B_024

鉄道路線モデルに基づくプログラムの動作の可視化
Visualization of Programs Based on Railroad Route Model

西尾 嘉矩 † NISHIO Yoshinori
六沢 一昭 ‡ ROKUSAWA Kazuaki

1. はじめに

本稿では，プログラムの動作を鉄道路線モデルに基づいて可視化するシステムについて述べる．

鉄道路線には，ループ状の路線や乗換駅がある．これらを見ると，プログラムにおける繰返し及び分岐に類似していることに気がつく．従って，プログラムを鉄道路線に置き換えることができれば，プログラムの動作を電車の動きによって可視化できる．

2. 鉄道路線モデル

2.1 鉄道路線モデルとは

鉄道路線モデルとは，プログラムを鉄道路線に対応させたモデルである．

プログラムと鉄道路線の対応関係を以下に示す．

- プログラムの制御構造 (繰返し, 分岐, call/return 命令) を鉄道の路線図に対応させる．
- プログラムの実行の流れを，線路上を走る電車の動きに対応させる．

2.2 実際の鉄道路線における電車及び乗客の移動

実際の鉄道路線には，ループ状の路線や乗換駅がある．ループ状の路線では，電車はその路線を回り続ける．乗換駅では，乗客は次に乗る路線の電車へ乗り換える．

乗換駅には，改札を出て別の駅の路線へ乗り換える乗換駅と，改札を出ずに同じ駅内で別の路線へ乗り換える乗換駅がある．

2.3 プログラムの制御構造と鉄道路線の対応

プログラムの制御構造と鉄道路線の対応を表1に示す．

表 1: プログラムの制御構造と鉄道路線の対応

制御構造	鉄道路線
繰返し	ループ状の路線
分岐	乗換駅
call/return 命令	乗換駅

- 繰返しは処理を繰り返す．この様子は，ループ状の路線を電車が回り続ける状況に類似している．ゆえに繰返しはループ状の路線に対応する．
- 分岐は条件に応じて次の処理を選択する．この様子は，乗換駅で，改札を出ることなく行き先に応じて別の路線へ乗り換える状況に類似している．ゆえに分岐は乗換駅に対応する．
- call/return 命令は，サブルーチンを呼び出す/呼び出し側に戻る処理を行う．これらの様子はいずれも，乗換駅で，改札を出て別の駅の路線へ乗り換える状況に類似している．ゆえに call/return 命令は乗換駅に対応する．

†千葉工業大学 大学院 情報科学研究科 情報工学専攻
‡千葉工業大学 情報科学部 情報工学科

2.4 複数プロセスと電車の対応

ここまで1つの電車の動きに着目してきたが，実際の鉄道では複数の電車が走っている．この複数の電車はプログラムの何に対応するだろうか．

1つのプロセスは1つの電車に対応するであろう．従って，複数のプロセスは複数の電車に対応すると考えられる．この対応関係を利用すると，複数のプロセスの実行は複数の電車の動きに，新たなプロセスの生成は新たな電車の生成に対応する．

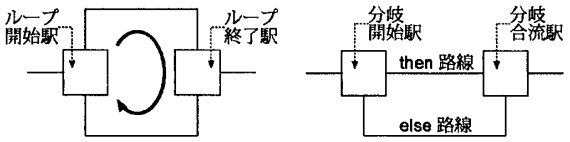
3. 可視化

本節では，鉄道路線モデルを用いてプログラムの動作をどのように可視化するかを述べる．

3.1 繰返し，分岐の可視化

繰返しはループ状の路線で表現する (図 1(a))．繰返しの実行中，電車はループ状の路線を矢印の向きに走る．

分岐は乗換駅で表現する (図 1(b))．本システムでは，この駅を分岐開始駅と呼ぶ．乗客が分岐開始駅で then 路線，又は else 路線に乗り換え，乗り換え先の路線を電車が走る．



(a) 繰返し　(b) 分岐

図 1: 繰返し，分岐に対応する路線図

3.2 call/return の可視化

call/return 命令はいずれも乗換駅で表現する (図 2)．本システムでは，この駅をそれぞれ call 駅，return 駅と呼ぶ．また，改札を出て向かう先の駅をサブルーチン開始駅と呼ぶ．

call, return の可視化を以下に示す．

call の可視化

1. 電車が call 駅に到着する．
2. サブルーチン開始駅から電車が出発する．

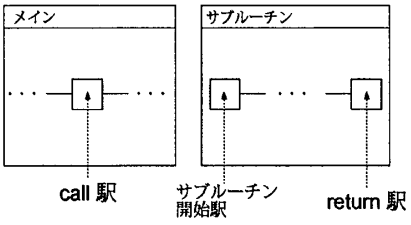


図 2: call/return 命令に対応する路線図

return の可視化

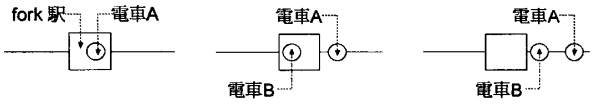
1. 電車が return 駅に到着する.
2. call 駅に停車していた電車が出発する.

