
系列パターンマイニングを用いた Dockerfileのビルドエラー予測

.....

オープンソースソフトウェア工学研究室
坂本 廉也

背景

| 社会の急激な情報化

- ソフトウェア開発競争が激化
- 開発現場はスピーディかつ柔軟な開発方式へ移行

スピーディかつ柔軟な開発方式と相性がいいDockerが普及

| Docker^[1]

- 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
- 多くの企業が利用^[2]

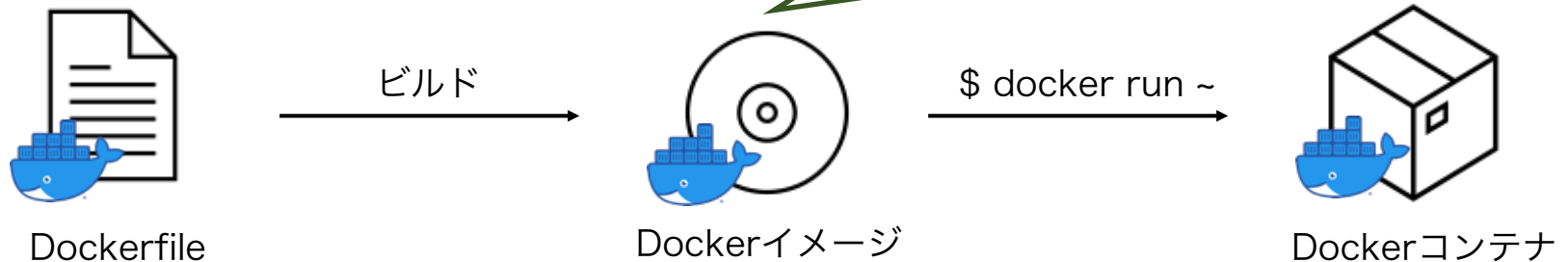
[1] <https://www.docker.com/>

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

コンテナ作成のながれ

Dockerイメージ（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^[3]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

