Junta

Brætspilsprogrammering

Kent M. C. Sebastian W. Elias K. O.
Simon B. J. Niels S. P. Martin B. M.

28. juni 2013

Disposition

1. Syntaktisk analyse
2. Kontekstuelle begrænsninger
3. Fortolker
4. Indbyggede typer og konstanter
5. Game Abstraction Layer
6. Simulator
7. Præsentation af spil, evaluering af krav og perspektivering

Del 1

Syntaktisk analyse

Leksikalsk analyse

- Lexemes $\rightarrow$ Tokens
- Ingen specific syntaks
- To klasser: Scanner og Token
  - Diverse scan-metoder
  - (**Type** tokenType, **String** value, **int** line, **int** offset)

Leksikalsk analyse

```
1  public Token scan() throws Exception {
2    while (isWhitespace()) { pop(); }
3    if (isEof()) { return token(Type.EOF); }
4    if (isDigit()) { return scanNumeric(); }
5    if (isUppercase()) { return scanUppercase(); }
6    if (isOperator()) { return scanOperator(); }
7    if (isLowercase()) { return scanKeyword(); }
8    if (current() == '"') { return scanString(); }
9    if (current() == '$') { return scanVar(); }
10   throw new ScannerError("Unidentified character: " + current(), token(Type.EOF));
11 }
```

Leksikalsk analyse

```
// An implementation of the traditional
// Noughts and Crosses game
type NacGame[] extends Game["Noughts and Crosses"] {
  define players = [
    NacPlayer[Crosses, "Crosses"],
    NacPlayer[Noughts, "Noughts"]
  ]
  define initialBoard = GridBoard[3, 3]
}
type NacPlayer[$pieceType, $name] extends Player[$name] {
  define winCondition[$gameState] =
    $gameState.findSquares[
      /friend (n friend n) | (e friend e) |
        (nw friend nw) | (ne friend ne ) friend/].size != 0
  define tieCondition[$gameState] =
    $gameState.board.isFull
  define actions[$gameState] =
  addActions[$pieceType[this], $gameState.board.emptySquares]
}
type Crosses[$owner] extends Piece[$owner]
type Noughts[$owner] extends Piece[$owner]
```

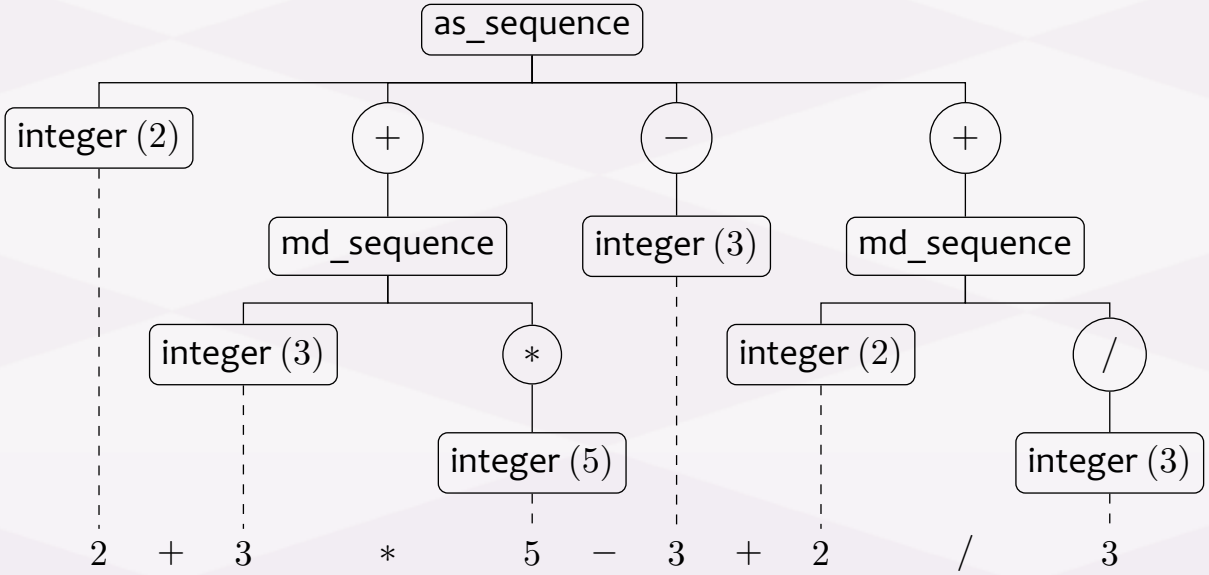
Grammatik

- Kontekstfri grammatik
- Understøttelse af LL-parsere
  - Strategi for parsing: Top-down
  - Ikke tvetydig
  - Ingen venstre-rekursion
- Systematisk opbygning (automatisk generering)
- Mulighed for hierarkisk opbygning (operatorpræcedens)

