Dockerfileの
ビルドエラー自動修正に向けた
ビルドエラー予測

オープンソースソフトウェア工学研究室
坂本 廉也

背景

ソフトウェア開発競争の激化

- 企業間のビジネス競争に直結する
- スピーディに高品質なソフトウェアの開発が必要

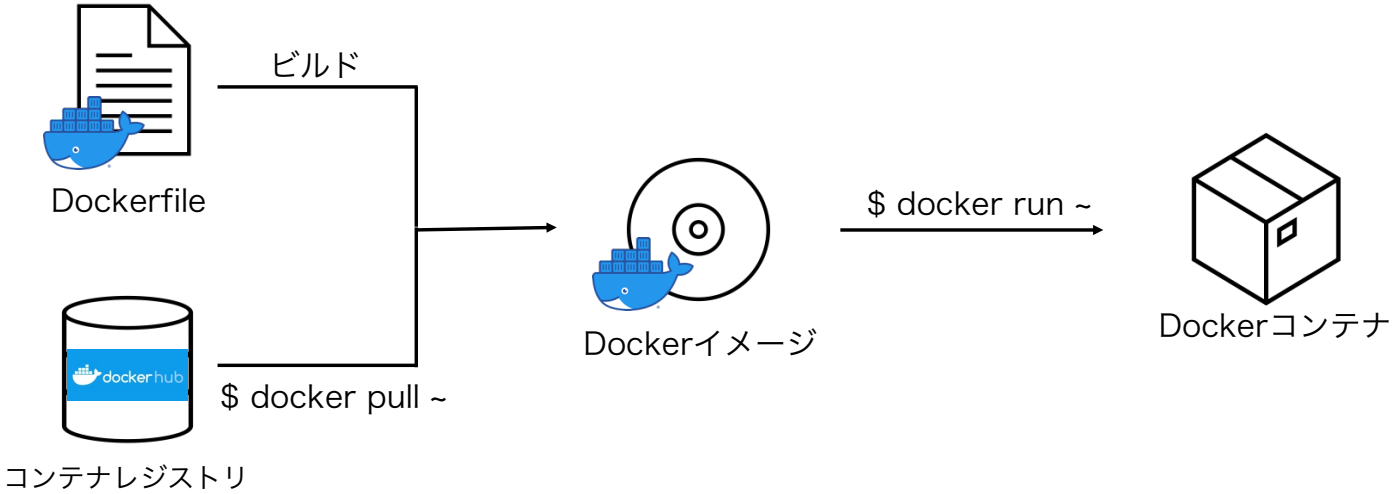
Dockerが積極的に利用される[1]

Docker[2]

- コンテナ型仮想化ソフトウェア
  - 独立した仮想アプリケーション環境（Dockerコンテナ）を動作
- メリット
  - デプロイの容易性　：本番環境への素早い展開
  - 環境の再現性　　：開発チームのメンバー間で環境を統一可能

[1] https://portworx.com/blog/2017-container-adoption-survey/
[2] https://www.docker.com/

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その実行環境を含むイメージファイル

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

```
FROM python:3-alpine

COPY [".", "/app/src/"]

WORKDIR /app/src/

RUN apk --update add ca-certificates \
    && update-ca-certificates \
    && rm -rf /var/cache/apk/*
```

既存イメージの機能を使用

⬇

ホストマシン上のファイルを
イメージへコピー

⬇

作業ディレクトリの指定

⬇

任意のシェルコマンドを使用

- Dockefrile内の命令を上から順に実行
  - 先に実行された命令が親として命令同士が親子関係を持つ[3]
  - 順番を考慮してDockerfileを記述する必要
- ただし，ビルドは失敗することがある（ビルドエラー）

研究動機

Dockerfileのビルドエラー

- 他の言語のビルドと比較してビルドエラー発生率が高い[4]
- 多くの修正時間が費やされる[5]
  - Docker特有の複雑な依存関係[6][7]

