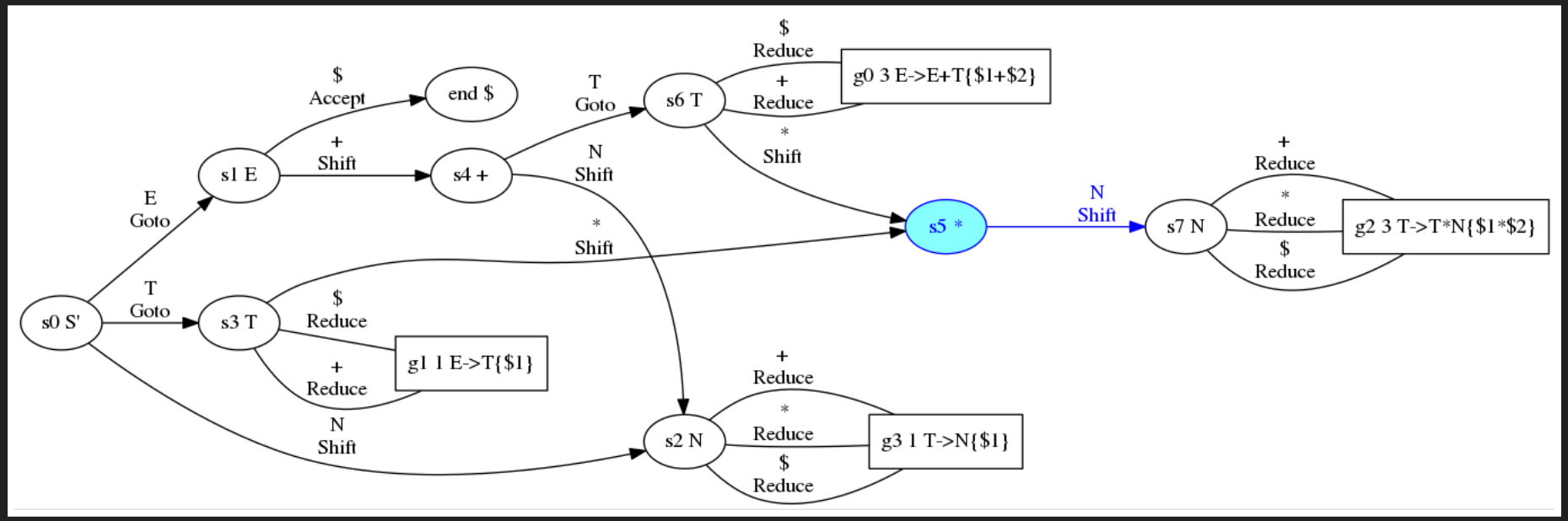
SHIFT S7

```
inputs 3 $
status 5 6 4 1 0
results * 2 + 1
```



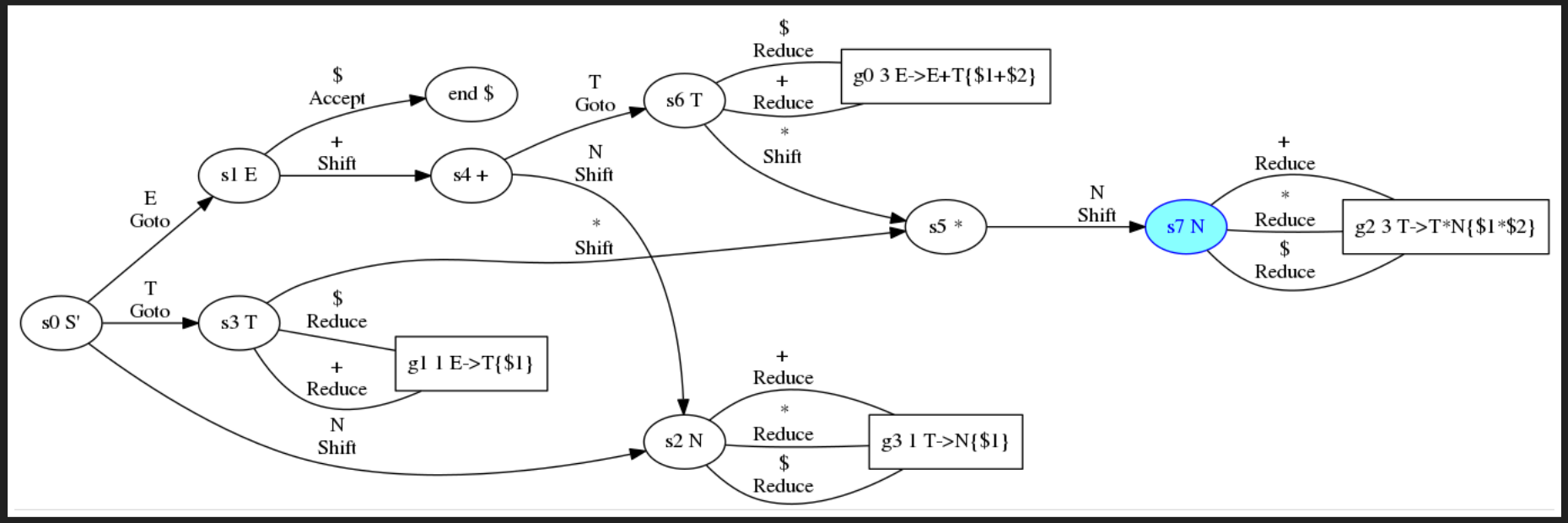
SHIFT S7

```
inputs 3 $ ステータスに7をpush 入力3を結果に移動
status 5 6 4 1 0
results * 2 + 1
```



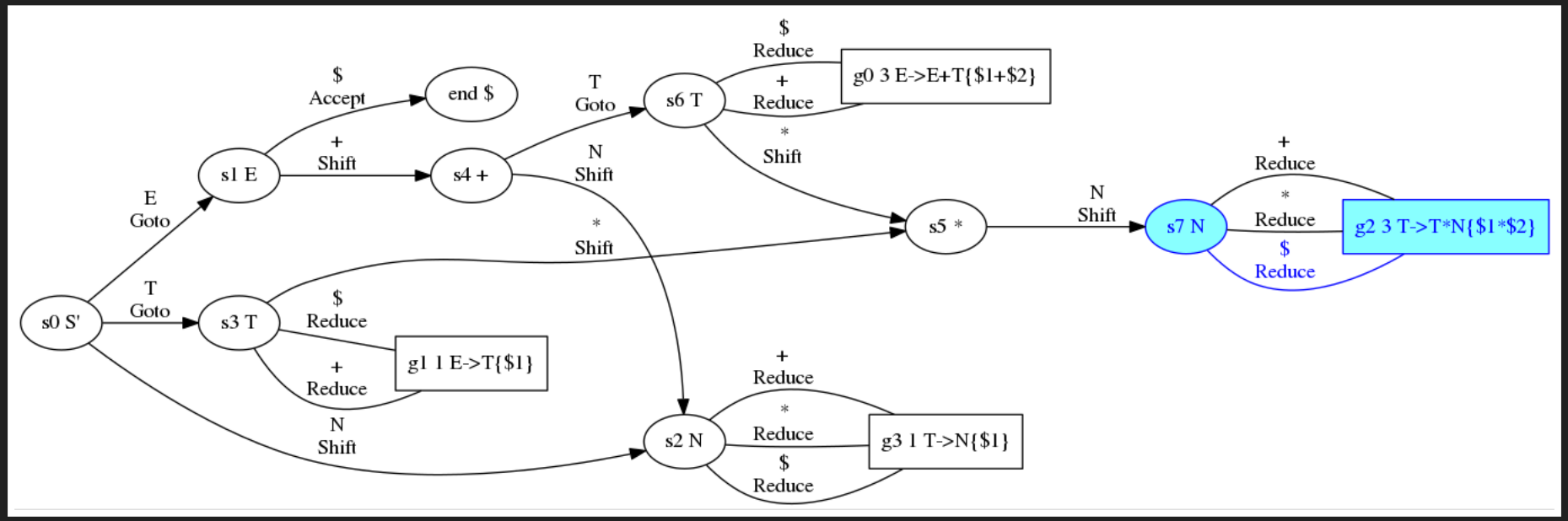
REDUCE G2

```
inputs $
status 7 5 6 4 1 0
results 3 * 2 + 1
```



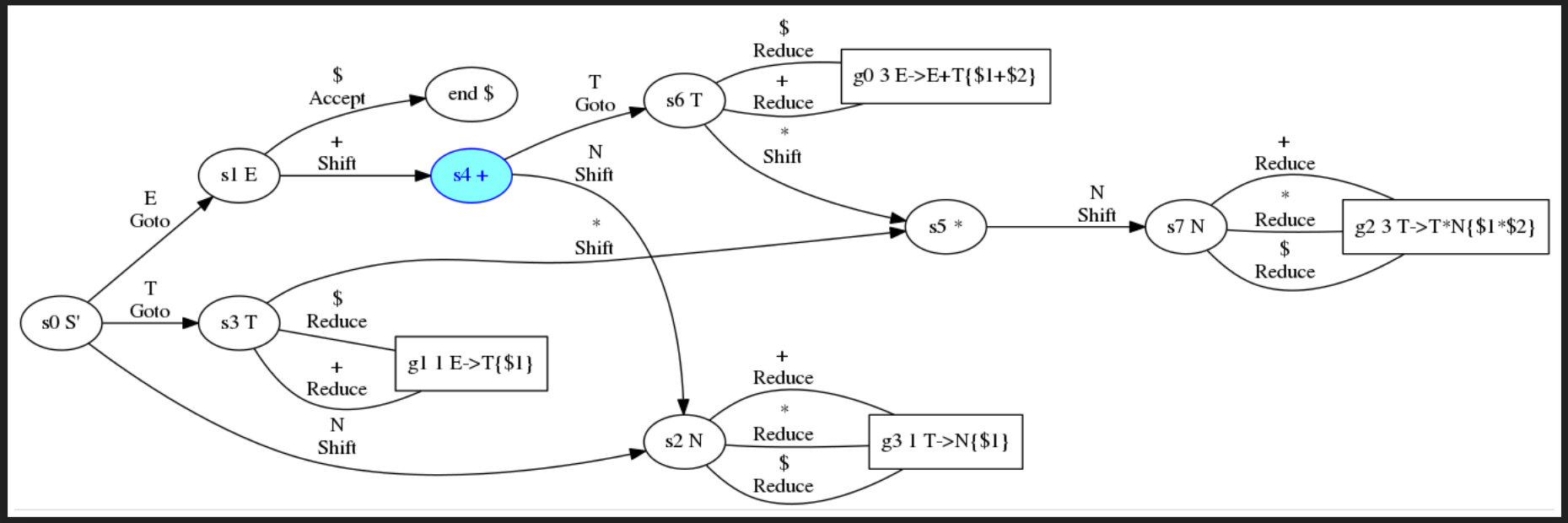
REDUCE G2

```
inputs $ 文法g2を見て、3個Pop、{$1*$2}を結果にpush、文法名Tを入力にpush
status 7 5 6 4 1 0
results 3 * 2 + 1
```



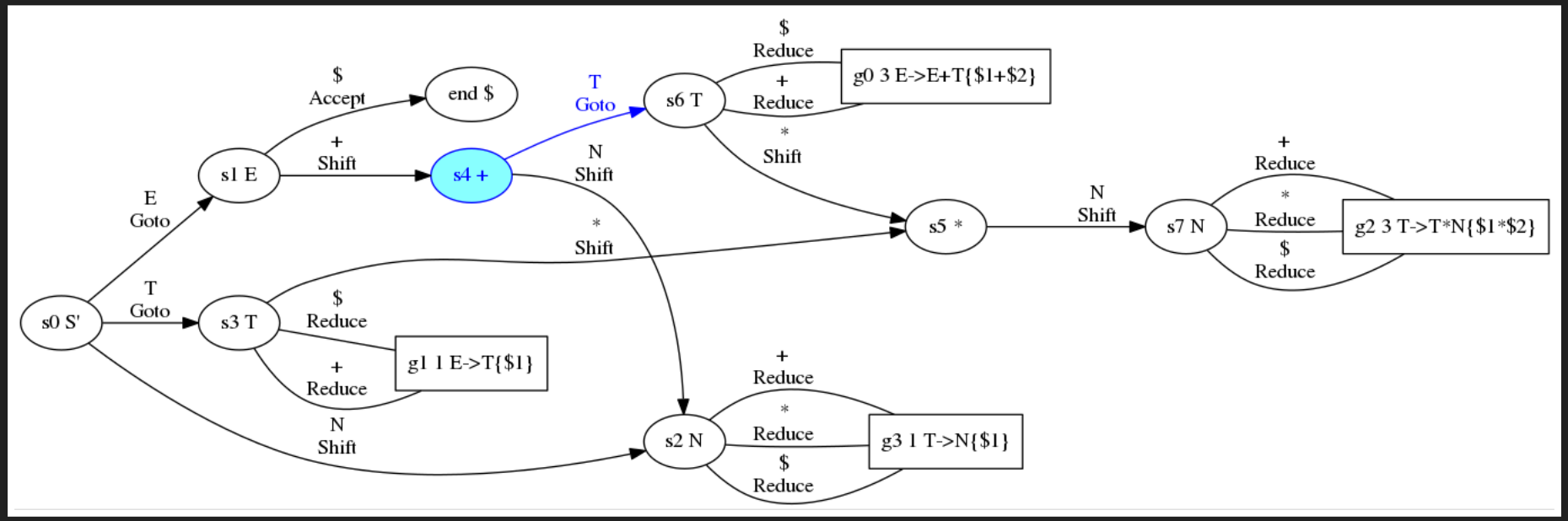
GOTO S6

```
inputs T $
status 4 1 0
results 6 + 1
```



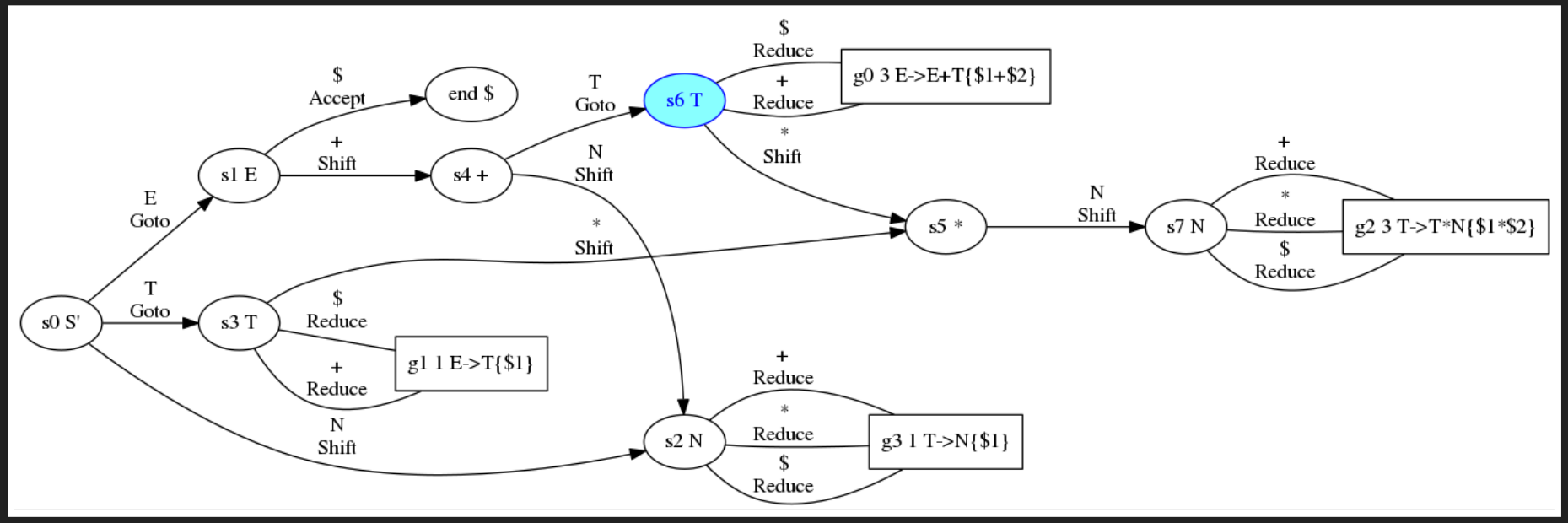
GOTO S6

```
inputs T $ ステータスに6をpush,入力Tを捨てる
status 4 1 0
results 6 + 1
```