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

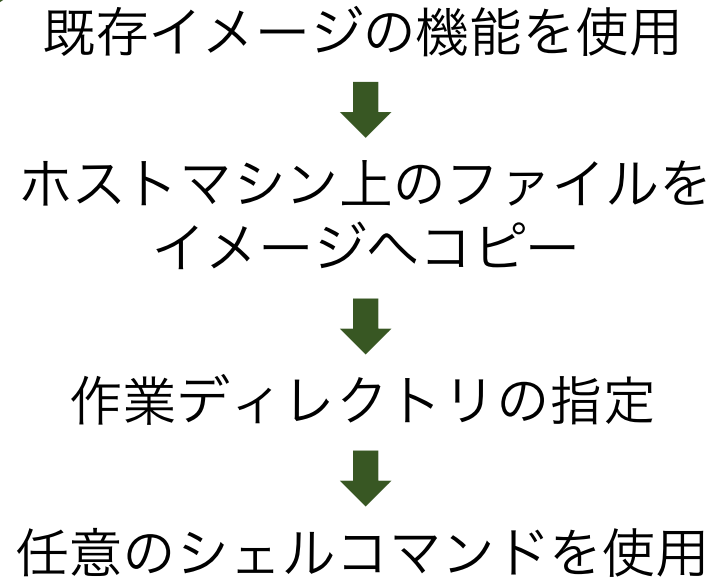
Dockerfileのビルド

```
FROM python:3-alpine

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

WORKDIR /app/src/

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



| Dockerfile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
- 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).

Shipwright^[8] | 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

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

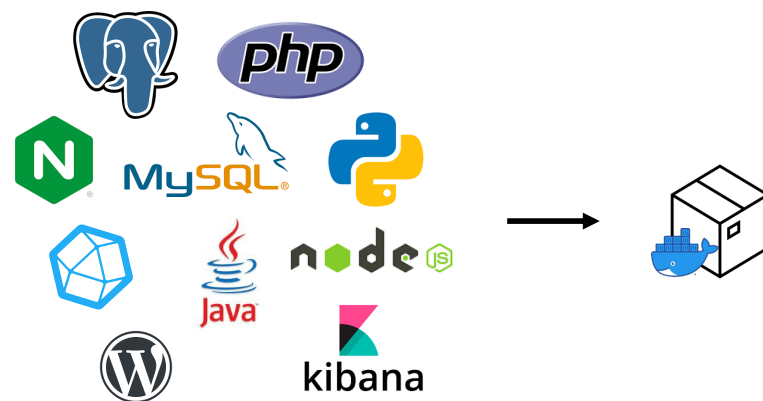
ビルドエラーを誘発する依存関係（命令の組み合わせ）を
考慮することで修正率向上を目指す

[8] 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).

目的

| Dockerfileには開発環境構築に必要な様々な命令が記述

- 命令の組合せは膨大
- 対応するエラーの種類も膨大



| 頻出な依存関係を抽出することで効率的に自動修正

1. シーケンシャル (系列) パターンマイニングで依存関係の抽出
2. 抽出した依存関係を自動修正

目的

ビルドエラー自動修正対象の抽出のためのビルドエラー予測

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

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%