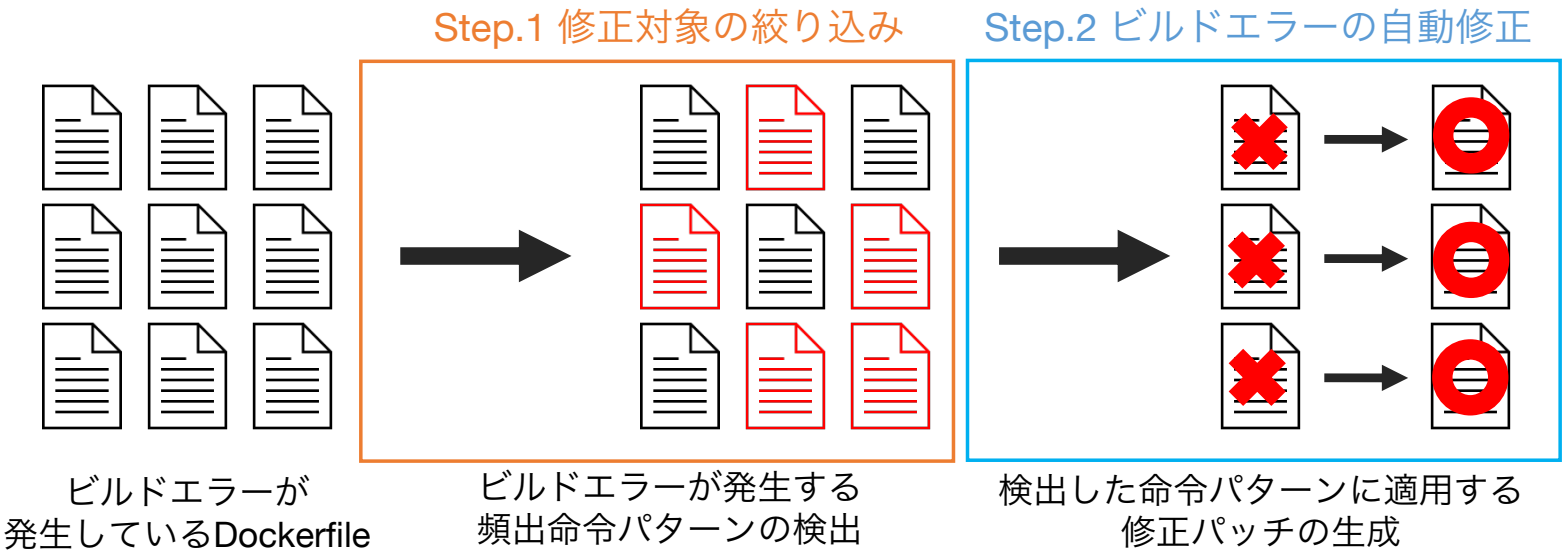
現代のスピーディなソフトウェア開発の障壁になる

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

[4] 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).
[5] 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).
[6] Chris Tomy, Tingmao Wang, Earl T. Barr, and Sergey Mechtaev. Modus: a Dat- alog Dialect for Building Container Images. In Proceedings of ESE/FSE ' 22, pp. 595–606 (2022).
[7]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).

自動修正までのながれ



1. 修正対象の絞り込み
   - 膨大なビルドエラーパターンへの修正パッチ生成は非現実的
     - あらゆるソフトウェアに関するエラーメッセージが存在
   - 頻繁に記述される命令のパターンを検出
2. ビルドエラーの自動修正

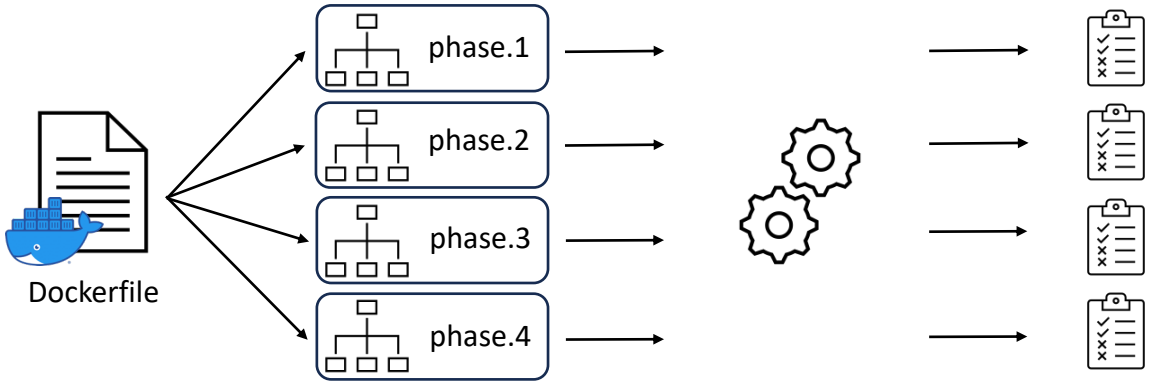
目的 & アプローチ

目的

Dockerfileのビルドエラー自動修正に向けたビルドエラーの予測

アプローチ

1. Dockerfile内の命令をAST（抽象構文木）に4段階で変換[8]
2. 順序木パターンマイニングによる頻出パターンの検出
3. 検出精度の評価



1. ASTに変換　2. 順序木パターンマイニング　3. 評価

Shipwright[9] | 先行研究

ルールベースのビルドエラー自動修正手法

1. エラーログをクラスタリング
2. クラスタリング結果から13の修正ルールを手作業で作成

→ 18.9%のビルドエラーを自動修正

Shipwrightの限界

- Dockefile固有の依存関係がビルドエラーを誘発[6][7]
  - エラーログのみの考慮では依存関係の解消は困難

Dockerfile全体の記述パターンのマイニングで
ビルドラーが発生する依存関係を抽出

[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 (2021).

