# mk semi lattice(logic)

## How to make a node-edge graph

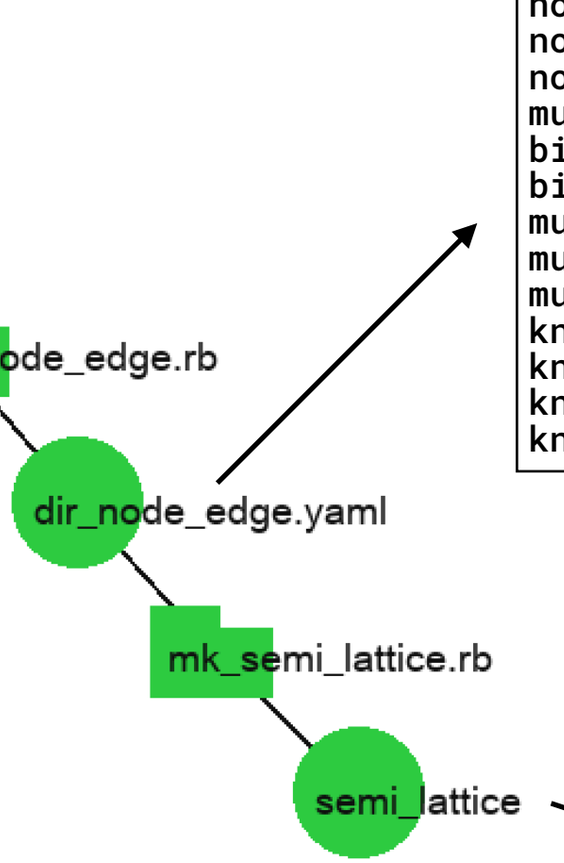
**Shigeto R. Nishitani, 2025/10/16**

# gem install mk_semi_lattice

# at w5_… > mk_semi_lattice

File: directories.yaml

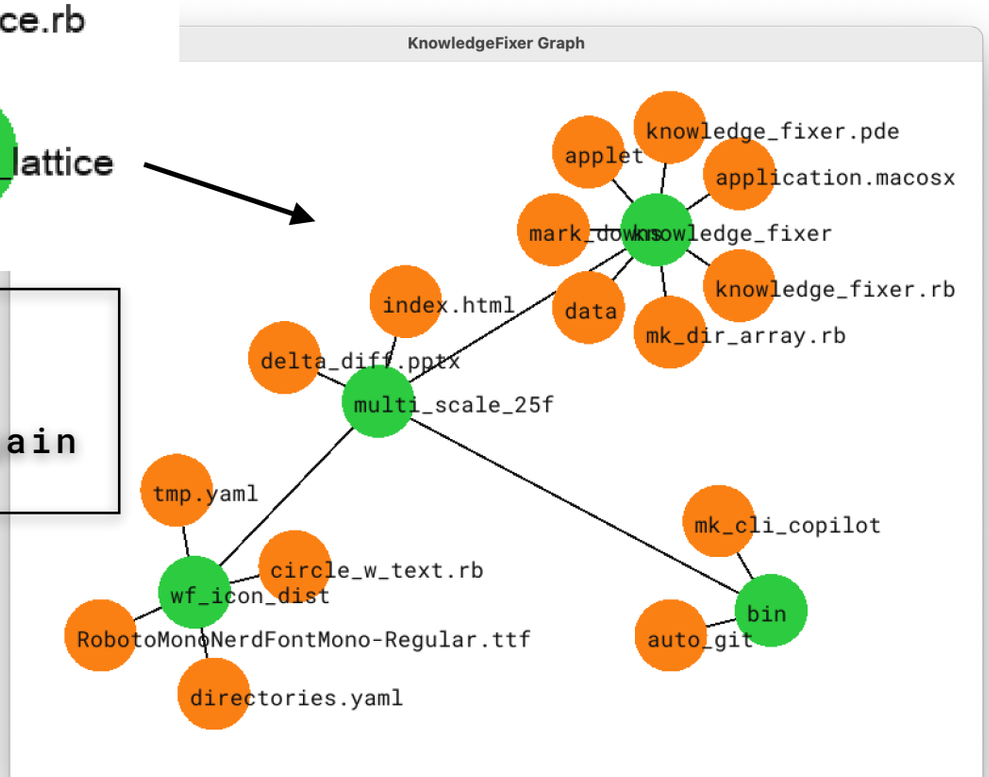
```yaml
---
- multi_scale_25f:
  - bin:
    - auto_git
    - mk_cli_copilot: []
  - delta_diff.pptx
  - index.html
  - knowledge_fixer:
    - applet: []
    - application.macosx: []
    - data: []
    - knowledge_fixer.pde
    - knowledge_fixer.rb
    - mark_downs: []
    - mk_dir_array.rb
  - wf_icon_dist:
    - RobotoMonoNerdFontMono-Regular.ttf
    - circle_w_text.rb
    - directories.yaml
    - tmp.yaml
```

