Go 1.13 Release Party, Aug 22 2019

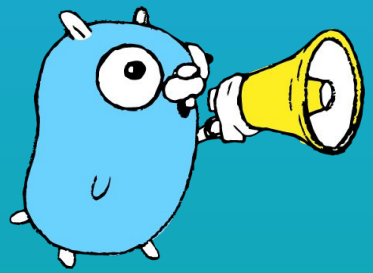
Go Modules

(Escape from GOPATH)

Geon Kim

GDG Golang KR

Speaker



About Go Modules

History

Makefiles, *goinstall*, and *go get*

Nov 2009:
Go 초기 릴리즈에선 Makefiles를 이용하여 빌드했다.
하지만, Makefiles는 다른 사람들과 코드를 공유하기에는 부족한 점이 많았다.

Feb 2010:
Go Team은 SCR에서 별다른 설정 없이 패키지를 다운 받을 수 있는 *goinstall*을 제안했다.
*goinstall*은 Import Path Convention을 소개했고, *goinstall*은 Makefiles를 대체하게 됐다.

Dec 2011:
Go Team은 흩어져있던 툴들을 하나의 커맨드로 묶고, 대체할 수 있는*go* 커맨드를 발표했다.
하지만, go 커맨드에는 패키지 버저닝에 대한 개념이 없었고, 이는 적어도 두 가지의 큰 결점을 불러오게 된다.

Versioning and API Stability

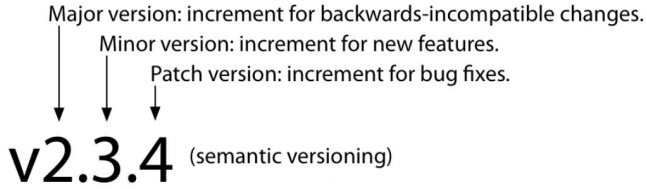
"주어진 업데이트가 어떠한 변경을 일으킬지 예측할 수 없다."

Nov 2013:
Go 1.2 FAQ에 패키지 버저닝에 대한 기본적인 조언이 추가됐다.

"공용으로 사용될 패키지는 하위 호환성을 지키기 위해 노력해야하고,
하위 호환성을 깨야하는 경우 새로운 import path와 함께 새로운 패키지를 만들어라"

Mar 2014:
Gustavo Niemeyer가 *gopkg.in*을 만들었다. *gopkg.in*은 단일 Git Repository의 각각의 다른 커밋 혹은 브랜치를 참조할 수 있는 Github Redirector로 *gopkg.in/yaml.v1*과 *gopkg.in/yaml.v2*와 같이 사용할 수 있다.

Aug 2015:
Dave Cheney에 의해 Go에서의 SEMVER(Sementic Versioning) 채택이 제안됐다.



Vendoring and Reproducible Builds

"재생산 가능한 빌드를 보장하거나 표현할 수 있는 방법이 없다."

Nov 2013:
Go 1.2 FAQ에 재생산 가능한 빌드를 위한 기본적인 조언이 추가됐다.

*"외부 패키지를 사용하고 있고, 그 패키지가 예상치 못한 방향으로 변경될까 걱정된다면,
가장 쉬운 해결책을 로컬 복제본을 만드는 것이다."*

Mar 2012:
Keith Rarick이 *Goven* 개발을 시작한다. *Goven*은 의존성들을 자신의 특정 Repository에 복제하고, 코드가 복제본을 가리킬 수 있도록 Import Path를 수정한다.

Nov 2013:
Keith Rarick은 보다 발전된 벤더링 도구인 *godep*을 발표한다. 현재의 Go Vendoring과 같은 방법으로 재생산 가능한 빌드를 가능하게 한다. *godep*은 Go Toolchain의 지원 없이 GOPATH를 수정하는 방법으로, 벤더링을 구현했다.

Vendoring and Reproducible Builds

Sep 2014:
Keith Rarick은 godep과 같은 Convention을 통해, Go가 외부 패키지에 대해 지원을 해줄 것을 제안한다.

