A Transformational Framework for Testing and Model Checking Implicit Invocation Systems

Hongyu Zhang, Jeremy S. Bradbury, James R. Cordy, Juergen Dingel

Software Technology Laboratory

School of Computing, Queen's University

Kingston, Ontario, Canada

DEBS '04

SUPPORTED BY NSERC CRSNG

Motivation

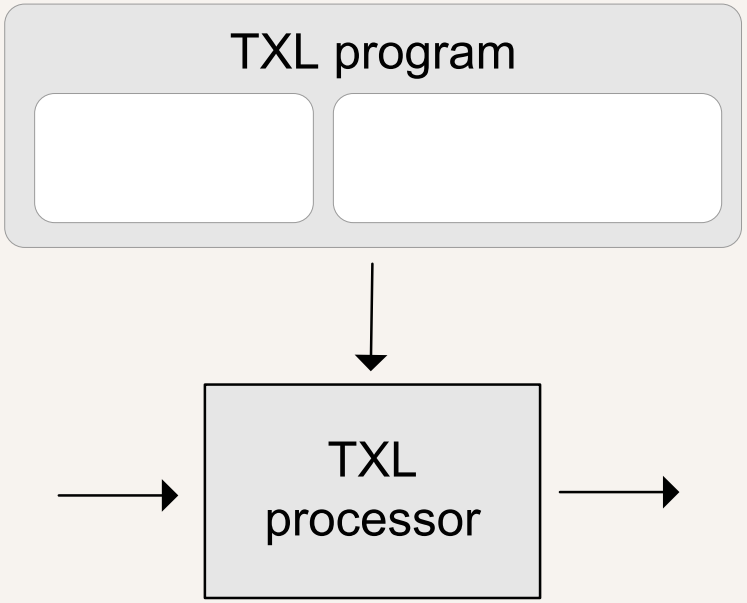
- Testing and model checking are two complementary verification and validation techniques:
  - testing can be lightweight but incomplete.
  - model checking tends to be more heavyweight but complete.
- A major problem is that testing and model checking usually require different software artifacts (semantic artifact gap).

Our Approach

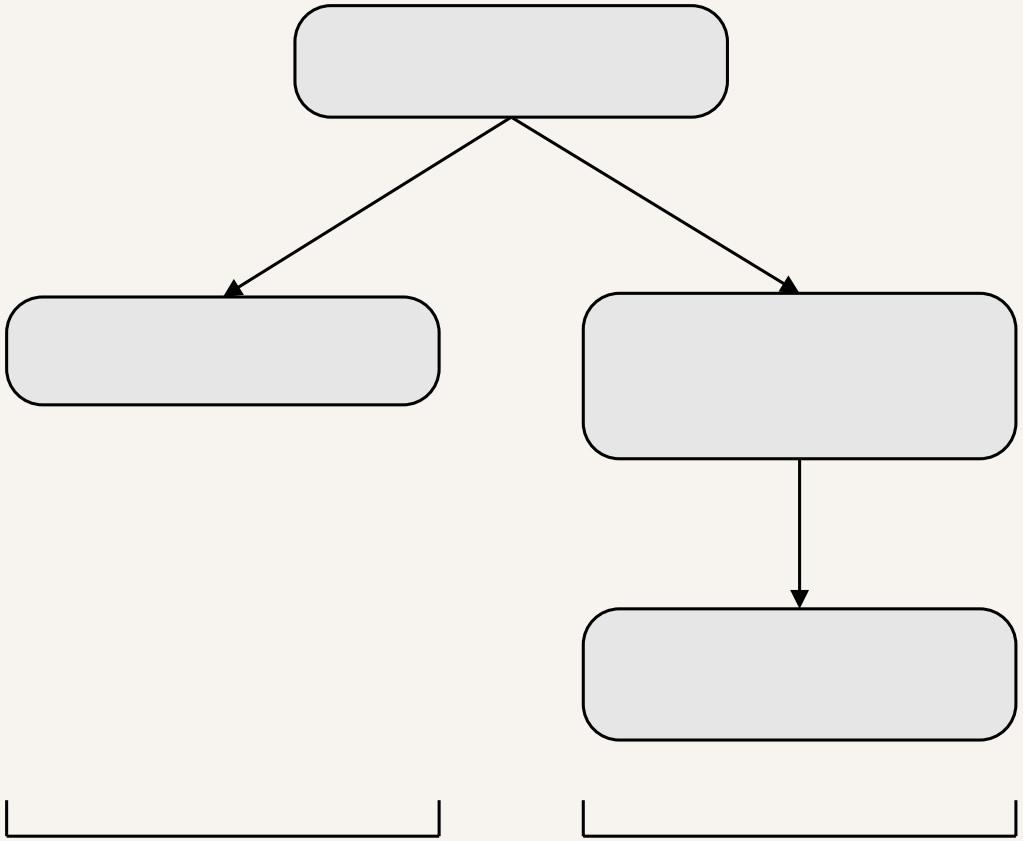
- We have developed a transformational framework for the testing and model checking of implicit invocation (II) systems.
- Why II?
  - II systems feature a lot of non-determinism due to concurrent execution of components.
  - II has become increasingly popular as an integration mechanism for loosely coupled components.

Transformation and TXL

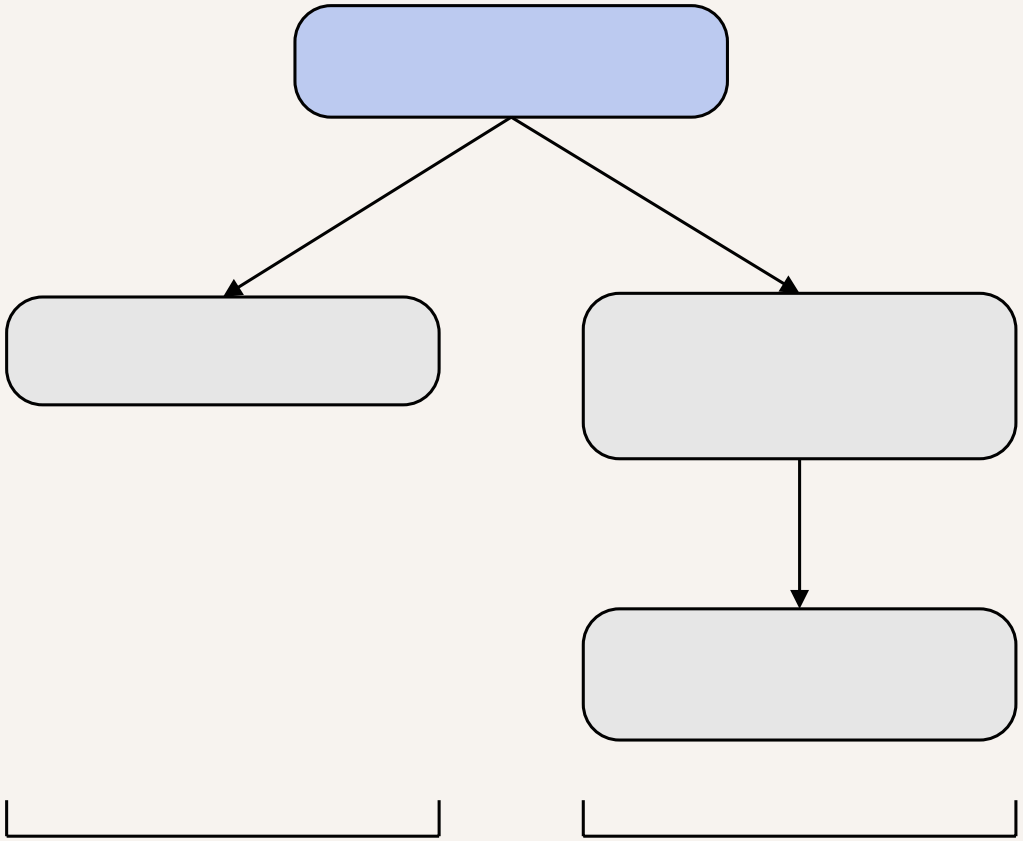
- The transformation tools in our framework are implemented using TXL.
  - TXL is a programming language specifically designed to support rule-based source-to-source transformation.
- Each tool written in TXL is fully automated and is based on formal rewriting rules.



