
頻出部分木マイニングを用いた ビルドエラーを誘発するDockerfileの 命令間の依存関係の自動修正

.....

オープンソースソフトウェア工学研究室
坂本廉也

背景

I ソフトウェアエコシステムの高速化

- ー 高度情報化社会での社会の急速な変化に追従
- ー 高速なソフトウェアデリバリーを実現する様々なツールが登場

I Docker^[1]

- ー コンテナ型仮想化ソフトウェアのデファクトスタンダード^[2]
 - 独立した仮想アプリケーション環境（コンテナ）を動作
 - スケーラビリティ/デプロイが容易になる
- ー IT企業の約79%がDockerを利用^[3]

[1] <https://www.docker.com/>

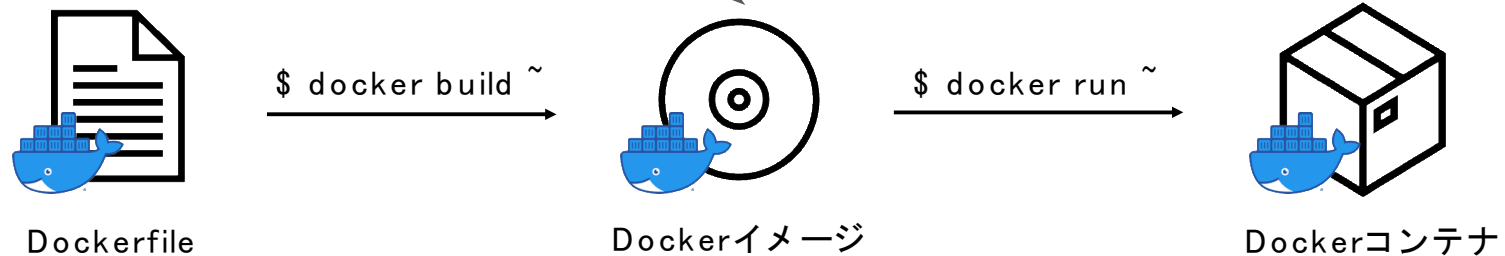
[2] A. Zerouali, V. Cosentino, G. Robles, J. Gonzalez-barahona, and T. Mens, “Conpan : A tool to analyze packages in software containers,” in MSR '19, 2019, pp. 592–596.

[3] <https://portworx.com/blog/2017-container-adoption-survey/>

コンテナ作成のながれ

Dockerイメージ（以下、イメージ）

- ー 開発者が求めるアプリケーションと実行環境を含むイメージファイル
- ー DockerHub^[4]などのコンテナレジストリなどで共有可能



Dockerfile

- ー イメージに含める内容を命令として記述したテキストファイル
- ー Dockerfileをビルドすることでイメージを作成

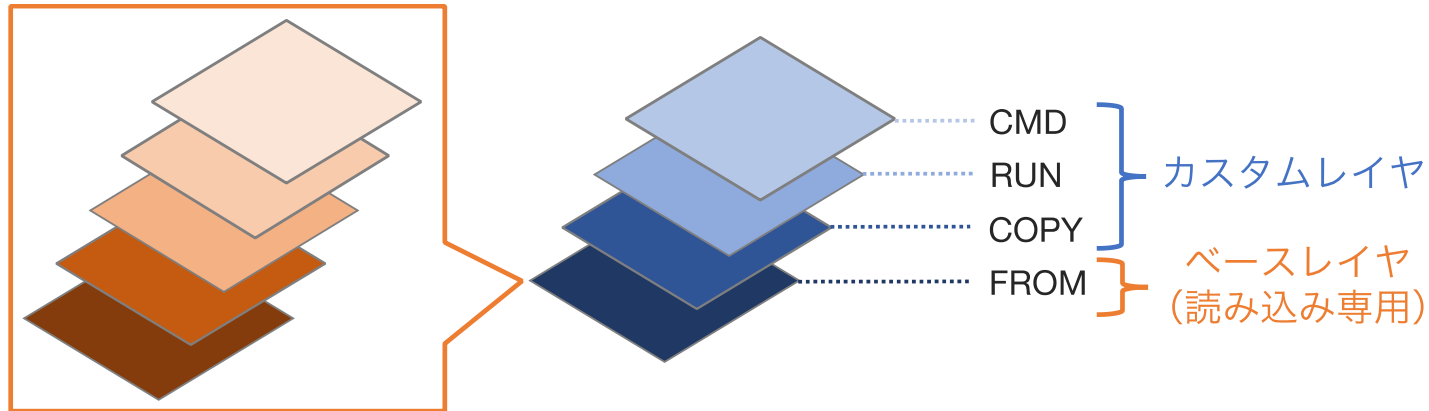
[4] <https://hub.docker.com/>

Dockerfileのビルド

- | | |
|----------------------------------|-----------------|
| 1 FROM ubuntu:22.04 | #ベースイメージの継承 |
| 2 COPY ["/src/", "."] | #ファイルをイメージへコピー |
| 3 RUN apt-get -y update && \ | #シェルコマンドの実行 |
| 4 apt-get install python=3.10 | |
| 5 CMD ["python", "hoge hoge.py"] | #コンテナ起動時にコマンド実行 |