Apr 2015:
Dave Cheney가 프로젝트 기반의 빌드 툴인 *gb*를 발표한다. Gb는 Import Path 재작성을 피하기 위해, GOPATH의 특정 Directory에 코드를 저장한다. 하지만, 이러한 방식은 많은 개발자들의 워크플로우와 잘 맞지 않았다.

2015 Spring:
Jason Buberel은 패키지 관리 도구들의 춘추전국시대의 막을 내리기 위해, Go 패키지 관리 랜드스케이프에 대한 설문을 진행한다. 이를 통해, Go Team은 go 커맨드가 Import Path의 재작성 없는 벤더링 지원을 필요로 한다는 것을 알게 되었다.

Aug 2015:
Go 1.5에서 Keith의 제안을 받아들여 벤더링을 시험적으로 도입했고, Go 1.6에서 이는 기본적으로 지원되게 된다.

An Official Package Management Experiment

Gophercon 2016:
패키지 관리에 관심이 있는 고퍼들이 모여 Go 패키지 관리를 위한 토론을 가졌고, 기존에 존재하던 여러 도구를 통합하고, 대체할 수 있는 새로운 패키지 관리 도구 만들기 위한 그룹이 형성됐다. 그 결과로 dep이 만들어진다.

Aug 2017:
Go 1.9의 릴리즈와 함께 *dep*을 사용할 수 있게 됐다. *dep* 역시 직접적인 Toolchain의 지원 없이 구현됐으며, 꽤 강력하고 유연했다. 하지만, dep은 패키지 버저닝과의 최종적인 go 커맨드 통합에 대한 프로토타입은 아니었다.

Mar 2018:
Russ Cox에 의해 Go Toolchain의 패키지 버저닝 지원이 제안됐다.

A Proposal for Package Versioning in Go

The Import Compatibility Rule:
새 패키지와 이전 패키지가 같은 Import Path를 공유하고 있다면, 새로운 패키지는 하위 호환이 돼야 한다.

Minimal Version Selection:
Oldest Allowed Version 선택 방식을 채택한 모듈 선택 방식이다. Minimal Version Selection은 모듈이 최소한의 요구사항 리스트를 갖을 수 있게 한다.

Defining Go Modules:
모듈은 특정 Import Path Prefix를 공유하고 있는 패키지들의 집합이고, 버저닝의 한 단위이다. 만약 GIt을 통해 개발을 하고 있다면, 저장소에 Tag를 추가하여 모듈을 버저닝할 수 있다.

The go command:
go 커맨드가 모듈과 함께 사용할 수 있도록 변경된다. 또한, *go.mod* 파일이 워크스페이스의 루트를 나타내기 때문에 GOPATH를 더 이상 사용하지 않아도 된다.