II Transformation Framework



Il Language (IIL)



II Language (IIL)

```
system SetAndCounter {
    external event EnvAdd {1..N}, EnvRemove {1..N};
    event Insert(int {1..2} numElements);
    event Delete(int {1..2} numElements);

    dispatcher delivers Insert, Delete {
        if (Insert.count > Delete.count) {
            deliver Immediate Insert;
            deliver Random Delete;
        }
        else {
            deliver Random Insert;
            deliver Immediate Delete;
        }
    }

    int {0..3} setSize;

    SetAndCounter() {
        Set s = new Set();
        Counter c = new Counter();

        bind EnvAdd to s.Add();
        bind EnvRemove to s.Remove();
        bind Insert to c.CountIns(Insert.numElements);
        bind Delete to c.CountDel(Delete.numElements);

        property AlwaysCatchesUp =
             (G F (setSize = c.counter));
        property ...
     }
```

```
component Set
    announces Insert, Delete
    accepts EnvAdd, EnvRemove {
    int {0..2} value;

    Add() {
        value = {1,2}; //nondeterministic choice
        if ((setSize + value) < 4) {
            setSize = setSize + value;
            announce Insert(value);
        }
    }

    Remove() {
        ...
    }
}

component Counter
    accepts Insert, Delete {
    int {0..3} counter = 0;

    CountIns(int {1..2} number) {
        counter = counter + number;
    }

    CountDel(int {1..2} number) {
        ...
    }
}
```

II Language (IIL)

```
external event EnvAdd {1..N}, EnvRemove {1..N};
event Insert(int {1..2} numElements);
event Delete(int {1..2} numElements);
```

Event Declarations

```
system SetAndCounter {

    dispatcher delivers Insert, Delete {
        if (Insert.count > Delete.count) {
            deliver Immediate Insert;
            deliver Random Delete;
        }
        else {
            deliver Random Insert;
            deliver Immediate Delete;
        }
    }

    int {0..3} setSize;

    SetAndCounter() {
        Set s = new Set();
        Counter c = new Counter();

        bind EnvAdd to s.Add();
        bind EnvRemove to s.Remove();
        bind Insert to c.CountIns(Insert.numElements);
        bind Delete to c.CountDel(Delete.numElements);

        property AlwaysCatchesUp =
             (G F (setSize = c.counter));
        property ...
    }
}
```

```
component Set
    ...rt, Delete
    ..., EnvRemove {
    ...ue;

    Add() {
        value = {1,2}; //nondeterministic choice
        if ((setSize + value) < 4) {
            setSize = setSize + value;
            announce Insert(value);
        }
    }

    Remove() {
        ...
    }
}

component Counter
    accepts Insert, Delete {
    int {0..3} counter = 0;

    CountIns(int {1..2} number) {
        counter = counter + number;
    }

    CountDel(int {1..2} number) {
        ...
    }
}
```

II Language (IIL)

```
system SetAndCounter {
    external event EnvAdd {1..N},
    event Insert(int {1..2} numEle
    event Delete(int {1..2} numEle

    dispatcher delivers Insert, Delete {
        if (Insert.count > Delete.count) {
            deliver Immediate Insert;
            deliver Random Delete;
        }
        else {
            deliver Random Insert;
            deliver Immediate Delete;
        }
    }

    int {0..3} setSize;

    SetAndCounter() {
        Set s = new Set();
        Counter c = new Counter();

        bind EnvAdd to s.Add();
        bind EnvRemove to s.Remove();
        bind Insert to c.CountIns(Insert.numElements);
        bind Delete to c.CountDel(Delete.numElements);

        property AlwaysCatchesUp =
             (G F (setSize = c.counter));
        property ...
     }
}
```

Component Declarations

```
component Set
    announces Insert, Delete
    accepts EnvAdd, EnvRemove {
    int {0..2} value;

    Add() {
        value = {1,2}; //nondeterministic choice
        if ((setSize + value) < 4) {
            setSize = setSize + value;
            announce Insert(value);
        }
    }

    Remove() {
        ...
    }
}

component Counter
    accepts Insert, Delete {
    int {0..3} counter = 0;

    CountIns(int {1..2} number) {
        counter = counter + number;
    }

    CountDel(int {1..2} number) {
        ...
    }
}
```

II Language (IIL)

```
system SetAndCounter {
    external event EnvAdd {1..N}, EnvRemove {1..N};
    event Insert(int {1..2} numElements);
    event Delete(int {1..2} numElements);

    dispatcher delivers Insert, Delete {
        if (Insert.count > Delete.count) {
            deliver Immediate Insert;
            deliver Random Delete;
        }
        else {
            deliver Random Insert;
            deliver Immediate Delete;
        }
    }

    int {0..3} setSize;

    SetAndCounter() {
        Set s = new Set();
        Counter c = new Counter();

        bind EnvAdd to s.Add();
        bind EnvRemove to s.Remove();
        bind Insert to c.CountIns(Insert.numElements);
        bind Delete to c.CountDel(Delete.numElements);

        property AlwaysCatchesUp =
             (G F (setSize = c.counter));
        property ...
     }
}
```

Announce Statements

```
component Set
    announces Insert, Delete
    accepts EnvAdd, EnvRemove {
    int {0..2} value;

    Add() {
        value = {1,2}; //nondeterministic choice
        if ((setSize + value) < 4) {
            setSize = setSize + value;
            announce Insert(value);
        }
    }

    Remove() {
        ...
    }
}

component Counter
    accepts Insert, Delete {
    int {0..3} counter = 0;

    CountIns(int {1..2} number) {
        counter = counter + number;
    }

    CountDel(int {1..2} number) {
        ...
    }
}
```

II Language (IIL)

```
system SetAndCounter {
    external event EnvAdd {1..N}, EnvRemove {1..N};
    event Insert(int {1..2} numElements);
    event Delete(int {1..2} numElements);

    dispatcher delivers Insert, Delete {
        if (Insert.count > Delete.count) {
            deliver Immediate Insert;
            deliver Random Delete;
        }
        else {
            deliver Random Insert;
            deliver Immediate Delete;
        }
    }

    int {0..3} setSize;

    SetAndCounter() {
        Set s = new Set();
        Counter c = new Counter();

        bind EnvAdd to s.Add();
        bind EnvRemove to s.Remove();
        bind Insert to c.CountIns(Insert.numElements);
        bind Delete to c.CountDel(Delete.numElements);

        property AlwaysCatchesUp =
             (G F (setSize = c.counter));
        property ...
    }
```

Dispatcher Declaration

```
component Set
    announces Insert, Delete
    accepts EnvAdd, EnvRemove {
    int {0..2} value;

                2}; //nondeterministic choice
                e + value) < 4) {
                = setSize + value;
                announce Insert(value);
            }
        }

        Remove() {
            ...
        }
    }

component Counter
    accepts Insert, Delete {
    int {0..3} counter = 0;

    CountIns(int {1..2} number) {
        counter = counter + number;
    }

    CountDel(int {1..2} number) {
        ...
    }
}
```