系列パターンマイニング^[9]

| 大量データから順序を考慮した頻出な組合せを抽出

- ビルドエラーを誘発するDockerfileの依存関係を抽出
 - Min_sup（最小サポート値）で頻出度を指定

| PrefixSpan^[10]

- 系列パターンマイニングを行うための代表的なアルゴリズム
 - 1アイテムだけの頻度を計算し、頻出アイテムの射影DBを求める
 - 可能な拡張パターン全てが頻出かどうかを計算し射影DBを求める
- Dockerfileの各行を1つのアイテムとしてPrefixSpanを適用

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

[10] J.Pei et al.: "PrefixSpan,: mining sequential patterns efficiently by prefix-projected pattern growth," Proceedings 17th ICDE' 01,2001, pp. 215-224

マシンリソース的制約

| データセット内のユニークアイテム数は66,885個

– 全アイテム数は122,439個

| 66,885個の頻出回数をカウントし、射影DBを求めるためには膨大なメモリ領域が必要

– プロセスが強制終了

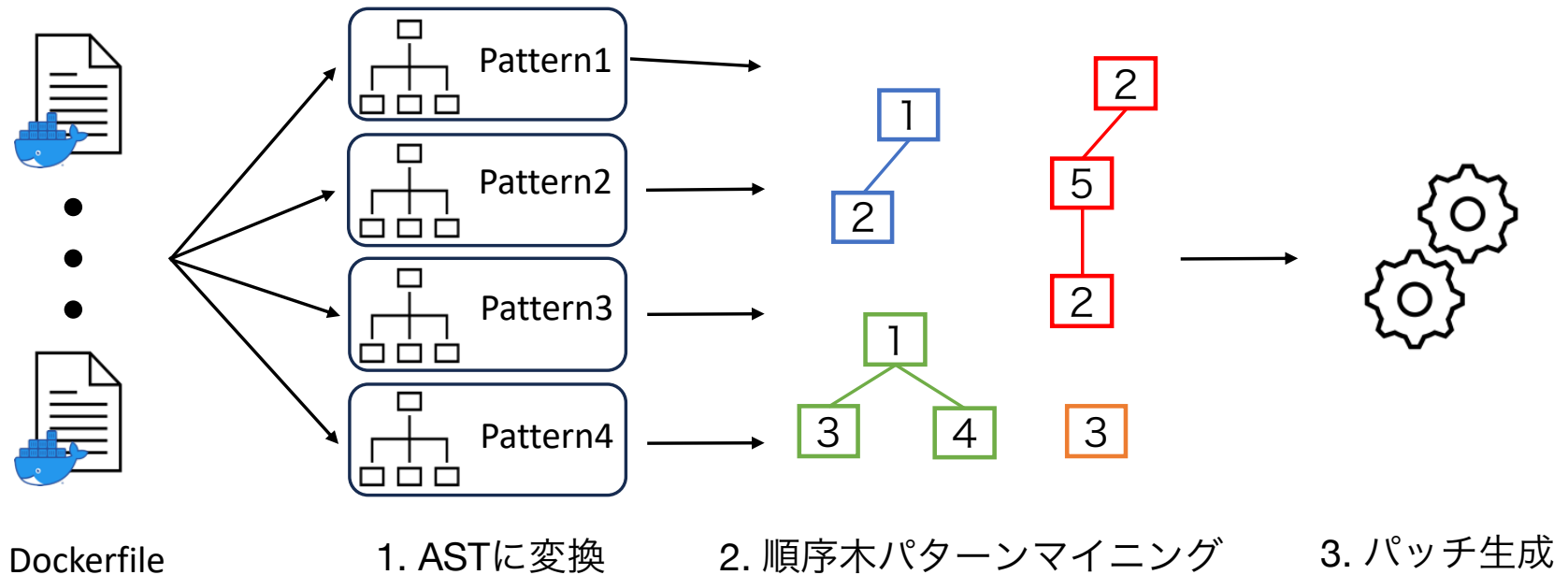
– メモリ節約のための工夫が必要

方針

① 飽和系列パターンマイニングアルゴリズムの使用 ※補足資料

② データの工夫（抽象構文木への変換）

データの工夫



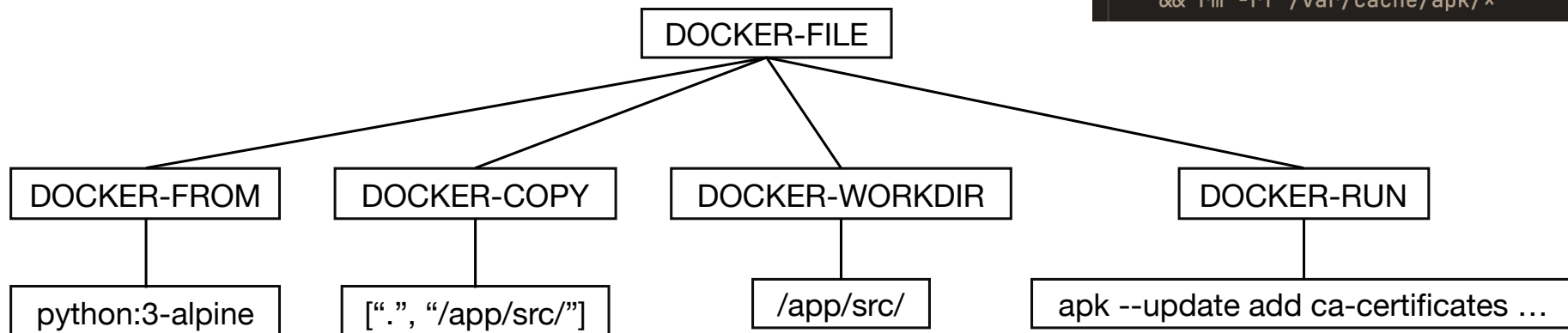
| Step1 : ビルドエラーが発生するDockerfileを4種類のAST (抽象構文木) に変換

| Step2 : ASTに対してパターンマイニングを適用

| Step3 : 抽出した依存関係への修正パッチを生成

ASTへの変換

| Pattern1 : 命令ごとにノードへ格納



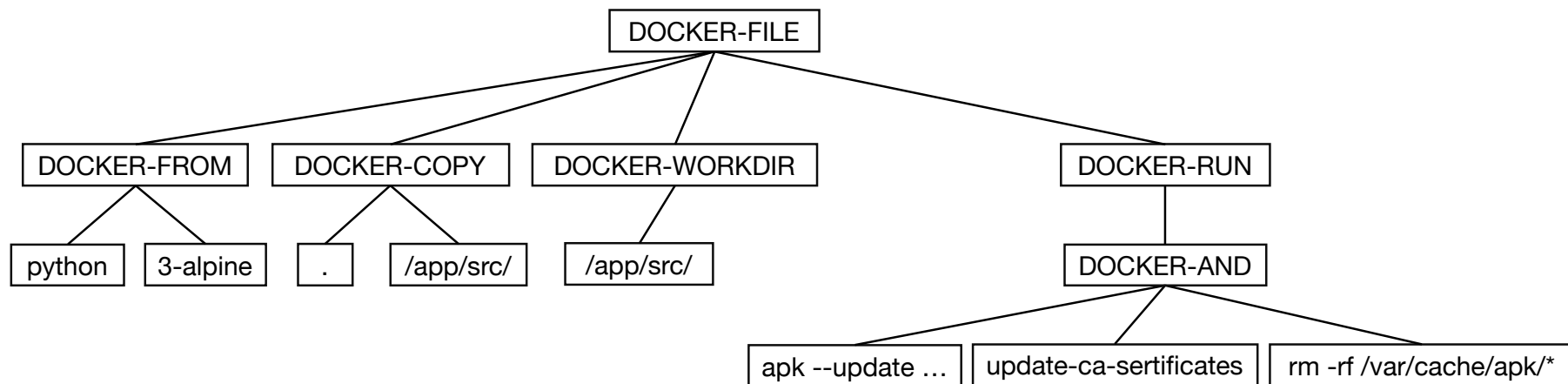
```
FROM python:3-alpine

