

頻出部分木マイニングを用いた Dockerfileのビルドエラーを誘発する 依存関係の抽出手法の提案

オープンソースソフトウェア工学研究室
坂本 廉也

背景

| ソフトウェアエコシステムの高速化

- 高度情報化社会での社会の急速な変化に追従
- 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

| Docker^[1]

- 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
- IT企業の約79%がDockerを利用^[3]

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

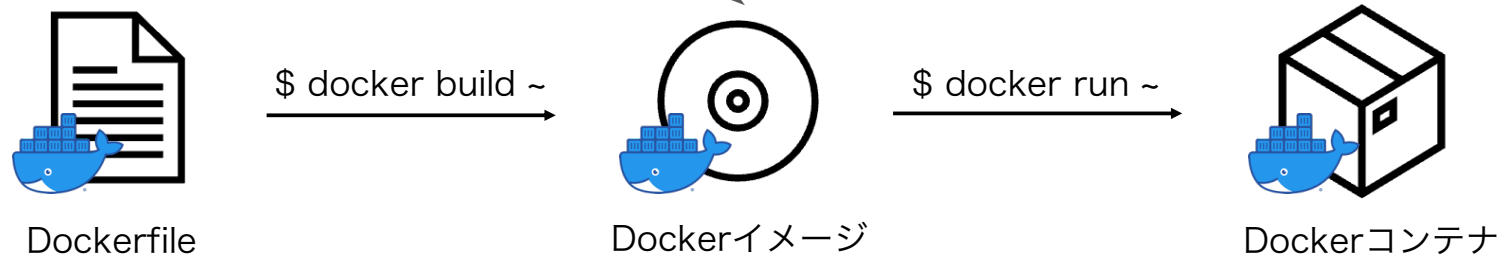
[2] A. Zerouali, V. Cosentino, G. Robles, J. Gonzalez-barahona, and T. Mens, “Conpan : A tool to analyze packages in software containers,” in MSR '19, 2019, pp. 592–596.

[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）

- 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

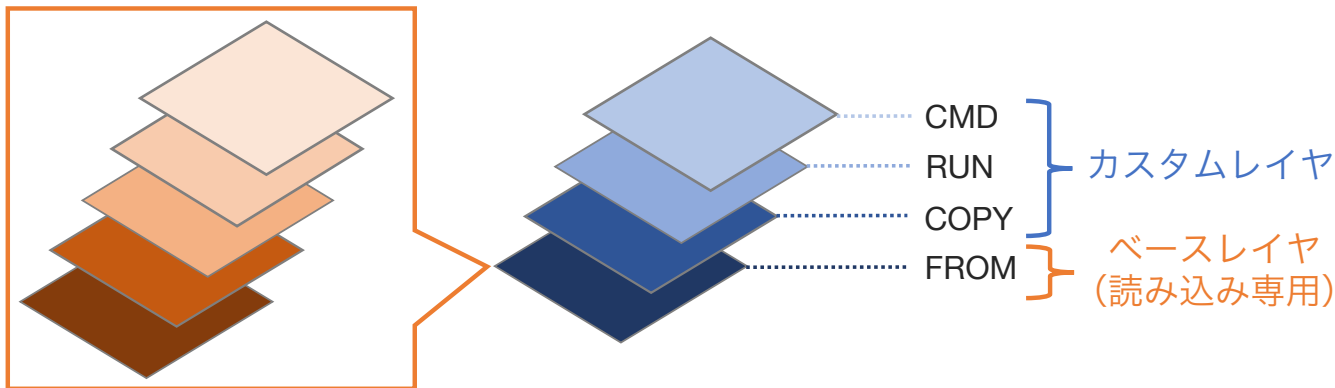
[4] <https://hub.docker.com/>

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



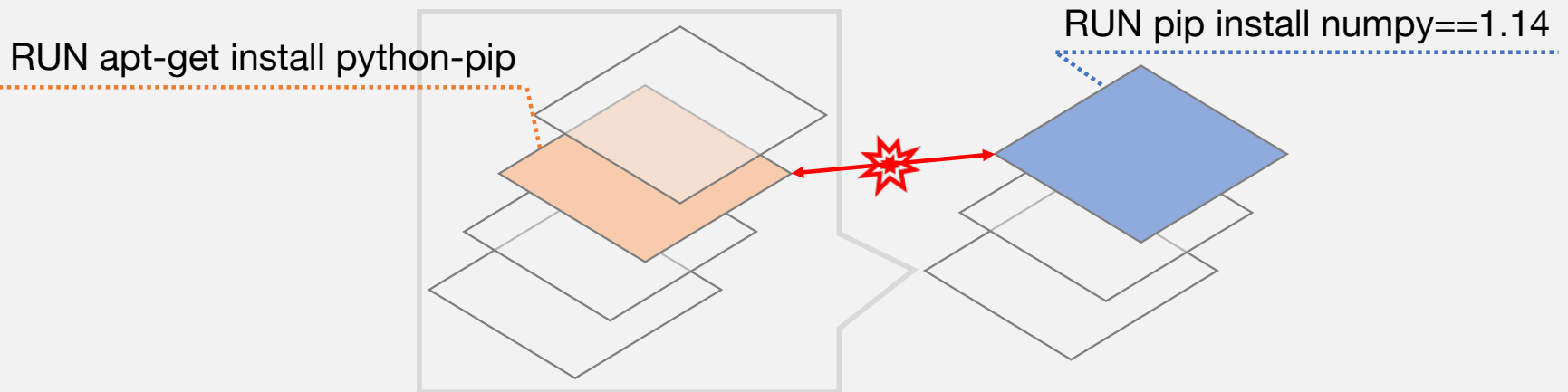
様々な原因で**ビルドエラー**が発生する（ビルドに失敗する）ことがある

ビルドエラーの原因

命令間の依存関係, 構文ミス, 権限 など様々

– “開発者が関知しない依存関係”によってビルドエラーが発生

- 開発者自身に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

[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).