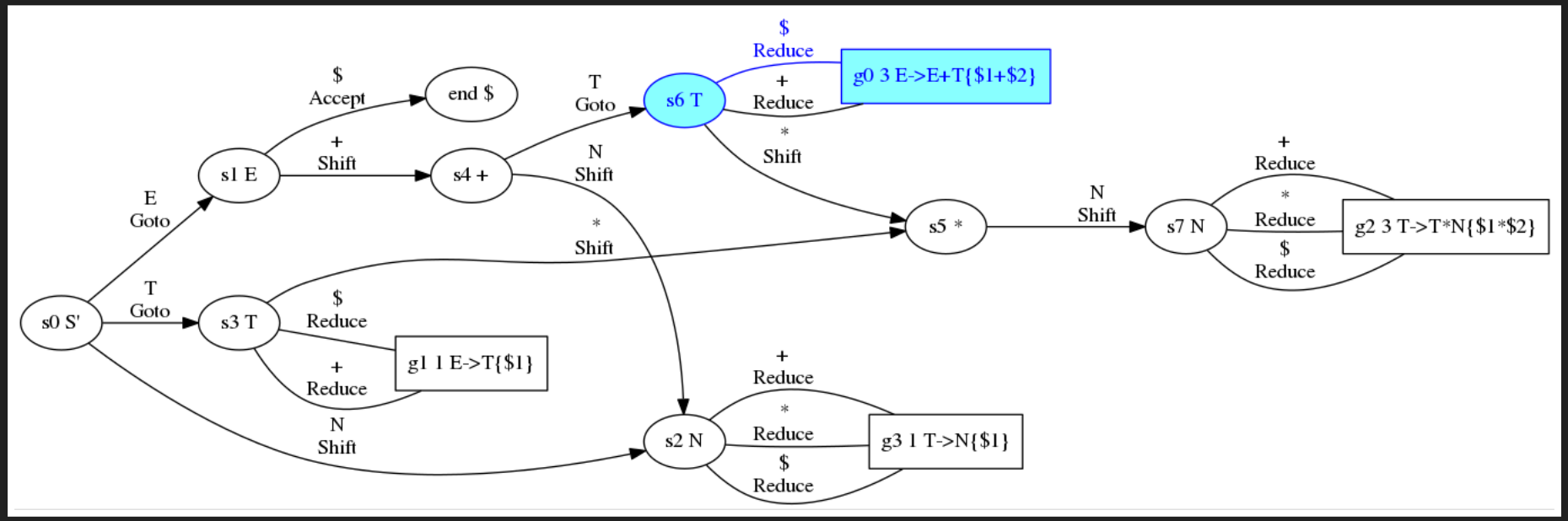
REDUCE G0

```
inputs $
status 6 4 1 0
results 6 + 1
```



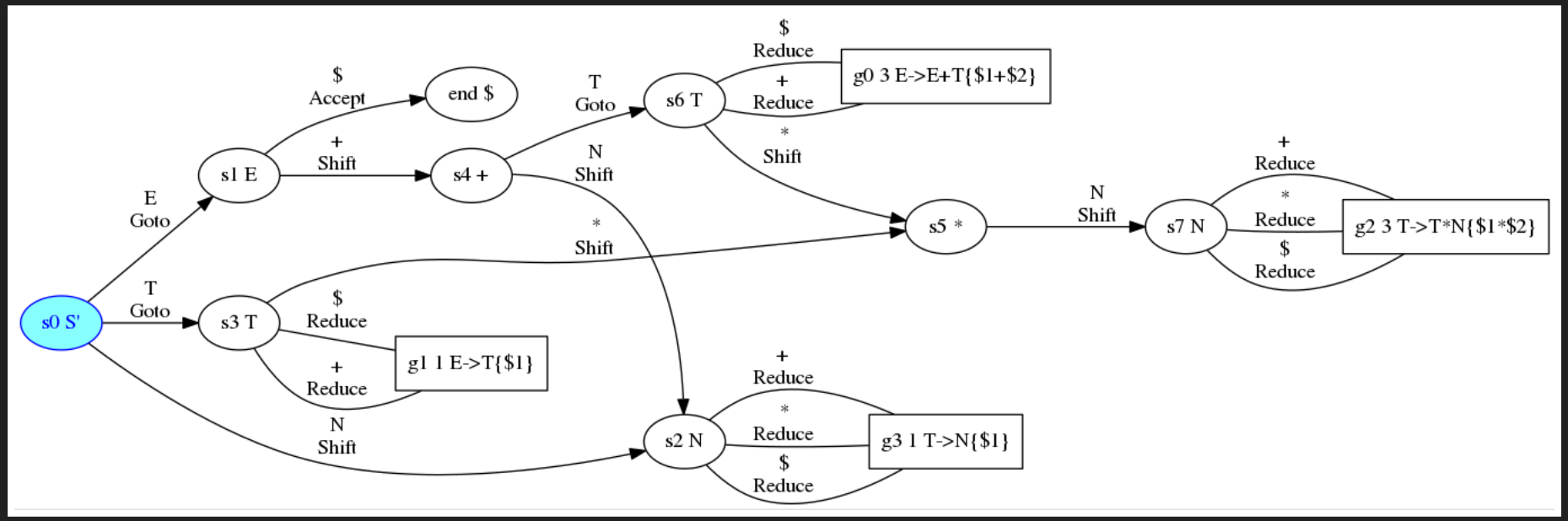
REDUCE G0

```
inputs $ 文法g0を見て、3個Pop、{$1+$2}を結果にpush、文法名Eを入力にpush
status 6 4 1 0
results 6 + 1
```



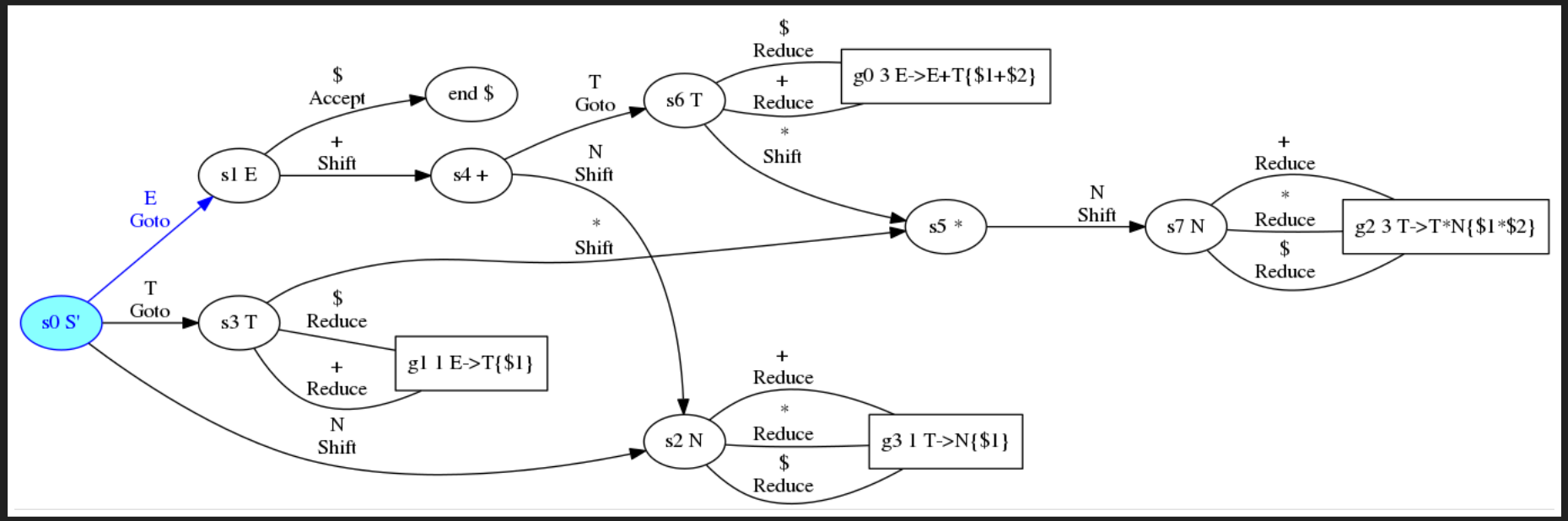
GOTO S1

```
inputs E $
status 0
results 7
```



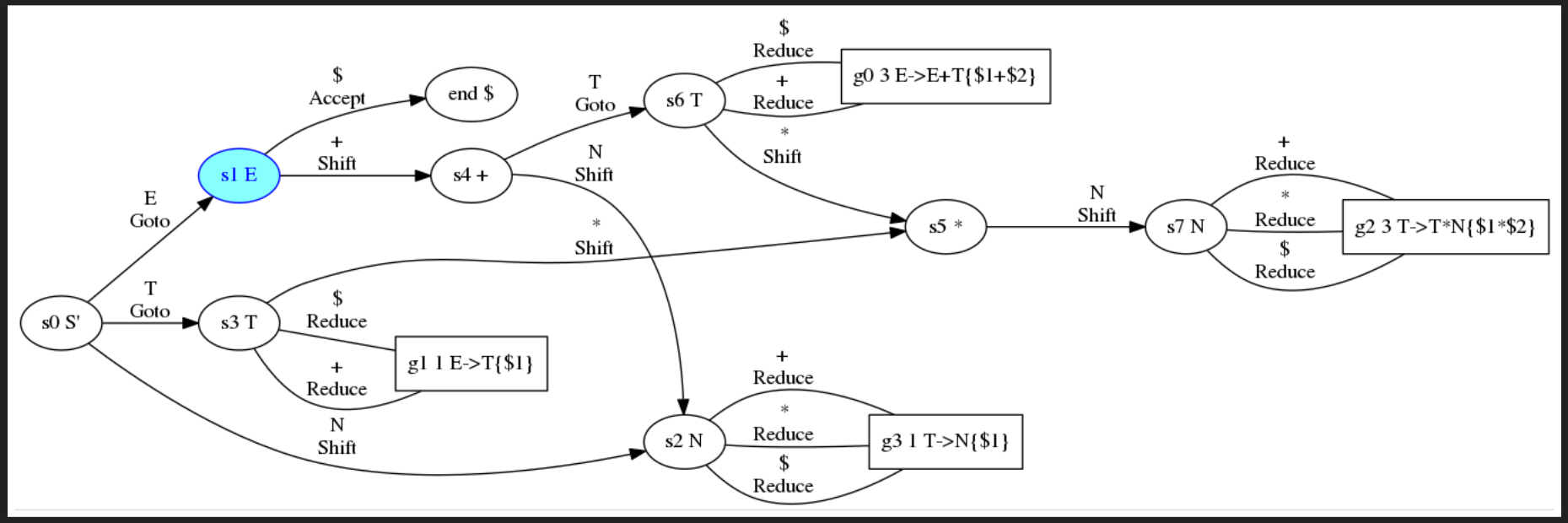
GOTO S1

```
inputs E $ ステータスに1をpush,入力Eを捨てる
status 0
results 7
```



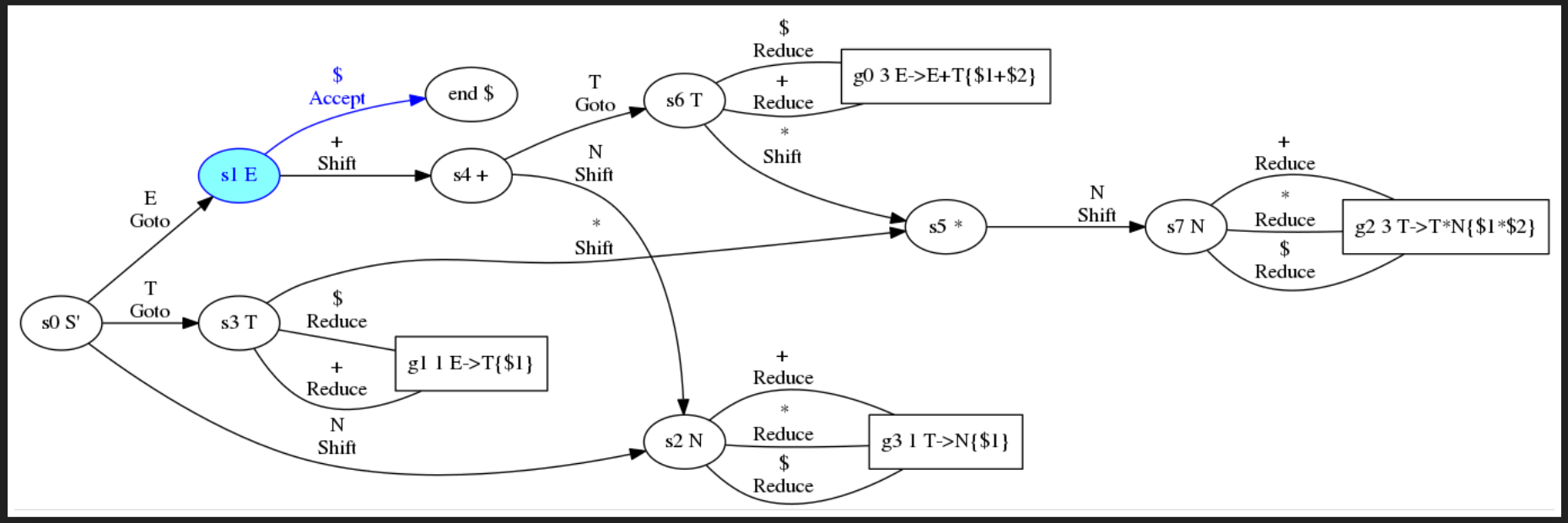
ACCEPT

```
inputs $
status 1 0
results 7
```



