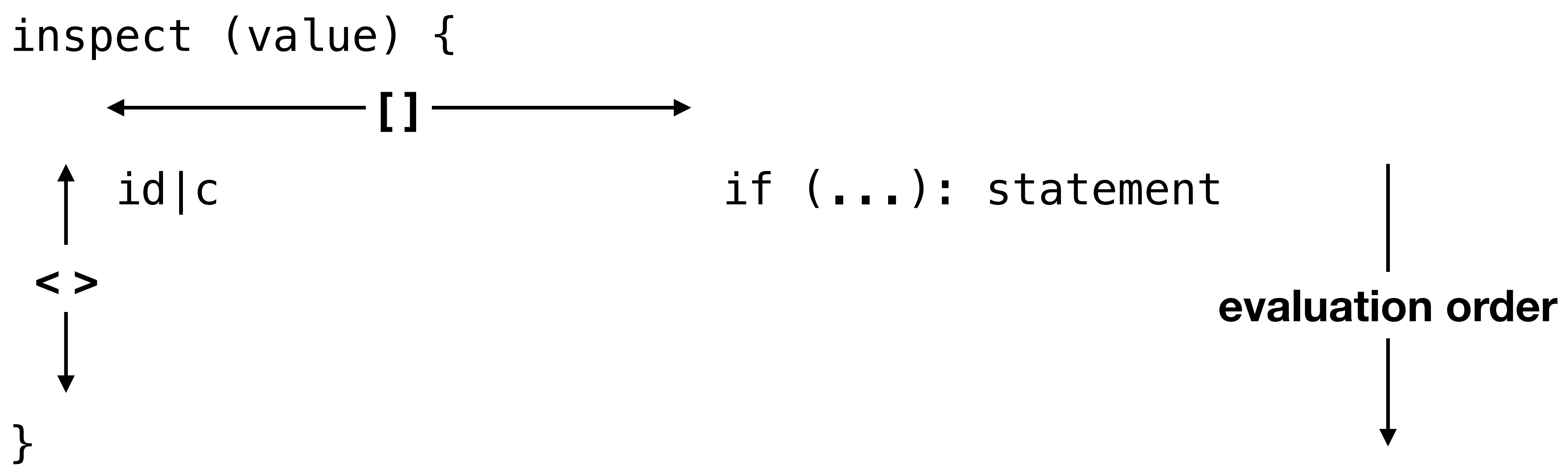
Pattern Matching

P1260 / P1308

Overview



```
inspect (value) {

    id|c                    if (...): statement

}
```

Customization Points:

- `c           | (c! pattern) | (c? pattern)`
- `c.match(v) | c.extract(v) | c.try_extract(v)`

Other compound patterns:

- Dereference pattern - `*x`
- Binding pattern - `id @ pattern`

Refutability

Refutability

- **Irrefutable**: Pattern cannot fail to match for any value
  - e.g., `[x, y]`
- **Refutable**: Pattern can fail to match for some value
  - e.g., `[x, 0]`

Structured Bindings

- Irrefutable patterns only (status-quo)
  - e.g., Rust, Swift
- Refutable patterns / throw an exception on failure
  - e.g., Scala, Haskell

Extensibility

Patterns in Other Languages

- **`1 | 2`** matches if 1 or 2 matches
- **`0..9`** matches if v is **within** the interval `[0, 9]`
- **`[x, 0] & [0, y]`** matches if **both** patterns match

Example: any_of

```
template <typename... Ts>
struct any_of : std::tuple<Ts...> {
  using tuple::tuple;

  template <typename U>
  bool match(const U& u) const {
    return std::apply([&](const auto&... xs) {
      return (... || xs == u);
    }, *this);
  }
};

int fib(int n) {
  inspect (n) {
    x if (x < 0): return 0;
    ^(any_of{1, 2}): return n;
    x: return fib(x - 1) + fib(x - 2);
  }
}
```

Example: `within`

```
struct within {
  int first, last;

  bool match(int n) const { return first <= n && n <= last; }
};


inspect (n) {
  ^(within{0, 9}): std::cout << n << " is in [0, 9].";
  _: std::cout << n << " is not in [0, 9].";
}
```

Example: `deref`

```
struct {
  template <typename T>
  auto&& extract(T&& arg) const {
    return *std::forward<T>(arg);
  }

  template <typename T>
  auto&& try_extract(T&& arg) const {
    return std::forward<T>(arg);
  }
} deref;

int* p = /* ... */;
inspect (p) {
  (deref! x): // unchecked
  (deref? x): // checked
}
```