
ビルドエラーを誘発する Dockerfileの依存関係の抽出

オープンソースソフトウェア工学研究室
坂本廉也

背景

| コンテナ型仮想化

- 軽量で高速な仮想環境（コンテナ）の構築/実行を実現する技術
- 近年急激に普及しているクラウド環境と相性が良い

| Docker^[?]

- 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
 - 独立した仮想アプリケーション環境（Dockerコンテナ）を動作
- コードから仮想環境を構築できるため、環境の再現性が高い
- IT企業の約79%がDockerを利用^[?]

[?] 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.

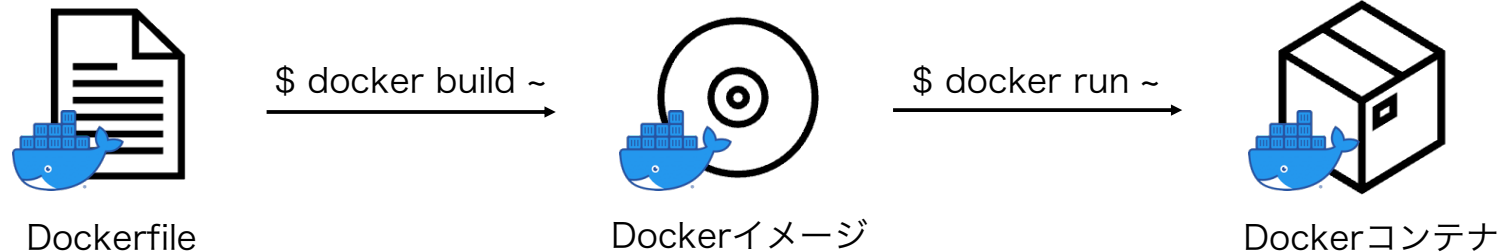
[?] <https://www.docker.com/>

[?] <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）

- 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
- DockerHub^[4]などのコンテナレジストリなどで共有可能



Dockerfile

- 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
- Dockerfileをビルドすることでイメージを作成

Dockerfileのビルド

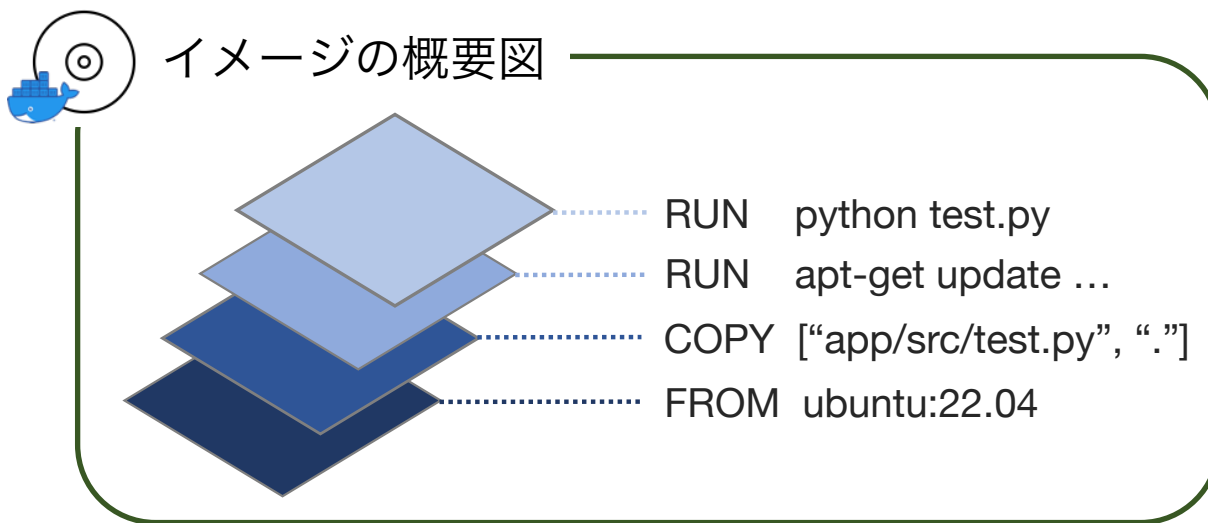
```
1 FROM ubuntu:22.04 #既存イメージの機能を継承
2 COPY ["src/test.py", "."] #ファイルをイメージへコピー
3 RUN apt-get -y update && \ #pythonをインストール
4     apt-get install python=3.10
5 RUN python test.py #pythonファイルの実行
```

| Dockerfileのビルド

- Dockerfile内の命令を上から順に実行
 - 先に実行された命令を親として命令同士が親子関係を持つ
- 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）