研究目的



ビルド時の複雑な依存関係を考慮することで広範な
ビルドエラーを自動修正可能

依存関係を考慮した自動修正手法

1. ビルドエラーを誘発する依存関係を抽出
2. 抽出した依存関係に対して修正パッチを作成し修正

Dockerfileのビルドエラーを誘発する依存関係の
抽出手法の提案

データセット

先行研究^[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

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%

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

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多くのビルドエラーに対して自動修正が可能

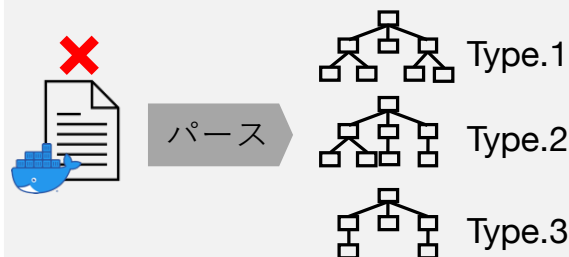
依存関係抽出のながれ

1. シェルの終了コードでビルドエラーが発生するDockerfileを分類
2. ビルドエラーが発生するDockerfileから、頻出パターンマイニングでビルドエラーを誘発する依存関係を抽出
 - － 抽象度の異なる3つのタイプの抽象構文木（AST）に変換
3. 修正ルールを作成し、自動修正
 - － 抽出された依存関係とビルドログを使用

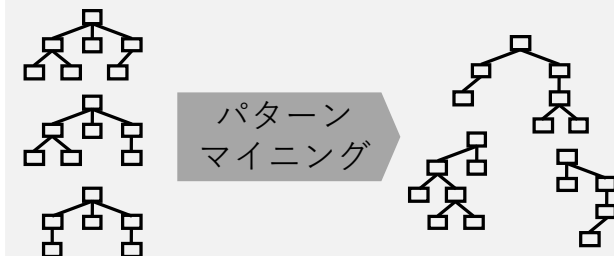
1. 終了コードで分類



2. DockerfileをASTへ変換

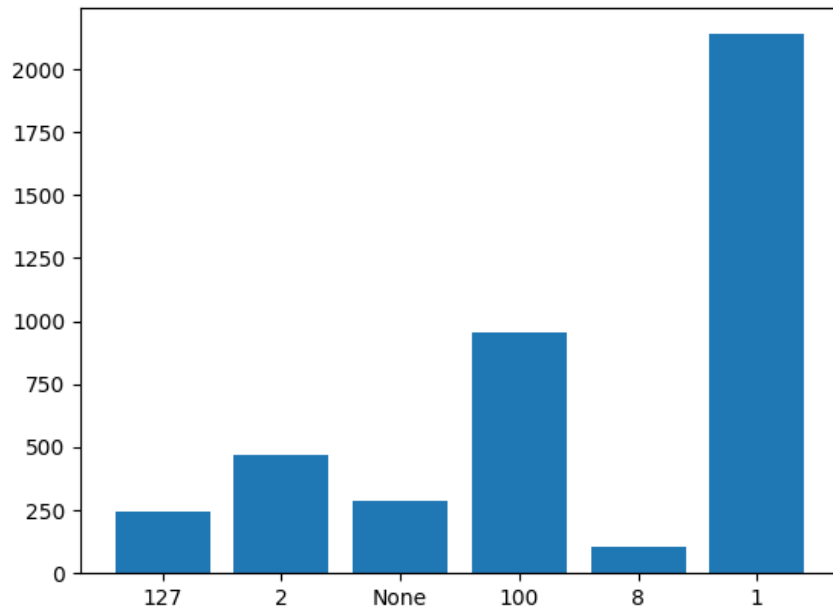


3. 系列パターンマイニング



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
 - 終了ステータスごとにエラーの種類が異なる
 - 100件以上存在する終了ステータスのみを対象
 - 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



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 : 不正なコマンドの使用
Code 1 : 一般的なエラー

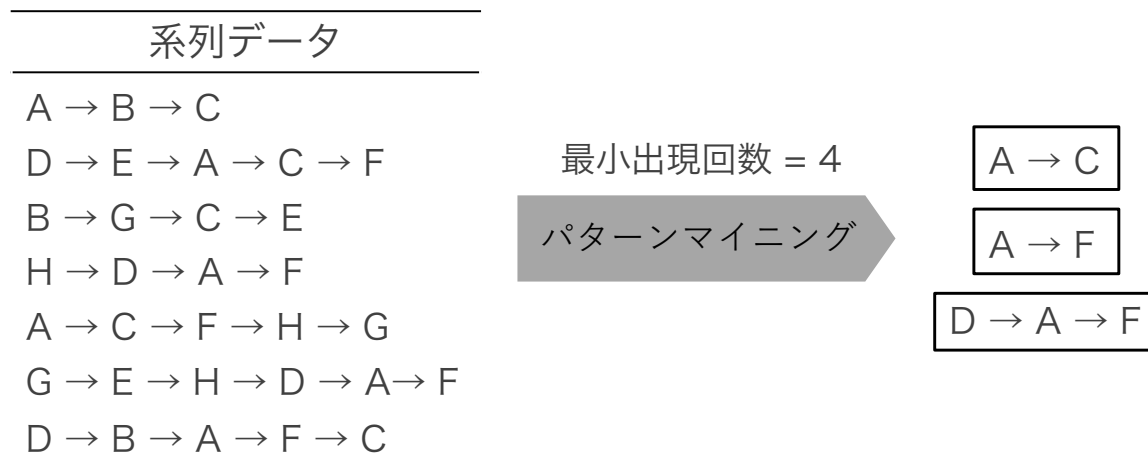
依存関係の抽出

依存関係の抽出

- ビルド時の命令の実行順序を考慮する必要
- 様々なアプリケーションの設定に関する命令に対応する必要

系列パターンマイニング^[10]