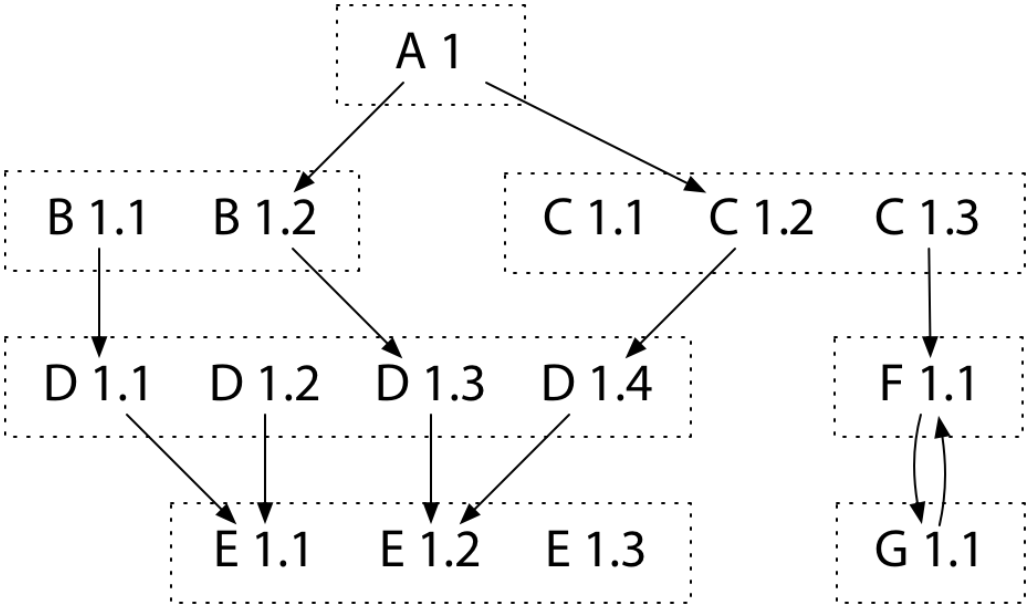
Minimal Version Selection

Newest Allowed Version
VS
Oldest Allowed Version

Operations

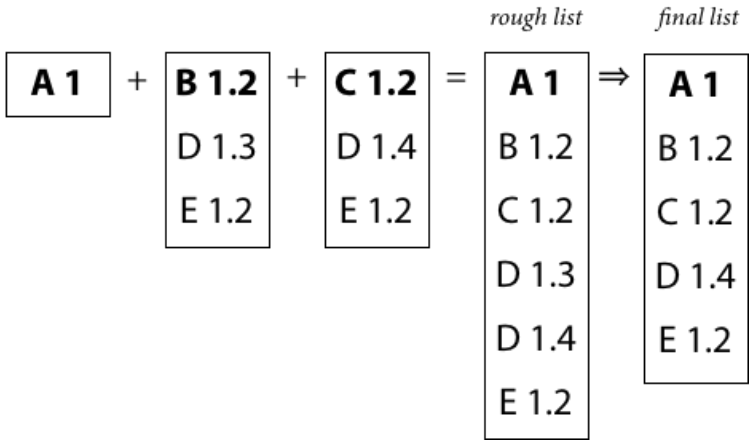
1. Construct Build List
2. Update All Modules
3. Upgrade One Module
4. Downgrade One Module