\$ docker build ~

命令を上から逐次実行



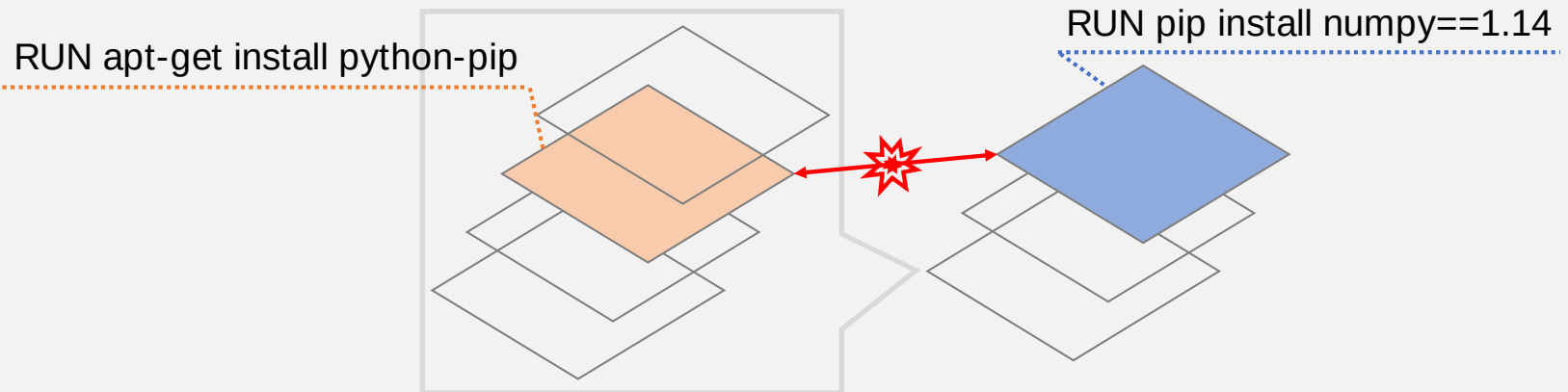
様々な原因でビルドエラーが発生する（ビルドに失敗する）

ビルドエラーとその原因

- 命令間の依存関係, 外部コンテキスト, 構文ミス など
 - 先に実行された命令との”開発者が関知しない依存関係”が発生

- 開発者自身が変更可能な範囲が分かりづらい

- ベースイメージへの変更ができず, アドホックな記述になり易い[5]



- 自身が記述した命令でも予期しない依存関係が発生

- カスタムレイヤ内で非互換が発生

[5] Chris Tomy, Tingmao Wang, Earl T. Barr, and Sergey Mechtaev. 2022. Modus: a Datalog dialect for building container images. In Proceedings of the European Software Engineering Conference and Symposium on the Foundations of Software Engineering, ESEC/FSE 2022, pp.595–606

研究目的

I Dockerfileのビルドエラー

- ビルドエラー発生率が高く^[6]修正に多くの時間が費やされる^[7]
 - ビルド時に発生する複雑な依存関係^{[5][8]}

I 研究動機

- Dockerfileのビルドエラー自動修正手法の提案によって現代のスピーディな開発へ貢献可能

目的

ビルドエラーを誘発するDockerfileの命令間の
依存関係の自動修正手法の提案

[6] Cito, J., Schermann, G., Wittern, J. E., Leitner, P., Zumberi, S. and Gall, H. C.: An Empirical Analysis of the Docker Container Ecosystem on GitHub, In Proceedings of MSR '17, pp.323–333 (2017).

[7] Wu, Y., Zhang, Y., Wang, T. and Wang, H.: An Empirical Study of Build Failures in the Docker Context, In Proceedings of MSR '20, pp. 76–80 (2020).

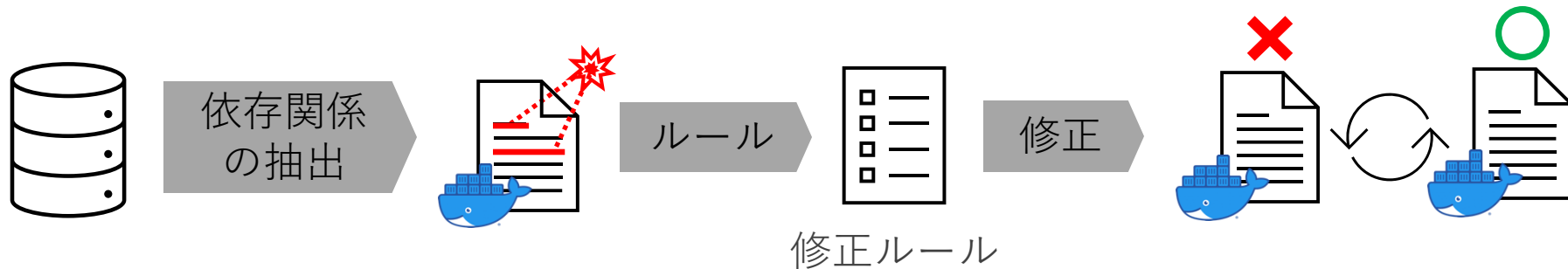
[8] 坂本 廉也, 東 裕之輔, 大平 雅雄: ビルドエラーを引き起こすDockerfileの依存関係の分析, ソフトウェア・シンポジウム2023, pp.61-70 (2023).

アプローチ

1. ビルドエラーを誘発する依存関係の抽出

- Dockerfileは開発に必要な設定に関する様々な記述を含む
 - 対応するエラーの種類も膨大なため、記述されやすい命令の組合せからビルドエラーを誘発する依存関係を特定

2. 抽出した依存関係から修正ルールを作成し、自動修正



データセット

I [9]で提供されているデータセット

- スター数10以上, 単一のDockerfileを持つGitHubリポジトリ
 - 破損している/現在は存在していない/マルチステージビルド を除外

Dockerfile件数	ビルドエラー発生件数	ビルドエラー発生割合
16,856件	4,554件	27.0%

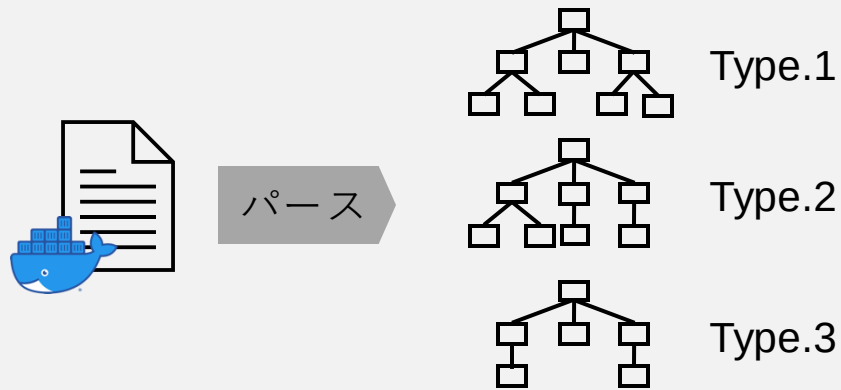
ビルドエラーの原因	Dockerfile数	割合
依存関係	3,259	71.6%
その他	1,295	28.4%
合計	4,554	100.0%