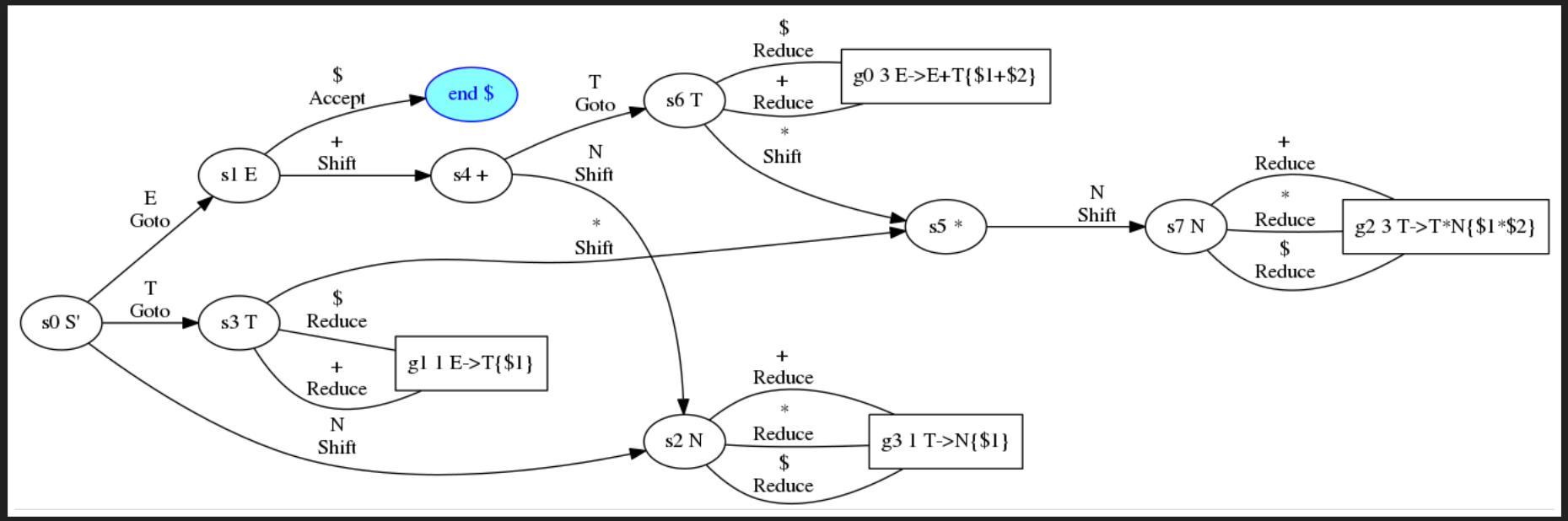
ACCEPT

```
inputs $ アクセプト
status 1 0
results 7
```



ACCEPT

```
inputs $ 結果 7 です
status 1 0
results 7
```

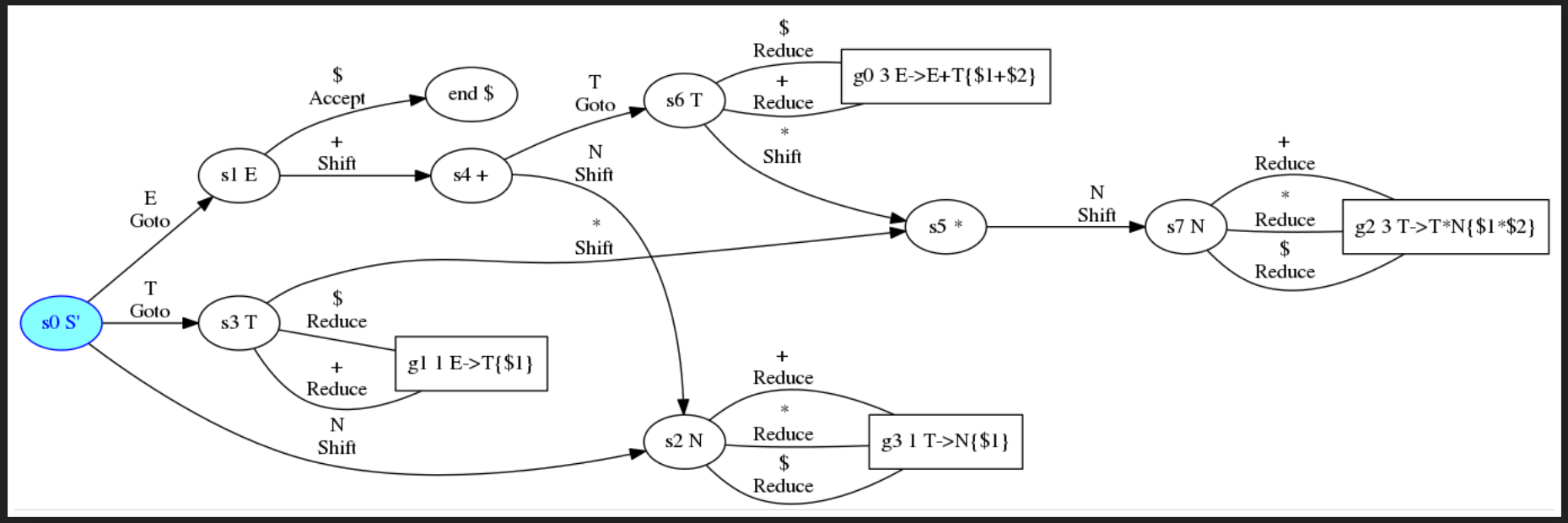


RESULT 1+2*3 = 7

2*3+4

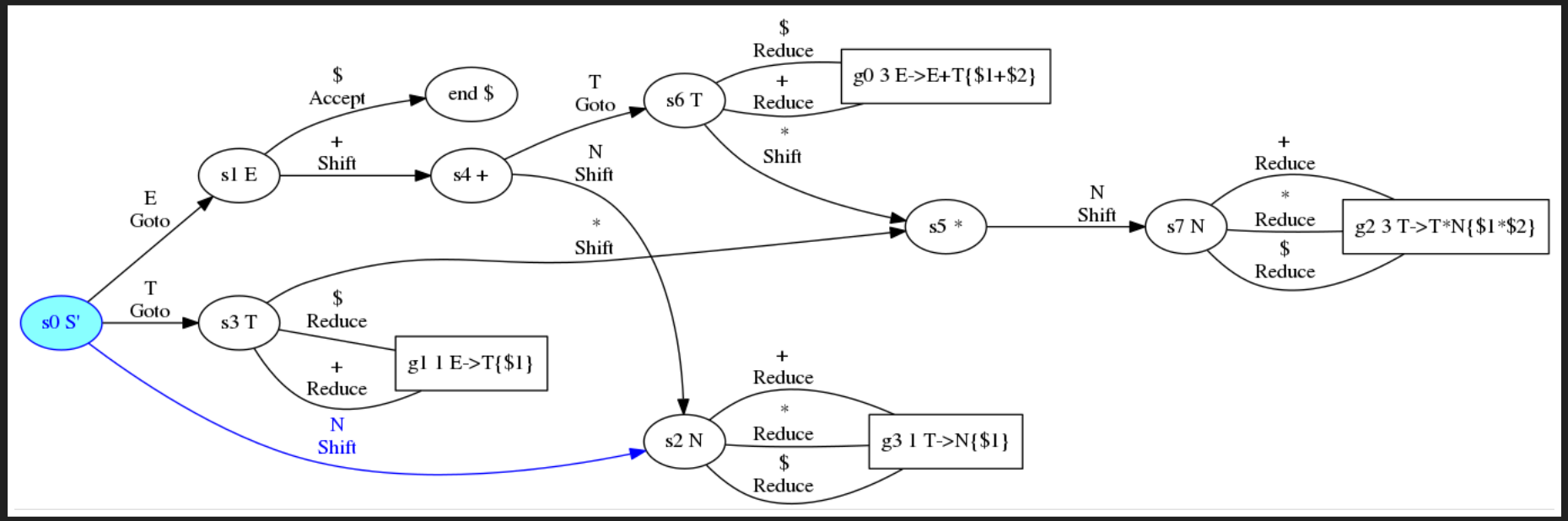
SHIFT S2

```
inputs 2 * 3 + 4 $
status 0
results
```



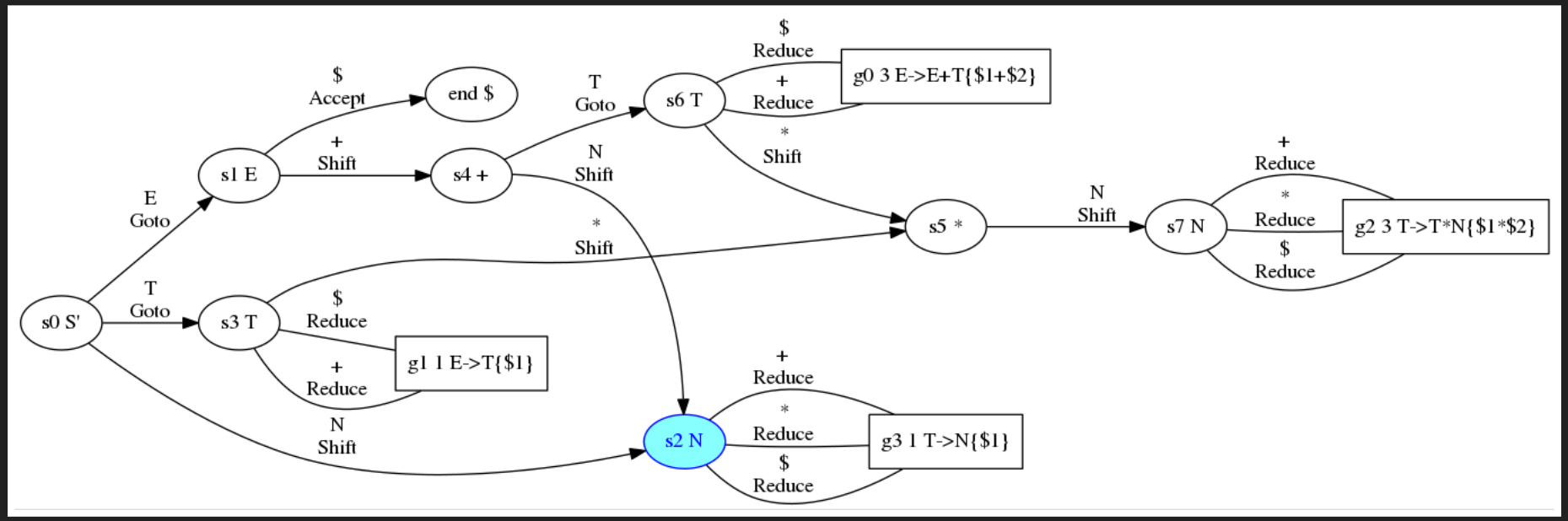
SHIFT S2

```
inputs 2 * 3 + 4 $ ステータスに2をpush 入力2を結果に移動
status 0
results
```



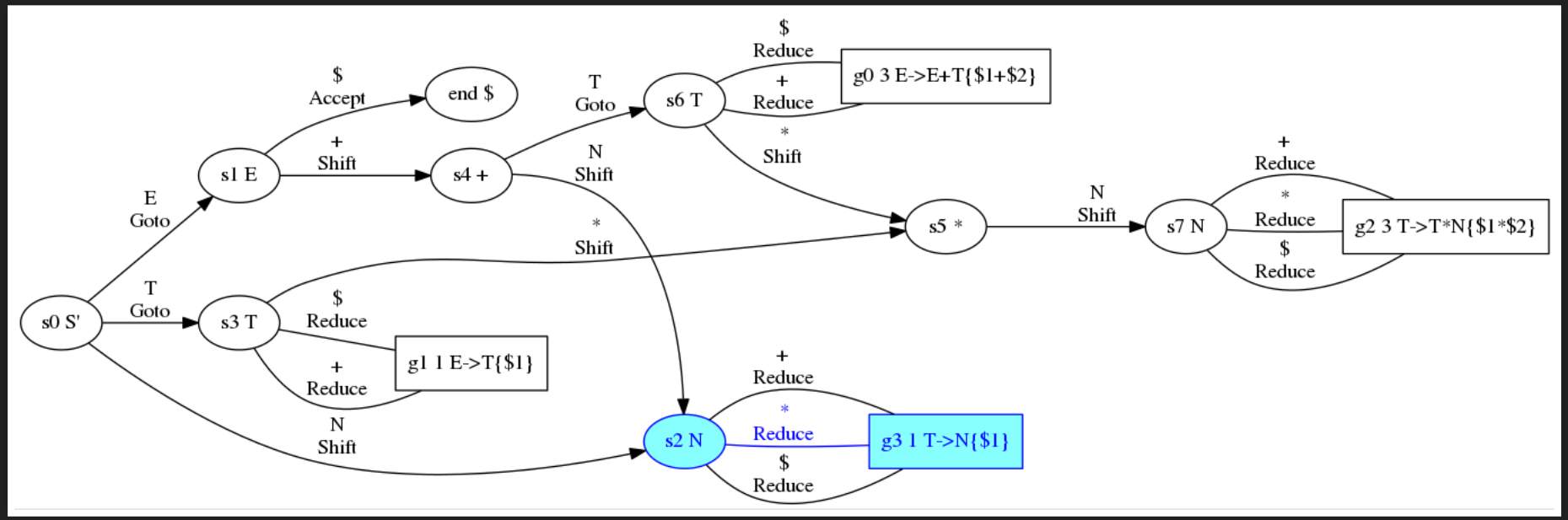
REDUCE G3

```
inputs * 3 + 4 $
status 2 0
results 2
```



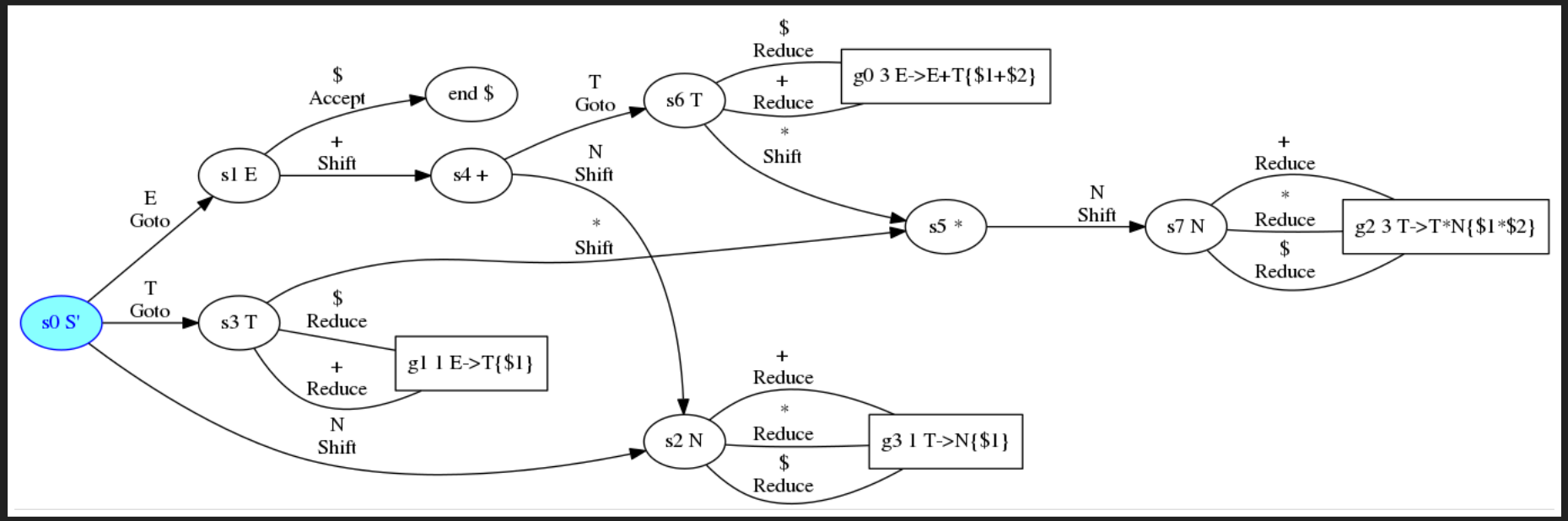
REDUCE G3

```
inputs * 3 + 4 $ 文法g3を見て、1個Pop、{$1}を結果にpush、文法名Tを入力にpush
status 2 0
results 2
```