II Language (IIL)

```
system SetAndCounter {
    external event EnvAdd {1..N}, EnvRemove {1..N};
    event Insert(int {1..2} numElements);
    event Delete(int {1..2} numElements);

    dispatcher delivers Insert, Delete {
        if (Insert.count > Delete.count) {
            deliver Immediate Insert;
            deliver Random Delete;
        }
        else {
            deliver Random Insert;
            deliver Immediate Delete;
        }
    }

    int {0..3} setSize;

    SetAndCounter() {
        Set s = new Set();
        Counter c = new Counter();

        bind EnvAdd to s.Add();
        bind EnvRemove to s.Remove();
        bind Insert to c.CountIns(Insert.numElements);
        bind Delete to c.CountDel(Delete.numElements);

        property AlwaysCatchesUp =
             (G F (setSize = c.counter));
        property ...
    }
}
```

Event-method Declarations

```
component Set
    announces Insert, Delete
    accepts EnvAdd, EnvRemove {
    int {0..2} value;

    Add() {
        value = {1,2}; //nondeterministic choice
        if ((setSize + value) < 4) {
            setSize = setSize + value;
            announce Insert(value);
        }
    }

    Remove() {
        ...
    }
}

component Counter
    accepts Insert, Delete {
    int {0..3} counter = 0;

    CountIns(int {1..2} number) {
        ...
    }

    CountDel(int {1..2} number) {
        ...
    }
}
```

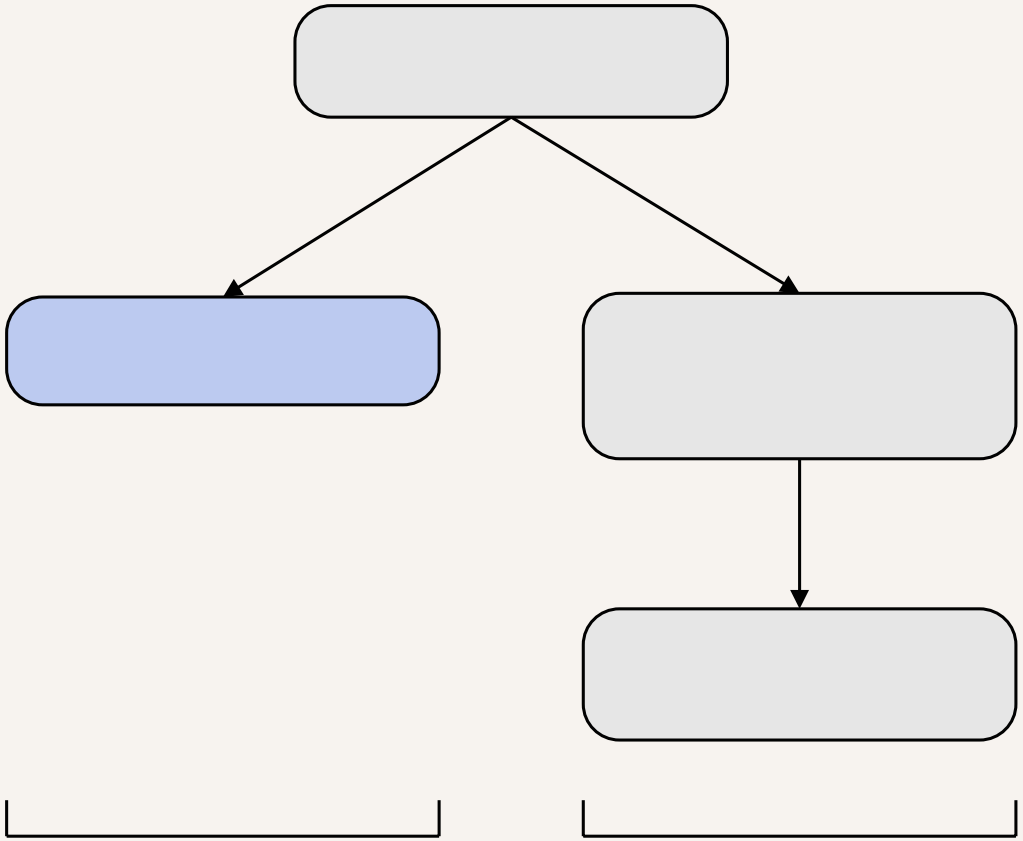
II Language (IIL)

```
system SetAndCounter {
    external event EnvAdd {1..N}, EnvRemove {1..N};
    event Insert(int {1..2} numElements);
    event Delete(int {1..2} numElements);

    dispatcher delivers Insert, Delete {
        if (Insert.count > Delete.count) {
            deliver Immediate Insert;
            deliver Random Delete;
        }
        else {
            deliver Random Insert;
            deliver Immediate Delete;
        }
    }

    int {0..3} setSize;

    SetAndCounter() {
        Set s = new Set();
        Counter c = new Counter();

        bind EnvAdd to s.Add();
        bind EnvRemove to s.Remove();
        bind Insert to c.CountIns(Insert.numElements);
        bind Delete to c.CountDel(Delete.numElements);

        property AlwaysCatchesUp =
             (G F (setSize = c.counter));
        property ...
    }
```

```
component Set
    announces Insert, Delete
    accepts EnvAdd, EnvRemove {
    int {0..2} value;

    Add() {
        value = {1,2}; //nondeterministic choice
        if ((setSize + value) < 4) {
            setSize = setSize + value;
            announce Insert(value);
        }
    }

    Remove() {
        ...
    }
}

component Counter
    accepts Insert, Delete {
    int {0..3} counter = 0;

    CountIns(int {1..2} number) {
        counter = counter + number;
    }

    CountDel(int {1..2} number) {
```