- 大量データから頻出系列データを抽出する手法



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

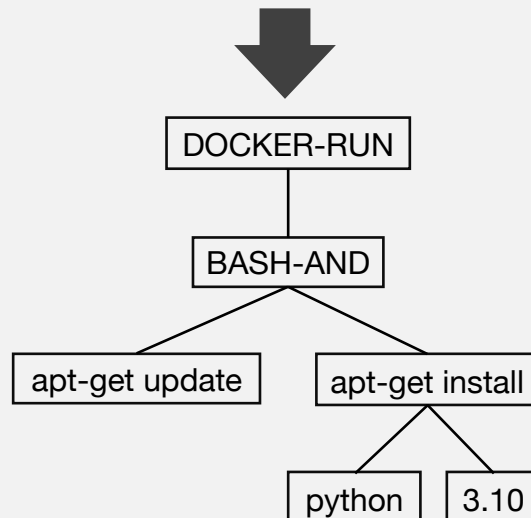
DockerfileをASTへ変換

- | 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
```

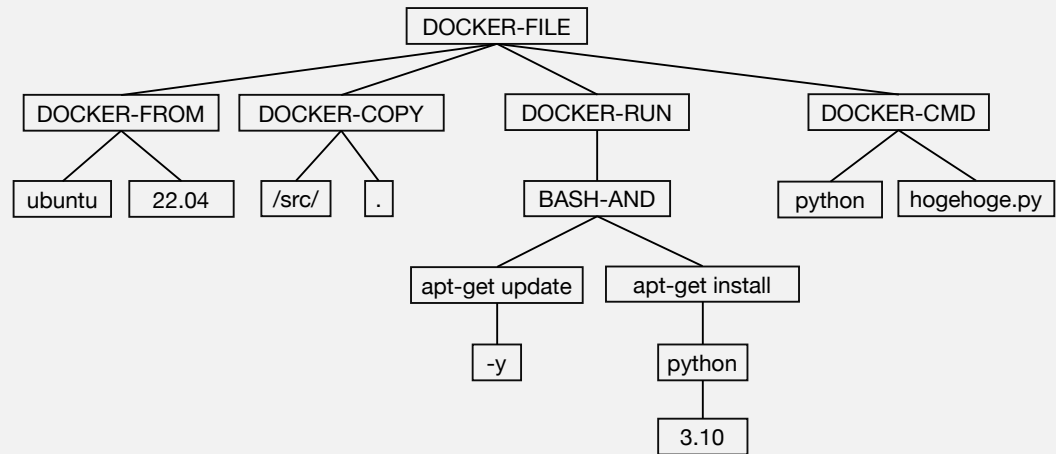
引数：シェルスクリプトや環境変数が埋め込まれる



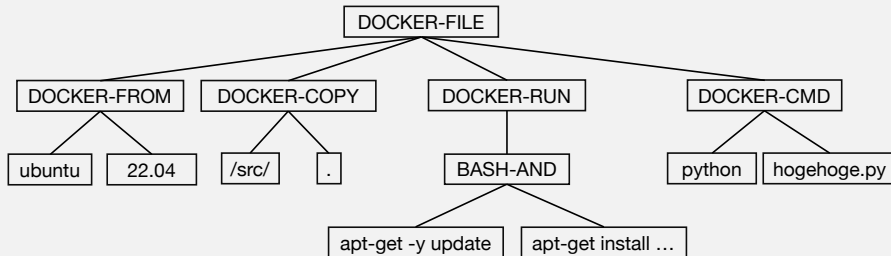
DockerfileをASTへ変換

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



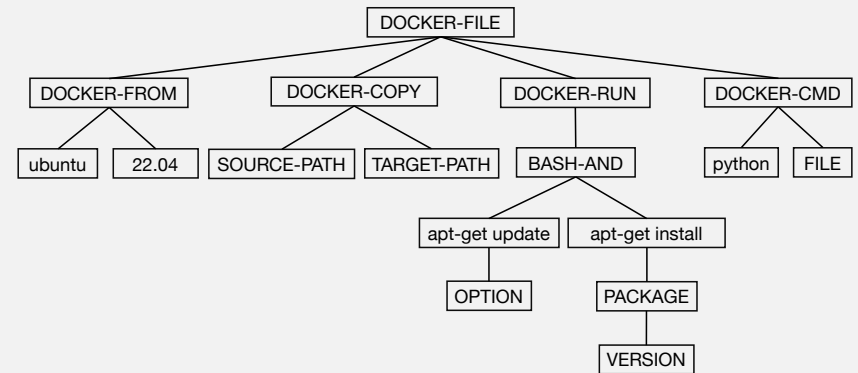
Type.2 : Dockerfile構文のパース