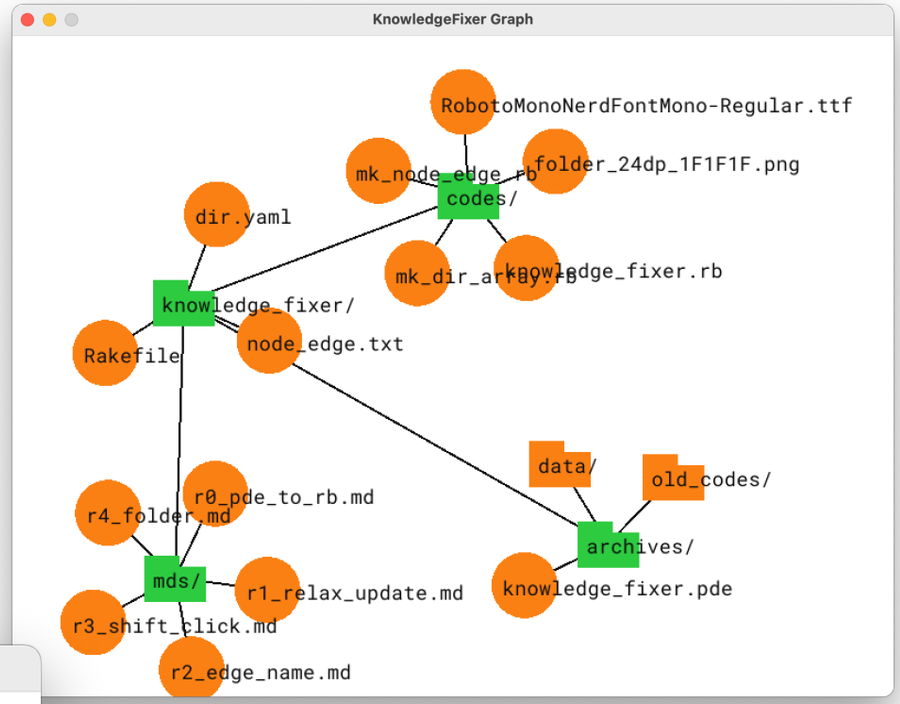


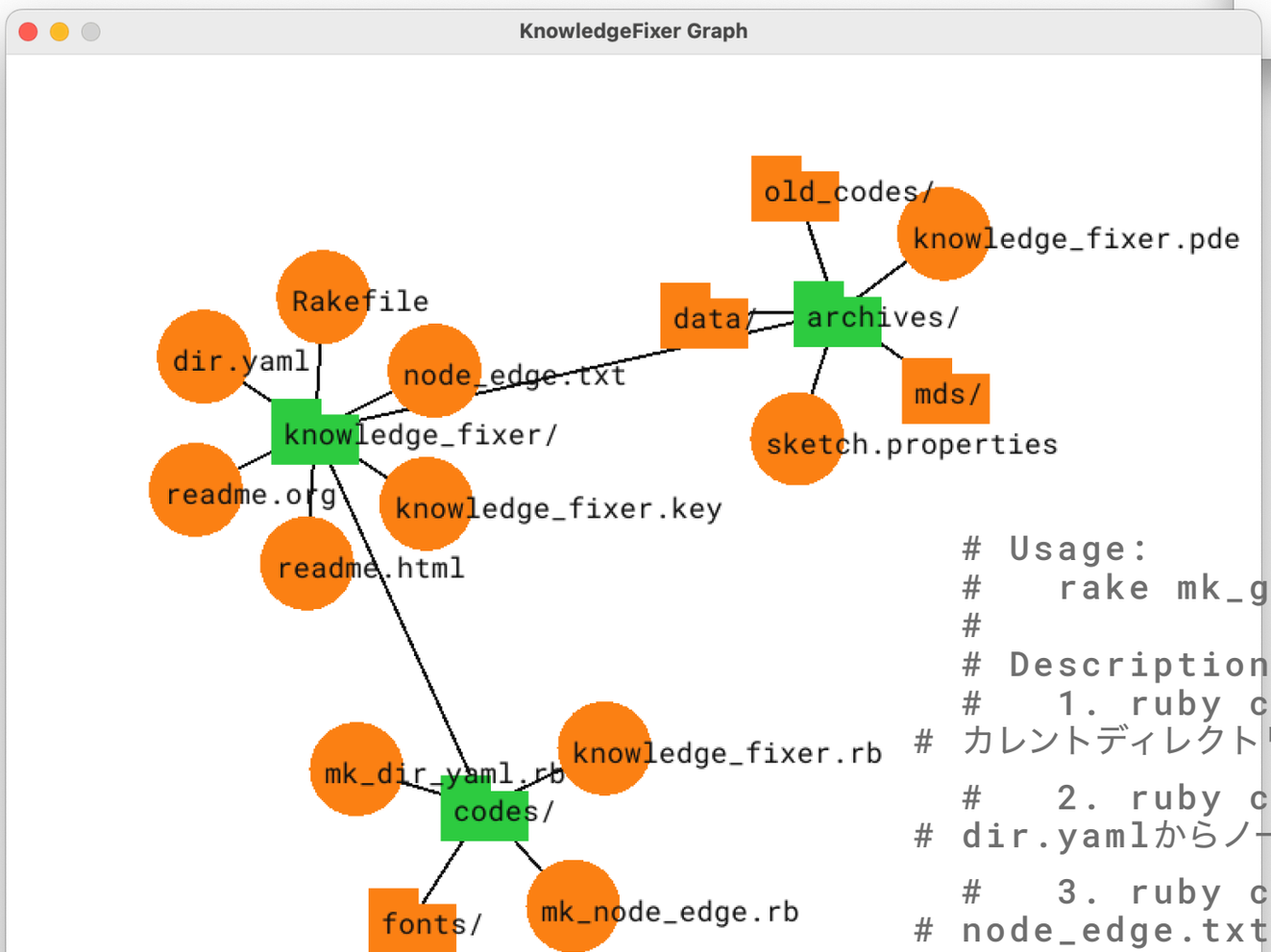

```
node applet
node application.macosx
node data
node knowledge_fixer.pde
node knowledge_fixer.rb
node mark_downs
node mk_dir_array.rb
node wf_icon_dist
node RobotoMonoNerdFontMono-Regular.ttf
node circle_w_text.rb
node directories.yaml
node tmp.yaml
multi_scale_25f--bin
bin--auto_git
bin--mk_cli_copilot
multi_scale_25f--delta_diff.pptx
multi_scale_25f--index.html
multi_scale_25f--knowledge_fixer
knowledge_fixer--applet
knowledge_fixer--application.macosx
knowledge_fixer--data
knowledge_fixer--knowledge_fixer.pde
knowledge_fixer--knowledge_fixer.rb
knowledge_fixer--mark_downs
knowledge_fixer--mk_dir_array.rb
wf_icon_dist--RobotoMonoNerdFontMono-Regular.ttf
wf_icon_dist--circle_w_text.rb
wf_icon_dist--directories.yaml
wf_icon_dist--tmp.yaml
```

```
> tree -L 2 .
.
├── Rakefile
├── archives
│   ├── mk_semi_lattice
│   ├── s0_pre_codes
│   ├── s1_pde2rb
│   ├── s2_relax
│   ├── s3_mk_dir_yaml
│   ├── s4_mk_node_edge_yaml
│   ├── s5_mk_lat_graph
│   └── s6_whole_steps
├── current_code -> ./archives/s6_whole_steps/
├── hyper_card
│   ├── hyper_card.yaml
│   └── hyper_card_node_edge.yaml
```

```
Usage:
click       : fix
drag        : move
shift-click : relax again
double click: open
```






```
# Usage:
#   rake mk_graph
#
# Description:
#   1. ruby codes/mk_dir_array.rb -L 2 .
# カレントディレクトリ以下2階層のディレクトリ構造をdir.yamlに出力
#   2. ruby codes/mk_node_edge.rb dir.yaml > node_edge.txt
# dir.yamlからノード・エッジ情報をnode_edge.txtに出力
#   3. ruby codes/knowledge_fixer.rb node_edge.txt
# node_edge.txtを使ってknowledge fixerを実行
```