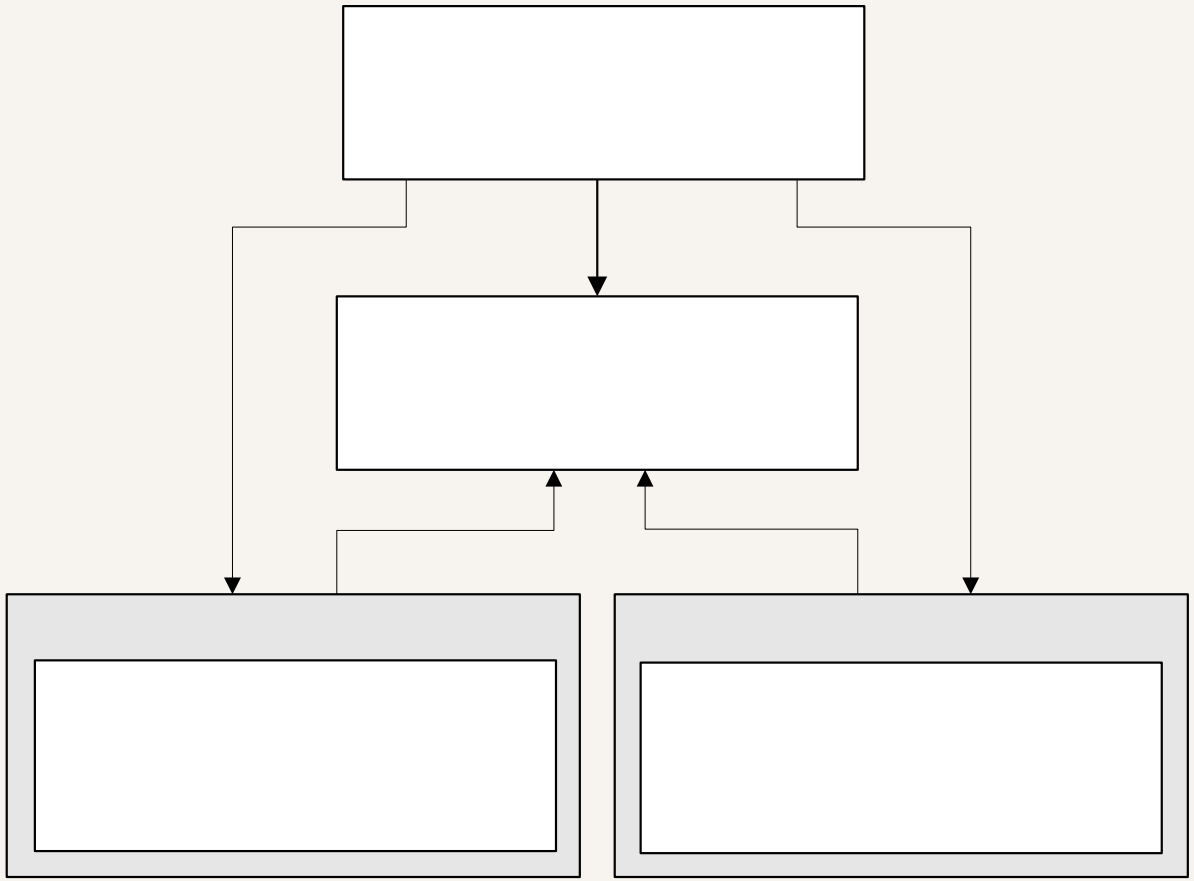
Property Declarations

Testing II Programs



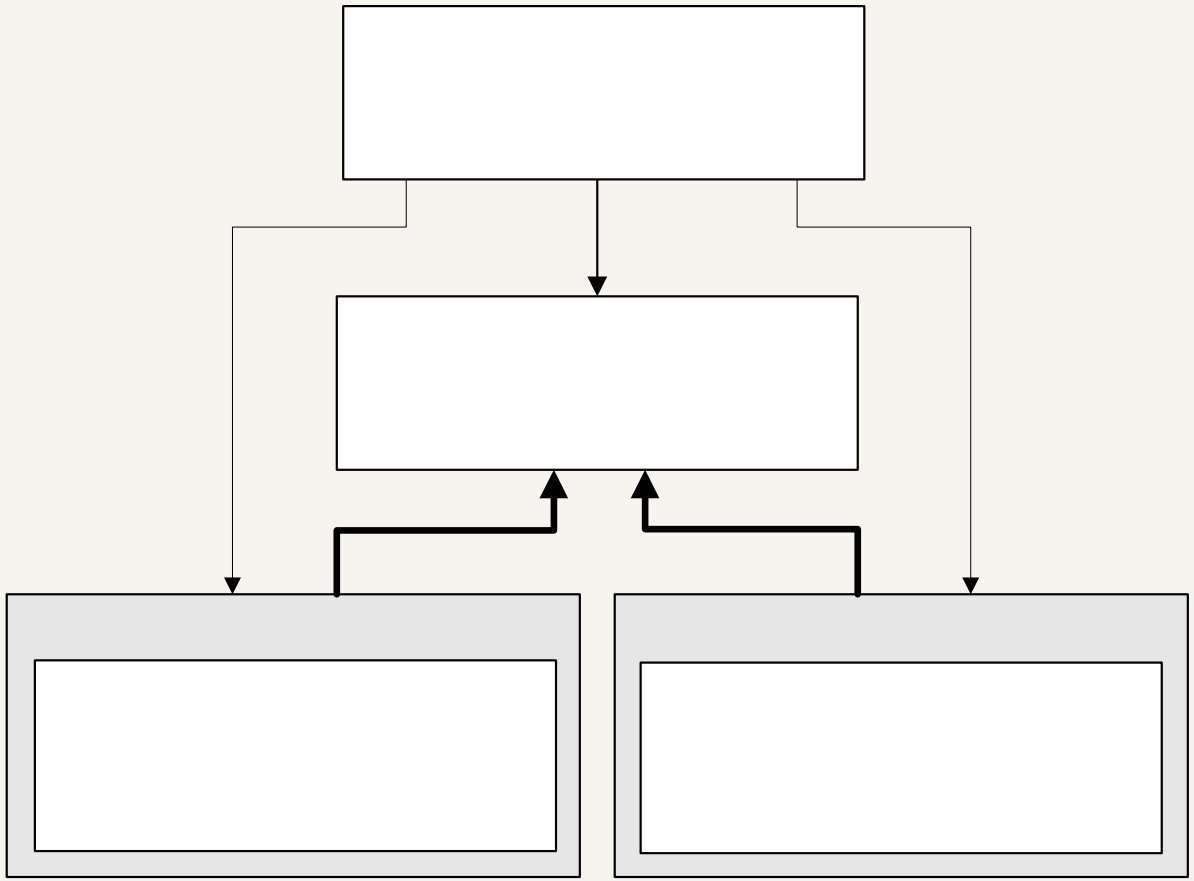
Testing IIL Programs

- Turing Plus is a concurrent programming language with random scheduling.
- The first challenge was developing an execution model of II in Turing Plus.



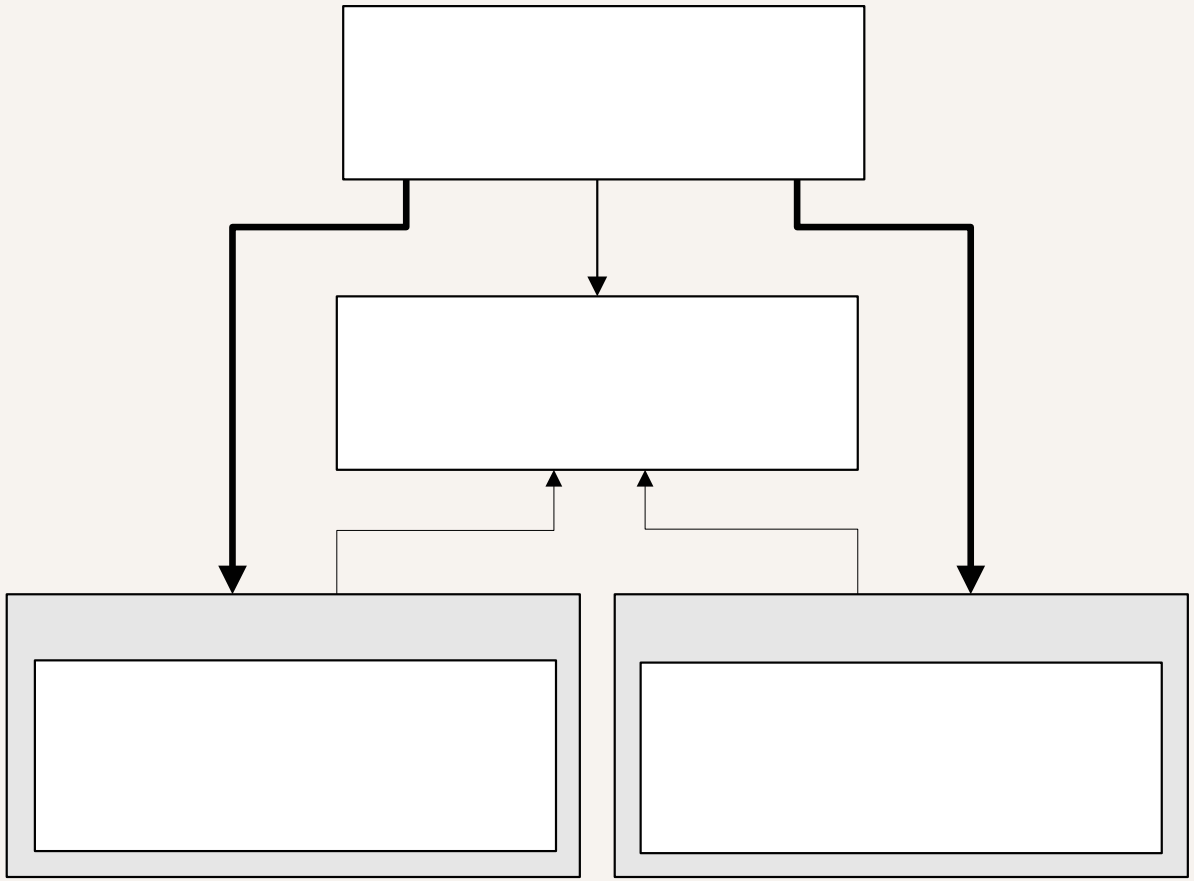
Testing IIL Programs

- Turing Plus is a concurrent programming language with random scheduling.
- The first challenge was developing an execution model of II in Turing Plus.