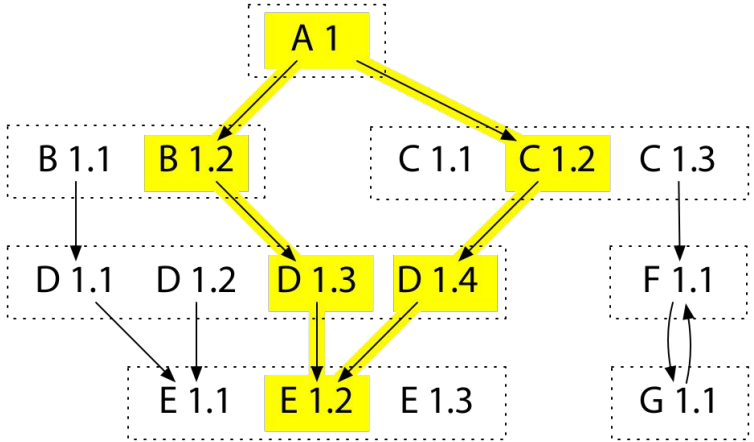
Requirement Graph



1. Construct build list

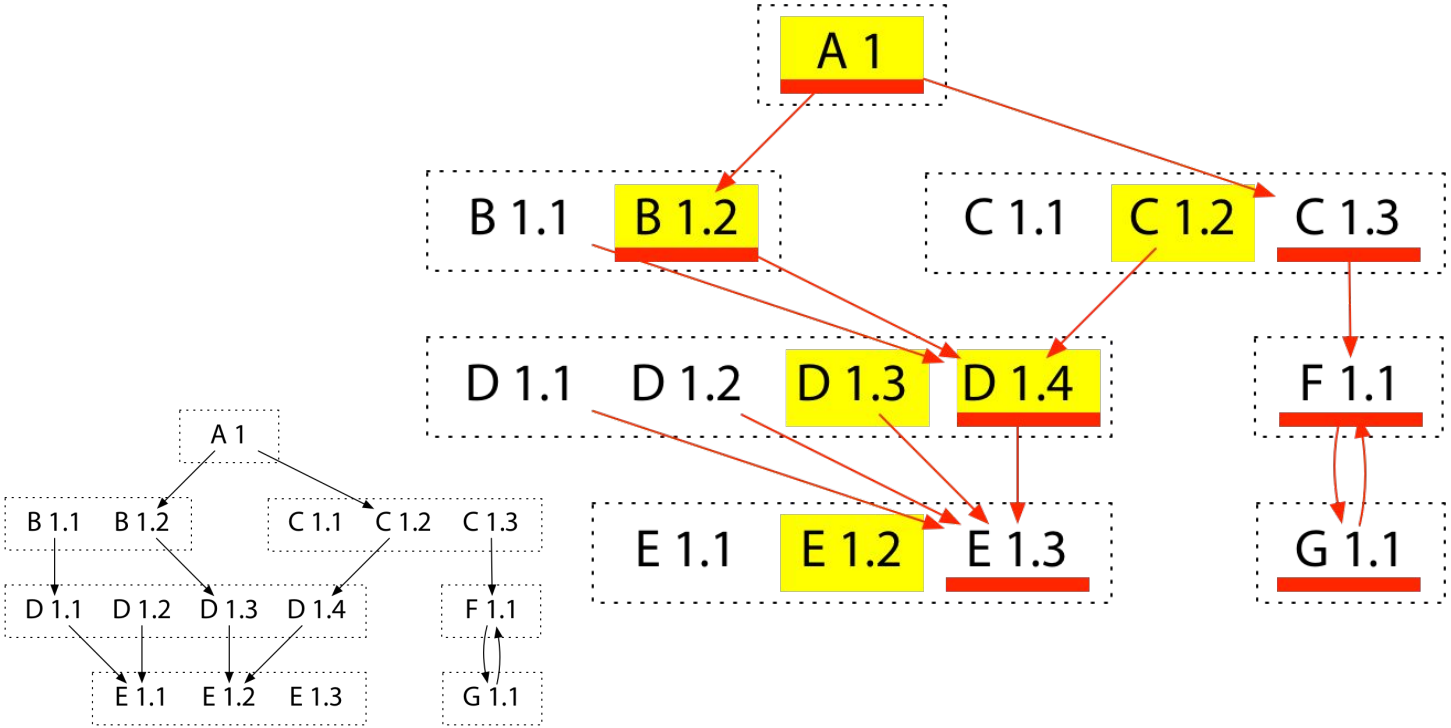
MVS


rough list
final list
A 1
+
B 1.2
D 1.3
E 1.2
+
C 1.2
D 1.4
E 1.2
=
A 1
B 1.2
C 1.2
D 1.3
D 1.4
E 1.2
⇒
A 1
B 1.2
C 1.2
D 1.4
E 1.2