※ビルドログの目視による調査

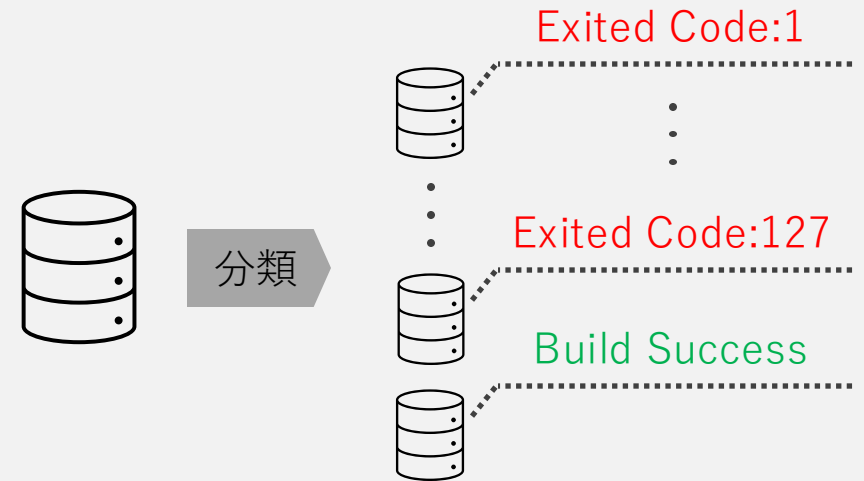
壊れた依存関係を修正することで効率的にビルドエラーを修正可能

依存関係の抽出

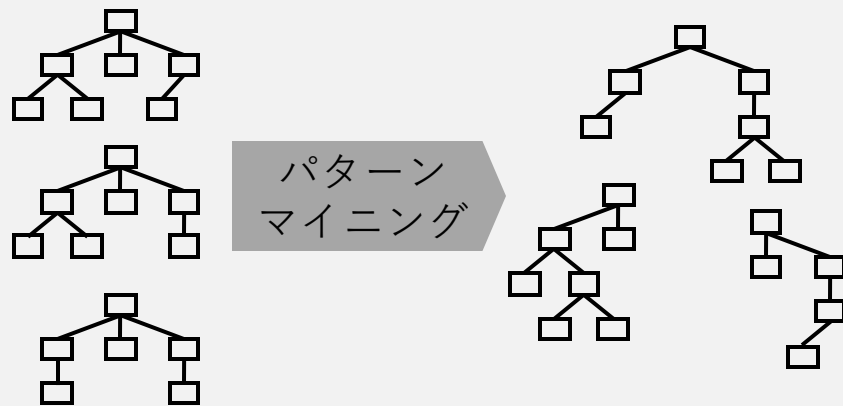
1. 3タイプの抽象度のASTに変換



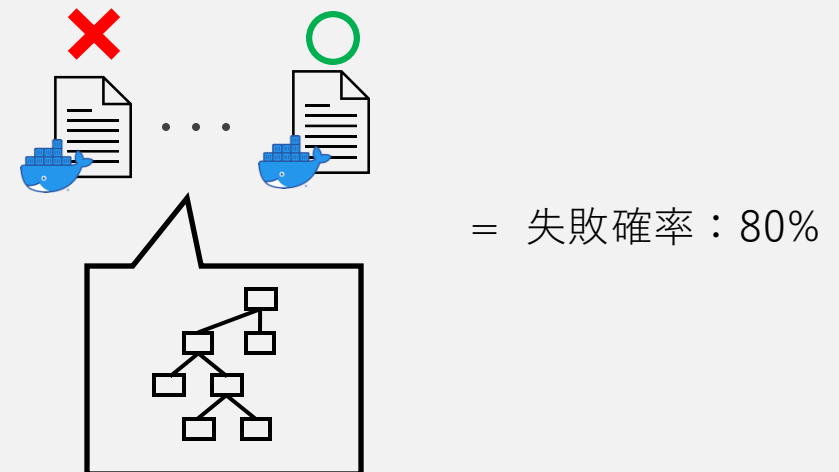
2. 終了コードで分類



3. 頻出部分木マイニングで部分木の抽出



4. 抽出した部分木の失敗確率を計算



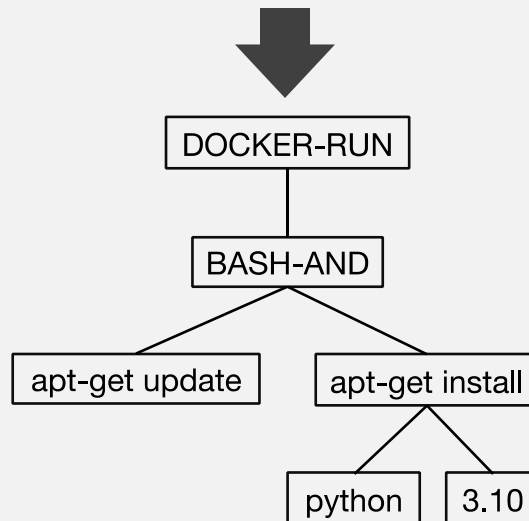
DockerfileをASTへ変換

- I Dockerfileを構造的に解釈可能な形（AST）へパース
 - 命令間の依存関係を保持したまま構文的な表現揺れを回避

命令：ビルド時の操作の種類を指定

```
3 RUN apt-get update && \  
4   apt-get install python=3.10
```