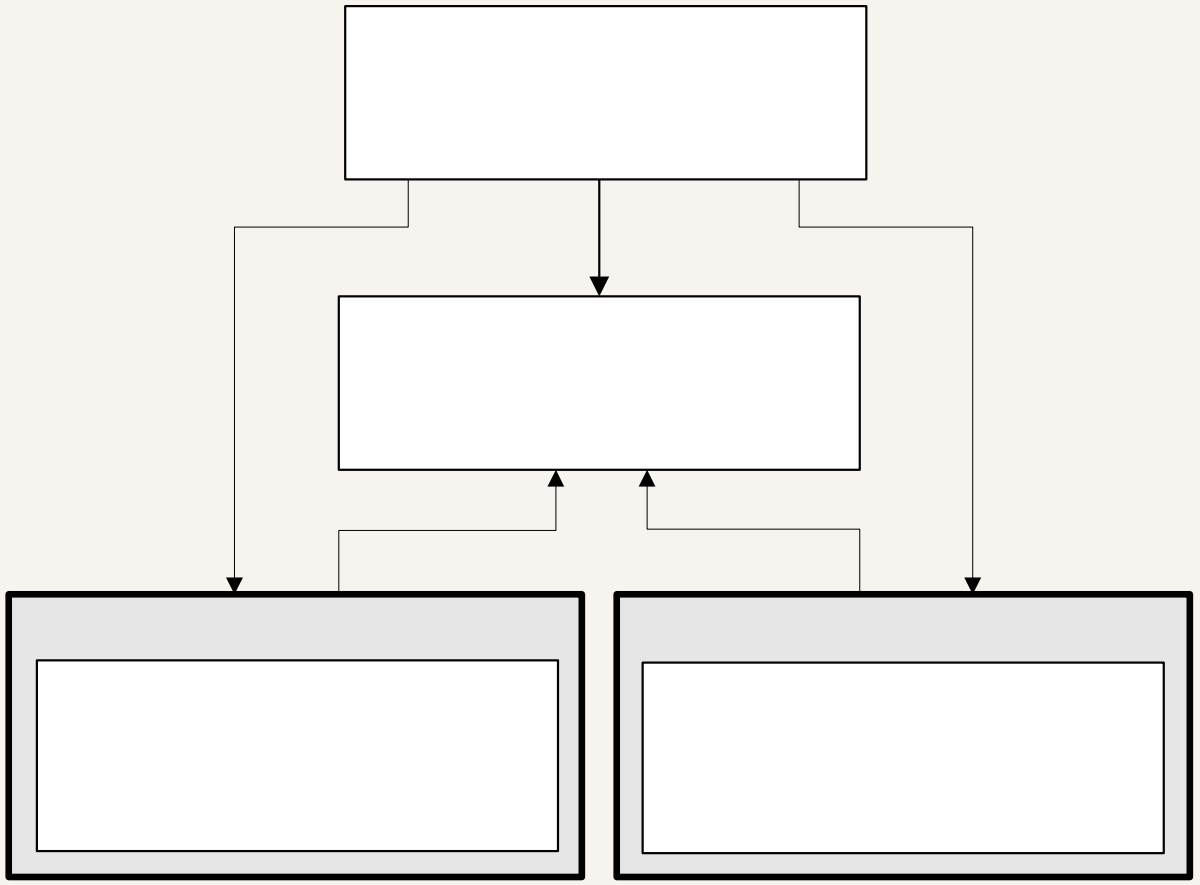
Testing IIL Programs

- Turing Plus is a concurrent programming language with random scheduling.
- The first challenge was developing an execution model of II in Turing Plus.



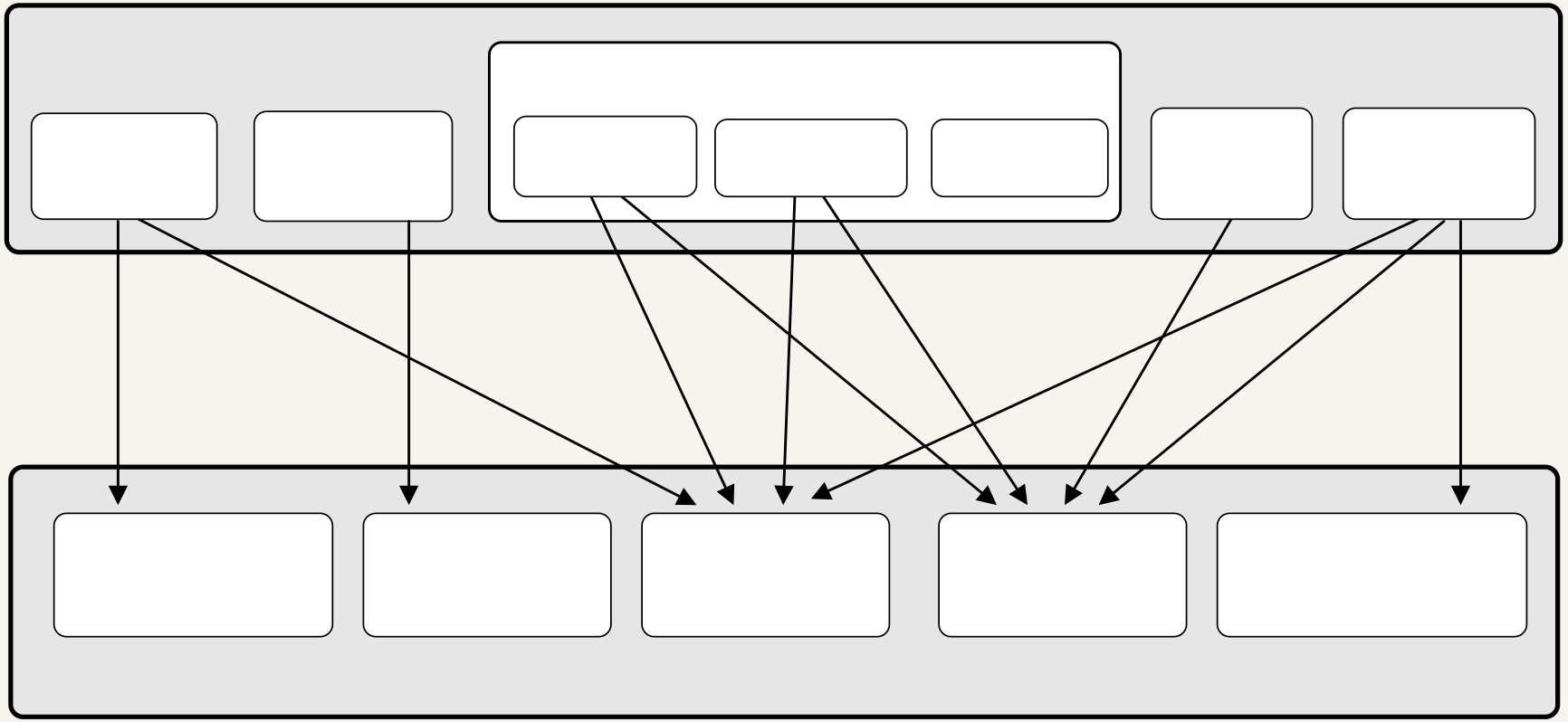
Testing IIL Programs

- Turing Plus is a concurrent programming language with random scheduling.
- The first challenge was developing an execution model of II in Turing Plus.



