Closing sync iterator on async wrapper close

https://github.com/tc39/ecma262/pull/2600

Sync iterator rejection

Sync iterator throw - sync iteration

```
function* iterator() {
  try {
    throw "reject";
  } finally {
    print("A");
  }
}

try {
  for (const x of iterator());
} catch (e) {
  print("B", e);
}
```

```
#### engine262, JavaScriptCore,
Moddable XS, SpiderMonkey, V8
A
B reject
```

Async iterator throw - async iteration

```
async function* iterator() {
  try {
    throw "reject";
  } finally {
    print("A");
  }
}

try {
  for await (const x of iterator());
} catch (e) {
  print("B", e);
}
```

```
#### engine262, JavaScriptCore,
Moddable XS, SpiderMonkey, V8
A
B reject
```

Async iterator rejection - async iteration

```
async function* iterator() {
  try {
    yield Promise.reject("reject");
  } finally {
    print("A");
  }
}

try {
  for await (const x of iterator());
} catch (e) {
  print("B", e);
}
```

```
#### engine262, JavaScriptCore,
SpiderMonkey, V8
A
B reject

#### Moddable XS
B reject
```

Iterator rejection - async iteration

```
function* iterator() {
  try {
    yield Promise.reject("reject");
  } finally {
    print("A");
  }
}

try {
  for await (const x of iterator());
} catch (e) {
  print("B", e);
}
```

```
#### engine262, JavaScriptCore,
Moddable XS, SpiderMonkey, V8
B reject
```

Surprising discrepancy

- Rejected promise is a valid value for a sync iterator
- Async wrapper awaits promise value and transforms rejection of value into rejection of iteration operation
- Iterator consumers (e.g. for-of) assume that iterator internally cleaned up before throwing / rejecting, and will not explicitly close it

=> impedance mismatch between producer and consumer

PR updates async wrapper to close sync iterator when the result was valid and not done

Throw called on async wrapper

Throw onto sync iterator

```
function* iterator() {
  try {
    yield 1;
  } catch (e) {
    print("iterator throw", e);
  } finally {
    print("iterator close");
  }
}

const i = iterator();
try {
  print("A", (await i.next()).value);
  await i.throw("reject");
} catch (e) {
  print("B", e.name || e);
}
```

```
#### engine262, JavaScriptCore, Moddable XS,
SpiderMonkey, V8
A 1
iterator throw reject
iterator close
```

Throw onto sync yield start, sync iterator

```
function* iterator() {
  try {
    yield 1;
  } catch (e) {
    print("iterator throw", e);
  } finally {
    print("iterator close");
  }
}

function* wrapper(i) {
  yield* i;
}

const i = wrapper(iterator());
try {
  print("A", (await i.next()).value);
  await i.throw("reject");
} catch (e) {
  print("B", e.name || e);
}
```

```
#### engine262, JavaScriptCore, Moddable XS,
SpiderMonkey, V8
A 1
iterator throw reject
iterator close
```

Throw onto async yield start, async iterator

```
async function* iterator() {
  try {
    yield 1;
  } catch (e) {
    print("iterator throw", e);
  } finally {
    print("iterator close");
  }
}

async function* wrapper(i) {
  yield* i;
}

const i = wrapper(iterator());
try {
  print("A", (await i.next()).value);
  await i.throw("reject");
} catch (e) {
  print("B", e.name || e);
}
```

```
#### engine262, JavaScriptCore, Moddable XS,
SpiderMonkey, V8
A 1
iterator throw reject
iterator close
```

Throw onto async yield start, async iterator without throw

```
async function* iterator() {
  try {
    yield 1;
  } catch (e) {
    print("iterator throw", e);
  } finally {
    print("iterator close");
  }
}

async function* wrapper(i) {
  yield* i;
}

const i = wrapper(removeThrow(iterator()));
try {
  print("A", (await i.next()).value);
  await i.throw("reject");
} catch (e) {
  print("B", e.name || e);
}
```

```
#### engine262, JavaScriptCore, Moddable XS,
SpiderMonkey, V8
A 1
iterator close
B TypeError
```

```
const removeThrow = (i) => {
  const kind =
    Symbol.asyncIterator in i ?
      Symbol.asyncIterator : Symbol.iterator;
  return {
    [kind]() {
      return this;
    },
    next(val) {
      return i.next(val);
    },
    return(val) {
      return i.return(val);
    },
  };
};
```

Throw onto async yield start, sync iterator without throw

```
function* iterator() {
  try {
    yield 1;
  } catch (e) {
    print("iterator throw", e);
  } finally {
    print("iterator close");
  }
}

async function* wrapper(i) {
  yield* i;
}

const i = wrapper(removeThrow(iterator()));
try {
  print("A", (await i.next()).value);
  await i.throw("reject");
} catch (e) {
  print("B", e.name || e);
}
```

```
#### engine262, JavaScriptCore, Moddable XS,
SpiderMonkey, V8
A 1
B reject
```

```
const removeThrow = (i) => {
  const kind =
    Symbol.asyncIterator in i ?
      Symbol.asyncIterator : Symbol.iterator;
  return {
    [kind]() {
      return this;
    },
    next(val) {
      return i.next(val);
    },
    return(val) {
      return i.return(val);
    },
  };
};
```

Another surprising discrepancy

- `iterator.throw` is never called by the language
- `yield*` does forward "throw" and closes the forwarded iterator if missing
- Missing "throw" is still a contract error, so `yield*` throws `TypeError`
- Async wrapper has different behavior: it doesn't close iterator, and forwards / rejects with value provided to "throw".

  => impedance mismatch and language inconsistency

PR updates async wrapper to close sync iterator when "throw" is missing

Should we change the rejection value as well?

Wrappers are not exposed to user code, so I believe `yield*` is the only way to make this mismatch visible