イメージのレイヤ構造

- | 各命令の実行内容を1つのレイヤに保持し，スタック
 - 依存関係を持つレイヤの集合としてイメージが成立

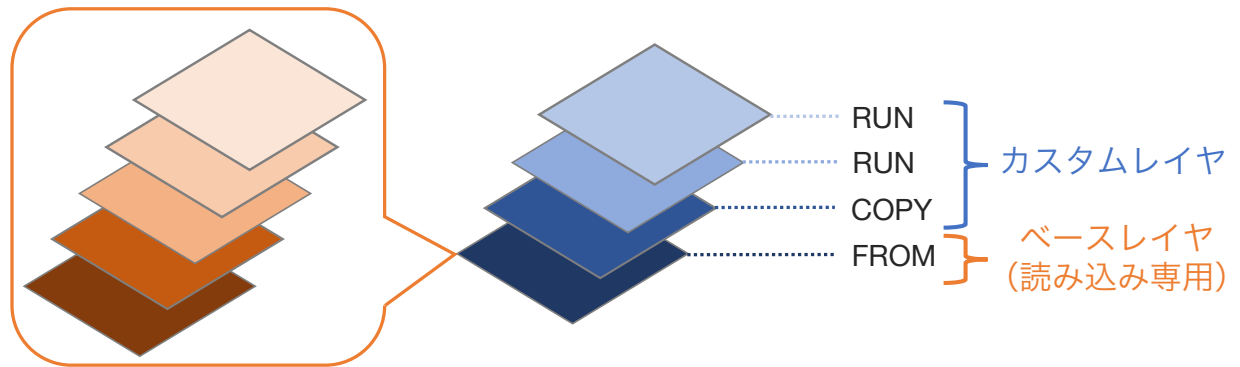


- | 実行命令と既に実行された命令の結果との互換性を検証
 - 互換性に問題が無い → レイヤをスタックし次の命令を実行
 - 互換性に問題がある → ビルドエラー

依存関係の複雑さ

| 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ないため、アドホックな記述になりやすい[5]



| 自身が記述した命令でも予期しない依存関係が発生

- 命令が実行されてはじめて依存関係が定義される

開発者が関知しない依存関係が原因でビルドエラーが発生

[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

研究動機

| Dockerfileのビルドエラー

- 他の言語がベースのプロジェクトのビルドと比較してビルドエラー発生率が高い^[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]}

研究動機

Dockerfileのビルドエラーを自動で修正することで
スピーディなソフトウェア開発へ貢献可能

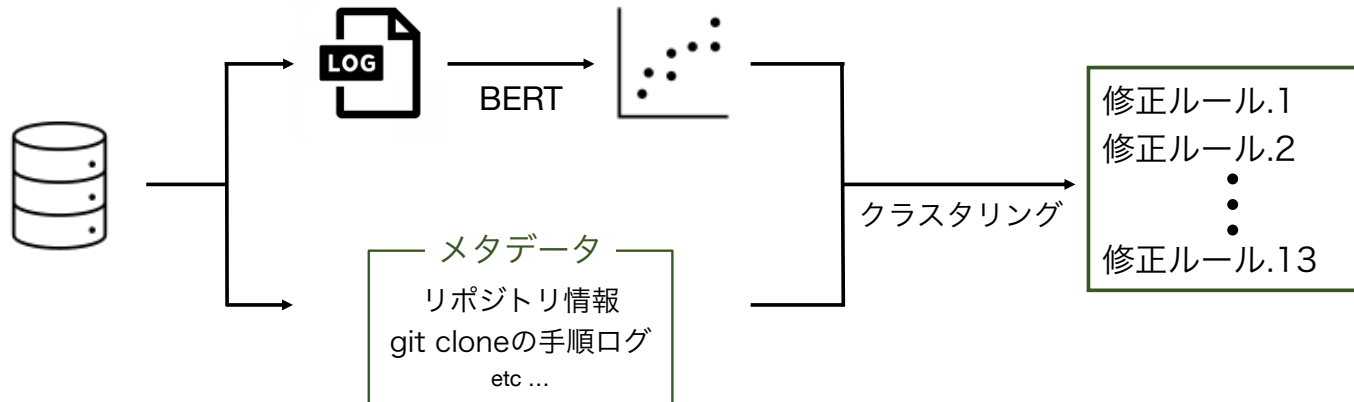
[6] Cito, J., Schermann, G., Wittern, J. E., Leitner, P., Zumberi, S. and Gall, H. C.: An Empirical Analysis of the Docker Container Ecosystem on GitHub, In Proceedings of MSR '17, pp.323–333 (2017).

[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).

先行研究 | Shipwright^[9]