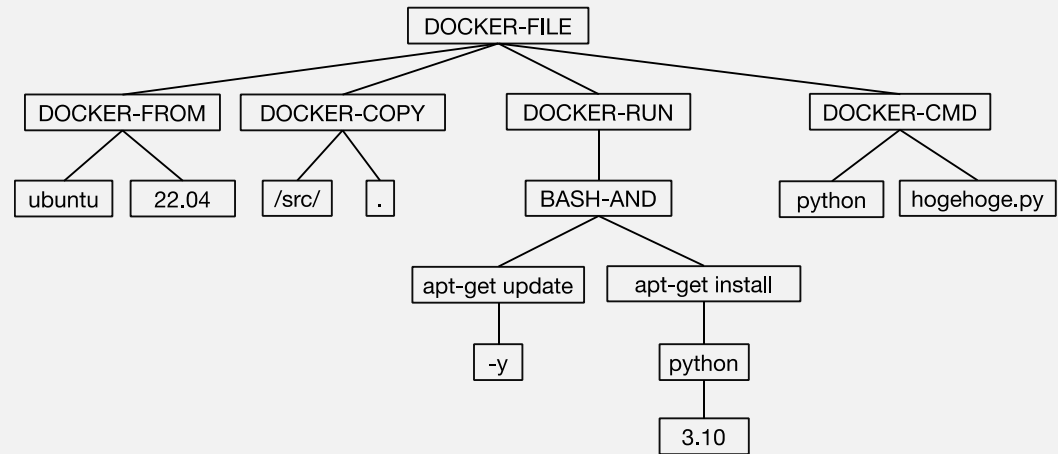
引数：シェルスクリプトや環境変数が埋め込まれる



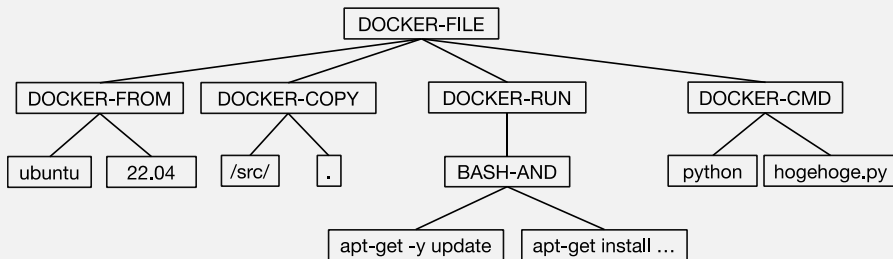
DockerfileをASTへ変換

Type.1：最小粒度でASTを構築

- 命令と引数をトークンに分割
- シェルスクリプトをパースし、ノードを追加



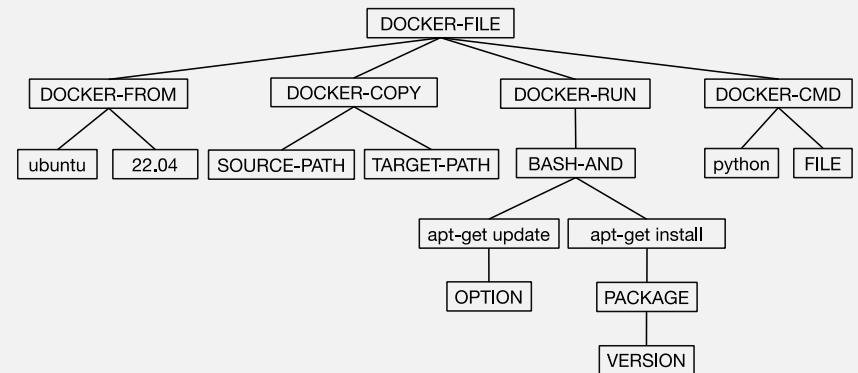
Type.2：Dockerfile構文のパース



- Dockerfileの構文レベルの依存関係を抽出



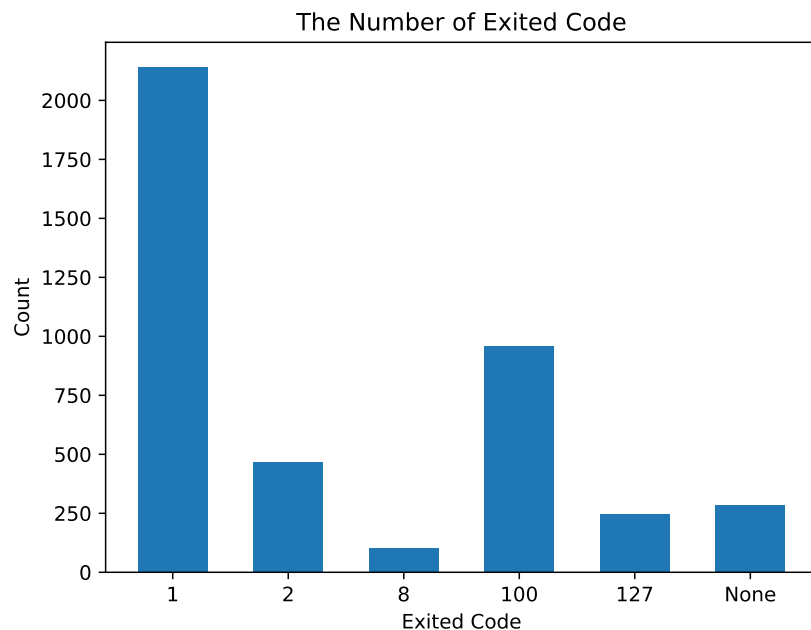
Type.3：ノードの抽象化



- Dockerfileの構造レベルの依存関係を抽出

ビルドエラーの分類

- シェルの終了ステータスを用いてビルドエラーを分類
 - ー 終了ステータスごとにエラーの種類が異なる
 - ー 先に終了ステータスで分類することで抽出されたパターンが引き起こすビルドエラーを解釈しやすくする
 - 100件以上存在する終了ステータスのみを対象

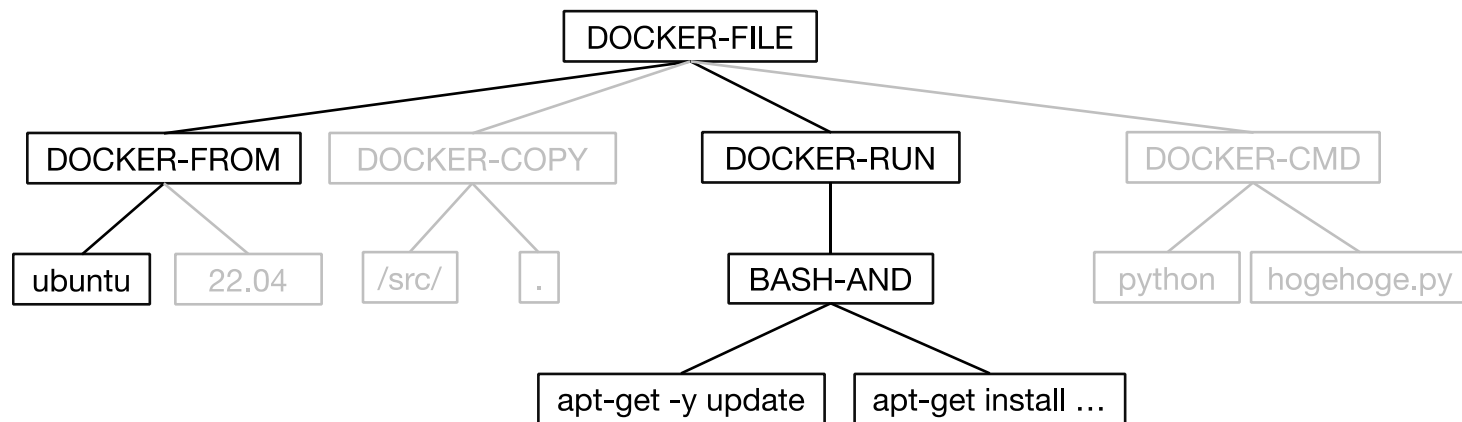


Code 1 : 一般的なエラー
Code 2 : ビルトインコマンドの誤用
Code 8 : 不正なコマンドの使用
Code 100 : ネットワーク異常
Code 127 : 不明なコマンドの使用
None : bash以外の箇所でのエラー