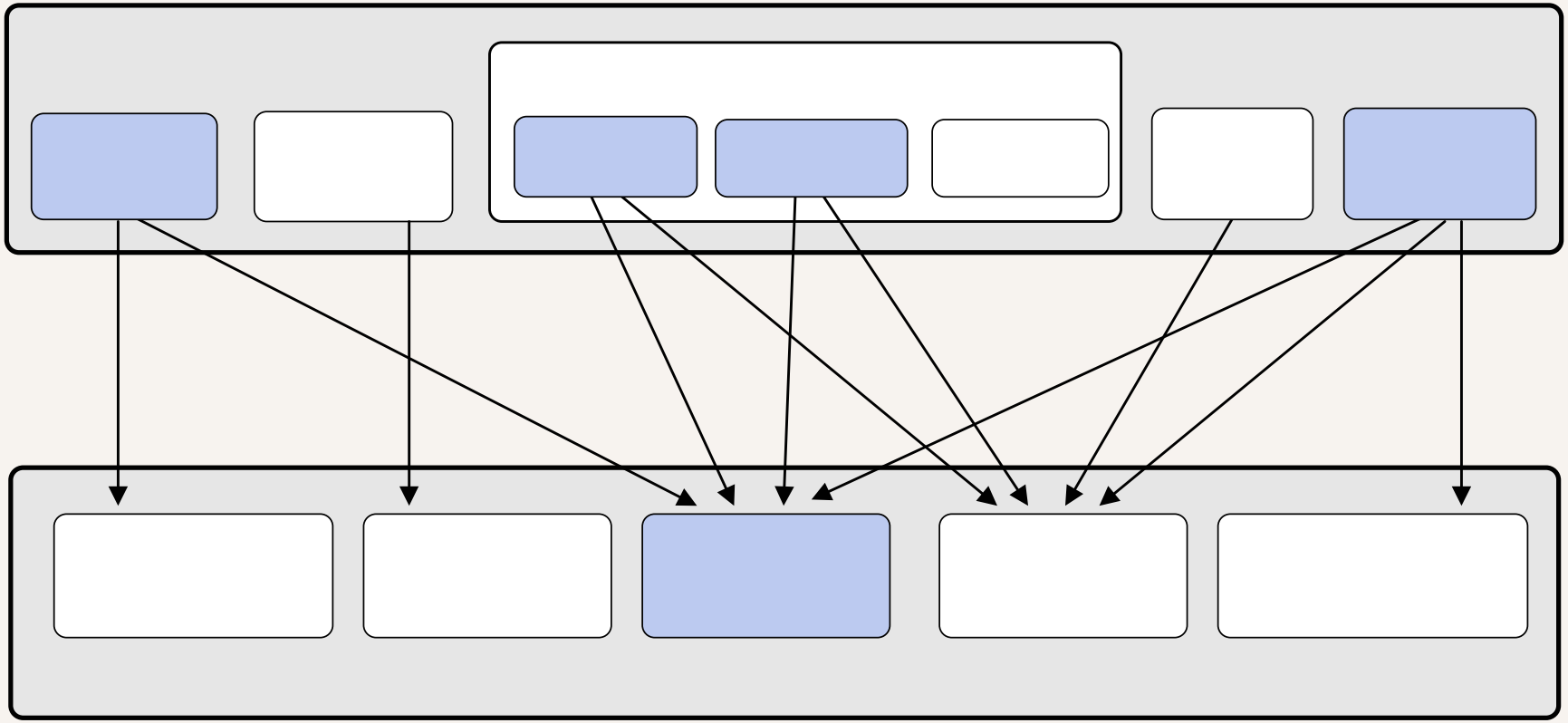
Testing IIL Programs

- The second challenge was transforming IIL to Turing Plus.