| ルールベース手法で18.9%のビルドエラーを自動修正



⚠ ビルドエラーを誘発する複雑な依存関係を考慮できていないため修正率が低い

—— 本研究の目的 ——

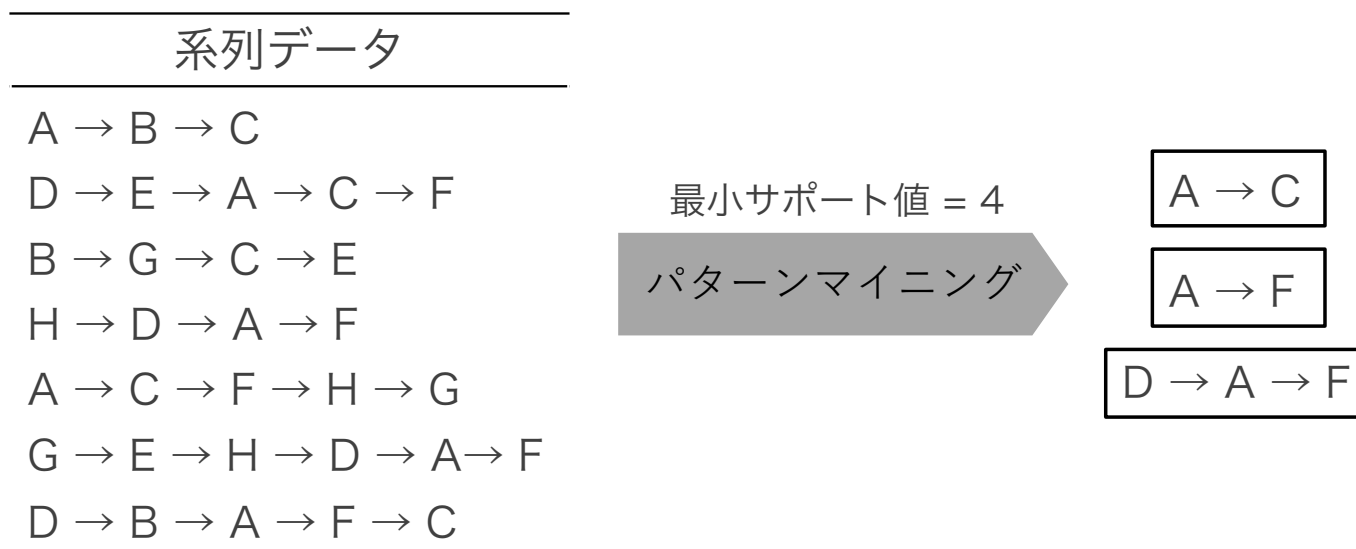
依存関係を考慮することで、Shipwrightが修正できないビルドエラーを自動修正する

[9] Henkel, J., Silva, D., Teixeira, L., d'Amorim, M. and Reps, T.: Shipwright: A Human-in-the-Loop System for Dockerfile Repair, In Proceedings of ICSE '21, pp. 1148–1160.

依存関係の抽出

| 系列パターンマイニング^[10]

- 大量データから順序を考慮した頻出する組合せを抽出する手法
 - 最小サポート値で頻出度を指定
- Dockerfile内の命令の記述順序を考慮した命令の組み合わせ（依存関係）を抽出する必要



[10] R.Agrawal, and R.Srikant: "Mining sequential patterns". Proceedings of the Eleventh International Conference on Data Engineering, 1995

データセット

- | Shipwrightのデータセットをフィルタリングして使用
- スター数10以上, 単一のDockerfileを持つGitHubリポジトリ

フィルタリング条件

1. 破損しているリポジトリ
2. 現在は存在していないリポジトリ
3. マルチステージビルドを用いたDockerfileを持つリポジトリ

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%

事前調査

| 依存関係が原因のビルドエラーの割合

- データセットに含まれるビルドログを目視で調査

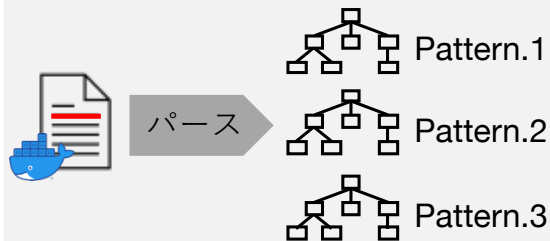
ビルドエラーの原因	Dockerfile数	割合
依存関係	3,259	71.6%
その他	1,295	28.4%
合計	4,554	100.0%

- ビルドエラーの原因の70%以上が依存関係が原因
- ビルドエラーを引き起こす依存関係を考慮することで効率的に自動修正できる

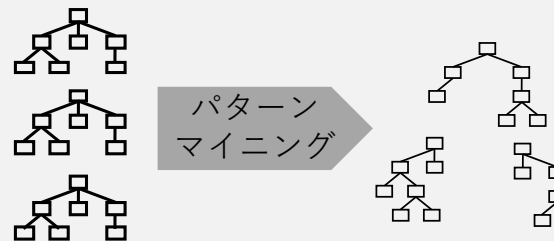
自動修正までのながれ

1. ビルドエラーが発生するDockerfileから，抽象度の異なる3種類の抽象構文木（AST）へ変換
2. 系列パターンマイニングを用いて，ASTからビルドエラーを誘発する依存関係を抽出
3. 修正ルールを作成し，自動修正

