Pattern Miner

Nil Geisweiller

SingularityNetathon 2018

OpenCog Foundation

Pattern Miner

- Find frequent patterns in the AtomSpace
- Patterns are Atomese programs, specifically pattern matcher queries
- Reboot
  - previous version from Shujing Ke (C++)
  - new version is URE oriented
    - More general
    - URE control
    - WIP

Pattern Miner: Algorithm

Initialize a collection of patterns

1. Select a pattern P from collection
2. Run P and extract valuations
3. Determine shallow abstractions from values
4. Specialize P by composing it with shallow abstractions
5. Add specializations of P with enough support to the collection
6. Repeat till termination

Pattern Miner: Algorithm

- AtomSpace: { (Inheritance A B), (Inheritance A C) }
- Min support: 2

Initialize a collection of patterns

1. Select a pattern P from collection
2. Run P and extract valuations
3. Determine shallow abstractions from values
4. Specialize P by composing it with shallow abstractions
5. Add specializations of P with enough support to the collection
6. Repeat till termination

Pattern Miner: Algorithm

- AtomSpace: { (Inheritance A B), (Inheritance A C) }
- Min support: 2

Initialize a collection of patterns: { (Lambda X X) }

1. Select a pattern P from collection
2. Run P and extract valuations
3. Determine shallow abstractions from values
4. Specialize P by composing it with shallow abstractions
5. Add specializations of P with enough support to the collection
6. Repeat till termination

Pattern Miner: Algorithm

- AtomSpace: { (Inheritance A B), (Inheritance A C) }
- Min support: 2

Initialize a collection of patterns: { (Lambda X X) }

1. Select a pattern P from collection: P = (Lambda X X)
2. Run P and extract valuations
3. Determine shallow abstractions from values
4. Specialize P by composing it with shallow abstractions
5. Add specializations of P with enough support to the collection
6. Repeat till termination

Pattern Miner: Algorithm

- AtomSpace: { (Inheritance A B), (Inheritance A C) }
- Min support: 2

Initialize a collection of patterns: { (Lambda X X) }

1. Select a pattern P from collection: P = (Lambda X X)
2. Run P and extract valuations: { X=(Inheritance A B), X=(Inheritance A C) }
3. Determine shallow abstractions from values
4. Specialize P by composing it with shallow abstractions
5. Add specializations of P with enough support to the collection
6. Repeat till termination

Pattern Miner: Algorithm

- AtomSpace: { (Inheritance A B), (Inheritance A C) }
- Min support: 2

Initialize a collection of patterns: { (Lambda X X) }

1. Select a pattern P from collection: P = (Lambda X X)
2. Run P and extract valuations: { X=(Inheritance A B), X=(Inheritance A C) }
3. Determine shallow abstractions from values: shabs(X)={ (Lambda Y Z (Inheritance Y Z)) }
4. Specialize P by composing it with shallow abstractions
5. Add specializations of P with enough support to the collection
6. Repeat till termination

Pattern Miner: Algorithm

- AtomSpace: { (Inheritance A B), (Inheritance A C) }
- Min support: 2

Initialize a collection of patterns: { (Lambda X X) }

1. Select a pattern P from collection: P = (Lambda X X)
2. Run P and extract valuations: { X=(Inheritance A B), X=(Inheritance A C) }
3. Determine shallow abstractions from values: shabs(X)={ (Lambda Y Z (Inheritance Y Z)) }
4. Specialize P by composing it with shallow abstractions: { (Put P (Lambda Y Z (Inheritance Y Z))) }
5. Add specializations of P with enough support to the collection
6. Repeat till termination

Pattern Miner: Algorithm

- AtomSpace: { (Inheritance A B), (Inheritance A C) }
- Min support: 2

Initialize a collection of patterns: { (Lambda X X) , (Lambda Y Z (Inheritance Y Z)) }

1. Select a pattern P from collection: P = (Lambda X X)
2. Run P and extract valuations: { X=(Inheritance A B), X=(Inheritance A C) }
3. Determine shallow abstractions from values: shabs(X)={ (Lambda Y Z (Inheritance Y Z)) }
4. Specialize P by composing it with shallow abstractions: { (Put P (Lambda Y Z (Inheritance Y Z))) }
5. Add specializations of P with enough support to the collection
6. Repeat till termination