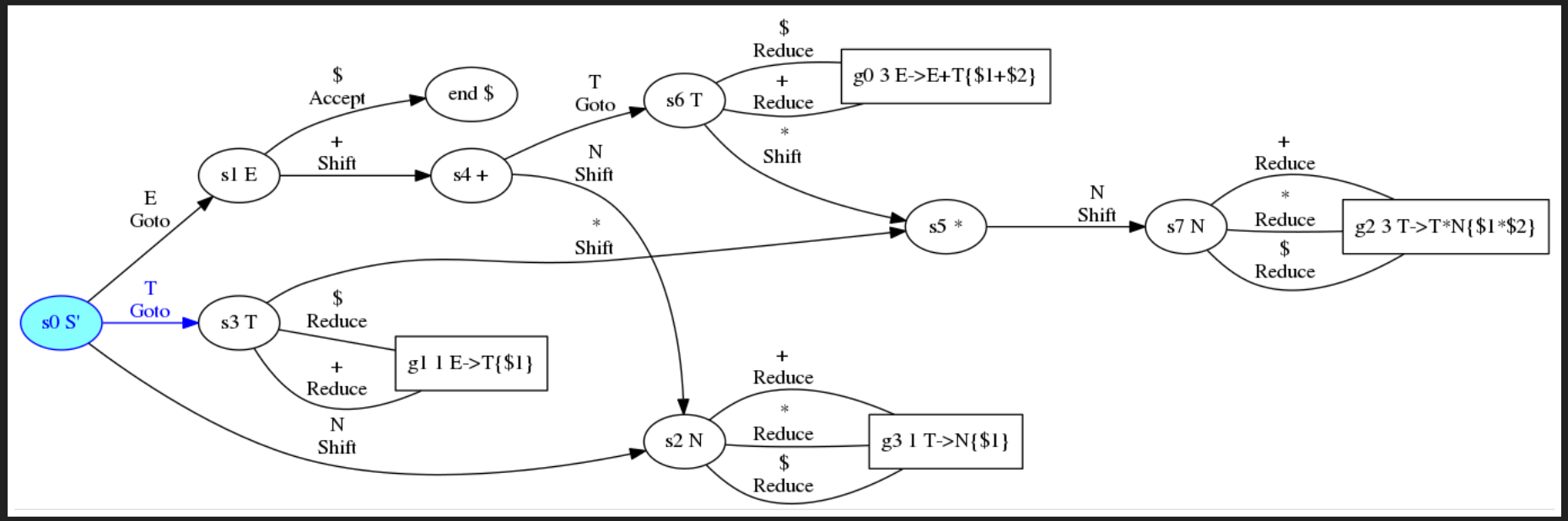
GOTO S3

```
inputs T * 3 + 4 $
status 0
results 2
```



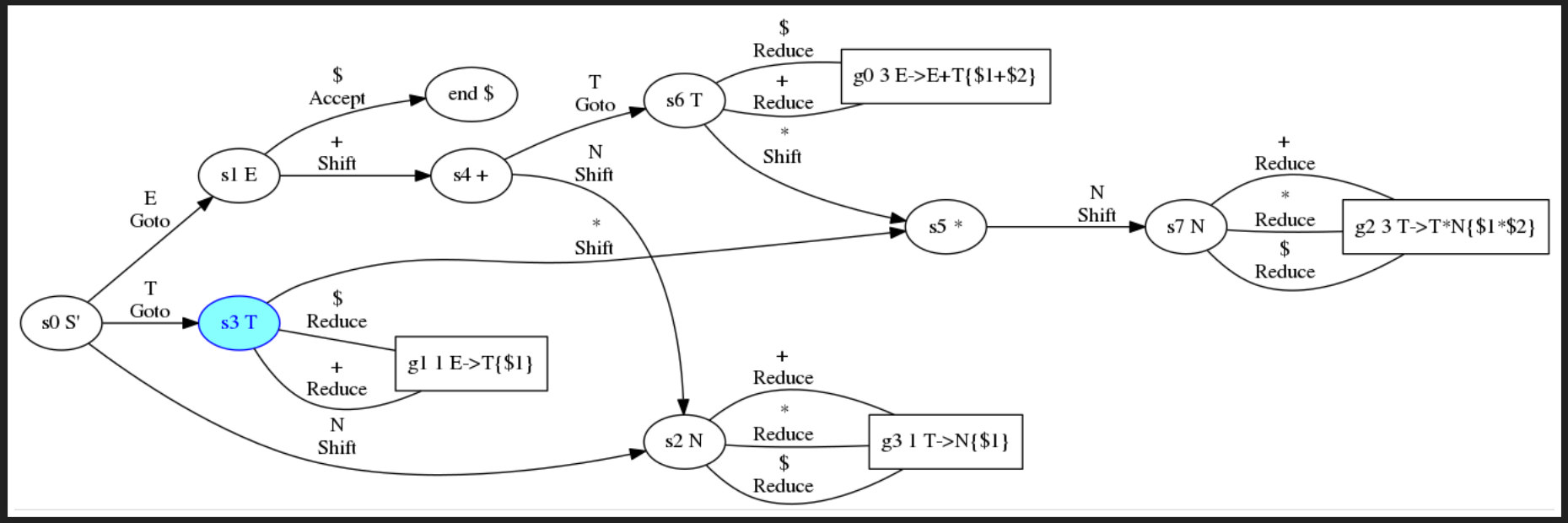
GOTO S3

```
inputs T * 3 + 4 $ ステータスに3をpush,入力Tを捨てる
status 0
results 2
```



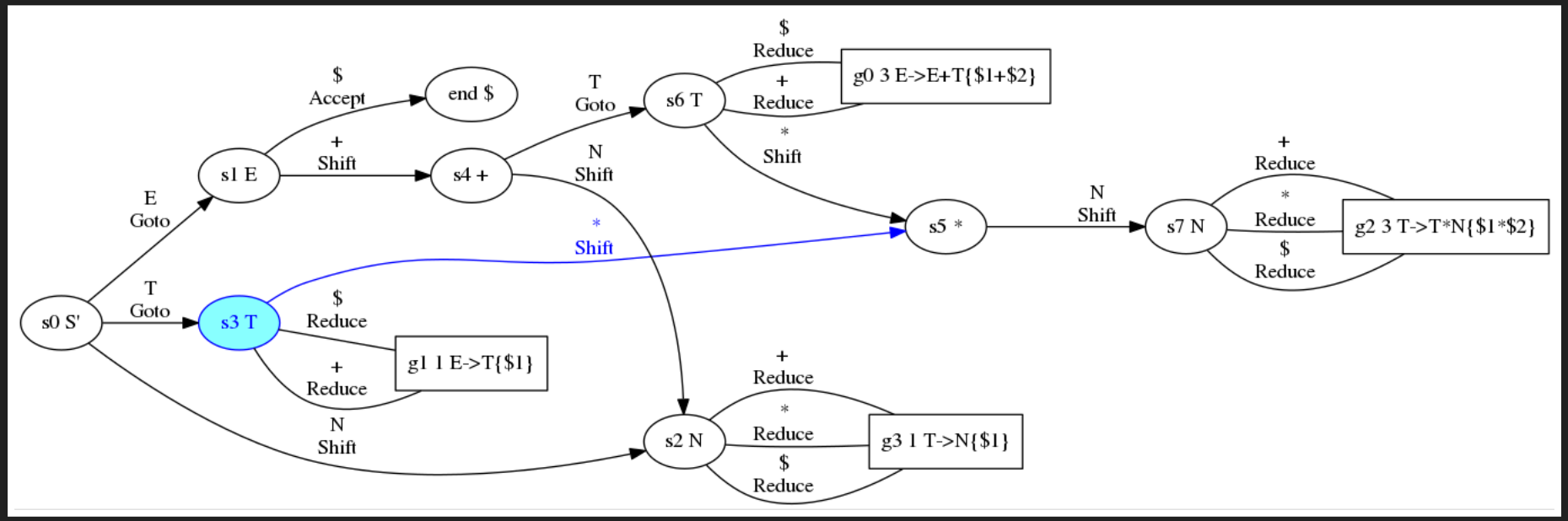
SHIFT S5

```
inputs * 3 + 4 $
status 3 0
results 2
```



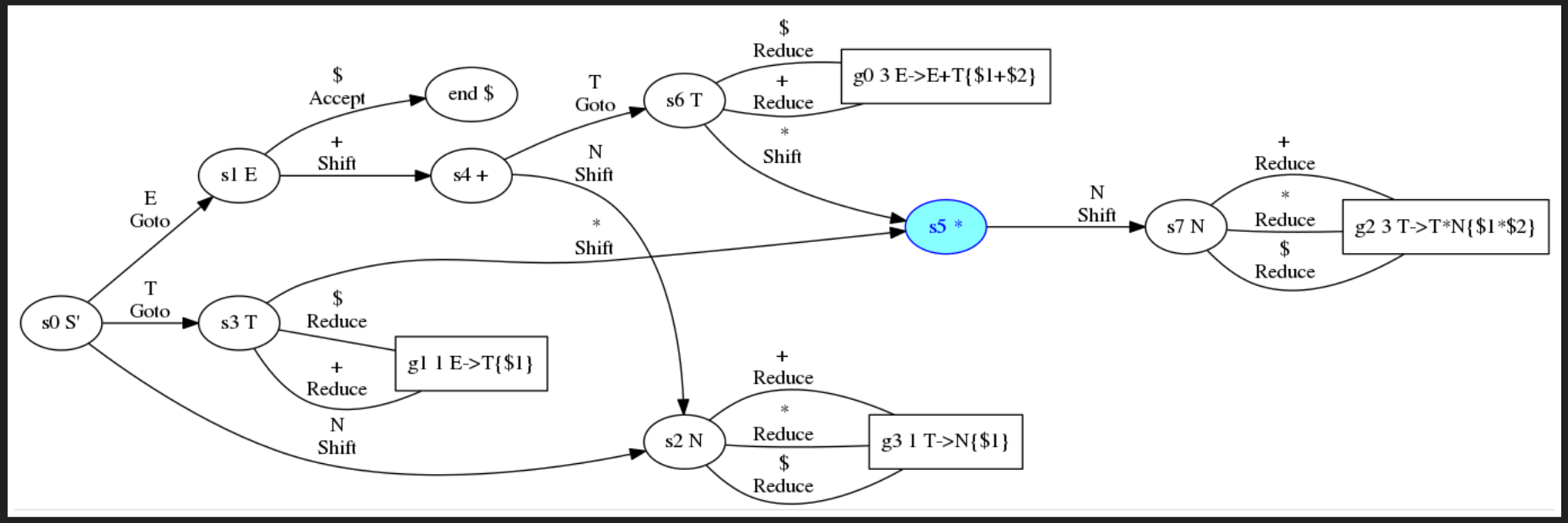
SHIFT S5

```
inputs * 3 + 4 $ ステータスに5をpush 入力*を結果に移動
status 3 0
results 2
```



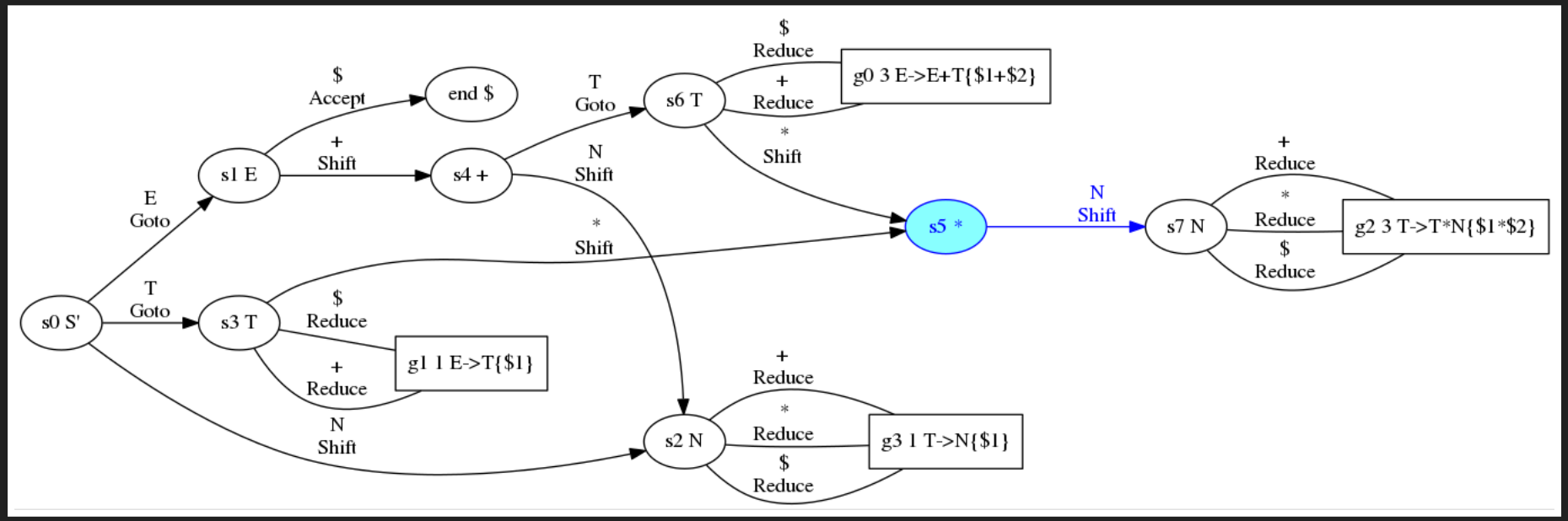
SHIFT S7

```
inputs 3 + 4 $
status 5 3 0
results * 2
```



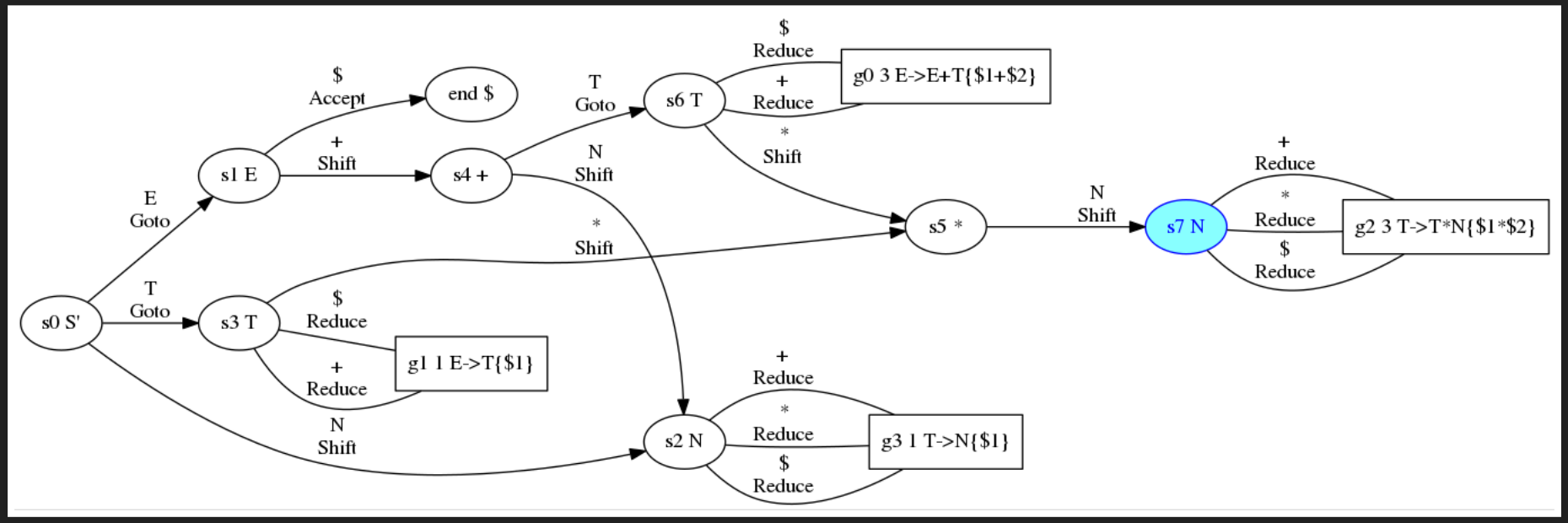
SHIFT S7

```
inputs 3 + 4 $ ステータスに7をpush 入力3を結果に移動
status 5 3 0
results * 2
```