1. DockerfileをASTへ変換



2. 系列パターンマイニング



3. 修正ルールの作成 & 自動修正



ASTへ変換

| Dockerfileの構文構造

- 命令と引数から構成

命令 : Dockerfile固有の命令

```
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

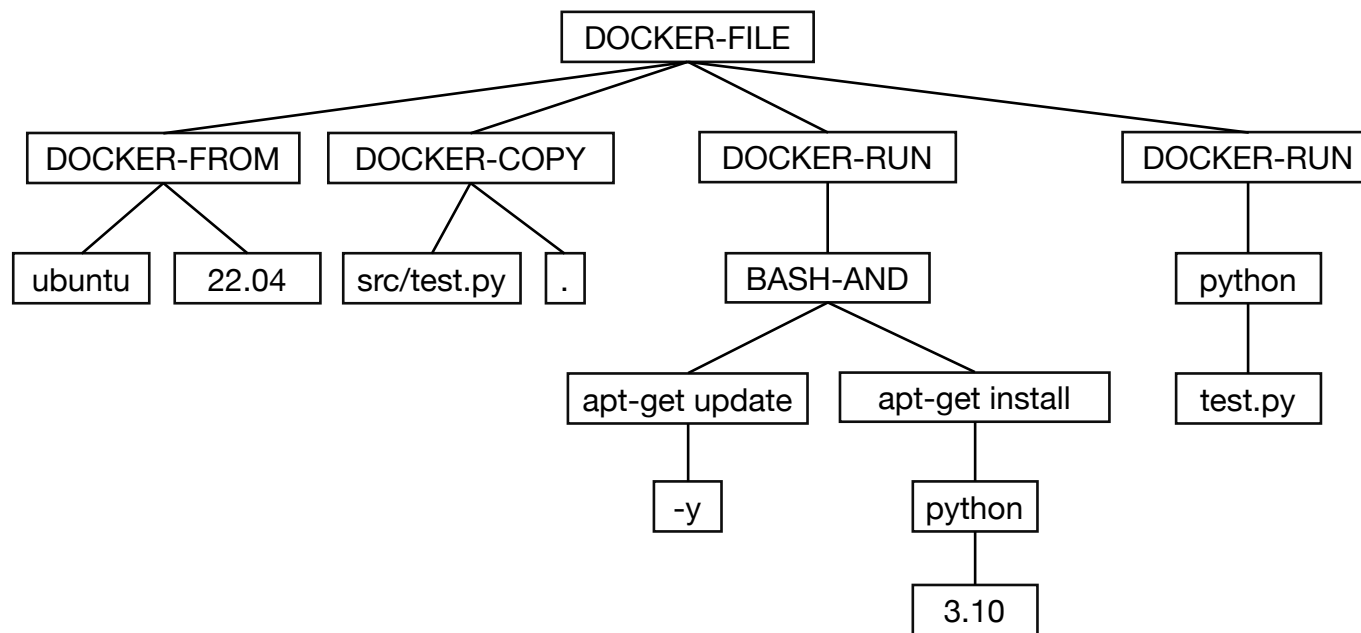
| 命令を意味的に解釈可能な形 (AST) へ変換

- 構文的な表現揺れを回避
- 順序木で表現することで命令の実行順序を保持できる

ASTへ変換 | Pattern.1

| Pattern.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

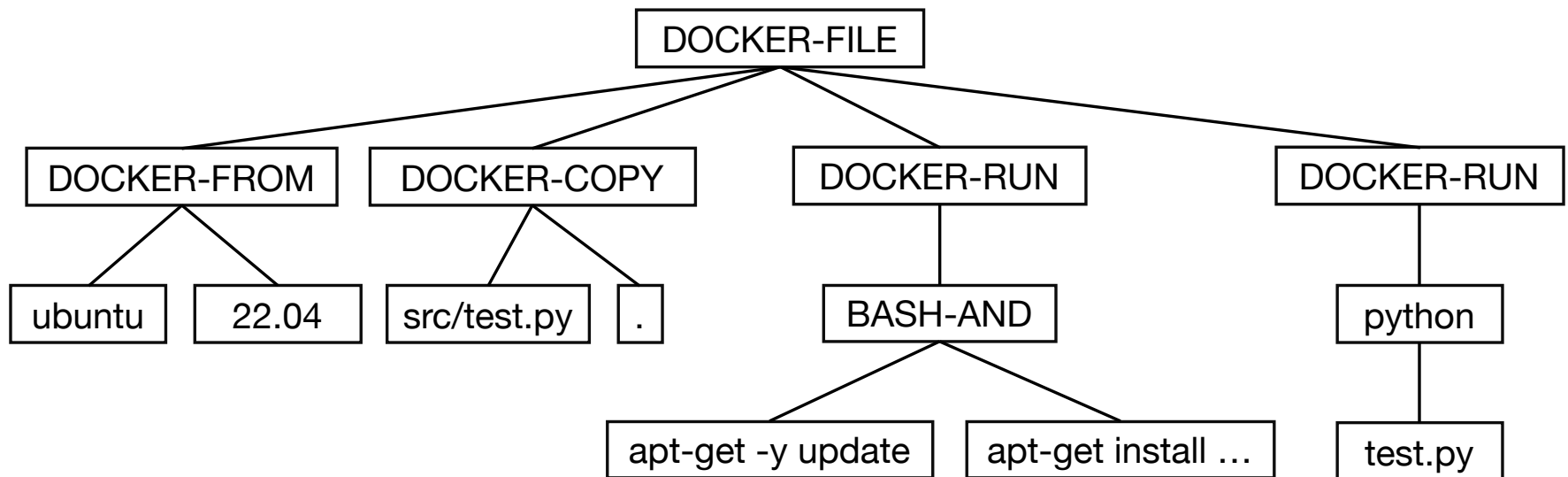


アイテム数が膨大になり,
パターンマイニング時に組み合わせ爆発が起こる可能性

ASTへ変換 | Pattern.2