# double_click_action

```
  if clicked_node
    is_double, now = double_click?(clicked_node, last_click_node, last_click_time)
    double_click_action(clicked_node) if is_double
    last_click_time = now
    last_click_node = clicked_node
  end
```

```
def double_click_action(clicked_node)
  comm = nil
  if clicked_node.file_path
    if File.directory?(clicked_node.file_path)
      if RbConfig::CONFIG['host_os'] =~ /darwin/
        comm = "open -a Terminal '#{clicked_node.file_path}'"
      else
        comm = "wt.exe -p Ubuntu-24.04 --colorScheme 'Tango Light'
      end
    else
      if RbConfig::CONFIG['host_os'] =~ /darwin/
        comm = "open '#{clicked_node.file_path}'"
      else
        comm = "explorer.exe '#{clicked_node.file_path}'"
      end
    end
    puts comm
    system comm
  else
    puts "no link error"
  end
end
```

#### ステップごとの動作

1. **クリックされたノードがあるか判定**
   `if clicked_node` で、クリック位置にノードがあった場合のみ処理を進めます。

2. **ダブルクリック判定**
   `if last_click_node == clicked_node && last_click_time && (now - last_click_time < 0.4)`
   - 前回クリックしたノードと同じノードをクリックしているか
   - 前回クリック時刻が記録されているか
   - 前回クリックから0.4秒以内か

   これらを満たす場合、ダブルクリックとみなします。

3. **ファイルオープンコマンドの生成**
   - ノードにファイルパスがある場合のみ処理
   - ディレクトリなら
     - Macなら `open -a Terminal 'ディレクトリパス'`
     - それ以外（Windows想定）なら `wt.exe ...`
   - ファイルなら
     - Macなら `open 'ファイルパス'`
     - それ以外なら `explorer.exe 'ファイルパス'`
   - コマンドを `comm` に格納し、標準出力に表示後、`system comm` で実行

