OUTLINE

1. Introduction
2. Paper summary
3. Invariants on Compound Types in VDM++
4. Other topics
OUTLINE

1. Introduction
2. Paper summary
3. Invariants on Compound Types in VDM++
4. Other topics
BACKGROUND (MORTEN)

- C++ guy
  - Low-level details
  - References vs. values
  - Object lifetimes
- Love discussing software paradigms
- MSc. Computer Engineering
- Looking into PhD related to static analysis and tooling
MOTIVATION

- Different perspective
  - Providing an educational example and comparing modelling styles
  - Not a critical system
  - Not focusing on “proving Chess”
- Exploring capabilities of VDM++
  - Interesting bugs with VDM++
- Everyone knows Chess
  - Understandable
  - Complex
1. Introduction
2. Paper summary
3. Invariants on Compound Types in VDM++
4. Other topics
Chess modelled in VDM++
- Explored different paradigms
- Initially OOP but then FP
  - VDM-SL like
  - Composite types - immutable data
  - Why?
Figure 1: Overview of the model structure.
OUTLINE

1. Introduction
2. Paper summary
3. Invariants on Compound Types in VDM++
4. Other topics
Writing a Chess model with OO structure
Implementing move operation
Odd behaviour occurred during tests

Figure 2: Initial OOP structure. Operations of Piece and subclasses left out.
class Board

types

  public BoardState = set1 of Piece

  inv s == forall p1, p2 in set s & p1 <> p2 => p1.position <> p2.position;

instance variables

  public board_state : BoardState;

operations

  public move: Piece * Piece`Coordinate ==> ()

  move(piece, coord) == (
    let dead_piece = {p | p in set board_state & p.position = coord} in
    board_state := board_state \ dead_piece;
    piece.position := coord
  )

  pre piece in set board_state and coord in set piece.possible_moves(board_state);
class Board

types
  public BoardState = set1 of Piece
  inv s == forall p1, p2 in set s & p1 <> p2 => p1.position <> p2.position;

instance variables
  public board_state : BoardState;

operations
  public move: Piece * Piece\`Coordinate ==> ()
  move(piece, coord) == (let dead_piece = {p | p in set board_state & p.position = coord} in
    board_state := board_state \ dead_piece;
    piece.position := coord)

  pre piece in set board_state and coord in set piece.possible_moves(board_state);
class Board

types
    public BoardState = set1 of Piece
    inv s == forall p1, p2 in set s & p1 <> p2 => p1.position <> p2.position;

instance variables
    public board_state : BoardState;

operations
    public move: Piece * Piece`Coordinate ==> ()
    move(piece, coord) == (
        let dead_piece = {p | p in set board_state & p.position = coord} in
            board_state := board_state \ dead_piece;
            piece.position := coord
    )
    pre piece in set board_state and coord in set piece.possible_moves(board_state);
Seems fine, right?
EXECUTING MODEL

class Board
types
  public BoardState = set1 of Piece
  inv s == forall p1, p2 in set s & p1 <> p2 => p1.position <> p2.position;

instance variables
  public board_state : BoardState;

operations
  public move: Piece * Piece`Coordinate ==> ()
  move(piece, coord) == (
    let dead_piece = {p | p in set board_state & p.position = coord} in
    board_state := board_state \ dead_piece;
    piece.position := coord
  )
  pre piece in set board_state and coord in set piece.possible_moves(board_state);
OLD BEHAVIOUR

Debugging move:
1. dead_piece removed from board_state
2. Invariant for board_state checked
3. piece position updated
4. Invariant for board_state checked
   ▶ Since piece refers to an object inside board_state
5. BoardState invariant violated

Invariant was checked on board_state with dead_piece in it
The actions:

- Posted issue on GitHub
- More complex than anticipated
- Lead to discussion related to VDMJ internals
- Fixed within 14 days by Nick Battle
- *But then...*