- Dockerfileの構文レベルの依存関係を抽出



Type.3 : ノードの抽象化

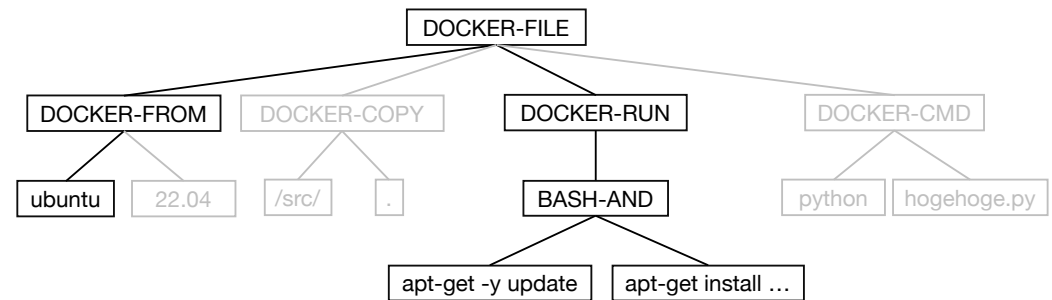


- Dockerfileの構造レベルの依存関係を抽出

ASTへの系列パターンマイニング

順序付き頻出部分木マイニング

- 順序木の集合から順序関係を保持した頻出部分木の抽出手法
- 本研究への適用
 - ASTは順序木で実装しているため、順序関係を保持できる



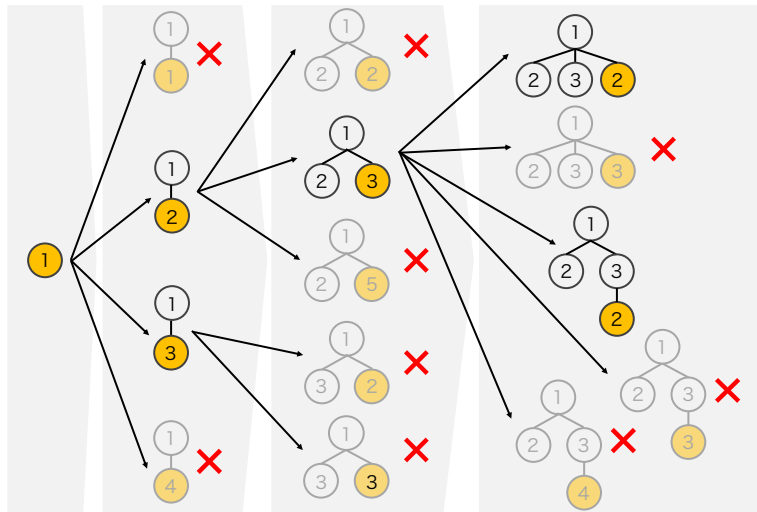
適用に際しての課題

- 一般的に、半構造データに対する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

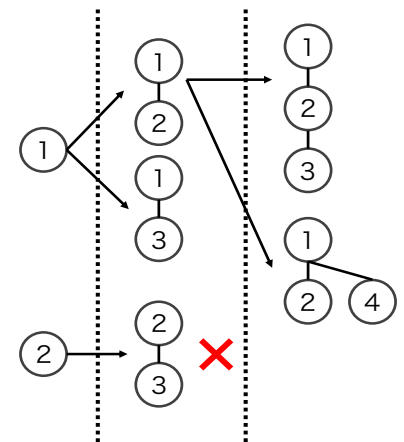
CMTreeMiner^[11]