| Pattern2 : シェルスクリプトを行レベルでノードに追加

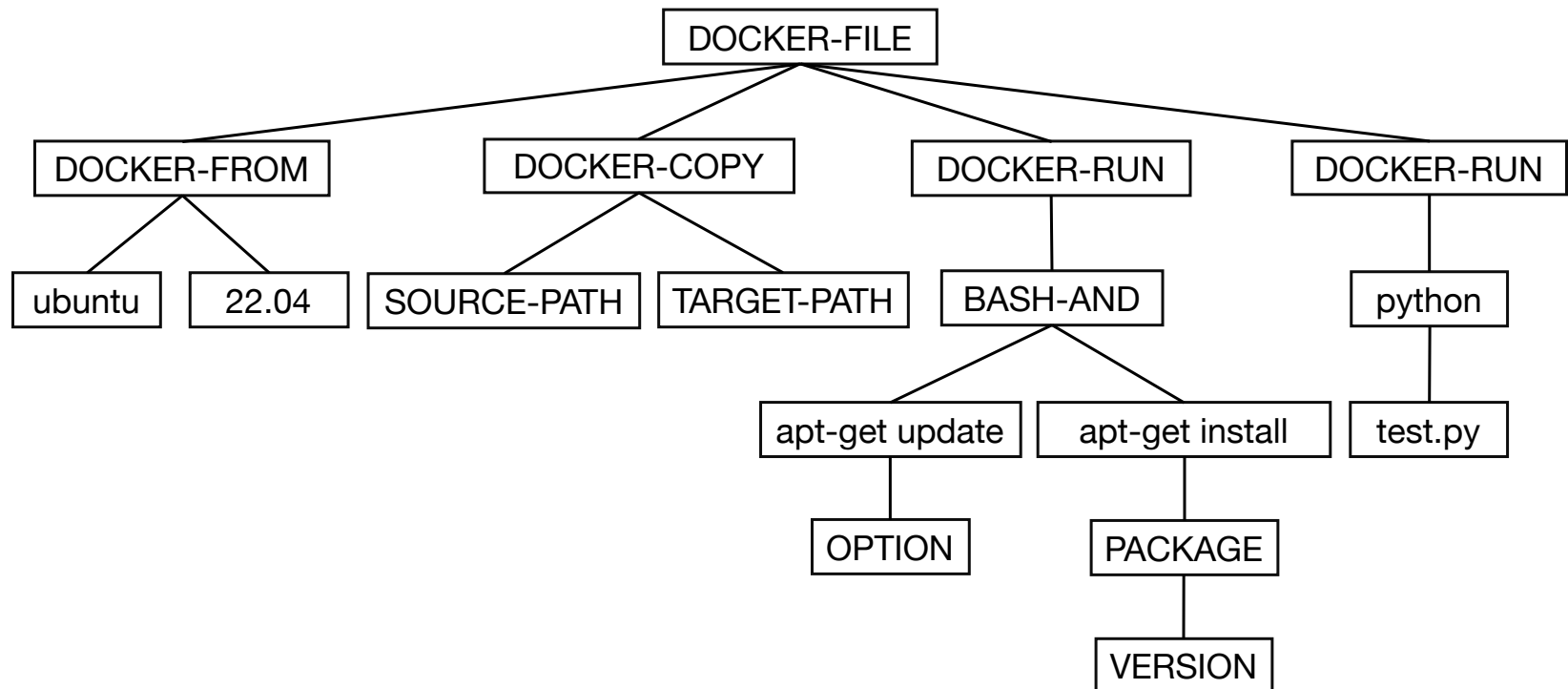
- トップレベルの構文だけのパースによってアイテム数を削減
- Dockerfileの構文レベルの頻出パターンを抽出可能



ASTへ変換 | Pattern.3

| Pattern3 : パッケージ, URL, パスなどの抽象化

- Pattern.1 から情報を抽象化することでアイテム数を削減
- Dockerfileの構造レベルの頻出パターンを抽出可能



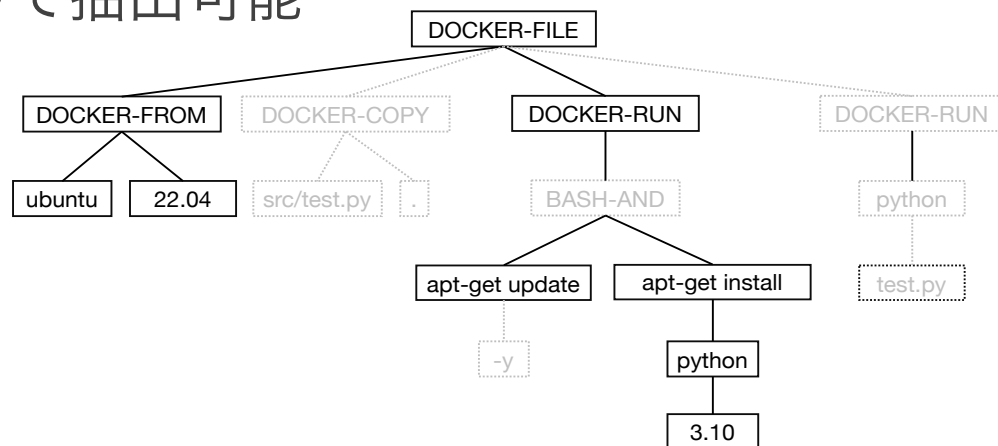
系列パターンマイニング | TreeMiner

| 順序付き頻出部分木マイニング

- 順序木に対する系列パターンマイニング手法

| TreeMinerアルゴリズム

- データセットの木構造の集合から頻出な部分木（ある木の部分集合）を抽出するアルゴリズム
- 子孫関係を頻出部分木として抽出可能



TreeMiner | 結果