4. **ファイルパスがなければエラー表示**
   `"no link error"` を出力

5. **クリック時刻・ノードの記録**
   ダブルクリック判定用に、`last_click_time` と `last_click_node` を更新

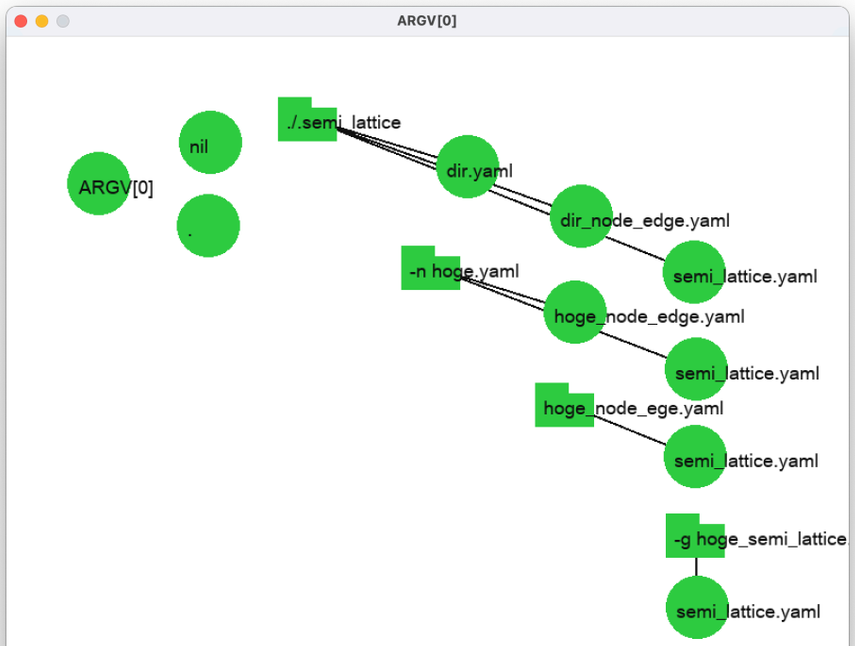
この処理により、ノードをダブルクリックすると、そのノードに紐づくファイルやディレクトリがOSに応じた方法で開かれます。

daddygongon: #file:mk_semi_lattice.rb:93-111 を切り出して,
double_click_actionという関数にしてください.

GitHub Copilot: ### [mk_semi_lattice.rb](file:///Users/bob/git_hub/mk_semi_lattice/lib/mk_semi_lattice.rb)

`93-111` のダブルクリック判定とファイルオープン処理を
`double_click_action(clicked_node, last_click_node, last_click_time)`
関数に切り出します。
`last_click_time` と `last_click_node` の更新は呼び出し元で行うようにします。

# yaml selection



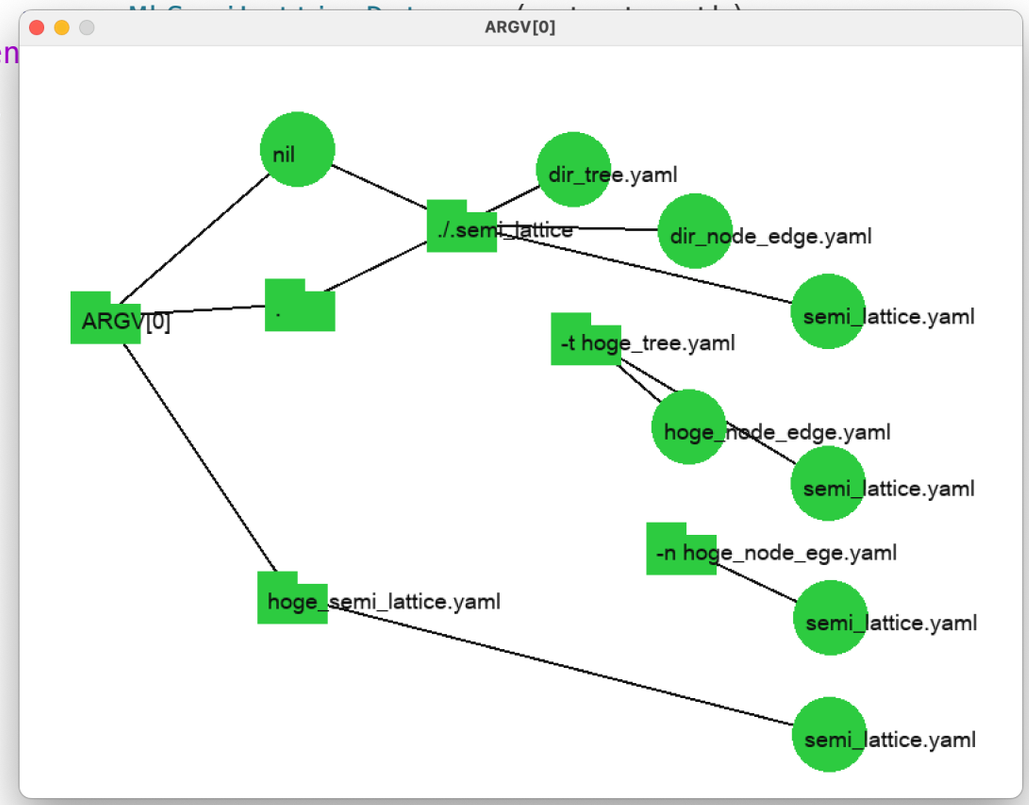

```ruby
path = ARGV[0] || '.'
$semi_dir = File.join(path, '.semi_lattice')

if options[:node_file]
  input_path = options[:node_file]
  parent_dir = File.dirname(input_path)
  base = File.basename(input_path, File.extname(input_path))
  output_path = File.join(parent_dir, "#{base}_node_edge.yaml")
  semi_lattice_yaml = File.join(parent_dir, "semi_lattice.yaml")
  if File.exist?(semi_lattice_yaml)
    warn "Warning: #{semi_lattice_yaml} already exists. Aborting.".colorize(:red)
    exit 1
  end
  MkSemiLattice::MkNodeEdge.new(input_path: input_path, output_path: output_path)
  file = output_path # ← ここでfileにoutput_pathを代入
elsif path == '.'
  Dir.mkdir($semi_dir) unless Dir.exist?($semi_dir)
  dir_yaml_path = File.join($semi_dir, 'dir.yaml')
  MkSemiLattice::MkDirYaml.new(path: path, layer: options[:layer],
                               output_file: dir_yaml_path)
  MkSemiLattice::MkNodeEdge.new(input_path: dir_yaml_path,
                               output_path: File.join($semi_dir, 'dir_node_edge.yaml'))
end

file ||= ARGV[0] || File.join($semi_dir, "dir_node_edge.yaml")
semi_lattice_yaml_path = File.join($semi_dir, "semi_lattice.yaml")

if options[:graph_file]
  file = options[:graph_file]
  app = MkSemiLatticeData.new(file, with_semi_lattice_yaml: true)
elsif File.exist?(semi_lattice_yaml_path)
  file = semi_lattice_yaml_path
  app = MkSemiLatticeData.new(file, with_semi_lattice_yaml: true)
else
  app = MkSemiLatticeData.new(file)
end
```