パターンマイニングの効率化

パターンマイニング

- データ集合から重要な組合せを抽出
- 大規模データから効率的にマイニングするアルゴリズムが存在
  - Apriori[10], Prefix-Span [11], BIDE [12] など

本研究の大規模データセットではメモリ使用量が膨大

マシンリソース節約の工夫

- DockerfileをASTへ変換してパターンマイニングを効率化
- 命令や引数の抽象化
  - パスや固有名詞，URLやパッケージ名の抽象化 など

データセット

- Shipwrightのデータセットをフィルタリング
  - スター数10以上，単一のDockerfileを持つGitHubリポジトリ
  - 20,526件のDockerfile

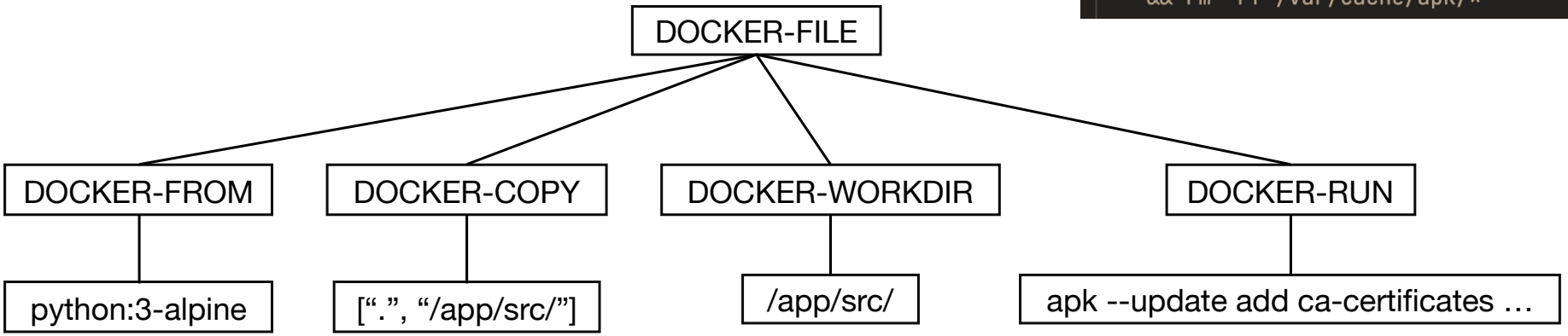
1. 破損しているリポジトリ
2. 現在は存在していないリポジトリ
3. マルチステージビルドを用いたDockerfileを持つリポジトリ

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	4,555件	27.0%

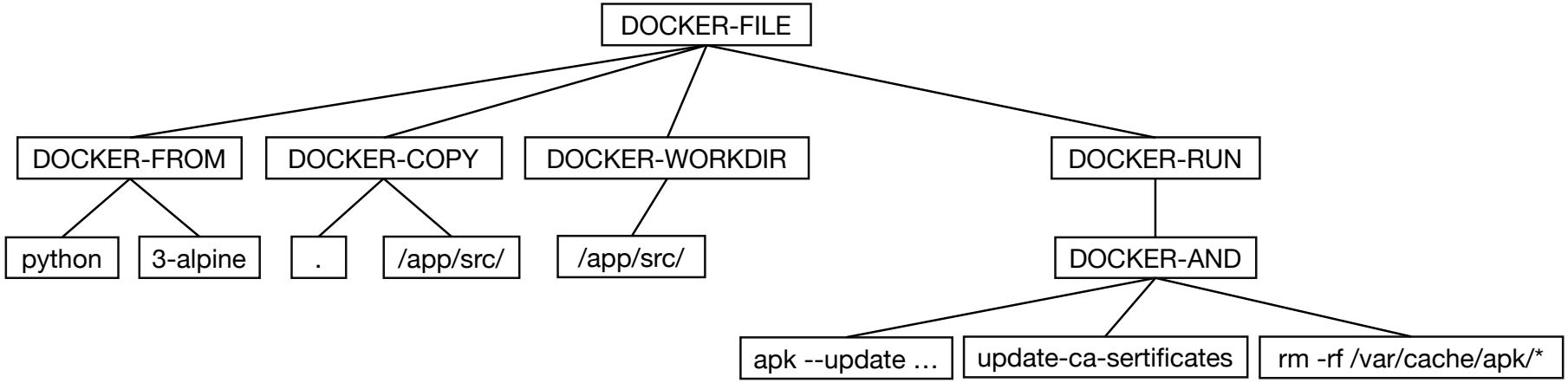
ASTへの変換

```
FROM python:3-alpine

COPY [".", "/app/src/"]

WORKDIR /app/src/

RUN apk --update add ca-certificates \
    && update-ca-certificates \
    && rm -rf /var/cache/apk/*
```