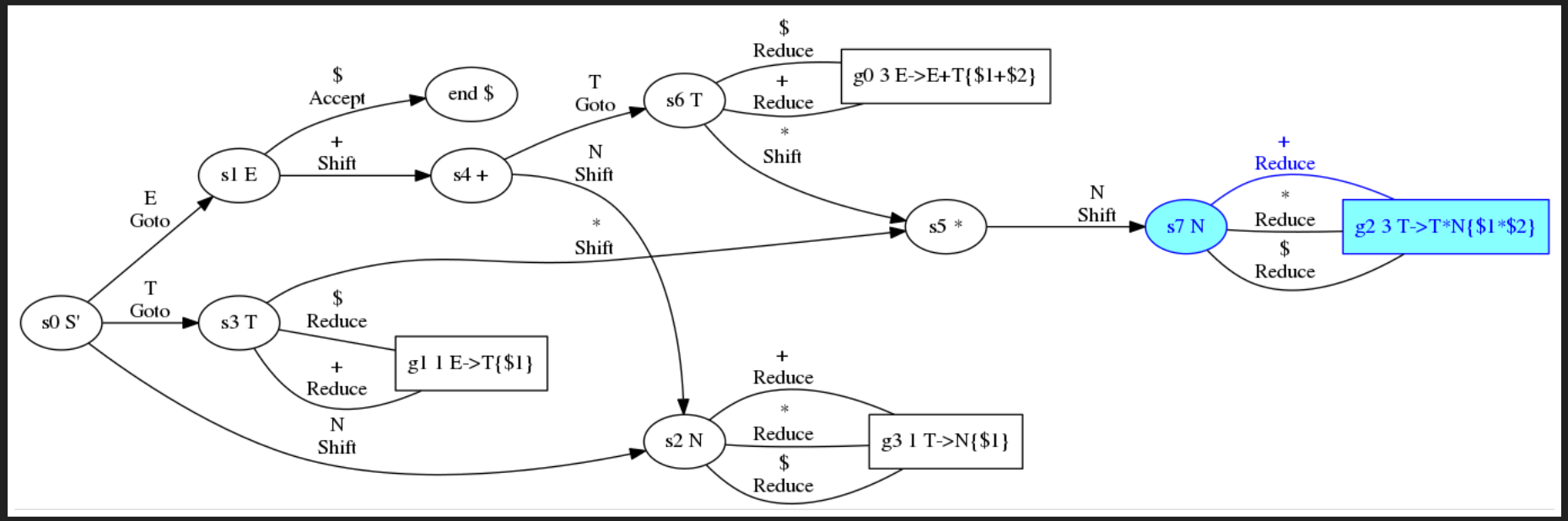
REDUCE G2

```
inputs + 4 $
status 7 5 3 0
results 3 * 2
```



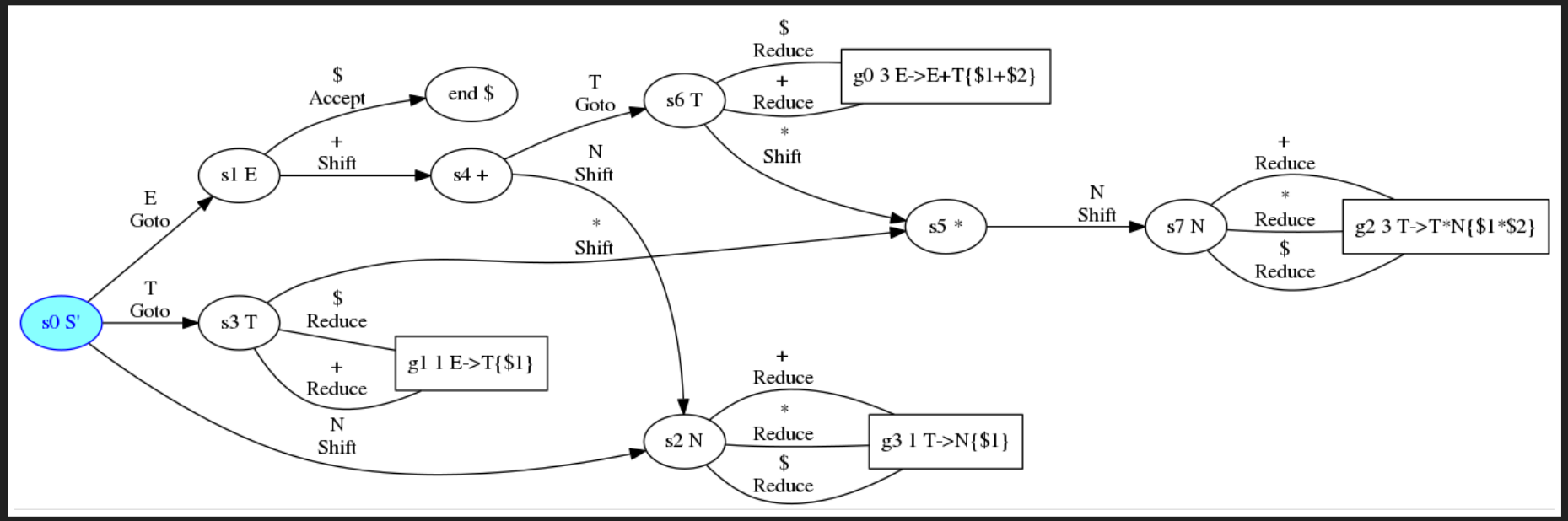
REDUCE G2

```
inputs + 4 $ 文法g2を見て、3個Pop、{$1*$2}を結果にpush、文法名Tを入力にpush
status 7 5 3 0
results 3 * 2
```



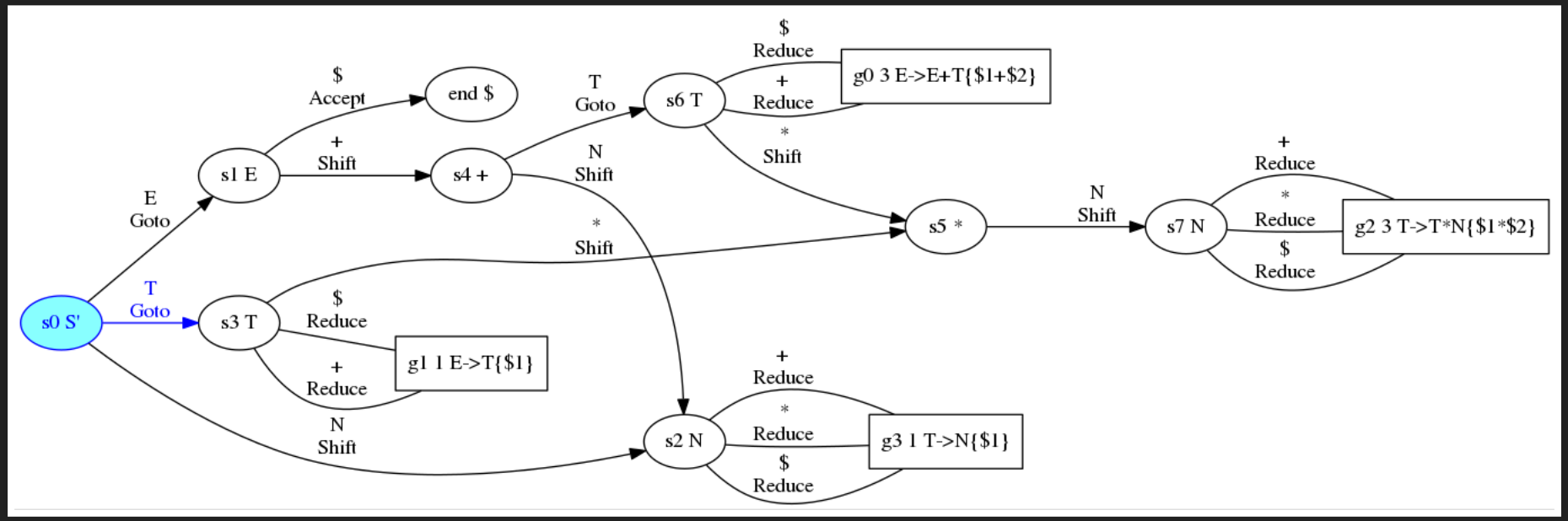
GOTO S3

```
inputs T + 4 $
status 0
results 6
```



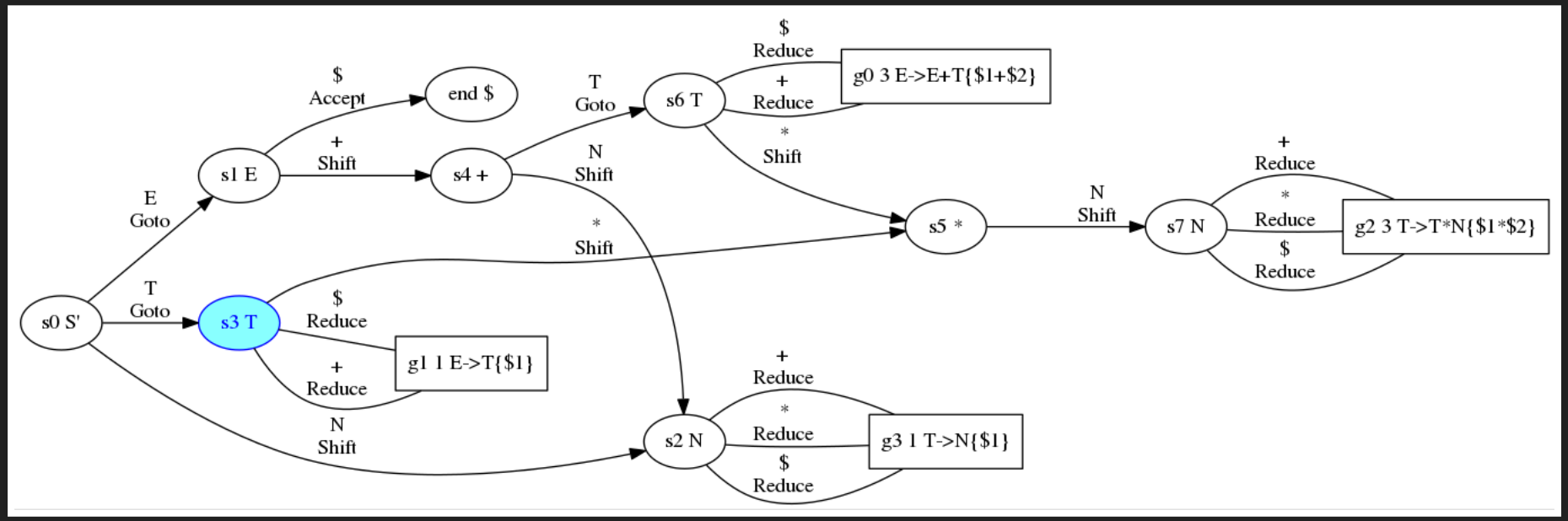
GOTO S3

```
inputs T + 4 $ ステータスに3をpush,入力Tを捨てる
status 0
results 6
```



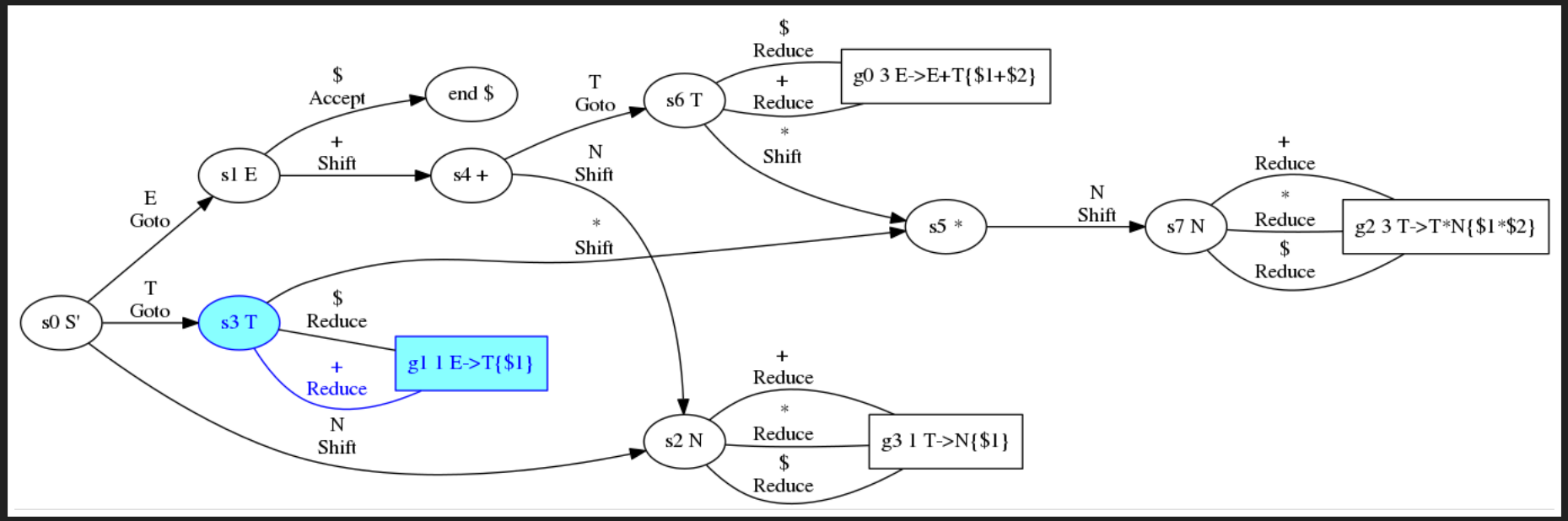
REDUCE G1

```
inputs + 4 $
status 3 0
results 6
```