| TreeMinerで出力されたパターン数

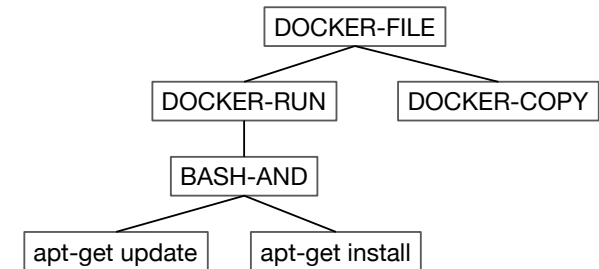
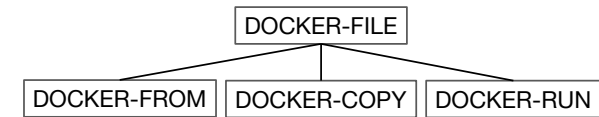
Pattern.1	Pattern.2	Pattern.3
11,356件	430件	173件

- 抽象度を高くすると出力パターン数が少なくなる
 - 抽象度が高くなるとアイテム数が減少するためアイテムの組み合わせ数も減少
- 手作業でビルドエラーを誘発する依存関係とそれ以外を分類

TreeMiner | 結果

| Pattern.1

- 抽象度の高いパターンしか抽出できない
 - サポート値を下げると計算が終わらない
- 得られた頻出部分木の例
 - 引数に関するパターンが抽出されない
 - 具体的なパッケージ名が抽出されない



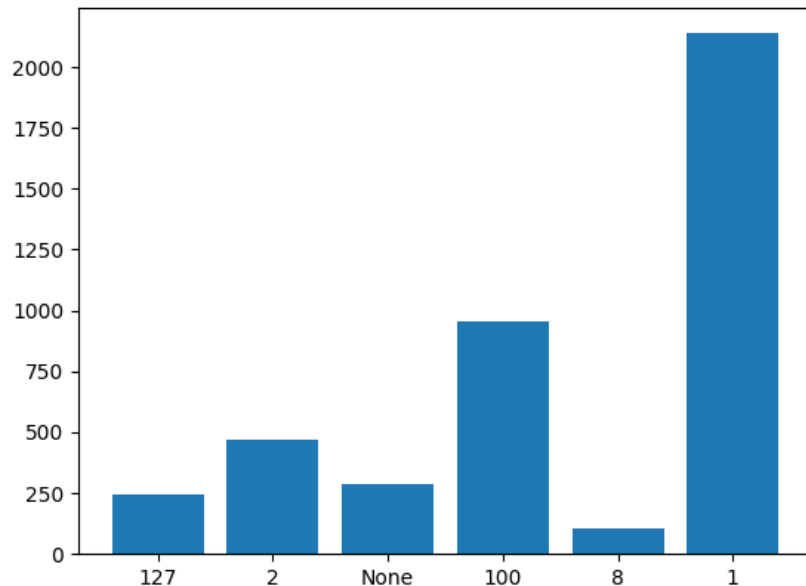
| Pattern.2 & Pattern.3

- Pattern.1と同様にサポート値を下げると計算が終わらない