A 1
B 1.1
B 1.2
C 1.1
C 1.2
C 1.3
D 1.1
D 1.2
D 1.3
D 1.4
F 1.1
E 1.1
E 1.2
E 1.3
G 1.1

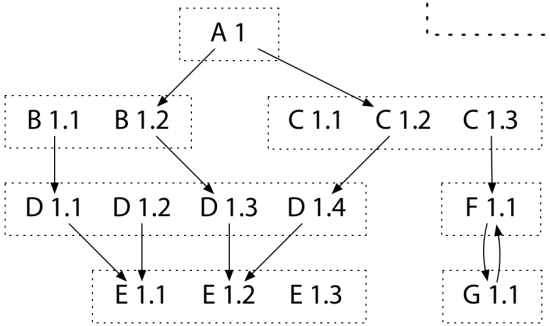
2. Upgrade All Modules

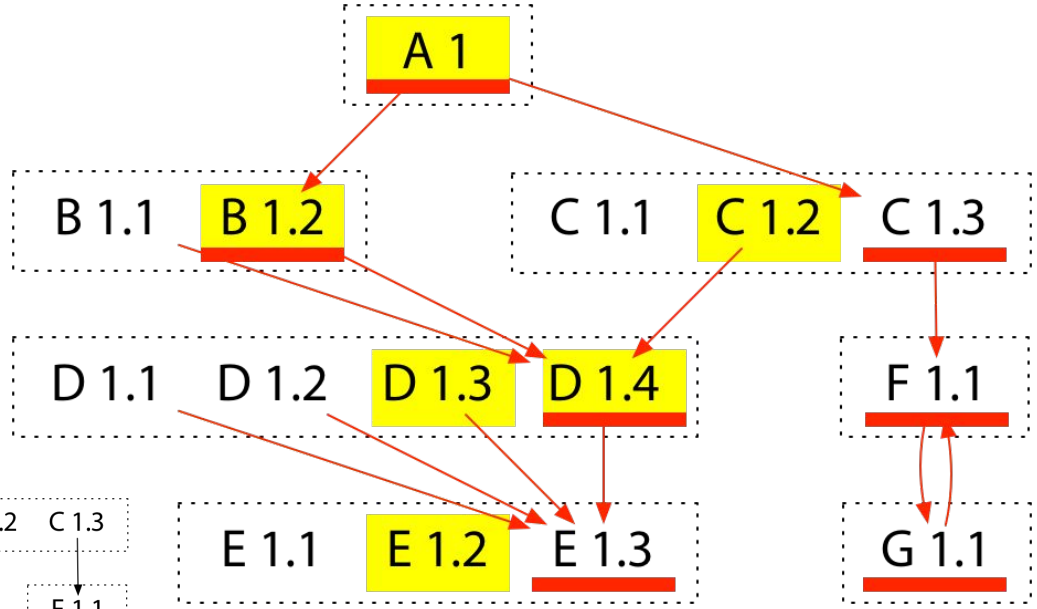
MVS


A 1
B 1.1
B 1.2
C 1.1
C 1.2
C 1.3
D 1.1
D 1.2
D 1.3
D 1.4
F 1.1
E 1.1
E 1.2
E 1.3
G 1.1
A 1
B 1.1
B 1.2
C 1.1
C 1.2
C 1.3
D 1.1
D 1.2
D 1.3
D 1.4
F 1.1
E 1.1
E 1.2
E 1.3
G 1.1