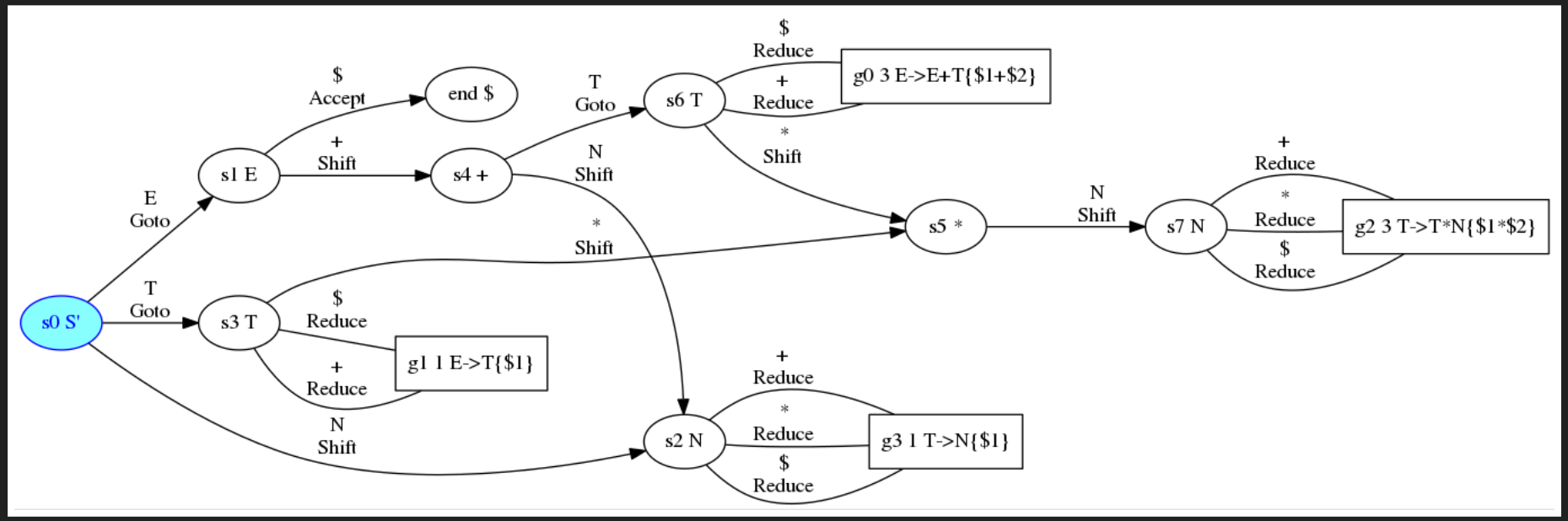
REDUCE G1

```
inputs + 4 $ 文法g1を見て、1個Pop、{$1}を結果にpush、文法名Eを入力にpush
status 3 0
results 6
```



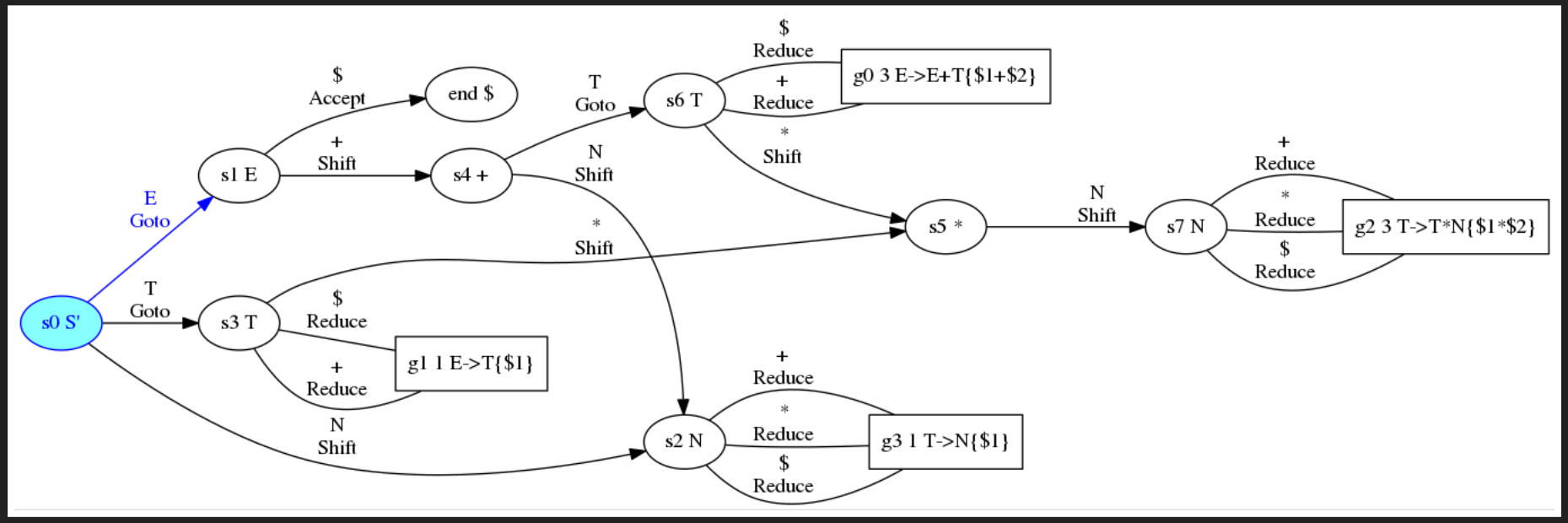
GOTO S1

```
inputs E + 4 $
status 0
results 6
```



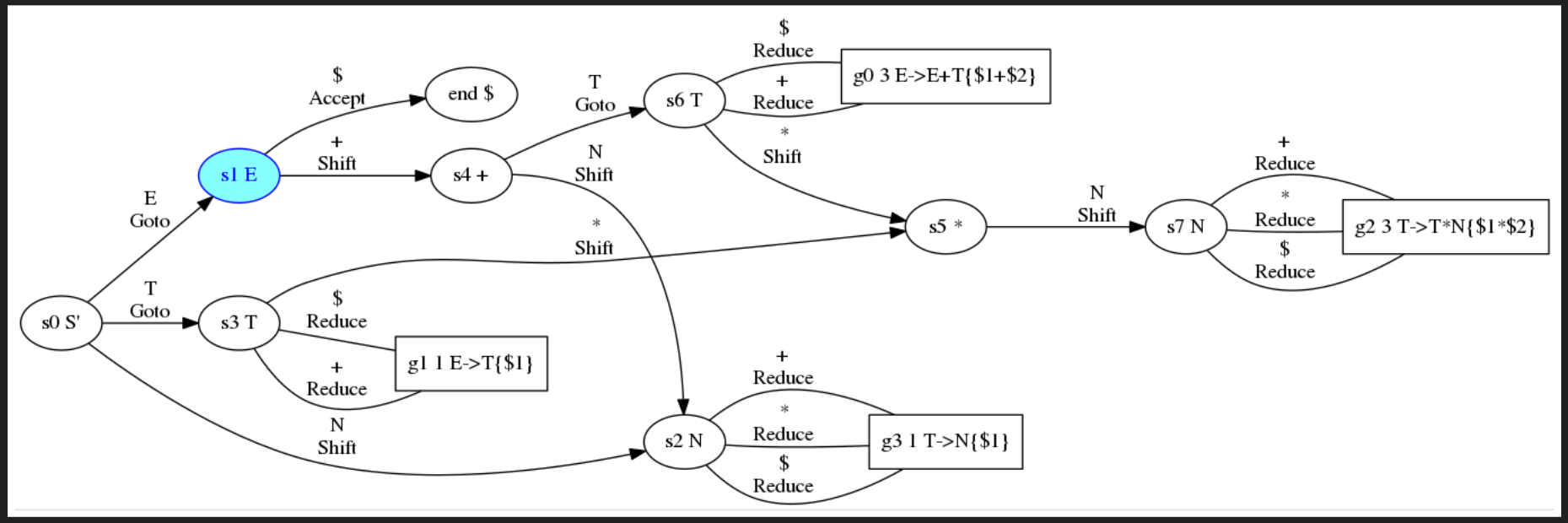
GOTO S1

```
inputs E + 4 $ ステータスに1をpush,入力Eを捨てる
status 0
results 6
```



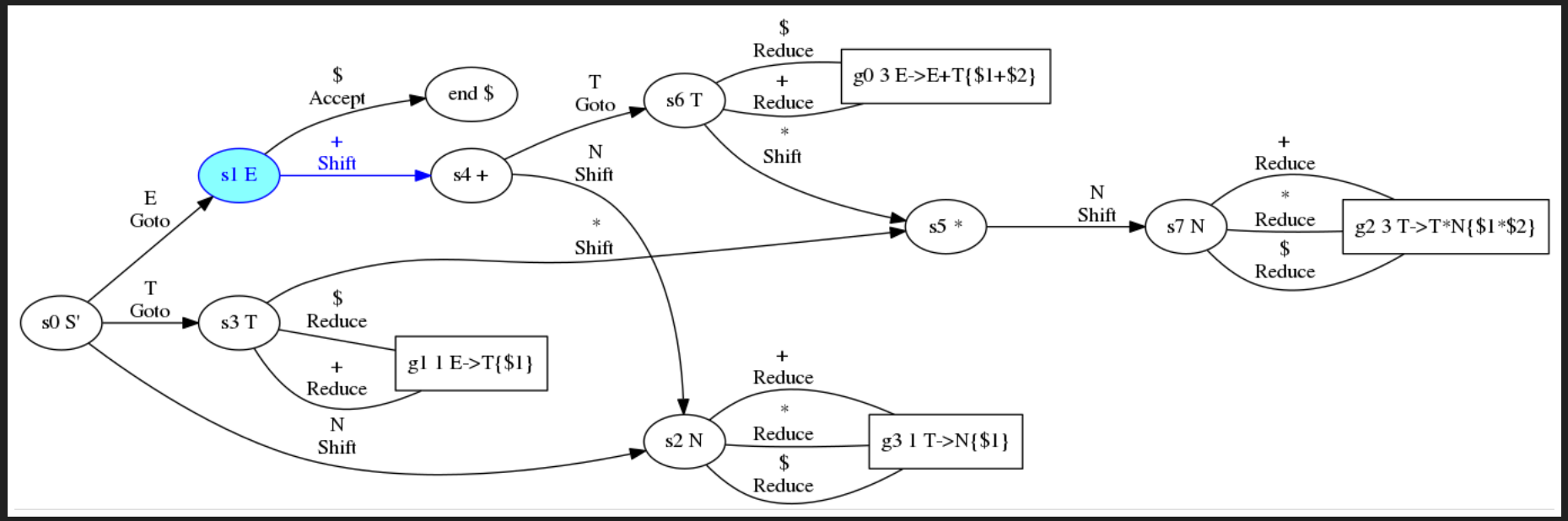
SHIFT S4

```
inputs + 4 $
status 1 0
results 6
```



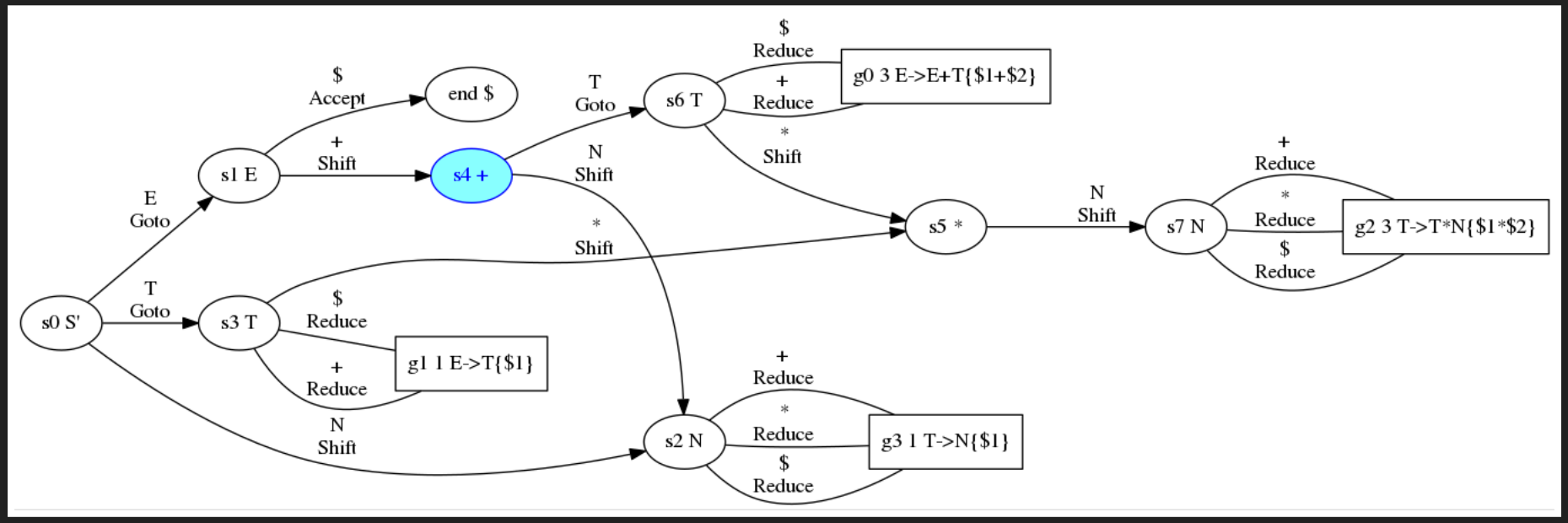
SHIFT S4

```
inputs + 4 $ ステータスに4をpush 入力+を結果に移動
status 1 0
results 6
```



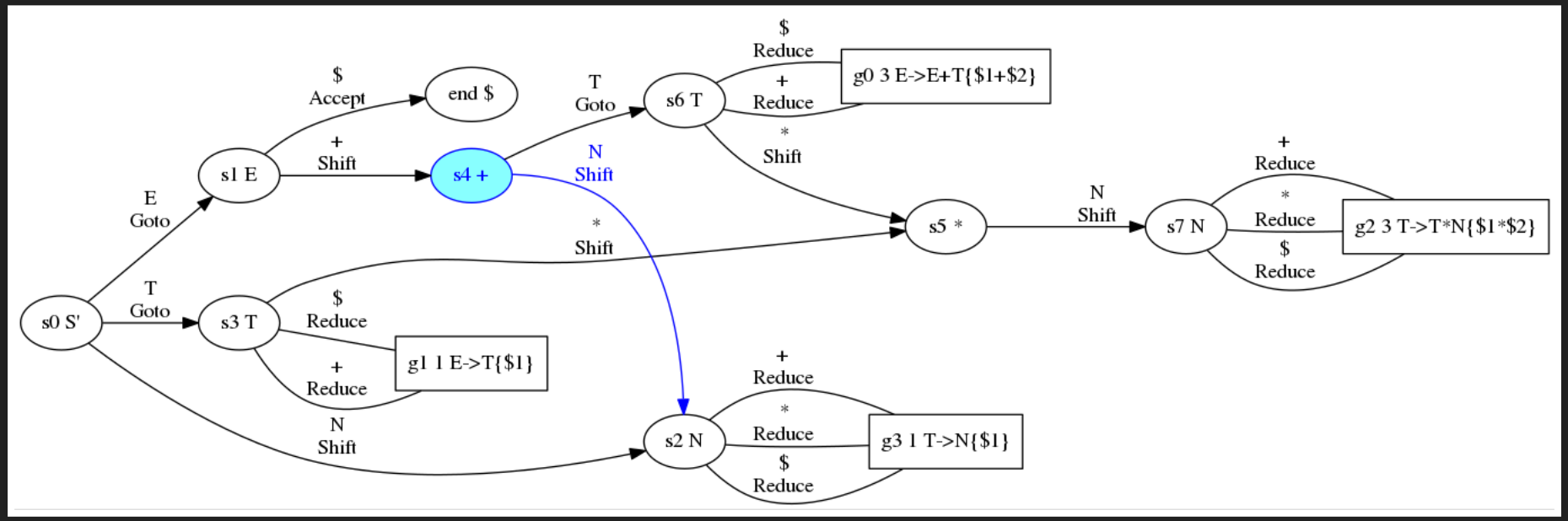
SHIFT S2

```
inputs 4 $
status 4 1 0
results + 6
```