Phase.1：命令ごとにノードへ格納



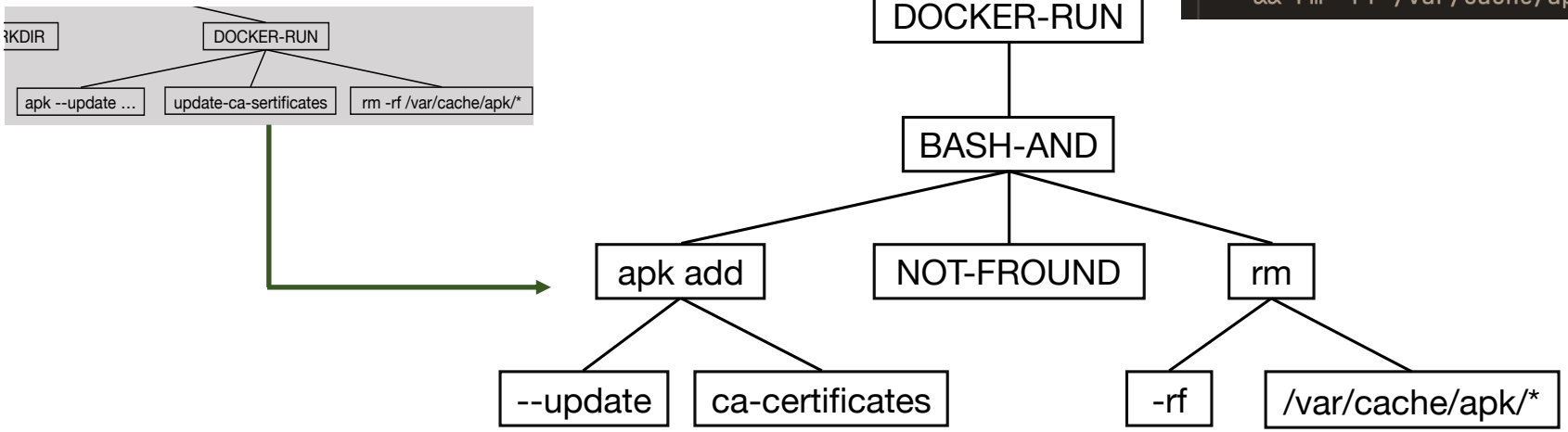
Phase.2：構文的に命令引数を分割



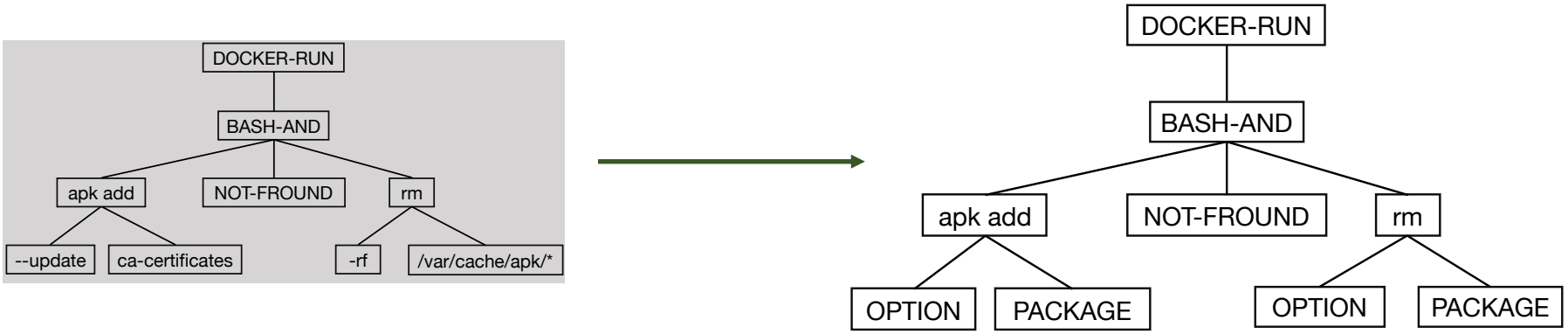
ASTへ変換

```
FROM python:3-alpine

COPY [".", "/app/src/"]

WORKDIR /app/src/

RUN apk --update add ca-certificates \
    && update-ca-certificates \
    && rm -rf /var/cache/apk/*
```

Phase.3：BASHコマンドのパース



Phase.4：パッケージ，URL，パスの抽象化



これからの予定

- 各フェーズの抽象度での予測精度の比較
  - データセットを4：1でマイニングデータとテストデータに分割
  - マイニングデータでビルドエラーが発生する命令を取得
  - テストデータでどの程度のカバレッジで拾えているか評価
  - 最小サポート値別でも比較（妥当な値を議論）
  - 各抽象度でどのようなルールが取得できているか議論