Grammatik

Niveau	Operatorpræcedens	
	Operator	Beskrivelse
1	`f[]`	Funktions- og listekald
2	`r.m r.m[]`	Metodekald
3	`-`	Unær negationsoperator
4	`* / %`	Multiplikation, division og modulo
5	`+ -`	Addition and subtraktion
6	`< > <= >=`	Sammenligningsoperatorer
7	`== != is`	Lighedsoperatorer og typekontrol
8	`and or`	Logisk og og $eller$
9	`not`	Logisk $ikke$
10	`if let set #`	if-, let-, set- og lambdaudtryk

Grammatik


as_sequence
integer (2)
+
md_sequence
integer (3)
*
integer (5)
−
integer (3)
+
md_sequence
integer (2)
/
integer (3)
2 + 3 * 5 − 3 + 2 / 3

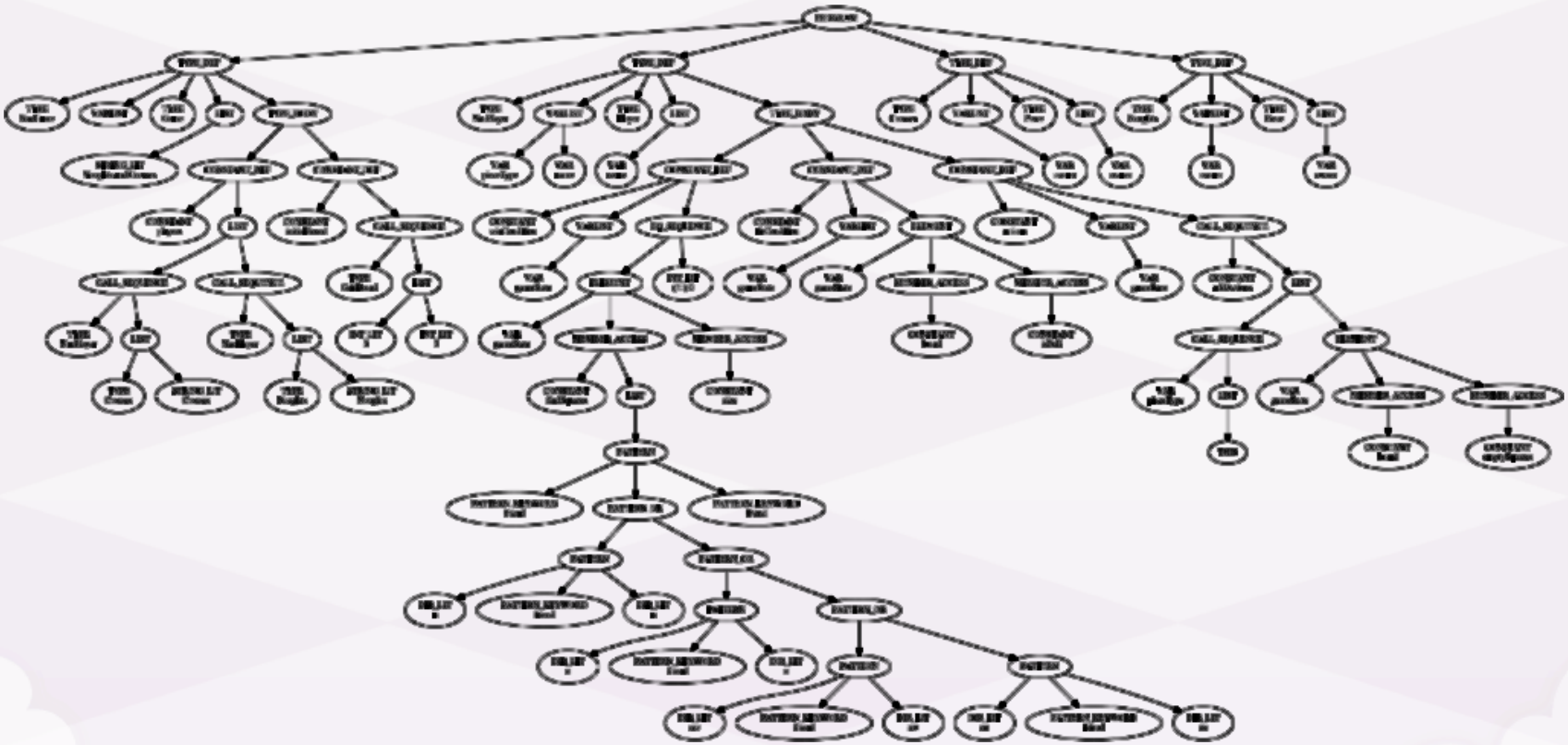
En Recursive Descent parser

- Håndskrevet parser vs. genereret parser
  - Automatisk generering
  - Tidskrævende
  - Pålidelighed
  - Skræddersy parseren
  - Bedre forståelse for opbygning af parsere
  - "Black box"
- Recursive descent, LL(1)-parser
- To klasser: Parser og AstNode
  - Diverse metodekald $\rightarrow$ et abstrakt syntakstræ
  - (**Type** type, **String** value, **int** line, **int** offset)