Link to discussion:
https://github.com/overturetool/vdm-vscode/issues/197
NEW BEHAVIOUR

```java
< repos/BreakingVDM++ > java -jar ~/vdmj_test/vdmj/vdmj/target/vdmj-4.5.0-SNAPSHOT-230305.jar -vdmpp -i SetObject
Parsed 3 classes in 0.053 secs. No syntax errors
Error 3366: Cannot access state field 'x' from this context in 'Board' (SetObjectReference.vdmpp) at line 18:28
Error 3366: Cannot access state field 'x' from this context in 'Board' (SetObjectReference.vdmpp) at line 18:36
Warning 5001: Instance variable 'board_state' is not initialized in 'Board' (SetObjectReference.vdmpp) at line 21:5
Type checked 3 classes in 0.157 secs. Found 2 type errors and 1 warning
Bye
```

Figure 3: New behaviour after fixing the issue.

Direct field access from functions (such as `inv.BoardState`) now prohibited
THE UNDERLYING ISSUE

VDM++ objects are references:

- Reference types vs. value types
- Mutable vs. immutable
- Aliasing

Some options with invariants\(^1\) on compound types of references:

1. Check invariant whenever an object that is referred to changes state
2. **Prohibit such invariants**

\(^1\) Similar points with to pre- and postconditions
How can we express the invariant?
Figure 4: Previous structure of the model.

Figure 5: New structure of the model. Essentially a VDM-SL specification.
class BM -- BoardModule

types

  public Board = set1 of PM`Piece

inv s == forall p1, p2 in set s & p1 <> p2 => p1.position <> p2.position;

functions

  public move: Board * PM`Piece * PM`Coordinate -> Board

move(board, piece, coord) == (

  let dead_piece = {p | p in set board & p.position = coord} in

  (board \ (dead_piece union {piece})) union

  {mk_PM`Piece(piece.type, coord, piece.color)}

)  

pre piece in set board_state and coord in set PM`possible_moves(piece, board_state);
class BM -- BoardModule

types
    public Board = set1 of PM`Piece
    inv s == forall p1, p2 in set s & p1 <> p2 => p1.position <> p2.position;

functions
    public move: Board * PM`Piece * PM`Coordinate -> Board
    move(board, piece, coord) == (let dead_piece = {p | p in set board & p.position = coord} in
        (board \ (dead_piece union {piece})) union
        {mk_PM`Piece(piece.type, coord, piece.color})
    )
    pre piece in set board_state and coord in set PM`possible_moves(piece, board_state);
class BM -- BoardModule

types
  public Board = set1 of PM`Piece
  inv s == forall p1, p2 in set s & p1 <> p2 => p1.position <> p2.position;

functions
  public move: Board * PM`Piece * PM`Coordinate -> Board
  move(board, piece, coord) == (
    let dead_piece = {p | p in set board & p.position = coord} in
    (board \ (dead_piece union {piece})) union
    {mk_PM`Piece(piece.type, coord, piece.color})
  )
  pre piece in set board_state and coord in set PM`possible_moves(piece, board_state);
The principles transfer

Reasoning about a functional model:
- Referential transparency
- No global state
- (Arguably) easier to test

Downsides:
- Difficult to model stateful aspects - e.g. “castling”
- (Arguably) less readable
OUTLINE

1. Introduction
2. Paper summary
3. Invariants on Compound Types in VDM++
4. Other topics
Further topics of interest:

- Castling and the importance of real-world data
- Different testing techniques
- Implementing simple moves
- String manipulation for PGN
Questions?

Morten Haahr Kristensen, Peter Gorm Larsen
201807664@post.au.dk, pgl@ece.au.dk
PlantInv: set of Alarm * map Period to set of Expert -> bool

PlantInv(as,sch) ==
(forall p in set dom sch & sch(p) <> {}) and
  (forall a in set as &
   forall p in set dom sch &
    exists expert in set sch(p) &
    a.GetReqQuali() in set expert.GetQuali());

--
a.GetReqQuali() in set expert.quali