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



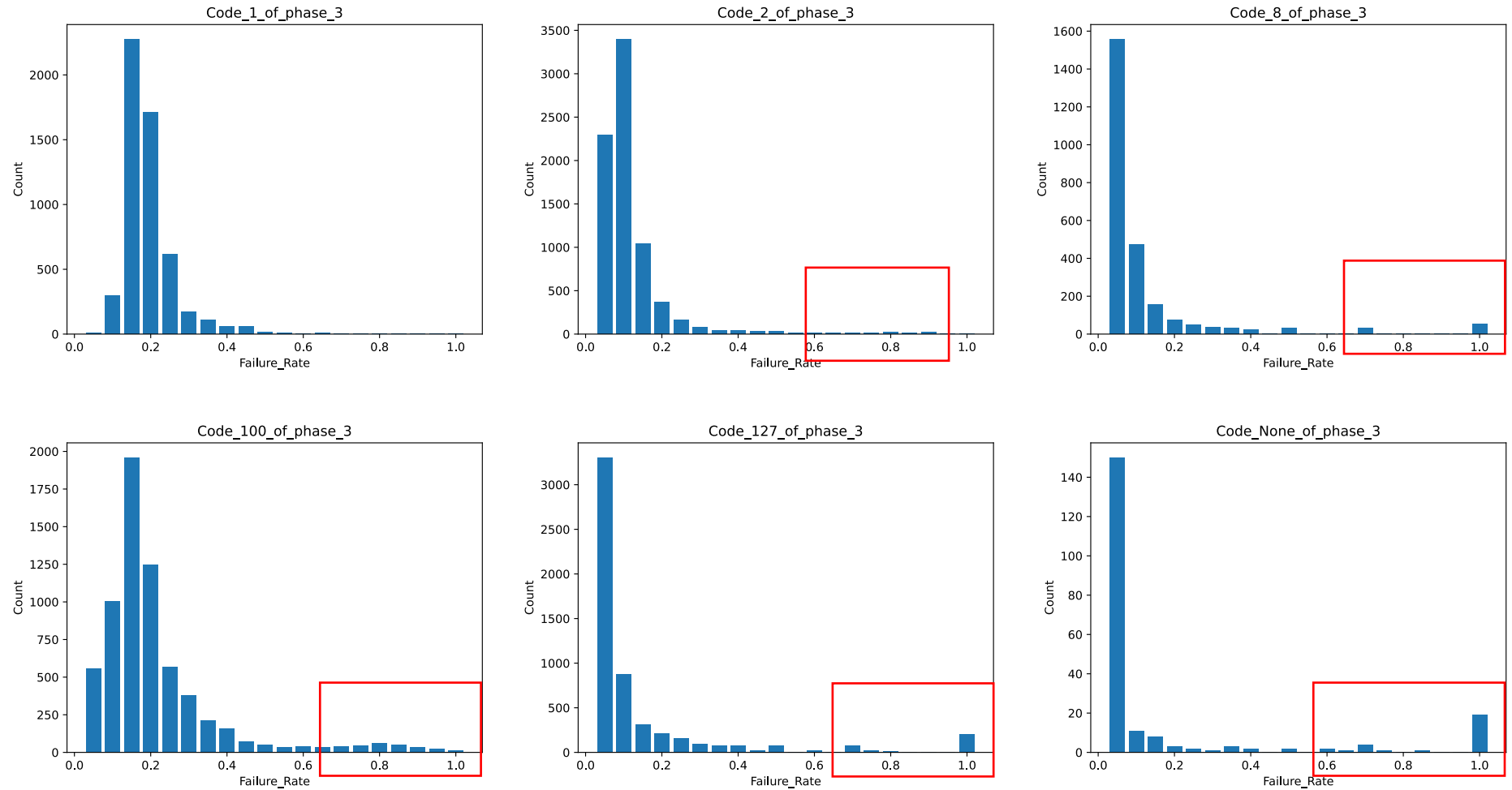
逐次探索空間の刈り込み



[11] 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)

結果 | Type.1

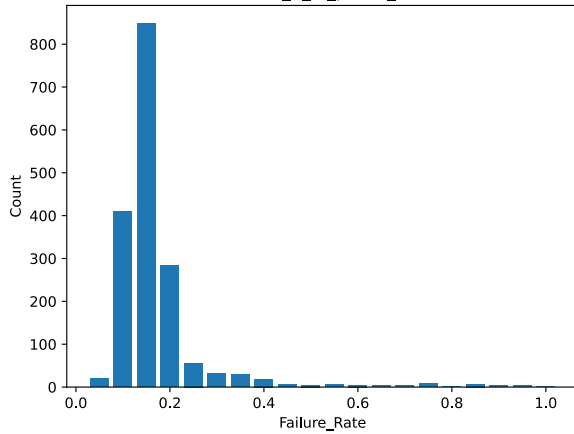
抽出した部分木をもつDockerfileがビルドに失敗する確率



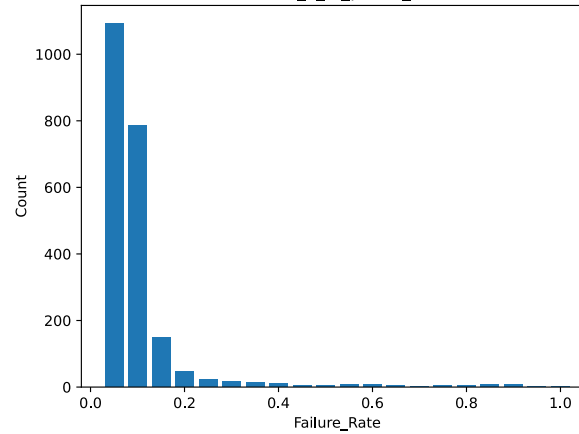
結果 | Type.2

抽出した部分木をもつDockerfileがビルドに失敗する確率

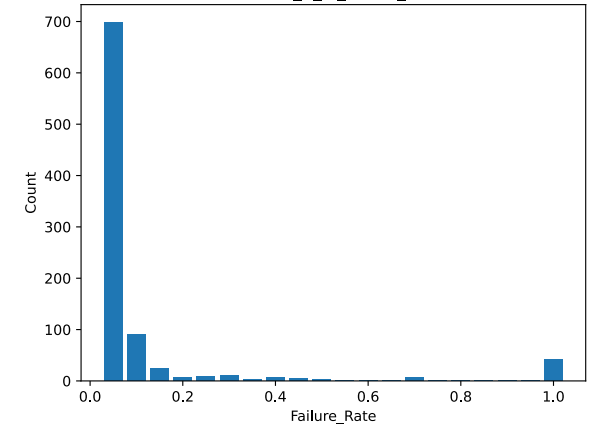
Code_1_of_phase_2



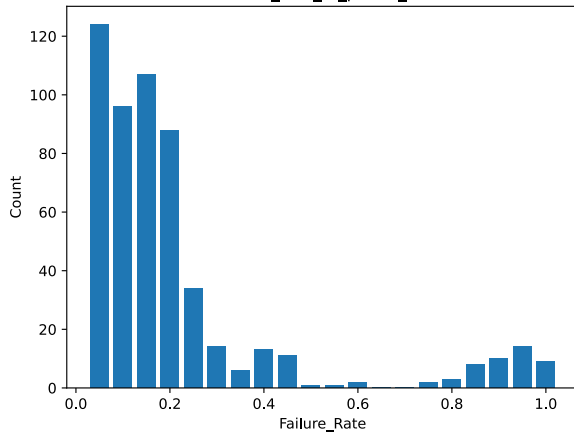
Code_2_of_phase_2