R. Compute a Minimal Requirement List

MVS

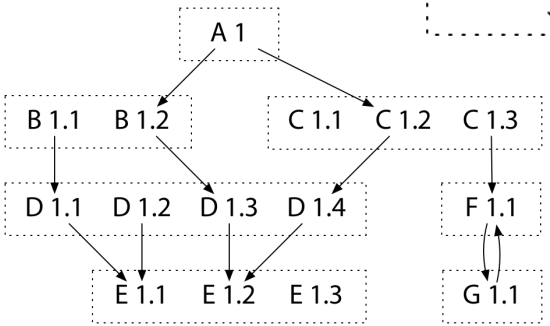


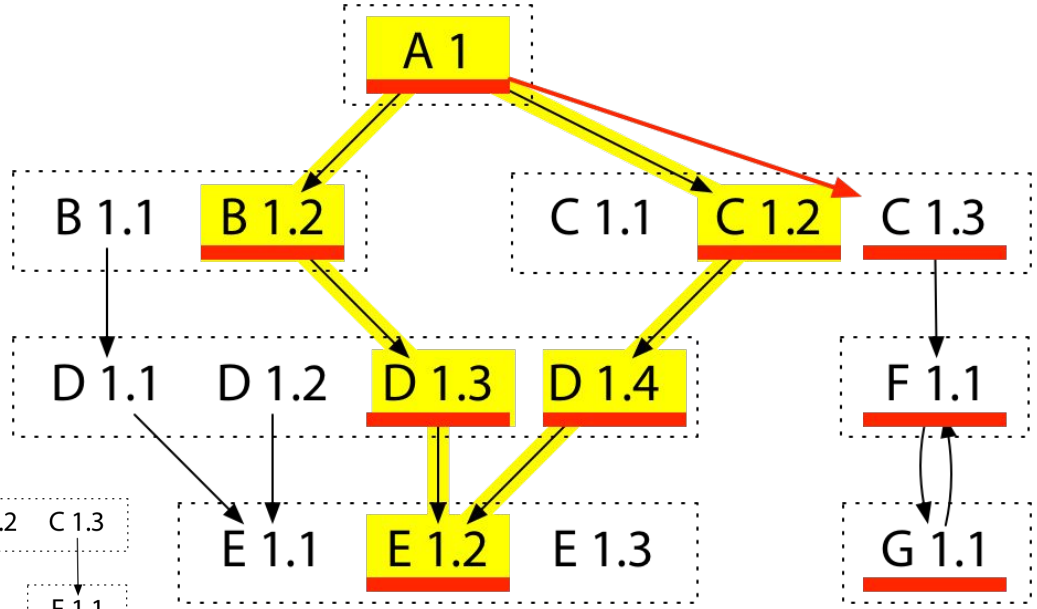


A 1 → C 1.3 → F 1.1 → G 1.1 → B 1.2 → D 1.4 → E 1.3

3. Upgrade One Module

MVS

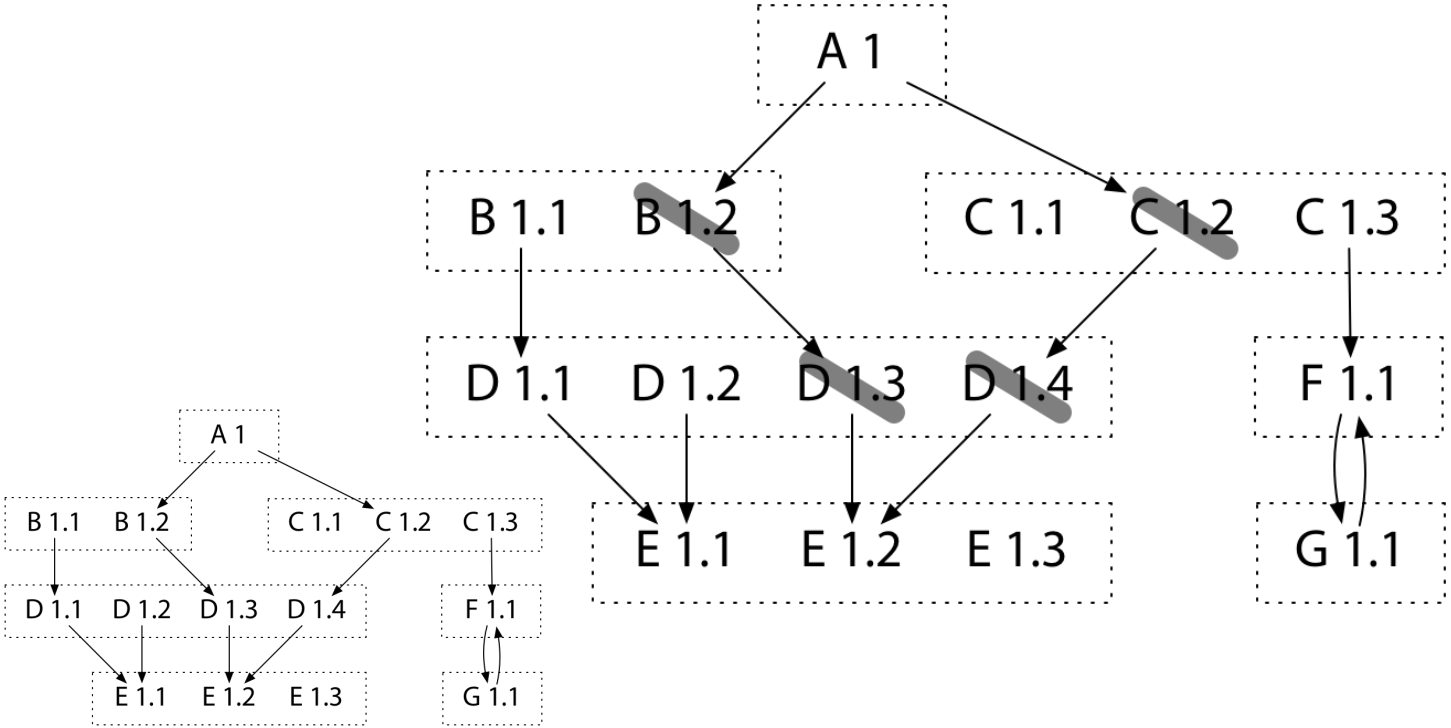




A 1 → C 1.3 → F 1.1 → G 1.1 → B 1.2 → D 1.4 → E 1.2

4. Downgrade One Module

MVS


A 1
B 1.1
B 1.2
C 1.1
C 1.2
C 1.3
D 1.1
D 1.2
D 1.3
D 1.4
F 1.1
E 1.1
E 1.2
E 1.3
G 1.1
A 1
B 1.1
B 1.2
C 1.1
C 1.2
C 1.3
D 1.1
D 1.2
D 1.3
D 1.4
F 1.1
E 1.1
E 1.2
E 1.3