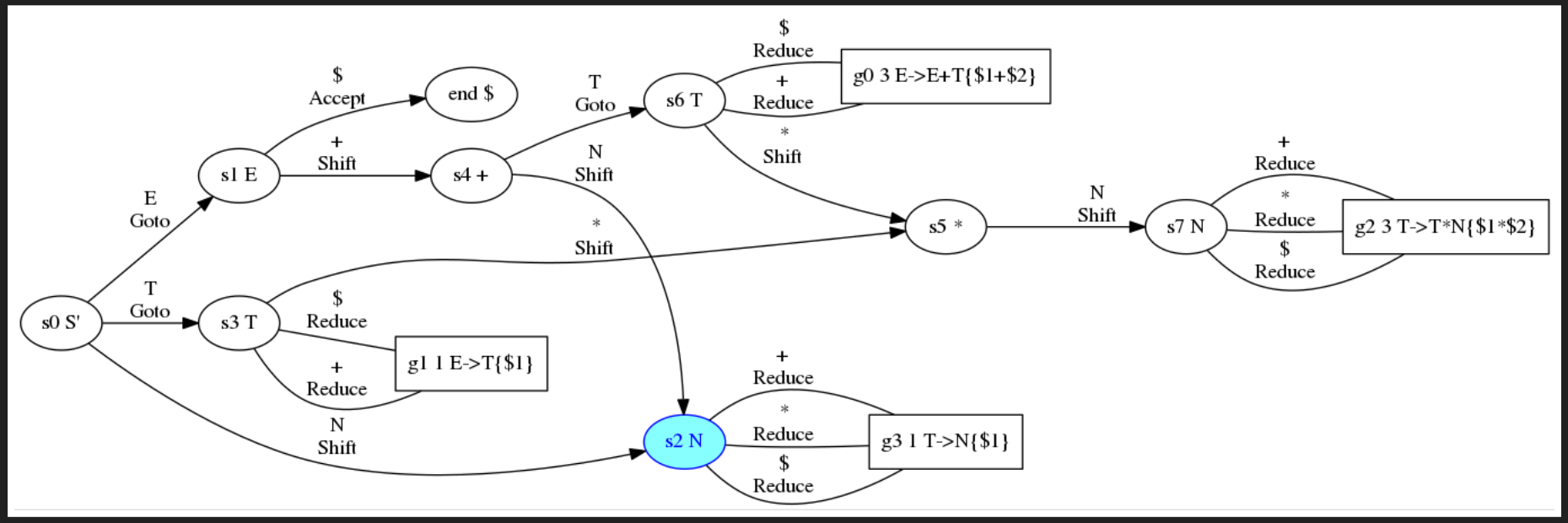
SHIFT S2

```
inputs 4 $ ステータスに2をpush 入力4を結果に移動
status 4 1 0
results + 6
```



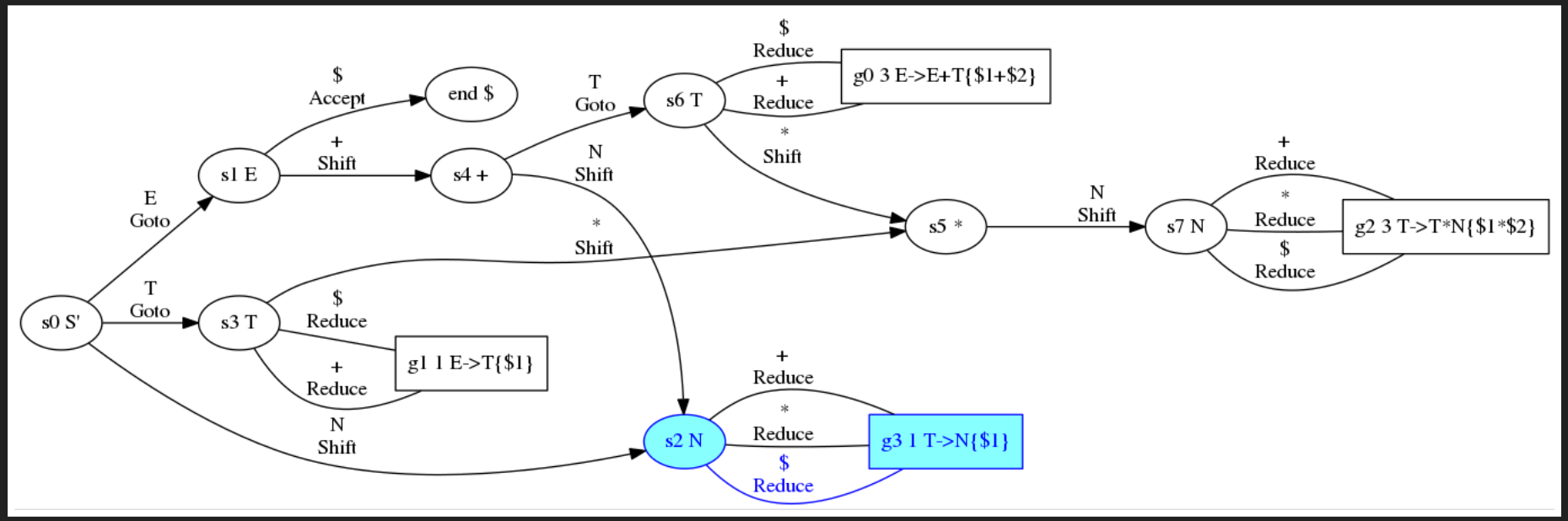
REDUCE G3

```
inputs $
status 2 4 1 0
results 4 + 6
```



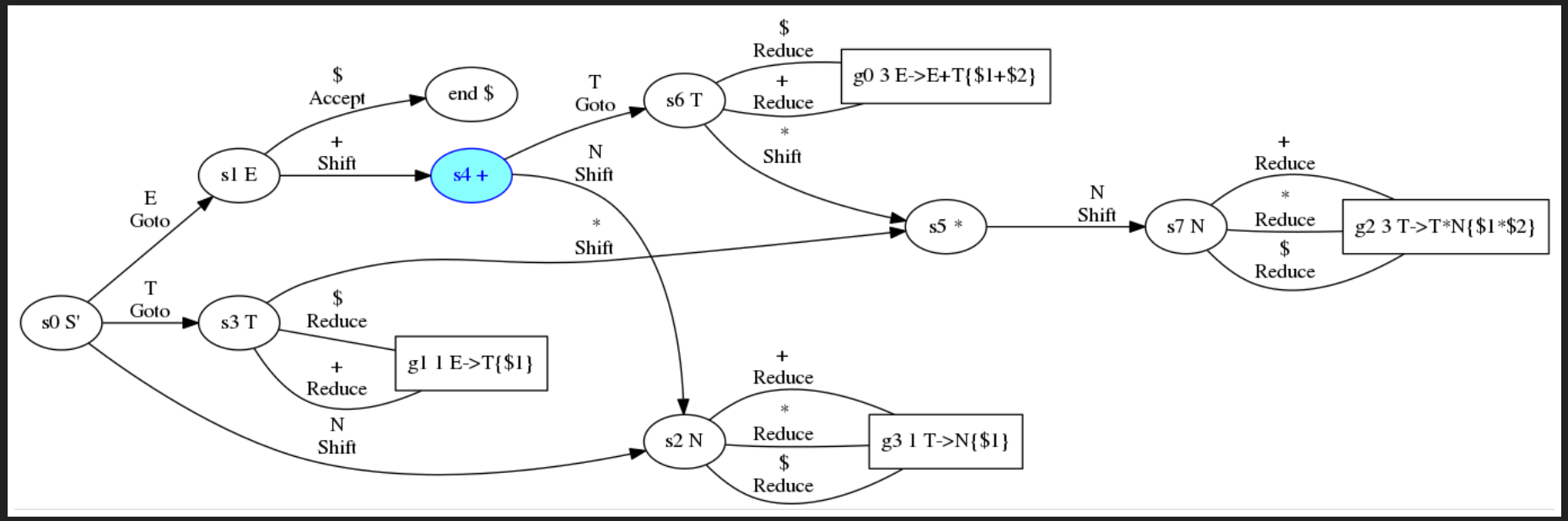
REDUCE G3

```
inputs $ 文法g3を見て、1個Pop、{$1}を結果にpush、文法名Tを入力にpush
status 2 4 1 0
results 4 + 6
```



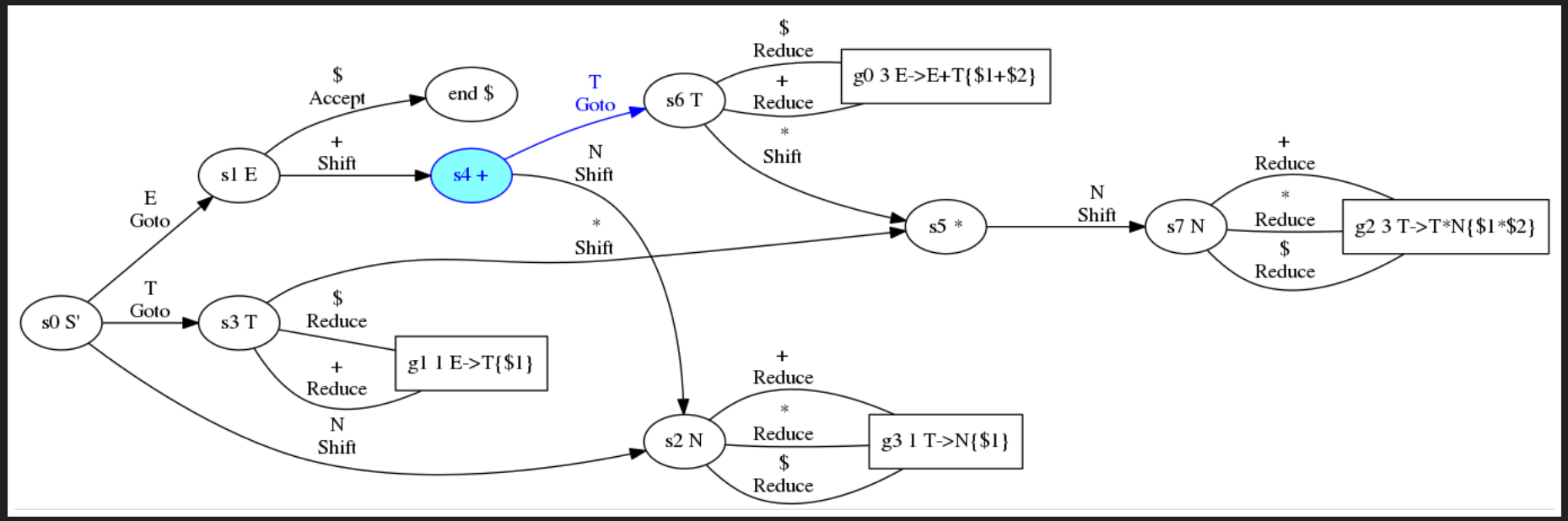
GOTO S6

```
inputs T $
status 4 1 0
results 4 + 6
```



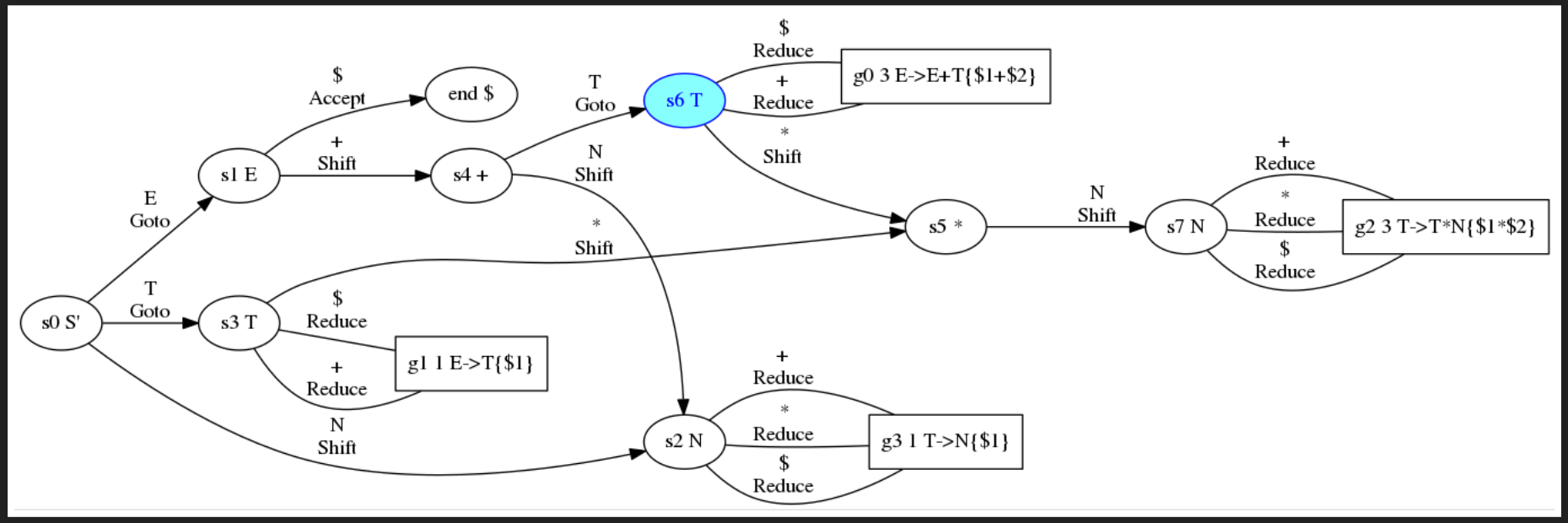
GOTO S6

```
inputs T $ ステータスに6をpush,入力Tを捨てる
status 4 1 0
results 4 + 6
```



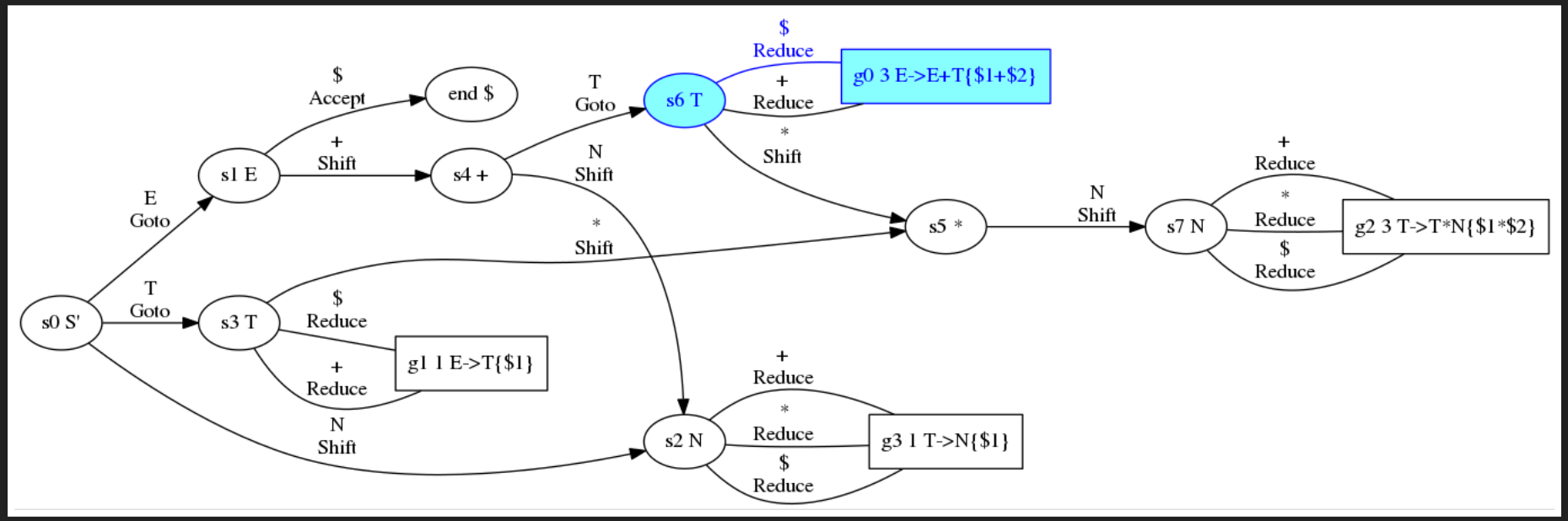
REDUCE G0

```
inputs $
status 6 4 1 0
results 4 + 6
```