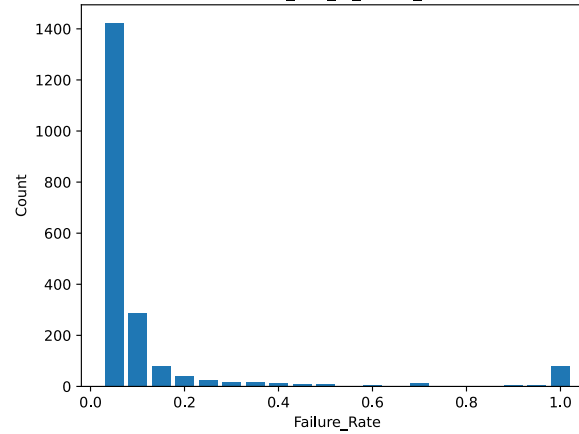
Code_8_of_phase_2



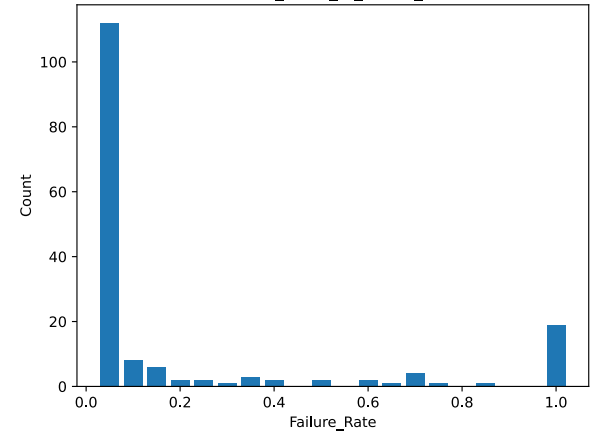
Code_100_of_phase_2



Code_127_of_phase_2



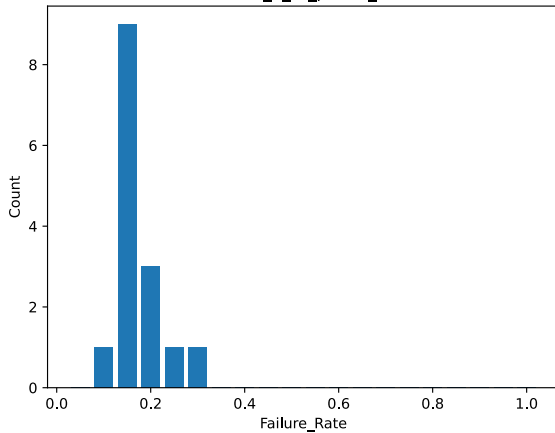
Code_None_of_phase_2



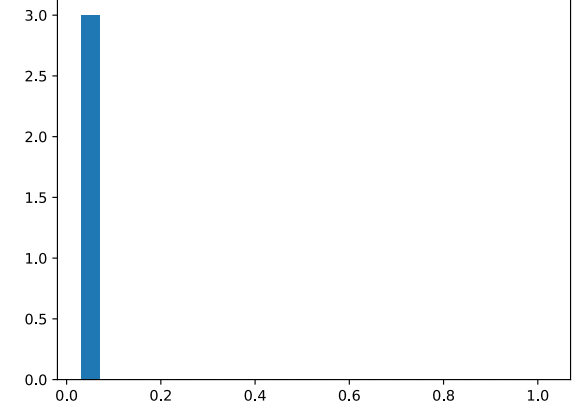
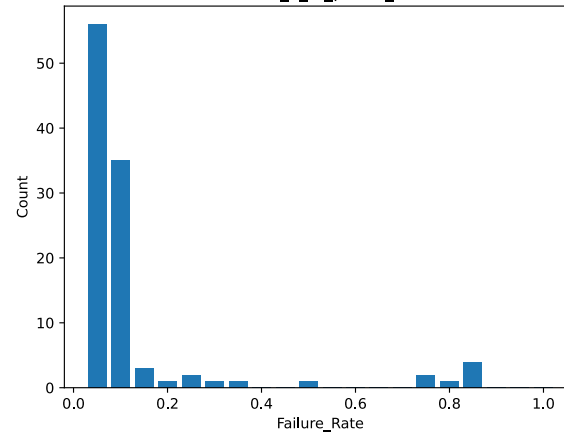
結果 | Type.3

抽出した部分木をもつDockerfileがビルドに失敗する確率

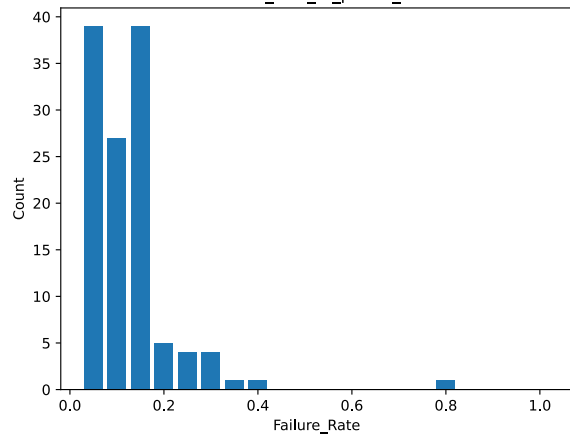
Code_1_of_phase_4



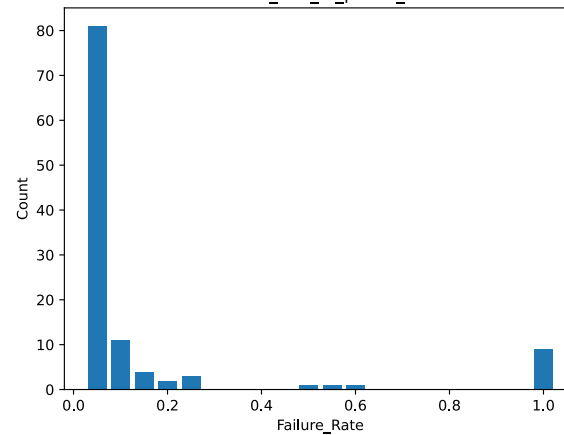