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

WORKDIR /app/src/

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

| Pattern2 : 構文的に命令引数を分割



ASTへの変換

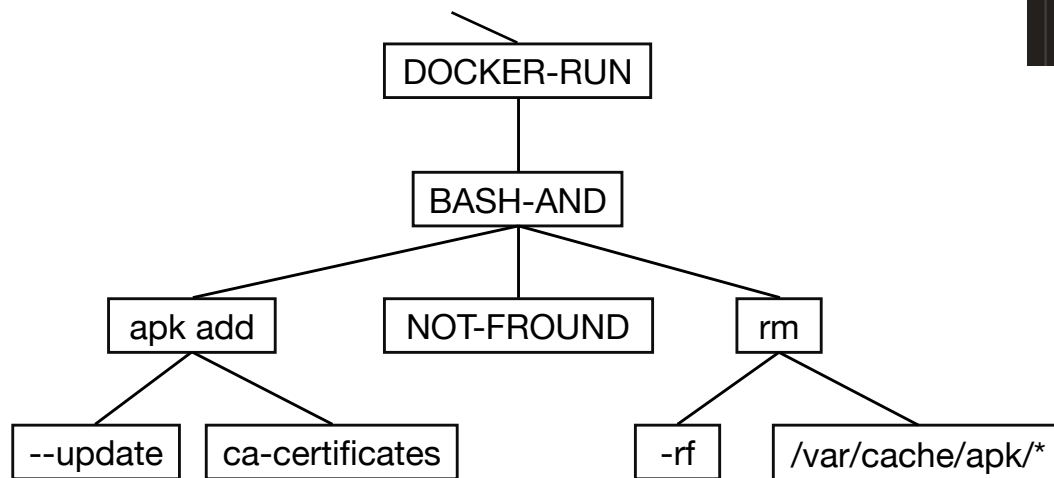
```
FROM python:3-alpine
```

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

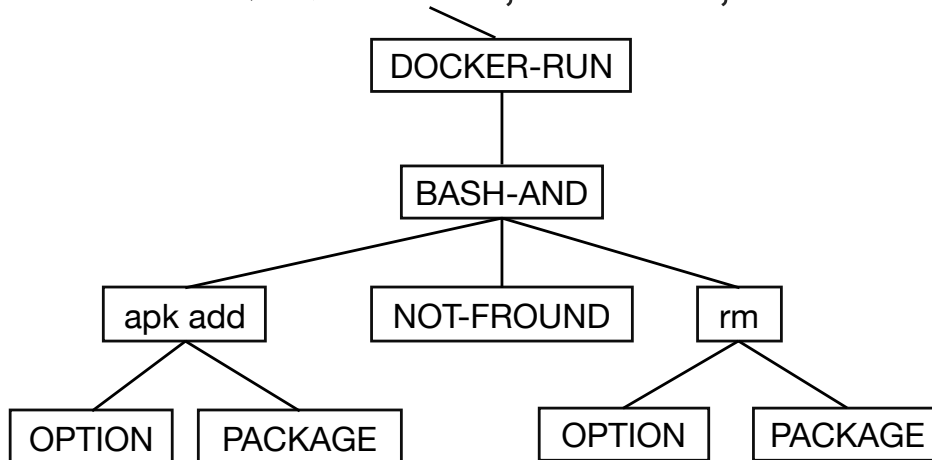
```
WORKDIR /app/src/
```

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

Pattern3 : BASHコマンドのパース



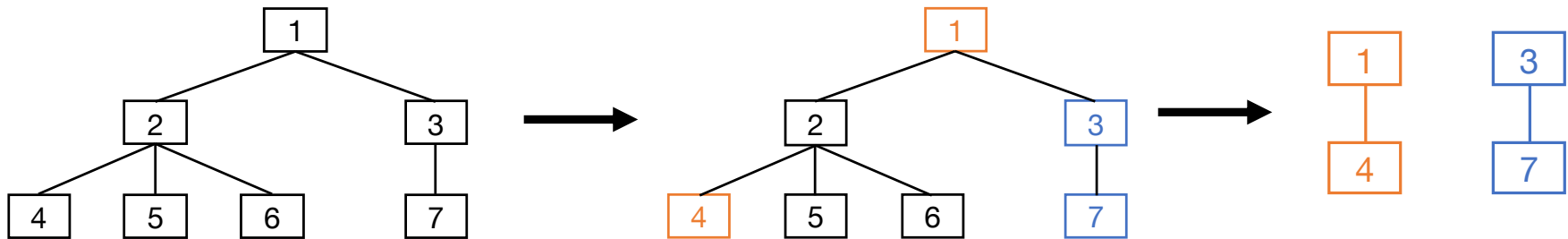
Pattern4 : パッケージ, URL, パスの抽象化



順序木パターンマイニング

| 順序木に対してパターンマイニングを実行

- TreeMiner^[11]を使用



| ASTへの適用

- ASTからビルドエラーを誘発する依存関係を抽出
- ビルドに成功するDockerfileにも含まれる頻出パターンを除外

[11] M.J Zaki.:Treeminer: An efficient algorithm for mining embedded ordered frequent trees. In Advanced Methods for Knowledge Discovery from Complex Data, pp. 123–151. Springer, 2005.

今後の予定

| TreeMinerを適用 / 精度評価

－ 定量的評価

1. 各パターンのASTをそれぞれ4 : 1 (分析用 : テスト用) に分割
2. 分析用データにTreeMinerを適用
3. ビルドに成功するDockerfileにも含まれる依存関係を除外
4. 抽出した依存関係がテストデータにどの程度含まれるか検証
 - Min_supの値によっても評価
5. 10回繰り返し, 平均値をとる
 - データの偏りへ対応するため

－ 定性的評価

- 目視による頻出パターンの理解

まとめ

| 目的

- ビルドエラー自動修正対象の抽出のためのビルドエラー予測

| アプローチ

- 系列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
 - 飽和系列パターンマイニング