順序付き頻出部分木マイニング

順序木に対する頻出パターンマイニング手法

- 順序木集合から順序を保持した頻出部分木のマイニング手法
- 最小サポート値以上の頻度で出現する部分木を計算



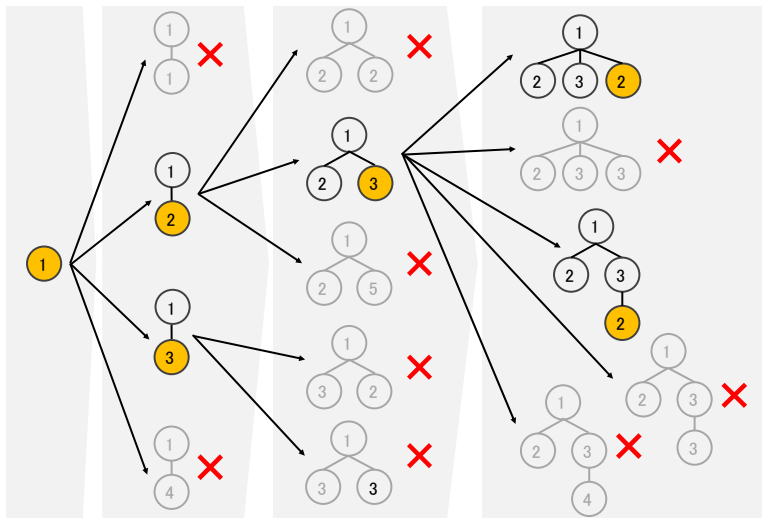
本研究への適用に際しての課題

- 半構造データへのパターンマイニングは計算コストが高い
→ 計算効率の良い“順序付き**飽和**部分木マイニング”の使用

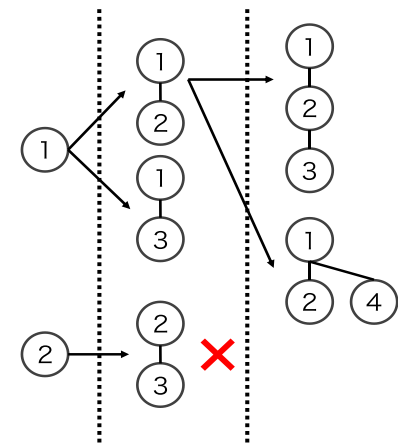
CMTreeMiner^[10]

順序付き飽和部分木マイニングを実現するアルゴリズム

1. サイズ $k-1$ の頻出部分木に対して新たな節点を末弟になるように最右枝に追加し，サイズ k の頻出部分木候補 T として抽出
2. 最小出現回数よりも T の出現回数が多ければ T を保持し，手順1を行う
 - － 飽和集合になり得ない T は探索空間から刈り込み



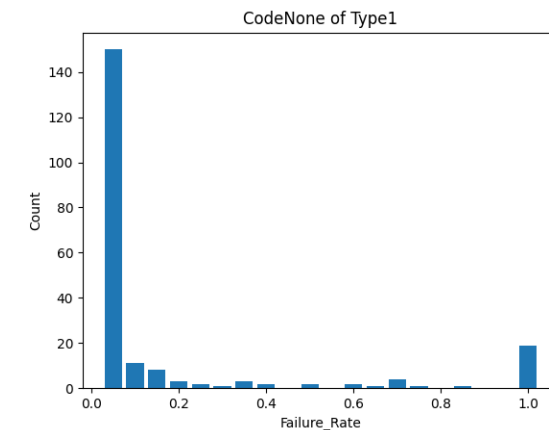
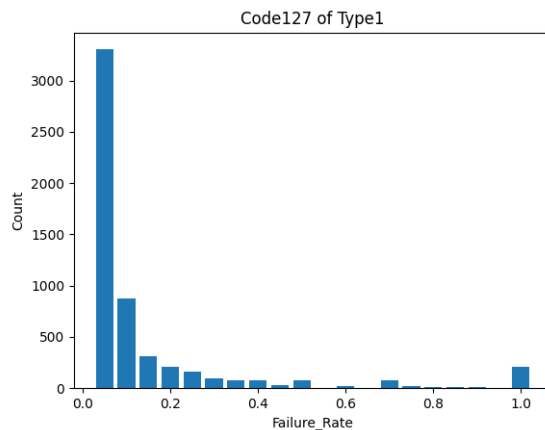
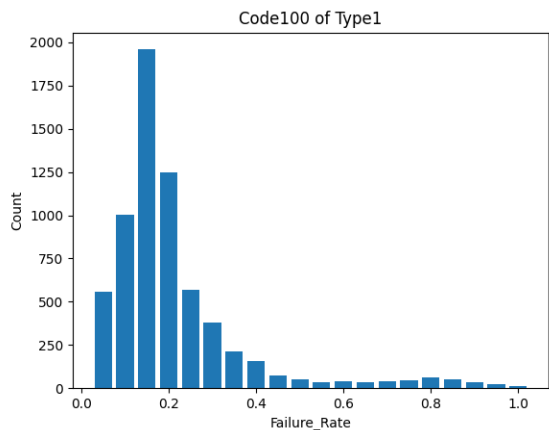
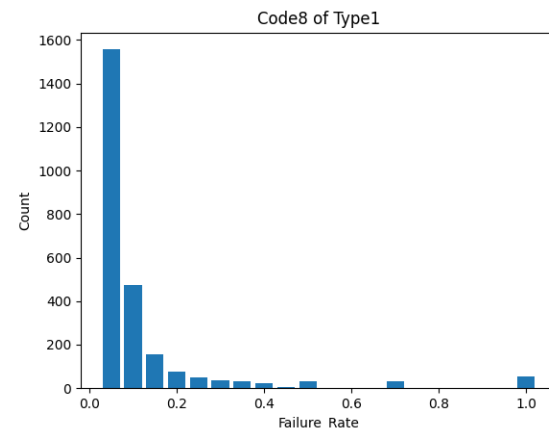
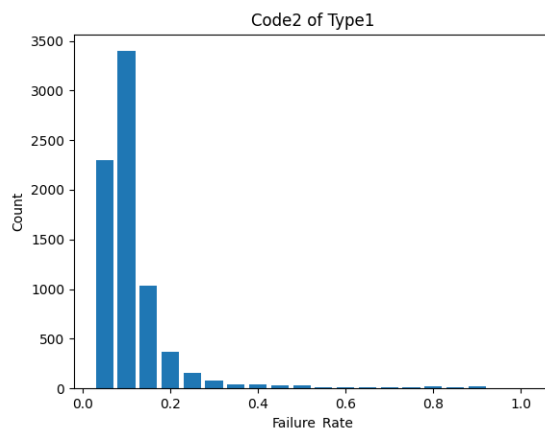
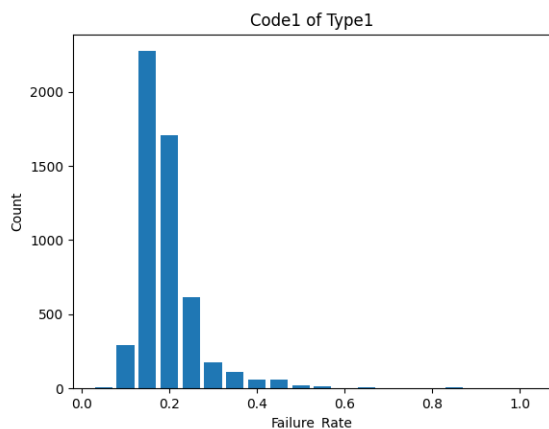
逐次探索空間の刈り込み