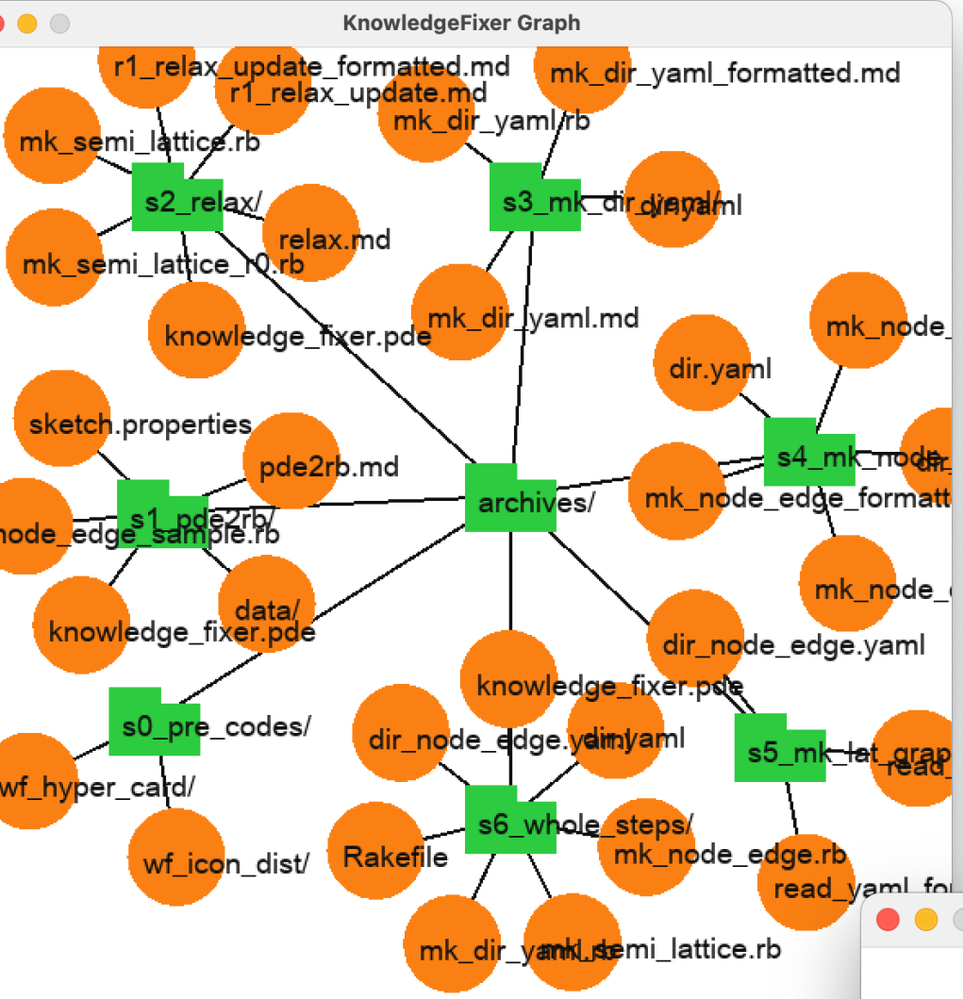
# final

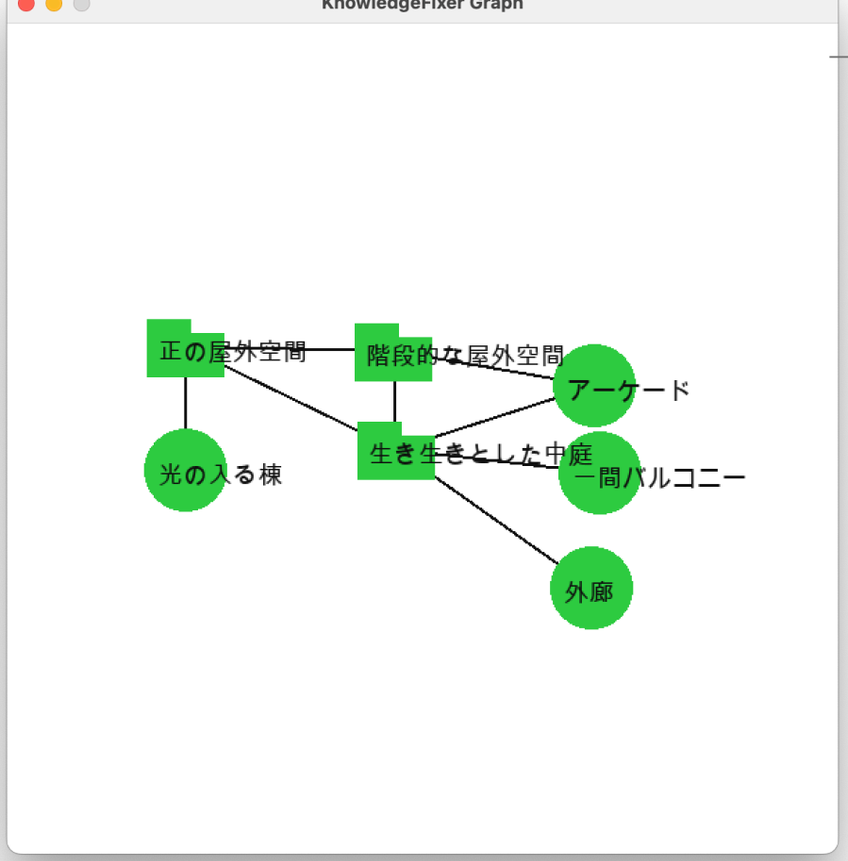
```ruby
parent_dir = Dir.pwd
semi_dir = File.join(parent_dir, '.semi_lattice')
semi_lattice_yaml_path = File.join(semi_dir, "semi_lattice.yaml")

init_file, init_step = if (ARGV[0]=='.' || ARGV[0].nil?) && !options[:file]
  if Dir.exist?(semi_lattice_yaml_path)
    [semi_lattice_yaml_path, :from_semi_lattice]
  else
    ['.', :from_dir]
  end
else
  [ARGV[0], options[:init_step]]
end

input_path, with_semi_lattice_yaml = case init_step
when :from_dir
  Dir.mkdir(semi_dir) unless Dir.exist?(semi_dir)
  in_path, out_path = init_file, File.join(semi_dir, 'dir_tree.yaml')
  MkSemiLattice::MkDirYaml.new(path: in_path, layer: options[:layer],
                               output_file: out_path)
  in_path, out_path = out_path, File.join(semi_dir, 'dir_node_edge.yaml')
  MkSemiLattice::MkNodeEdge.new(input_path: in_path,
                                output_path: out_path )
  [out_path, false]
when :from_tree
  init_file = options[:file]
  base = File.basename(init_file, File.extname(init_file))
  in_path, out_path= init_file, File.join(parent_dir, "#{base}_node_edge.yaml")
  MkSemiLattice::MkNodeEdge.new(input_path: in_path, output_path: out_path)
  [out_path, false]
when :from_node_edge
  if File.exist?(File.join(parent_dir, 'semi_lattice.yaml'))
    puts "Warning: semi_lattice.yaml already exists in current directory.".yellow
    exit 1
  end
  [options[:file], false]
when :from_semi_lattice
  [init_file, true]
end

# p [input_path, with_semi_lattice_yaml]

app = MkSemiLatticeData.new(input_path,
  with_semi_lattice_yaml: with_semi_lattice_yaml)
```