G 1.1

Theory

MVS

*"충족 가능성 문제(**充足可能性問題, satisfiability problem, SAT**)는 어떠한 변수들로 이루어진 논리식이 주어졌을 때, 그 논리식이 참이 되는 변수값이 존재하는지를 찾는 문제이다."*

결정 문제: *"어떠한 문제에 대해 예-아니오로 답할 수 있는 문제"*

P: *"결정론적 튜링 머신이 다항 시간 안에 풀 수 있는 결정 문제"*

NP: *"비결정론적 튜링 머신이 다항 시간 안에 풀 수 있는 결정 문제"*

NP-hard: *"모든 NP문제를 다항 시간 안에 환원할 수 있는 문제"*

NP-complete: *"NP-hard 문제 중 NP에 속한 문제"*

SAT in Minimal Version Selection

$$(X) \land (\neg Y) \land (\neg X \lor Z)$$

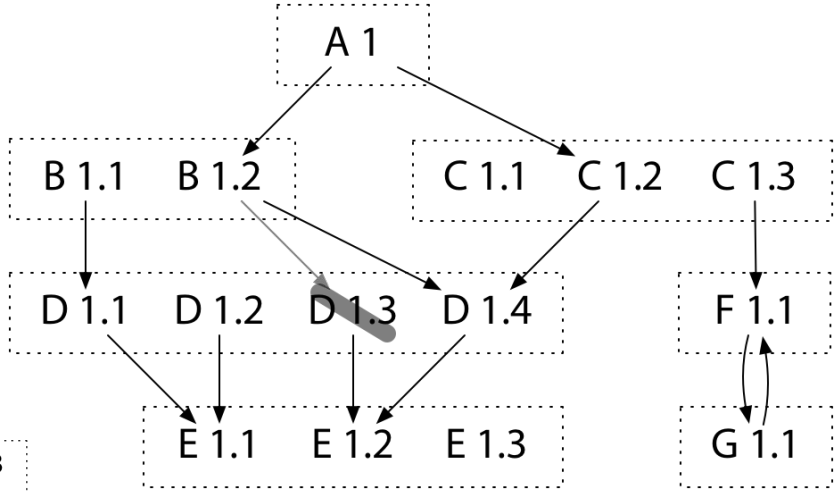
2-SAT: 모든 Clause가 2개 이하의 Literal로 이뤄진 SAT