[10] Chi, Y., Yang, Y., Xia, Y., Muntz, R.R.: CMtreeminer: Mining both closed and maximal frequent subtrees. In: The Eighth Pacific-Asia Conference on Knowledge Discovery and Data Mining, (2004)

ビルドエラー失敗確率の計算

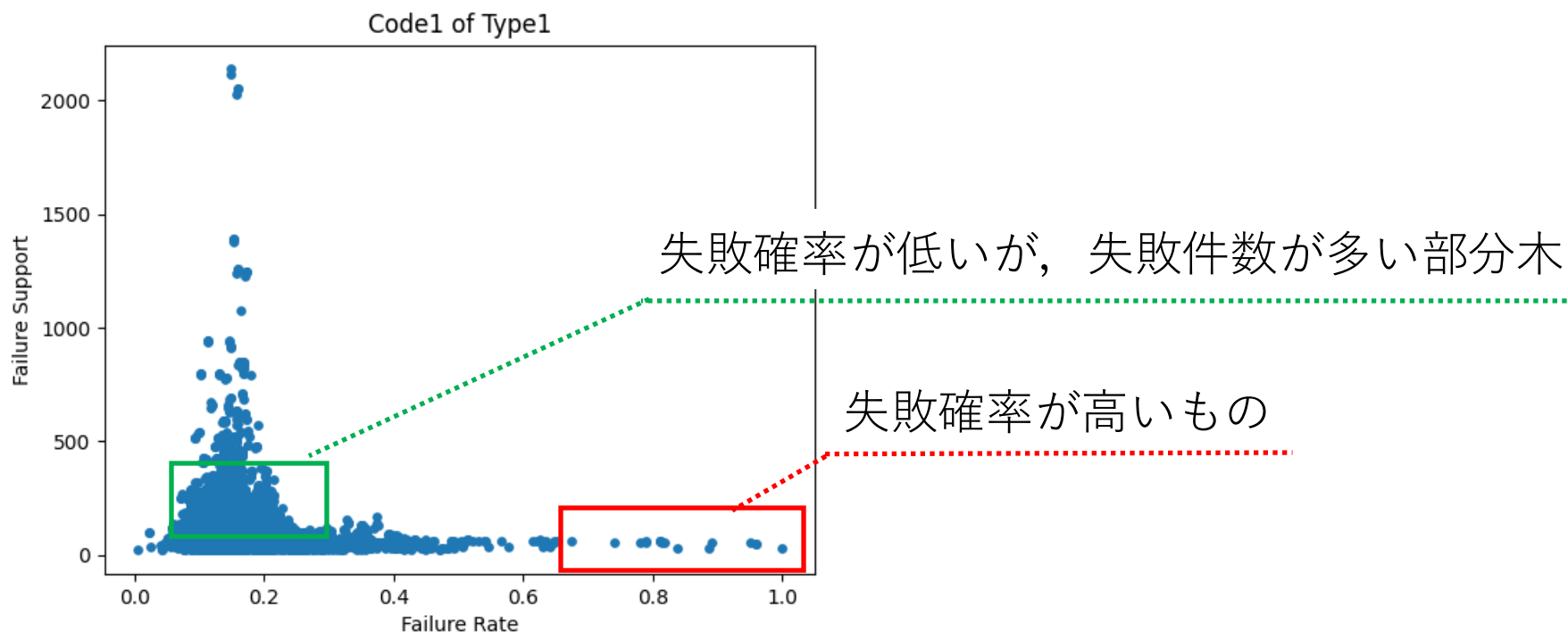
抽出した部分木をもつDockerfileがビルドに失敗する確率



自動修正の対象

進行中

- 失敗確率が高いもの
- 失敗確率が低いが、失敗件数が多い部分木
 - Dockerfileによく記述されしばしばビルドエラーが発生する



失敗確率が低いものの分析

- 目視調査

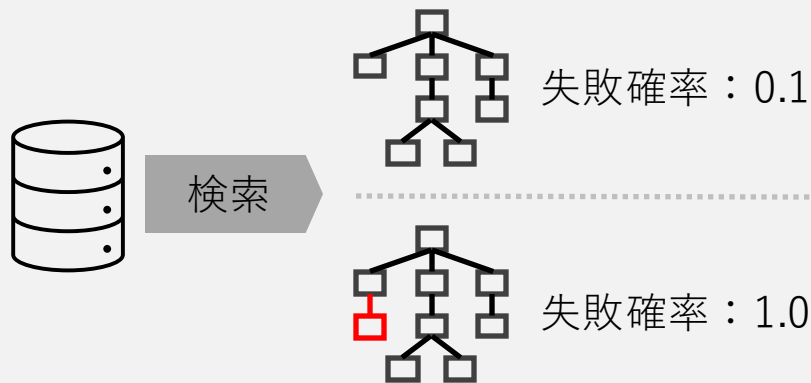
単体の部分木では理解できない部分木の分析

- ある前提条件の元でのみ失敗確率が高くなるものを探索
- アソシエーションルールの考え方
 - Support : 部分木A,Bのエラー件数 / 部分木A,BのDockerfile件数
 - Confidence : 部分木A,Bのエラー件数 / 部分木Aのエラー件数
 - Lift : Confidence / (部分木Bのエラー件数/部分木A,BのDockerfile件数)