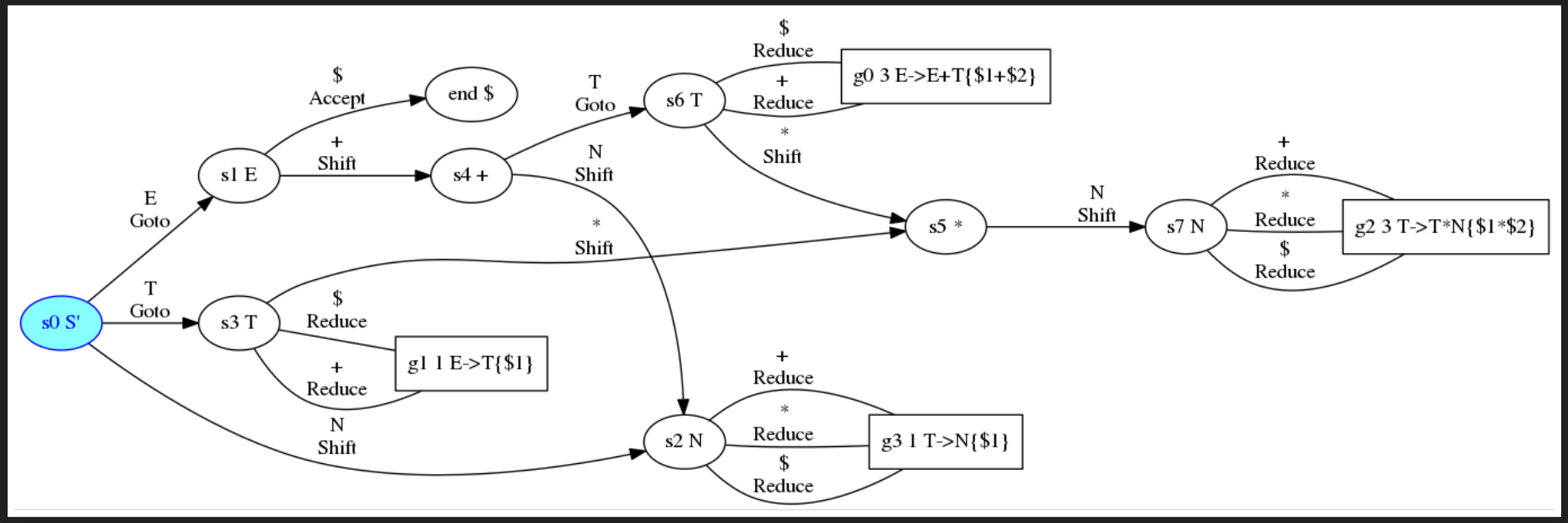
REDUCE G0

```
inputs $ 文法g0を見て、3個Pop、{$1+$2}を結果にpush、文法名Eを入力にpush
status 6 4 1 0
results 4 + 6
```



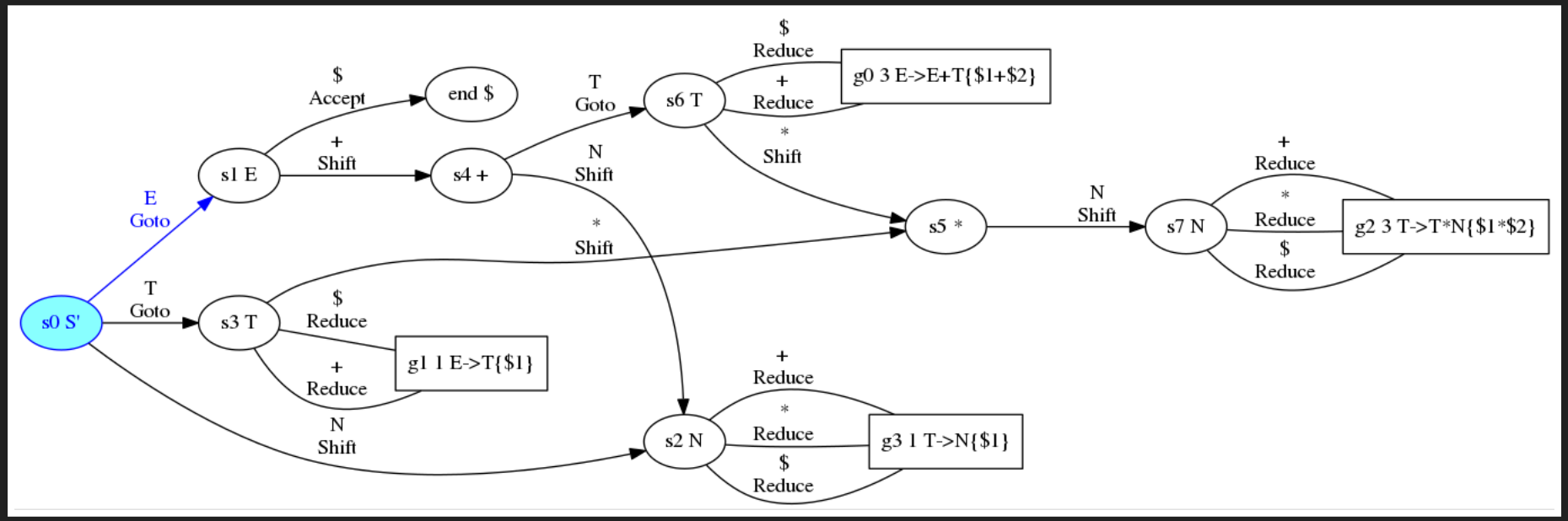
GOTO S1

```
inputs E $
status 0
results 10
```



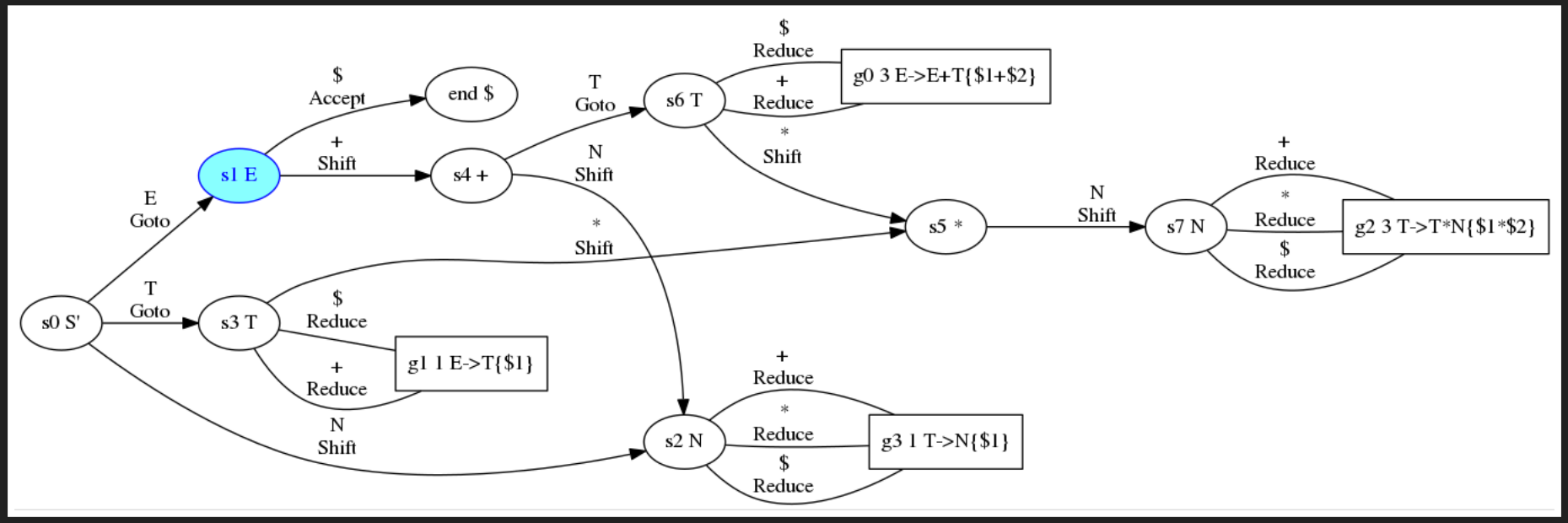
GOTO S1

```
inputs E $ ステータスに1をpush,入力Eを捨てる
status 0
results 10
```



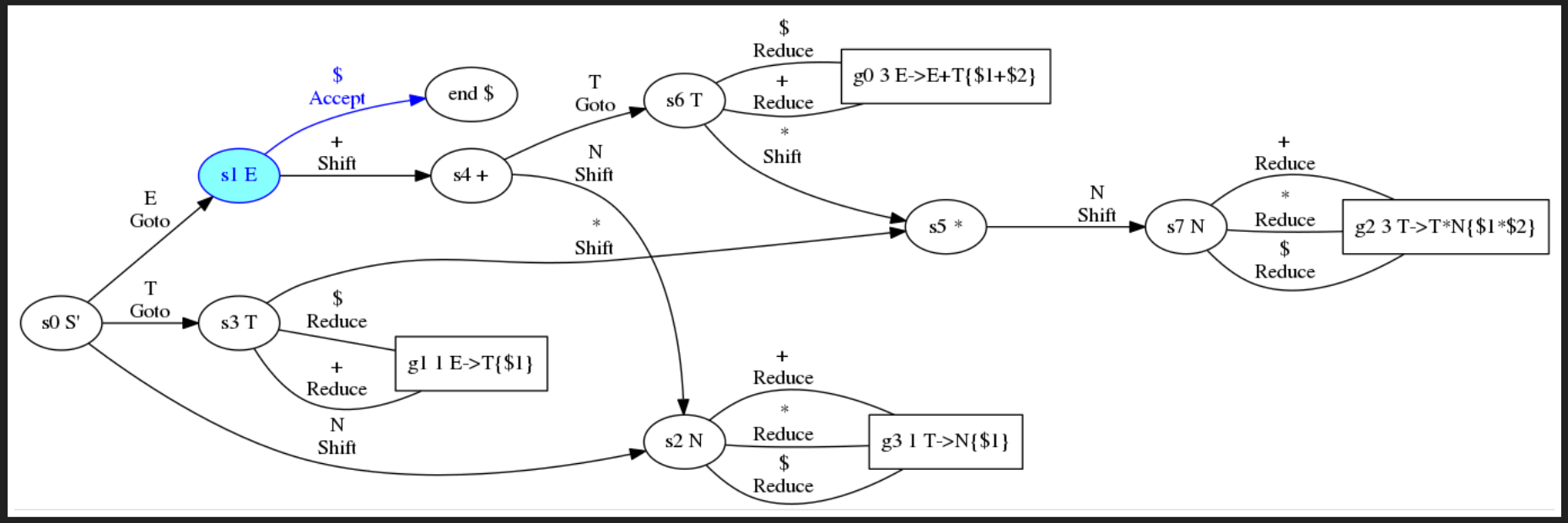
ACCEPT

```
inputs $
status 1 0
results 10
```



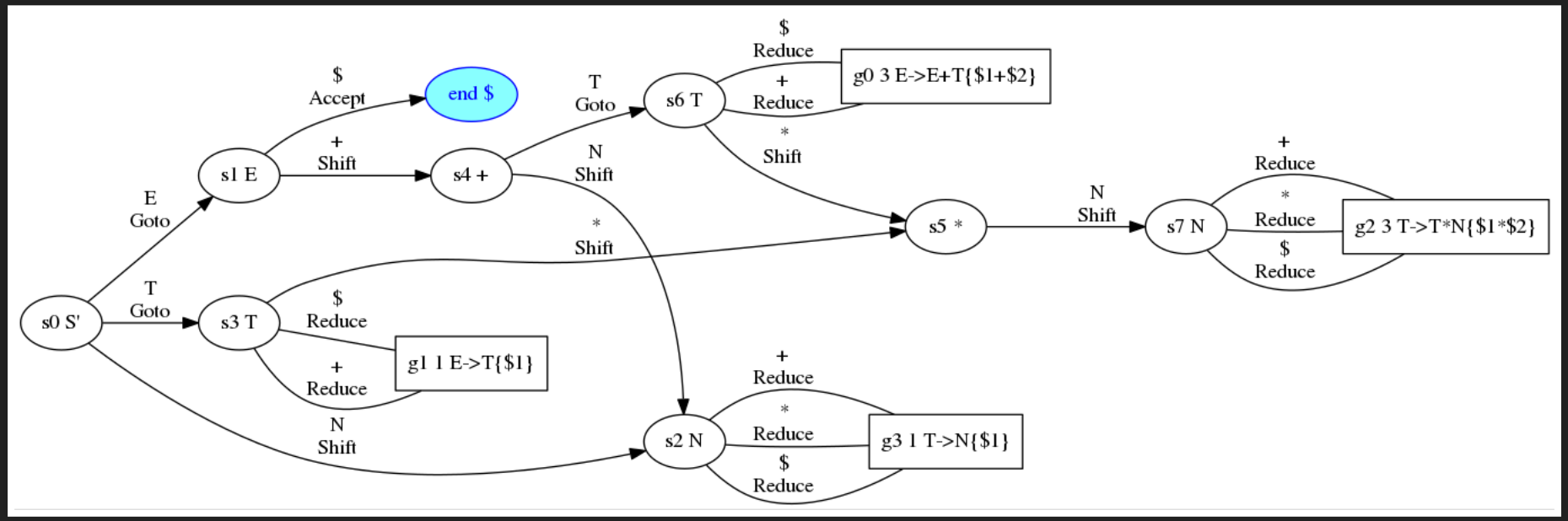
ACCEPT

```
inputs $ アクセプト
status 1 0
results 10
```



ACCEPT

```
inputs $ 結果 10 です
status 1 0
results 10
```



RESULT 2*3+4 = 10

- いくつかの例でパーサの動きを見てみました。
- よく分かってない方でもLR構文解析の動きが分かりましたでしょうか？

4. LR構文解析とは

- 構文解析を複数の状態に分け、そこでの入力によって次の状態に変化させていって構文解析するのがLR構文解析です。
- 状態を戻す場合には呼び出し経路を記憶しておく必要があるのでスタックが必要になります。
- 呼び出し経路を保存しておくスタックを状態スタックといい、結果を保存するスタックを結果スタックといいます。
- 通常LR構文解析器はYaccといったようなコンパイラコンパイラにより自動生成されるので動きを理解せずに使っている方も多いでしょう。
- しかしながら、LR構文解析の動作を理解すればよりパーサを制作するのに役立つかも知れません。

ご清聴ありがとうございました