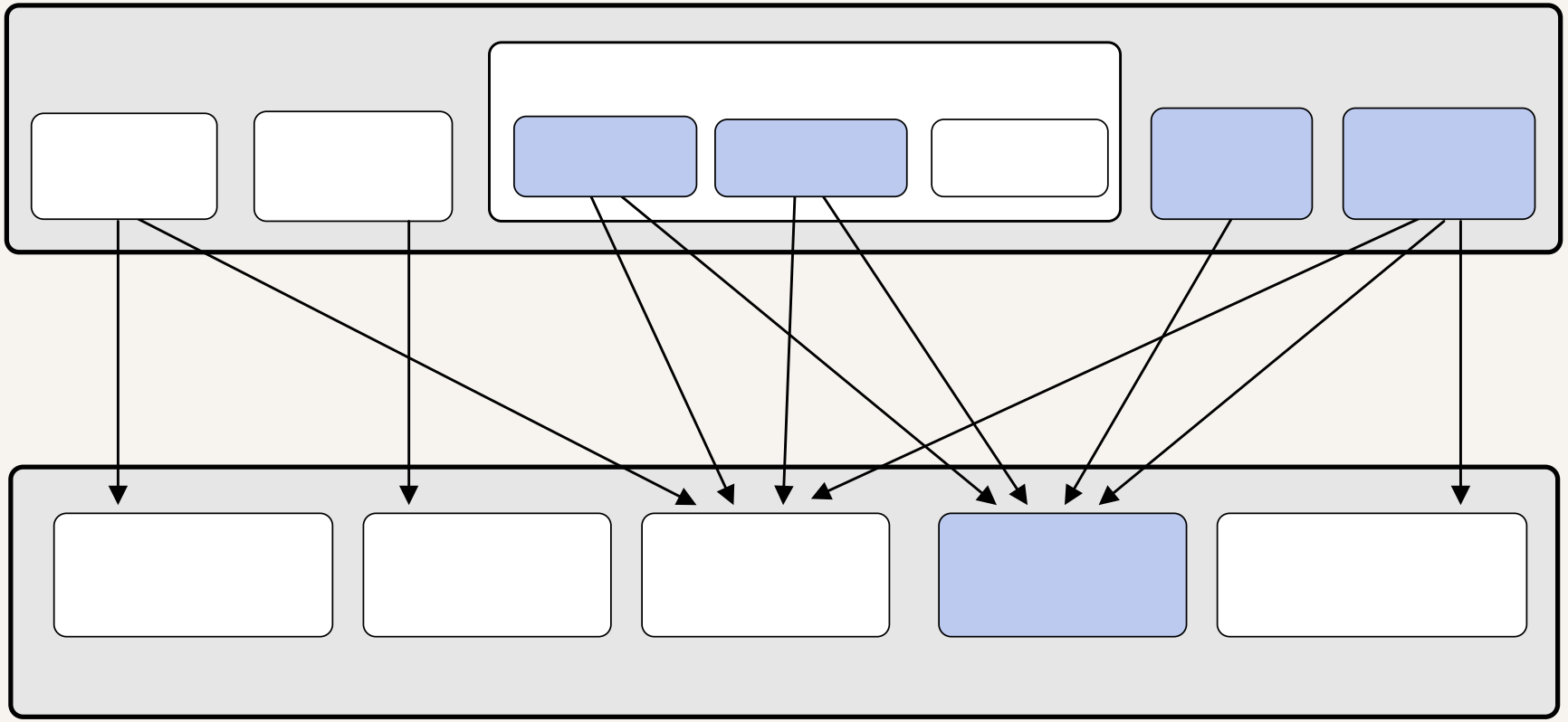
Testing IIL Programs

Component Transformation



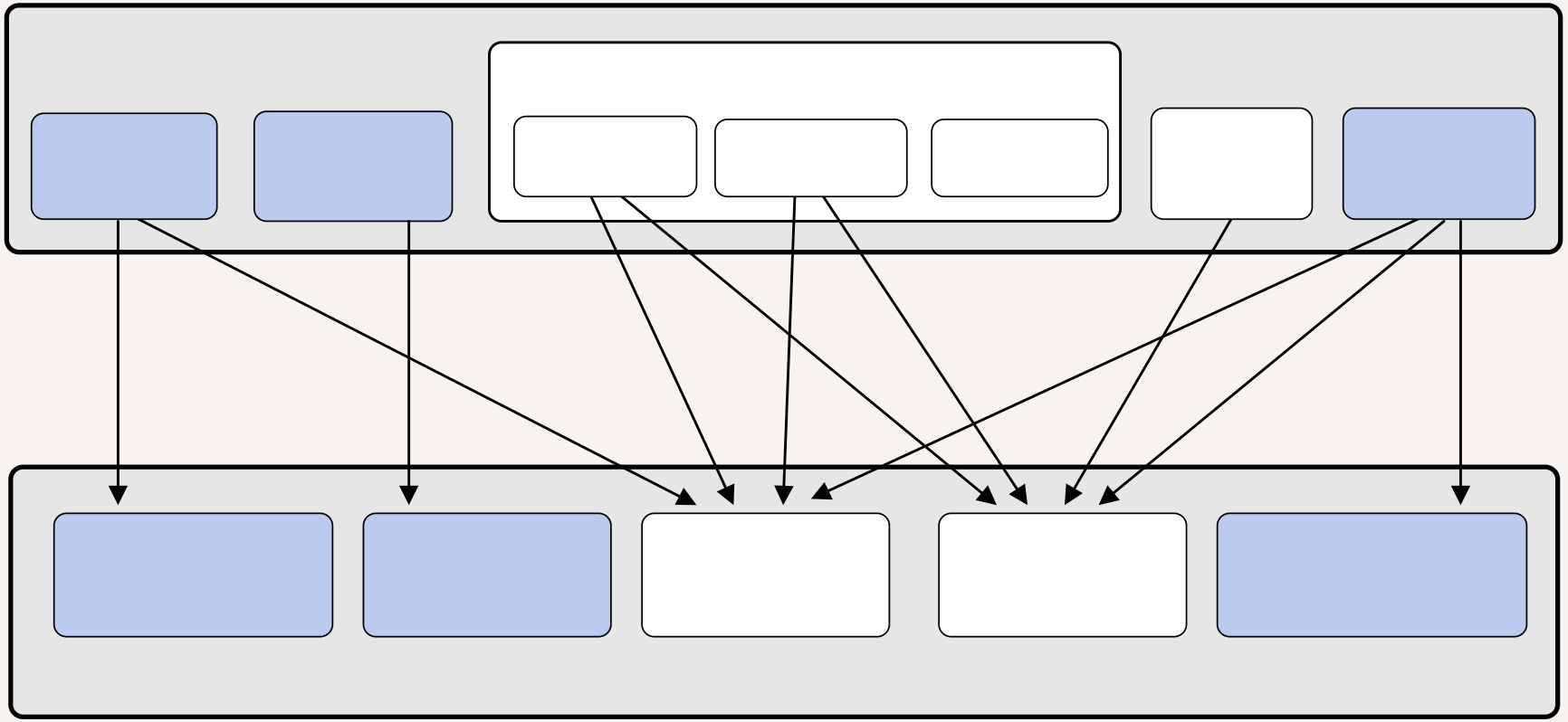
Testing IIL Programs

Dispatcher Transformation



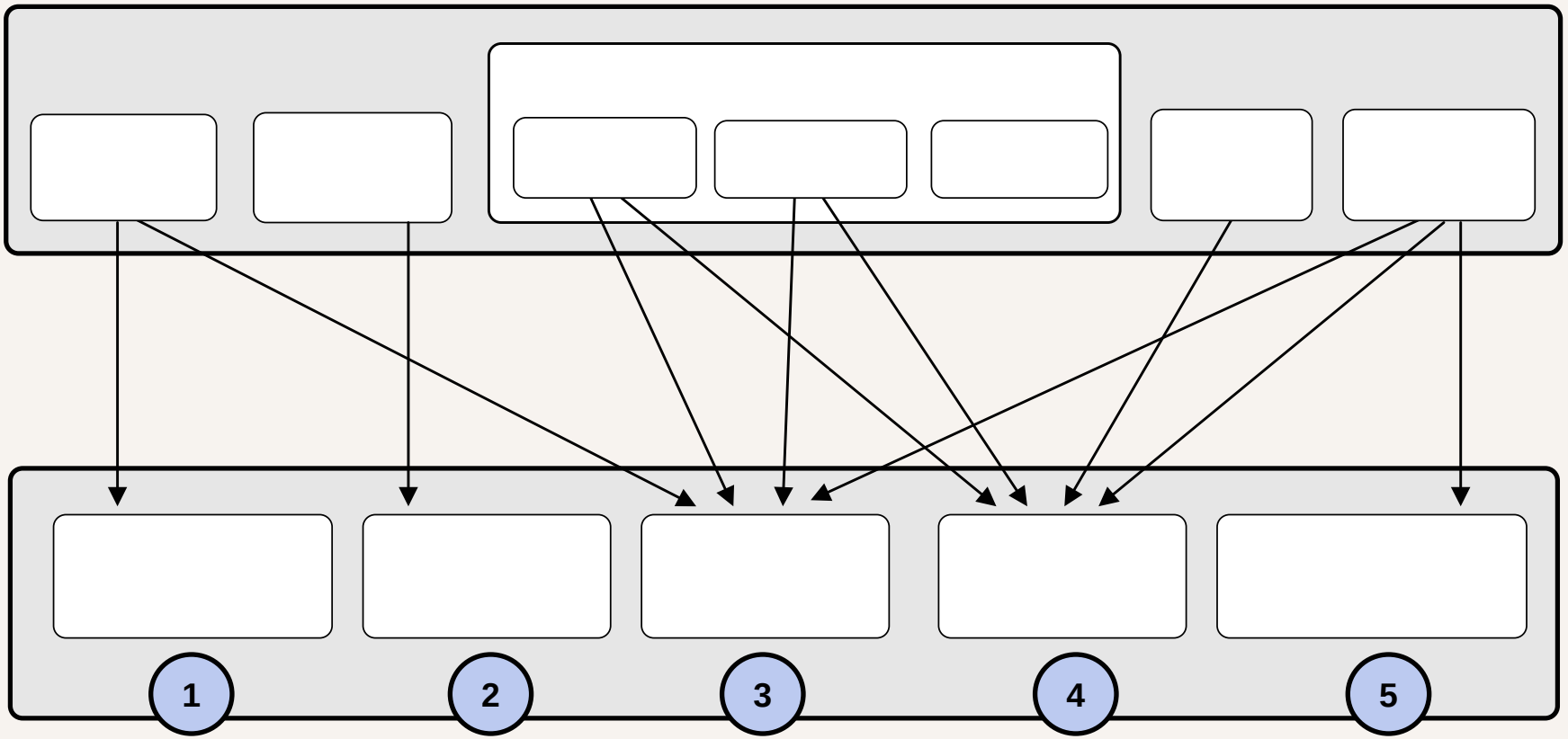
Testing IIL Programs

System and Environment Setup

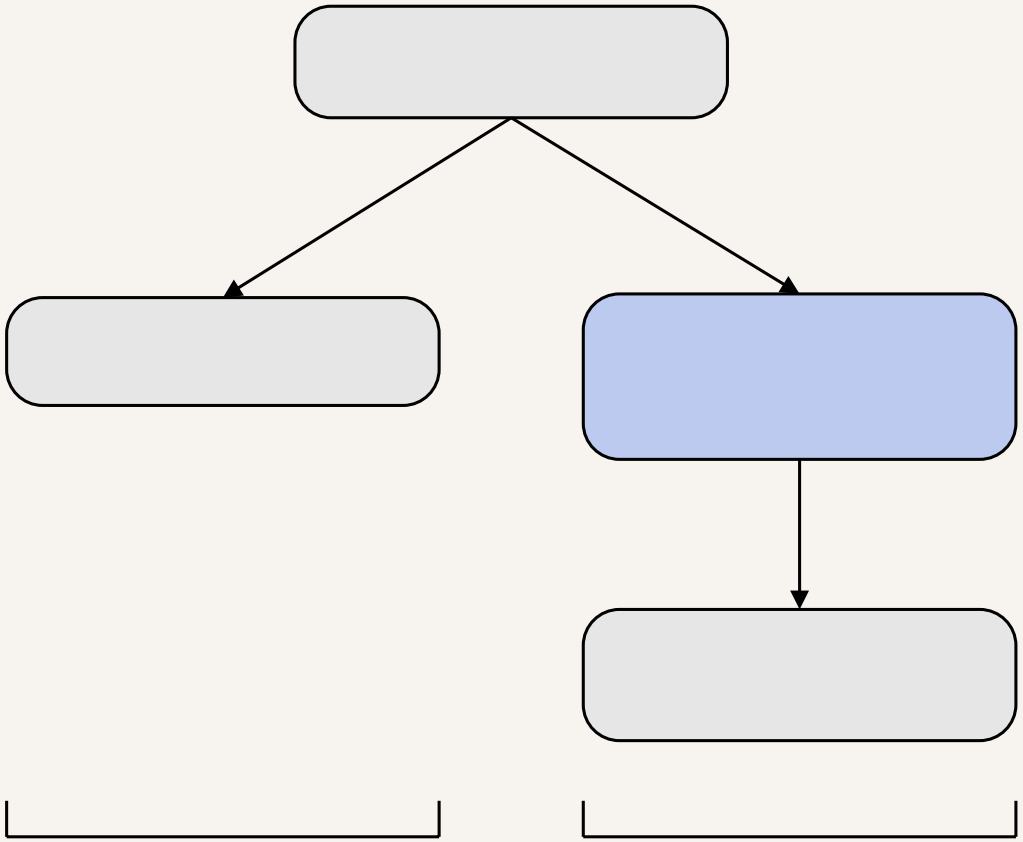


Testing IIL Programs

Re-Ordering of the System Body



Model Checking IIL Programs

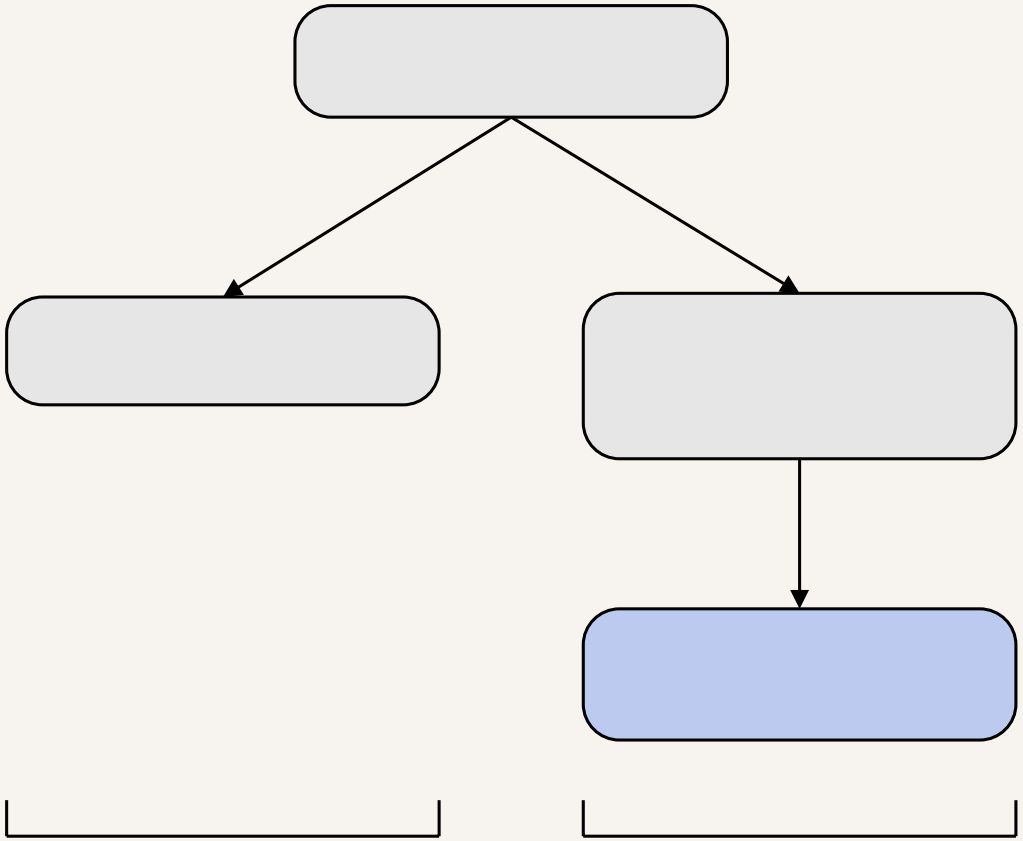


Model Checking IIL Programs

IIL to XML Transformation

- Uses another automatic transformation tool written in TXL.
- The transformation from IIL to XML involves two steps.
  - *Step 1:* the notational conveniences of IIL are removed and the program is reorganized to follow the program order required by the XML representation.
  - *Step 2:* the syntax translation from IIL to XML.

Model Checking IIL Programs



Model Checking IIL Programs

XML to SMV Transformation

- To model check systems written in IIL we use the existing approach we presented in [BD03].
  - This is an extension of work originally developed by Garlan, Khersonsky, and Kim in [GKK03].
- Uses an automatic transformation tool written in Java.
- Models can be checked using the Cadence SMV model checker.

[BD03] J. S. Bradbury and J. Dingel. Evaluating and improving the automatic analysis of implicit invocation system. In *Proc. of ESEC/FSE 2003*, pages 78–87, Sept. 2003.

[GKK03] D. Garlan, S. Khersonsky, and J. Kim. Model checking publish-subscribe systems. In *Proc. of the Int. SPIN Work. on Model Checking of Software*, May 2003.

Evaluation

- As an initial evaluation of our transformational framework we used three examples:
  - The Set-Counter example.
  - The Active Badge Location System (ABLS).