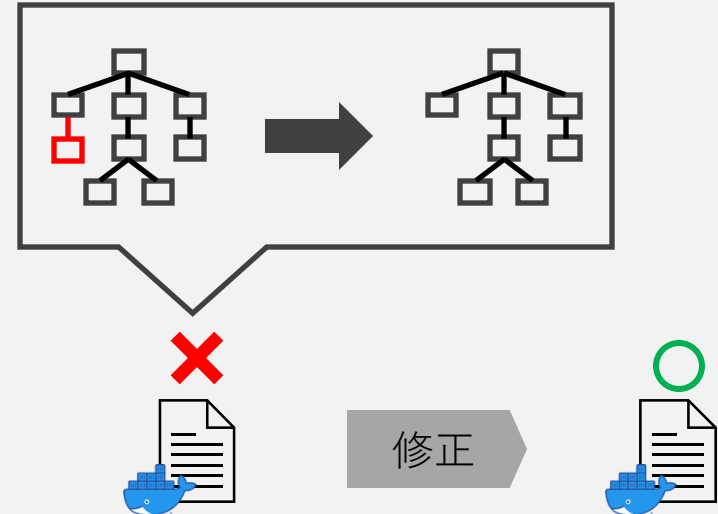
I ビルドエラーの原因となるノードを検索

- － 失敗確率が高い部分木を低い部分木へ置換

1. 失敗確率対照的な部分木の類似ペアを検索



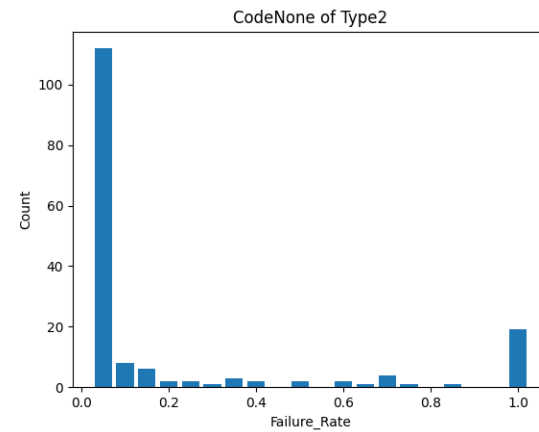
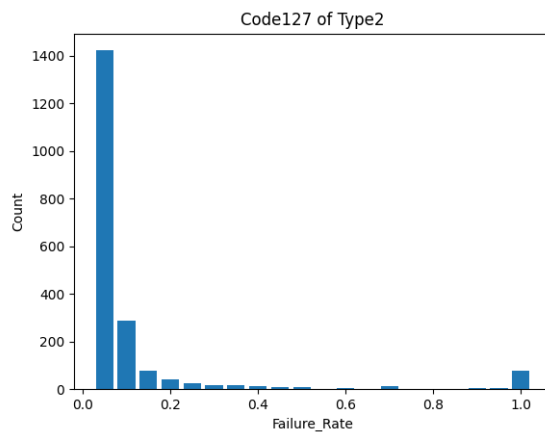
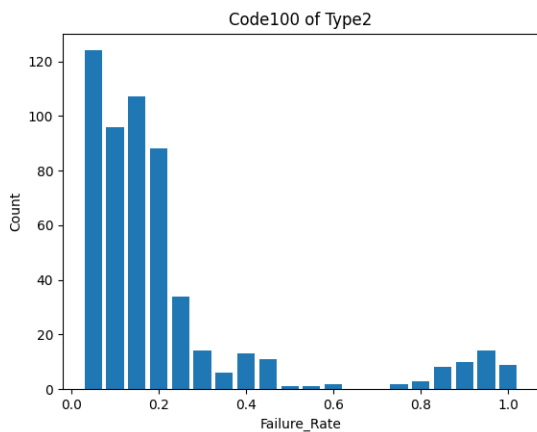
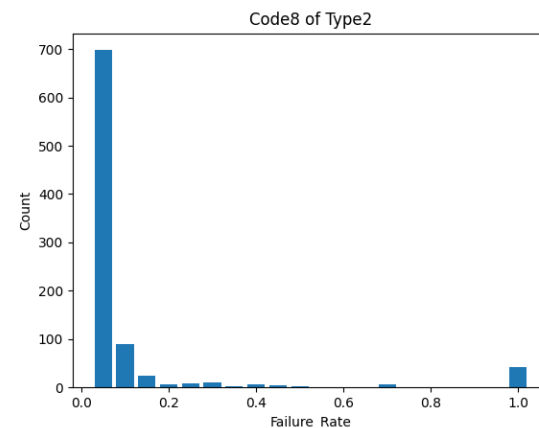
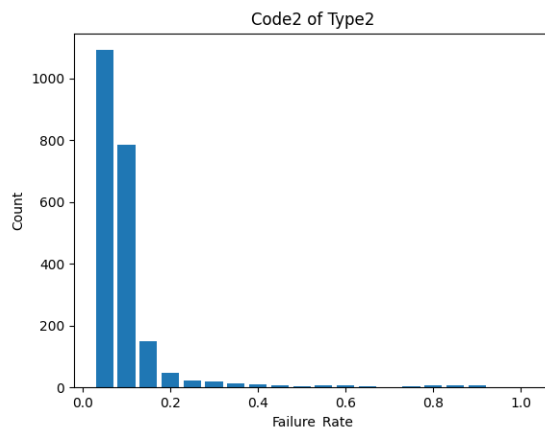
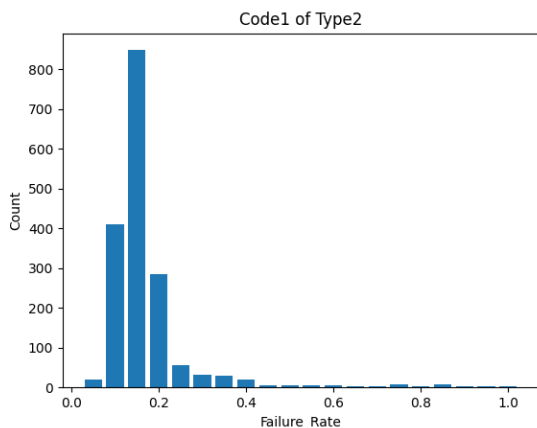
2. 成功するパターンと失敗するパターンを置き換え