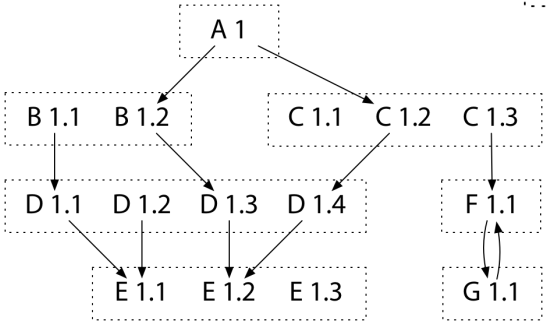
Horn-SAT: Horn Clause로 이뤄진 SAT

dual-Horn-SAT: dual-Horn Clause로 이뤄진 SAT

Exclude

MVS

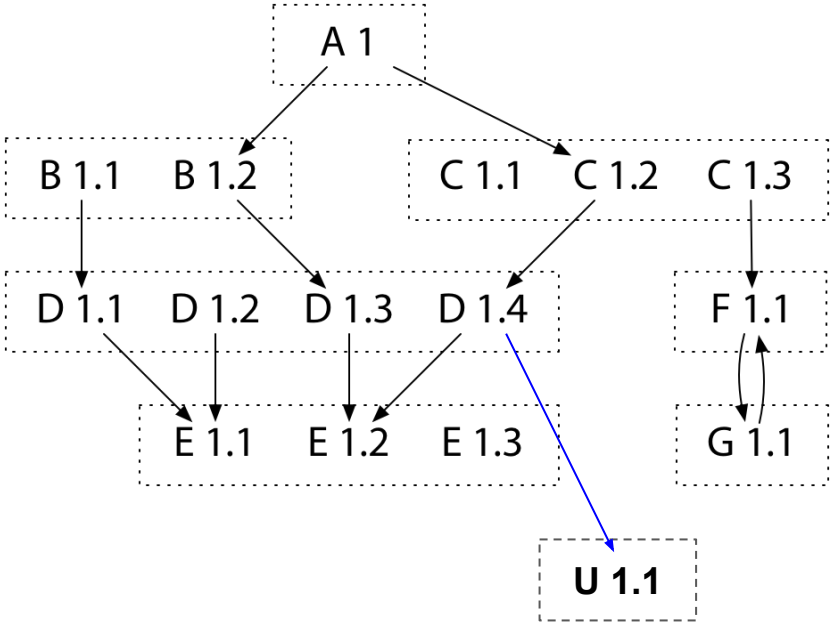




Do Not Use (¬ X ∨ ¬ Y)

Replace

MVS



Go Commands

Go Commands

go mod <command>

init: 현재 디렉터리에, 새로운 모듈을 생성한다.
edit: *go.mod* 파일을 수정한다.
download: 로컬캐시에 모듈을 다운로드 한다.
tidy: 누락된 모듈을 추가하고, 사용되지 않는 모듈을 제거한다.
verify: 로컬 캐시에 다운된 의존성들을 검증한다.
vendor: 의존성의 복사본이 저장된 **vendor** 디렉터리를 생성한다.
graph: 모듈 요구사항 그래프를 출력한다.
why: 특정 패키지나 모듈이 필요한 이유를 설명한다.

Go Commands

Module-aware go get

go get [-d] [-m] [-u] [-v] [-insecure] [build flags] [<path>@<module-query>]

Go Modules in 2019

Module Index

Module Authentication

Module Mirrors

Module Discovery

GOPROXY in Go 1.13

GOPROXY in Go 1.13

Environment variables

GOPROXY = https://proxy.golang.org,direct
GOSUMDB
GOPRIVATE
GONOPROXY
GONOSUMDB

GOPROXY in Go 1.13

```
go env -w GOPROXY=direct
go env -w GOSUMDB=off
```

GOPROXY in Go 1.13

Performance

Q&A