Code_2_of_phase_4



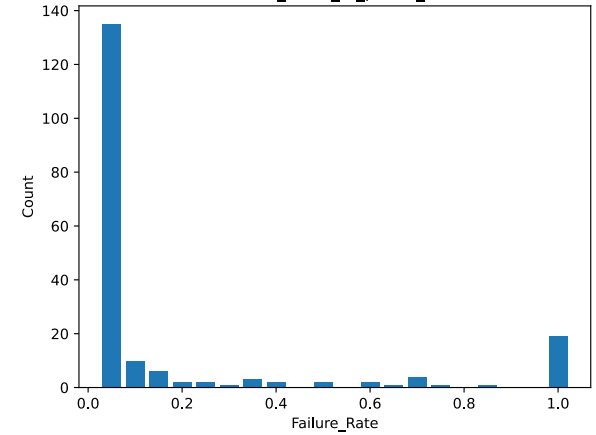
Code_100_of_phase_4



Code_127_of_phase_4



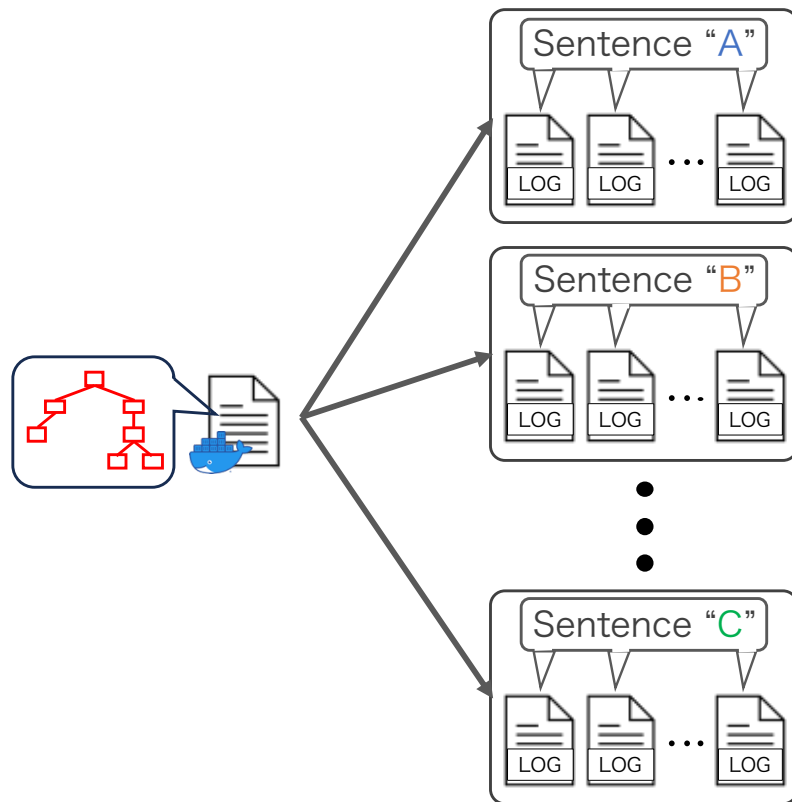
Code_None_of_phase_4



頻出部分木とエラーログの紐付け

ビルドエラーを誘発する依存関係が引き起こすビルドエラーの内容を理解

ー ビルドログ内のエラーメッセージを使用

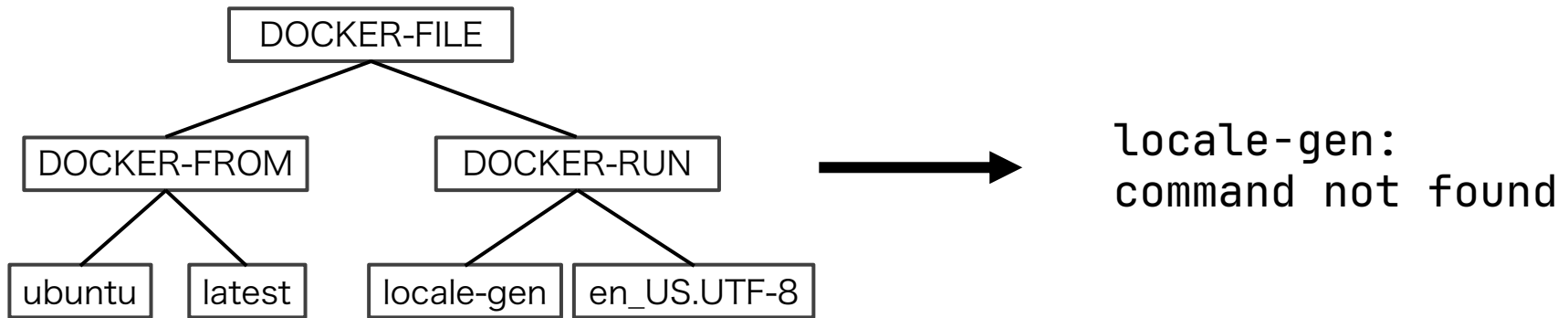


抽出できた依存関係

進行中

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

今後の予定

| ベースラインの検討中

- Dockerfileからビルドエラーに関連する依存関係を抽出するような研究は存在しない

| 抽出できた依存関係の確認