  - The Unmanned Vehicle Control System (UVCS).
- Errors in examples were discovered independently by testing and model checking.
- Examples written in Turing Plus typically 3-5 times larger and in SMV typically 5-10 times larger than IIL.

Conclusions

- We have presented a framework for specifying, testing, and model checking II systems.
  - A high-level language (IIL) for specifying II systems.
  - Two fully automatic, formally specified transformation tools:
    1. IIL to Turing Plus for execution and testing.
    2. IIL to SMV for model checking.
- We believe that our work provides a useful test bed for studying the relationship and possible synergies between testing and model checking.

Future Work - Evaluation

- To what extent can parallel testing be used to increase confidence in model checking results and the correctness of the model checker?
- How can testing be used to simplify or optimize model checking and vice-versa?
- Can we use similar transformation techniques to automatically derive run-time monitors for temporal safety properties?

Future Work – Generalization of II

- Currently we are restricted to a specific definition of II.
  - Centralized event dispatcher with static bindings.
- We would like to expand our framework to more general forms of publish-subscribe systems.
  - For example, II systems that support dynamic bindings, multiple dispatchers, and additional concurrency models.
- What other generalizations are interesting?

A Transformational Framework for Testing and Model Checking Implicit Invocation Systems

Hongyu Zhang, Jeremy S. Bradbury, James R. Cordy, Juergen Dingel

Software Technology Laboratory

School of Computing, Queen's University

Kingston, Ontario, Canada

DEBS '04

SUPPORTED BY NSERC CRSNG