- ビルドに成功するDockerfileにも存在する依存関係しか抽出できない

データの工夫

| 終了コードの件数が100件以上存在するものを対象

- 終了コードごとにエラーの種類が異なる
- 先に終了コード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

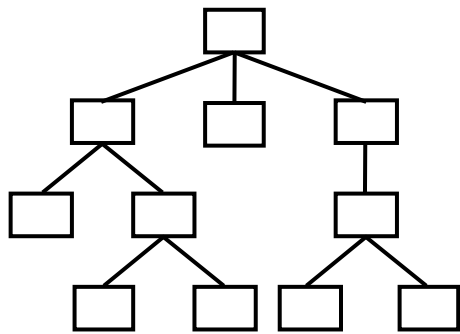


Code 127 : 不明なコマンドの使用
Code 2 : ビルトインコマンドの誤用
None : bash以外の箇所でのエラー
Code 100 : ネットワーク異常
Code 8 : サーバーからのエラー応答
Code 1 : 一般的なエラー

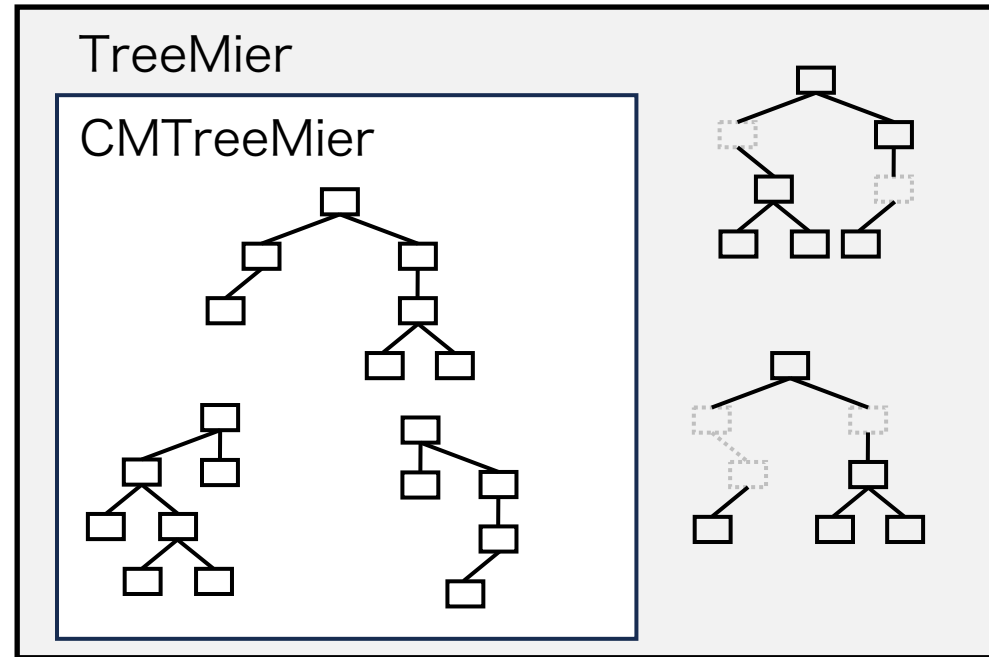
アルゴリズムの工夫

| CMTreeMiner[?]

- 順序付き頻出飽和部分木マイニング
- ある頻出部分木の部分集合は考慮しないため計算効率が良い



パターン
マイニング