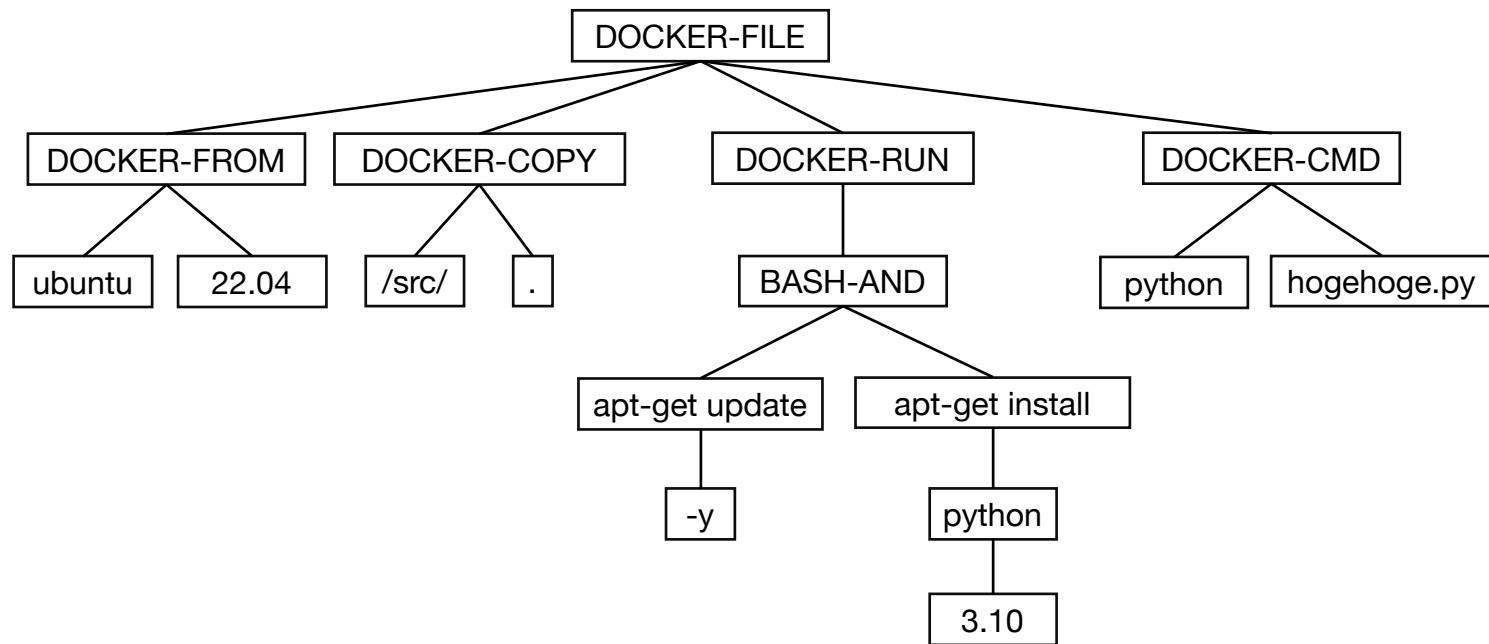
| 論文執筆

補 足 資 料

DockerfileをASTへ変換

| Type.1 : 最小粒度でASTを構築

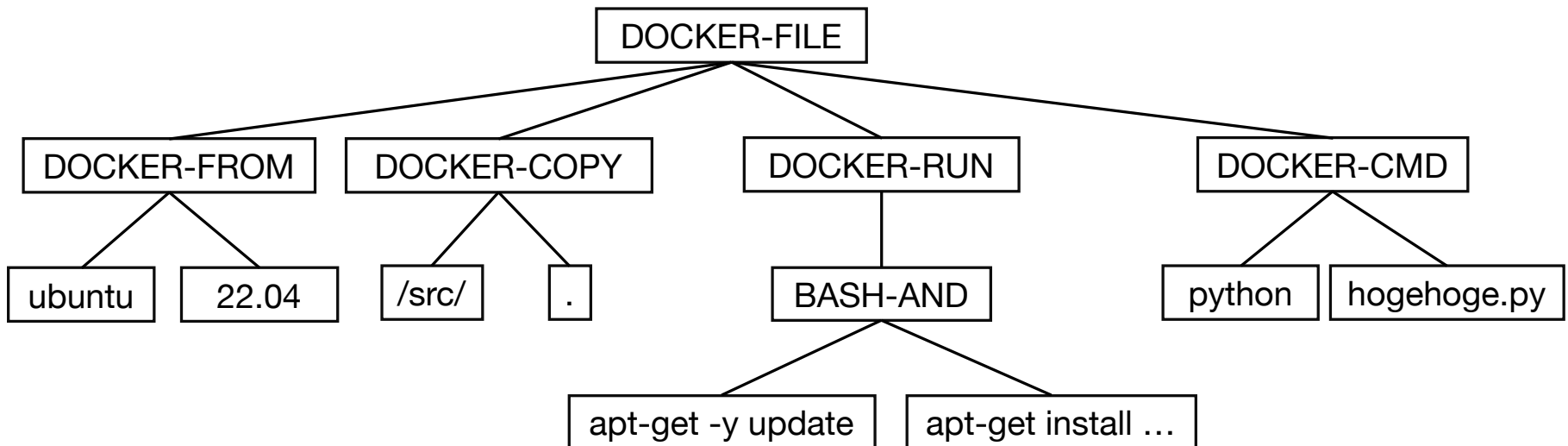
- 命令と引数をトークンに分割
- シェルスクリプトをパースし, ノードを追加



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

DockerfileをASTへ変換

- | Type.2 : 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の構文レベルの系列パターンを抽出可能



DockerfileをASTへ変換

| Type.3 : パッケージ, URL, パスなどの抽象化

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