補 足 資 料

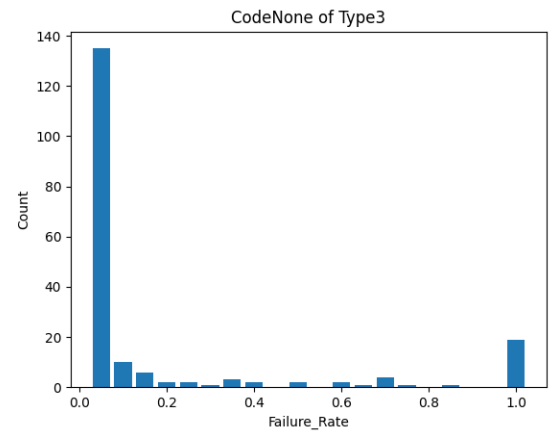
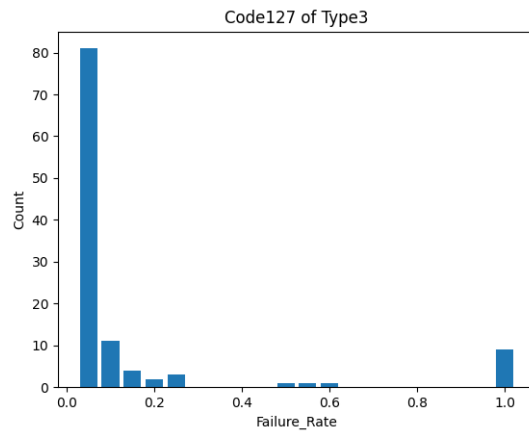
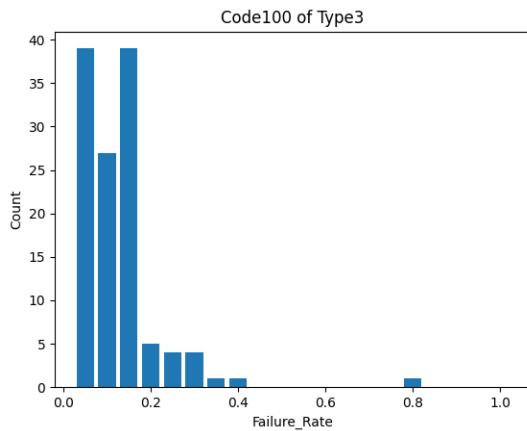
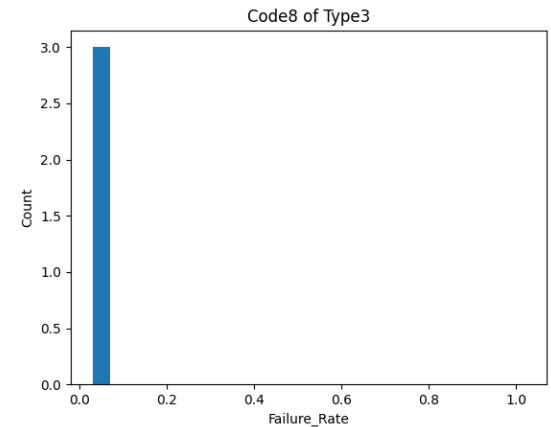
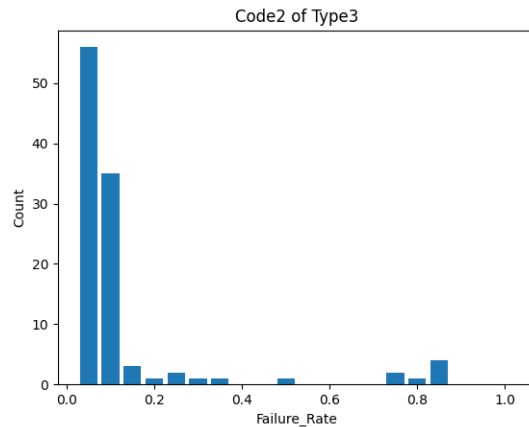
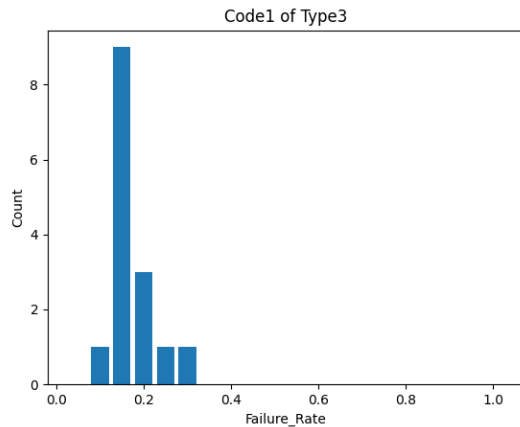
失敗確率の計算

抽出した部分木をもつDockerfileがビルドに失敗する確率



失敗確率の計算

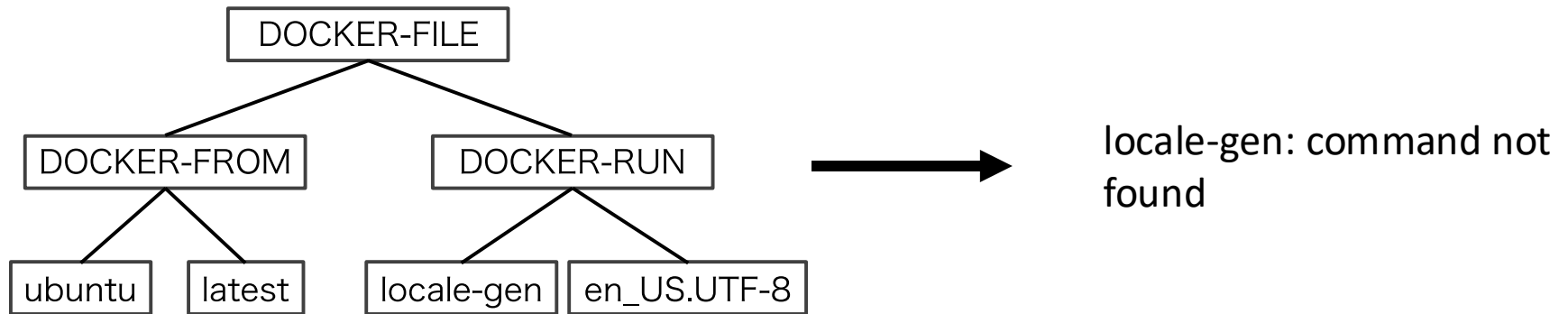
抽出した部分木をもつDockerfileがビルドに失敗する確率



抽出できた依存関係

例

- Type.1, 終了コード127番



- locale-gen コマンドはubuntuのビルトインコマンドではない