 - データの工夫

| 今後の予定

- TreeMinerの実行, 評価

対外発表状況

| 発表

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, 6月, pp.61-70

| 受賞

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, 6月, pp.61-70 (論文奨励賞)

補 足 資 料

方針① | 前処理

| Dockerfileの命令を構文的に分割

- RUN命令内の“&&演算子”を分割し，複数のRUN命令へ変換
- 改行を用いて記述された命令を複数命令へ分割
 - “\”で同命令内の命令を複数行に記述可能
- 空白行やコメント行を削除

```
FROM python:3-alpine

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

WORKDIR /app/src/

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

処理前のDockerfile



```
FROM python:3-alpine

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

WORKDIR /app/src/

RUN apk --update add ca-certificates

RUN update-ca-certificates

RUN rm -rf /var/cache/apk/*
```

前処理済みDockerfile

方針① | BIDE^[12]

| 飽和系列パターンマイニングアルゴリズム

- 頻出する飽和系列パターンを効率的に抽出
- 飽和パターンのみを保持するため大幅なメモリ節約が可能

| ビルドエラーが発生する前処理済みDockerfileへの適用

- 1行を1アイテムとしてBIDEを適用
- 抽出したパターンからビルドに成功するDockerfileにも含まれるパターンを除外

[12] J. Wang and J. Han: "BIDE: efficient mining of frequent closed sequences," Proceedings. 20th International Conference on Data Engineering, 2004, pp. 79-90,

方針① | BIDE

| 結果

- 最小サポート数を上げるにつれて頻出パターン数が大きく減少

Min_sup	抽出パターン数	ビルド成功パターン除外後パターン数
10	7452	5856
20	1831	1409
50	291	211
100	73	58