3.3 fork 命令の実行 (プロセスの生成) の可視化

fork 命令は fork 駅で表現する. fork 命令の実行は, 駅で新たな電車を生成することで可視化する.

fork の可視化を以下に示す (図 3).

1. 電車 A が fork 駅に到着する.
2. fork 駅で電車 B が生まれ, 電車 A は駅を出発する.
3. 電車 B が fork 駅を出発する.



(a) 場面 1　(b) 場面 2　(c) 場面 3

図 3: fork の可視化

4. 処理の流れ

本システムにおける可視化処理全体の流れを図 4 に示す. 本システムは, まず実行ログを作成し, それを基に可視化を行う.

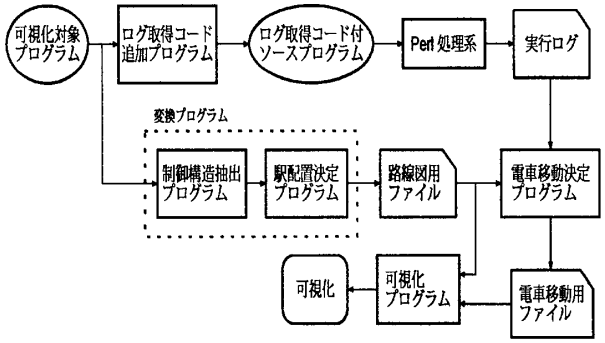


図 4: 処理の流れ

実行ログ
　可視化対象プログラム§ の実行ログを記述したファイル

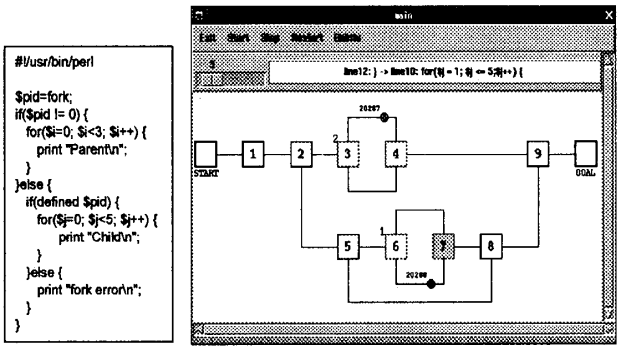
路線図用ファイル
　駅の配置位置を記述したファイル

電車移動用ファイル
　電車が走る路線を記述したファイル

§現在 Perl プログラムが可視化対象である.

5. 実行例

図 5(a) のプログラムの実行を可視化した際の動画の一場面を図 5(b) に示す. 図 5(b) の 2 つの黒丸は電車を示し, 黒丸の左上の数字はプロセス ID を表す (上方の電車のプロセス ID は 20287, 下方は 20288 である).



(a) プログラム　(b) 動画の一場面

図 5: 可視化例

図 5(a) のプログラム

3 行目 (`$pid=fork;`) で新たなプロセスを生成する. そのため, このプログラムの 3 行目以降は 2 つのプロセスが実行する.

20287 電車の動き

1. 「START 駅」を出発し,「駅 1 (fork 駅)」に到着する.
2. 「駅 1」で **20288 電車**が生まれる.
3. 「駅 1」を出発し,「駅 2 (分岐開始駅)」に到着する.
4. then 路線を走り,「駅 3 (ループ開始駅)」に到着する.
5. ループ路線を 3 周し,「駅 4 (ループ終了駅)」に到着する.
6. 「駅 4」を出発し,「駅 9 (分岐合流駅)」を通って「GOAL 駅」に到着する.

20288 電車の動き

1. 「駅 1」で生まれる.
2. 「駅 1」を出発し,「駅 2」に到着する.
3. else 路線を走り,「駅 5 (分岐開始駅)」に到着する.
4. then 路線を走り,「駅 6 (ループ開始駅)」に到着する.
5. ループ路線を 5 周し,「駅 7 (ループ終了駅)」に到着する.
6. 「駅 7」を出発し,「駅 8 (分岐合流駅)」,「駅 9」を通って「GOAL 駅」に到着する.

6. まとめ

本稿では, 鉄道路線モデルを説明し, このモデルに基づいてプログラムの動作を可視化するシステムについて述べた.

本システムを使用すると, ユーザは, プログラムの制御構造 (繰返し, 分岐, call/return 命令) を鉄道の路線図で, プログラムの実行の流れを, 線路上を走る電車の動きで見ることができる. また, 複数のプロセスの実行を複数の電車の動きで, 新たなプロセスの生成を新たな電車の生成で見ることができる.