Pattern Miner: Algorithm

- AtomSpace: { (Inheritance A B), (Inheritance A C) }
- Min support: 2

Initialize a collection of patterns: { (Lambda X X) , (Lambda Y Z (Inheritance Y Z)) }

1. Select a pattern P from collection:
2. Run P and extract valuations:
3. Determine shallow abstractions from values:
4. Specialize P by composing it with shallow abstractions:
5. Add specializations of P with enough support to the collection
6. Repeat till termination

Pattern Miner: Algorithm

- AtomSpace: { (Inheritance A B), (Inheritance A C) }
- Min support: 2

Initialize a collection of patterns: { (Lambda X X) , (Lambda Y Z (Inheritance Y Z)) }

1. Select a pattern P from collection: P = (Lambda Y Z (Inheritance Y Z))
2. Run P and extract valuations:
3. Determine shallow abstractions from values:
4. Specialize P by composing it with shallow abstractions:
5. Add specializations of P with enough support to the collection
6. Repeat till termination

Pattern Miner: Algorithm

- AtomSpace: { (Inheritance A B), (Inheritance A C) }
- Min support: 2

Initialize a collection of patterns: { (Lambda X X) , (Lambda Y Z (Inheritance Y Z)) }

1. Select a pattern P from collection: P = (Lambda Y Z (Inheritance Y Z))
2. Run P and extract valuations: { {Y=A, Z=B}, {Y=A, Z=C} }
3. Determine shallow abstractions from values:
4. Specialize P by composing it with shallow abstractions:
5. Add specializations of P with enough support to the collection
6. Repeat till termination

Pattern Miner: Algorithm

- AtomSpace: { (Inheritance A B), (Inheritance A C) }
- Min support: 2

Initialize a collection of patterns: { (Lambda X X) , (Lambda Y Z (Inheritance Y Z)) }

1. Select a pattern P from collection: P = (Lambda Y Z (Inheritance Y Z))
2. Run P and extract valuations: { {Y=A, Z=B}, {Y=A, Z=C} }
3. Determine shallow abstractions from values: shabs(Y)={A}
4. Specialize P by composing it with shallow abstractions:
5. Add specializations of P with enough support to the collection
6. Repeat till termination

Pattern Miner: Algorithm

- AtomSpace: { (Inheritance A B), (Inheritance A C) }
- Min support: 2

Initialize a collection of patterns: { (Lambda X X) , (Lambda Y Z (Inheritance Y Z)) }

1. Select a pattern P from collection: P = (Lambda Y Z (Inheritance Y Z))
2. Run P and extract valuations: { {Y=A, Z=B}, {Y=A, Z=C} }
3. Determine shallow abstractions from values: shabs(Y)={A}
4. Specialize P by composing it with shallow abstractions: { (Put P (List A Z)) }
5. Add specializations of P with enough support to the collection
6. Repeat till termination

Pattern Miner: Algorithm

- AtomSpace: { (Inheritance A B), (Inheritance A C) }
- Min support: 2

Initialize a collection of patterns: { (Lambda X X) , (Lambda Y Z (Inheritance Y Z)) , (Lambda Z (Inheritance A Z)) }

1. Select a pattern P from collection: P = (Lambda Y Z (Inheritance Y Z))
2. Run P and extract valuations: { {Y=A, Z=B}, {Y=A, Z=C} }
3. Determine shallow abstractions from values: shabs(Y)={A}
4. Specialize P by composing it with shallow abstractions: { (Put P (List A Z)) }
5. Add specializations of P with enough support to the collection
6. Repeat till termination

Pattern Miner: Status

- C++ prototype
  - efficient
  - exhaustive
  - limited control
  - no surprisingness
  - no filter (except for the initial patterns)
- URE prototype
  - very inefficient (anything is a shallow abstraction!)
  - exhaustive
  - potentially high level of control
  - no surprisingness
  - no filter (except for the initial patterns)