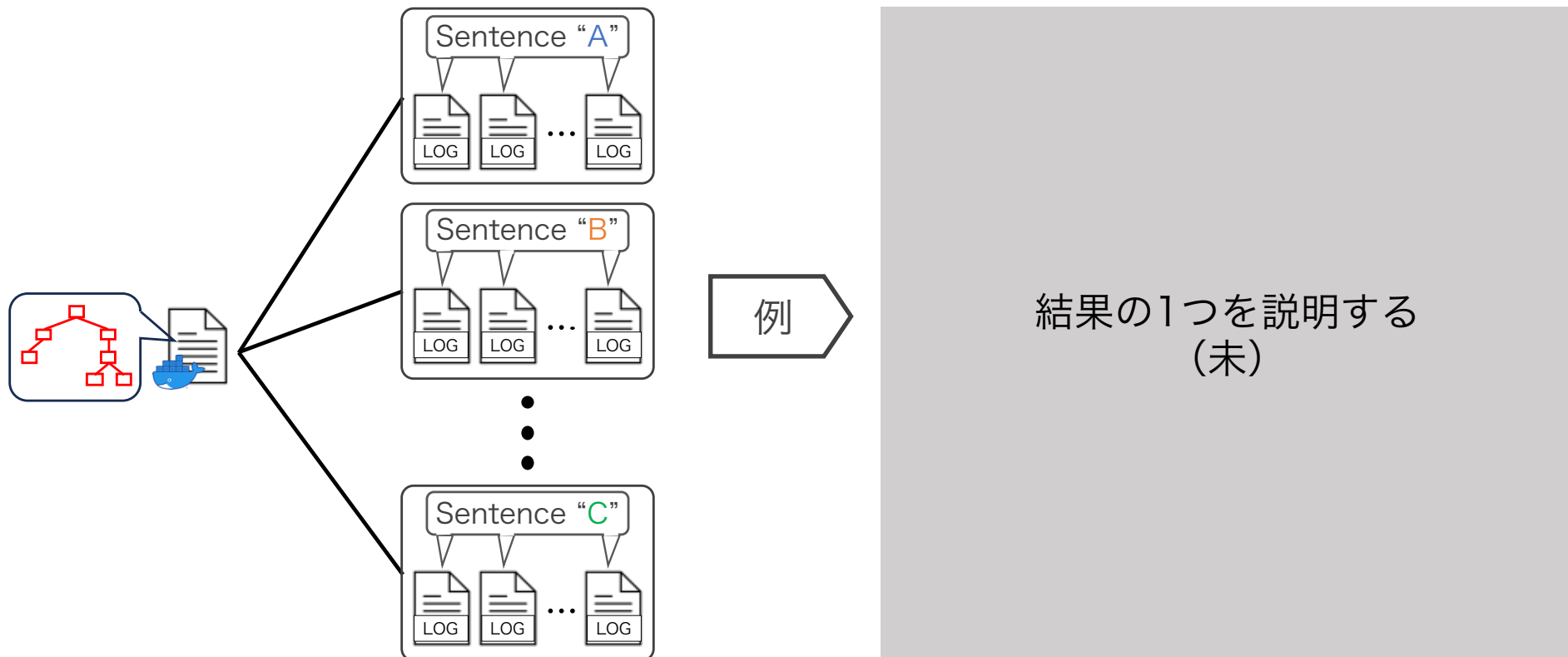


子孫関係を考慮できないため取りこぼしが発生する可能性

CMTreeMiner | 結果

| 定性的評価

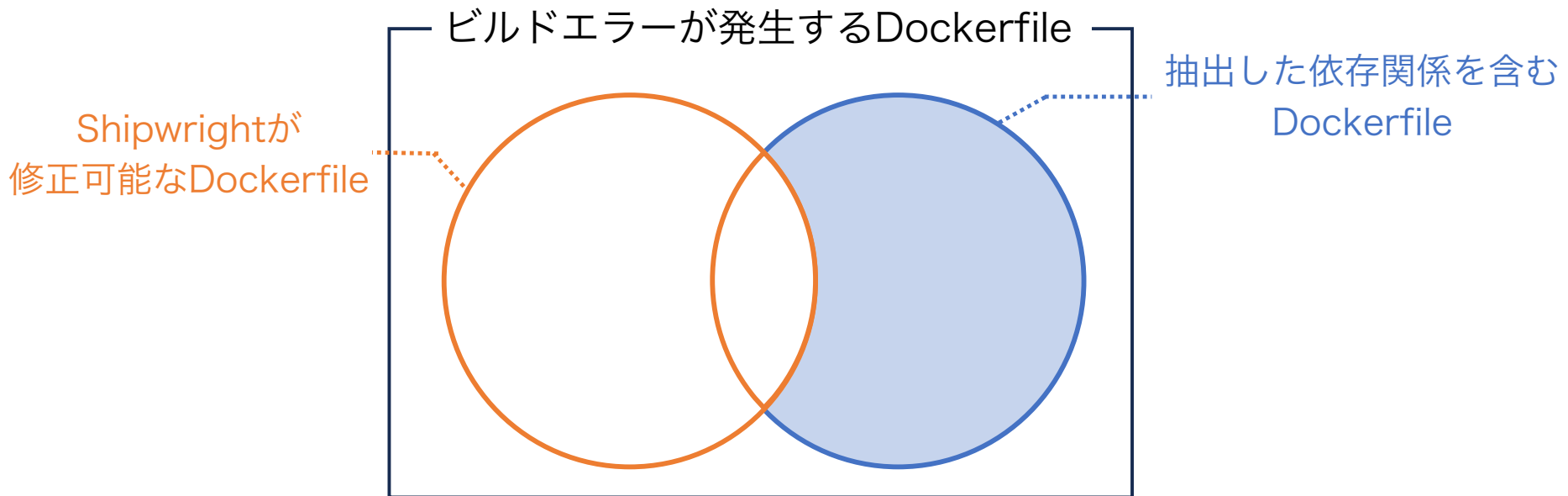
- 抽出した依存関係とビルドログからあるパターンがどのようなエラーを引き起こすかを理解



今後の予定 | ビルドエラーパターンの評価

| 定量的評価

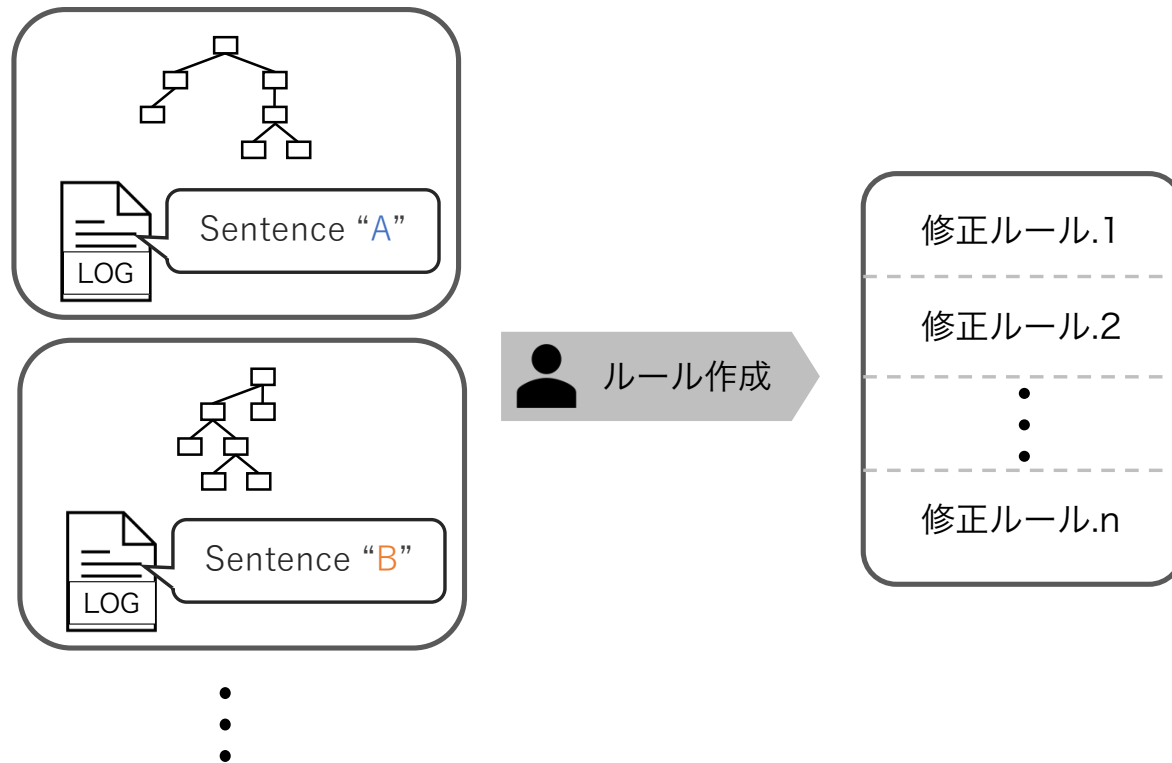
- Shipwrightが修正できないビルドエラーをどの程度抽出可能か
 - Shipwrightと異なるビルドエラーを対象に自動修正手法を提案することで広範なビルドエラーを修正可能



今後の予定 | 自動修正

| ルールベース

- ビルドエラーを誘発する依存関係とビルドログから修正ルールを作成



対外発表状況

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