# revising

```ruby
if options[:node_file]
  input_path = options[:node_file]
  app = MkSemiLatticeData.new(input_path)
elsif options[:tree_file]
  input_path = options[:tree_file]
  parent_dir = File.dirname(input_path)
  base = File.basename(input_path, File.extname(input_path
  output_path = File.join(parent_dir, "#{base}_node_edge.ya
  semi_lattice_yaml = File.join(parent_dir, "semi_lattice.y
  if File.exist?(semi_lattice_yaml)
    warn "Warning: #{semi_lattice_yaml} already exists. Abo
    exit 1
  end
  MkSemiLattice::MkNodeEdge.new(input_path: input_path, out
  file = output_path
  app = MkSemiLatticeData.new(output_path)
elsif path != '.'
  app = MkSemiLatticeData.new(path, with_semi_lattice_yaml
elsif File.exist?(semi_lattice_yaml_path)
                    iceData.new(semi_lattice_yaml_path, with_
else
  Dir.mkdir($semi_dir) unless Dir.exist?($semi_dir)
  dir_yaml_path = File.join($semi_dir, 'dir.yaml')
  MkSemiLattice::MkDirYaml.new(path: path, layer: options[
                               output_file: dir_yaml_path)
  output_path = File.join($semi_dir, 'dir_node_edge.yaml')
  MkSemiLattice::MkNodeEdge.new(input_path: dir_yaml_path,
                                output_path: output_path )
```






File: patt_lang_node_edge.yaml

```yaml
---
nodes:
- id: 1
  name: 正の屋外空間
  type: dir
- id: 2
  name: 階段的な屋外空間
  type: dir
- id: 3
  name: 生き生きとした中庭
  type: dir
- id: 4
  name: アーケード
  type: file
- id: 5
  name: 外廊
  type: file
- id: 6
  name: 一間バルコニー
  type: file
- id: 7
  name: 光の入る棟
  type: file
edges:
- from: 1
  to: 7
- from: 2
  to: 4
- from: 2
  to: 3
- from: 1
  to: 3
- from: 3
  to: 4
- from: 3
  to: 5
- from: 3
  to: 6
```

File: hyper_card.yaml

```yaml
---
HyperCard/:
  !=html(link...):
  知識の構造(ツリーではない)--qiita/:
    Tree==MindMap:
  chunking(7$\pm$2):
  知的生産の技術(梅棹)/:
    京大(B5)カード, 記憶の代わりに記録:
  toggle表示/:
    enCapsule(隠蔽)/:
      object指向(思考):
      Inherit(遺産継承):
      PolyMorph(多形, morpheme, phoneme):
```