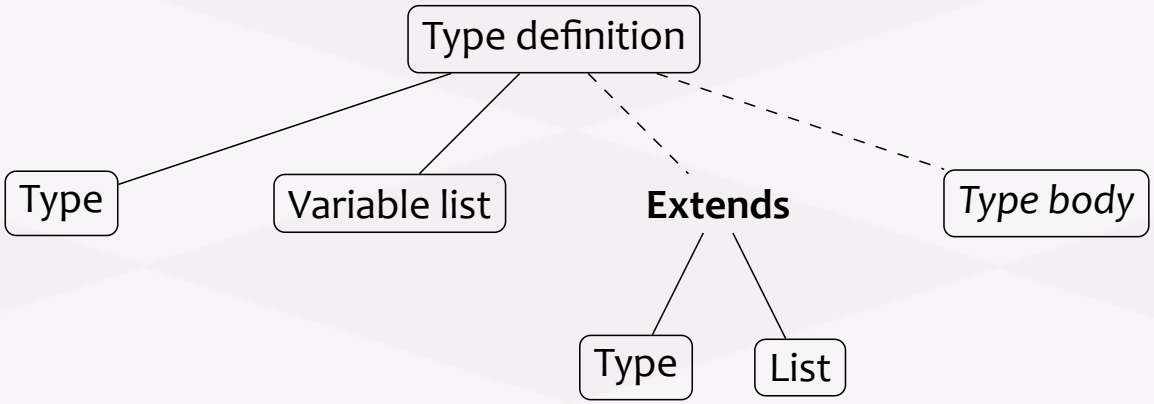
En Recursive Descent parser

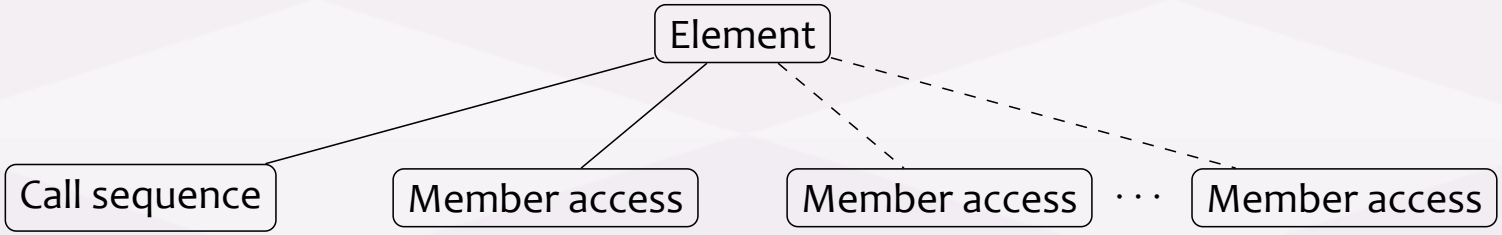
```
// An implementation of the traditional
// Noughts and Crosses game
type NacGame[] extends Game["Noughts and Crosses"] {
  define players = [
    NacPlayer[Crosses, "Crosses"],
    NacPlayer[Noughts, "Noughts"]
  ]
  define initialBoard = GridBoard[3, 3]
}
type NacPlayer[$pieceType, $name] extends Player[$name] {
  define winCondition[$gameState] =
    $gameState.findSquares[
      /friend (n friend n) | (e friend e) |
        (nw friend nw) | (ne friend ne ) friend/].size != 0
  define tieCondition[$gameState] =
    $gameState.board.isFull
  define actions[$gameState] =
  addActions[$pieceType[this], $gameState.board.emptySquares]
}
type Crosses[$owner] extends Piece[$owner]
type Noughts[$owner] extends Piece[$owner]
```

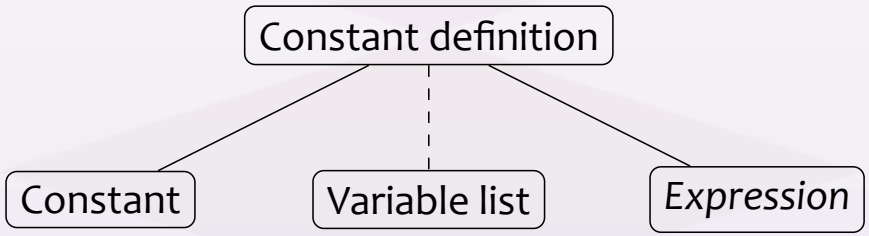
Det Abstrakte syntakstræ



Det Abstrakte syntakstræ



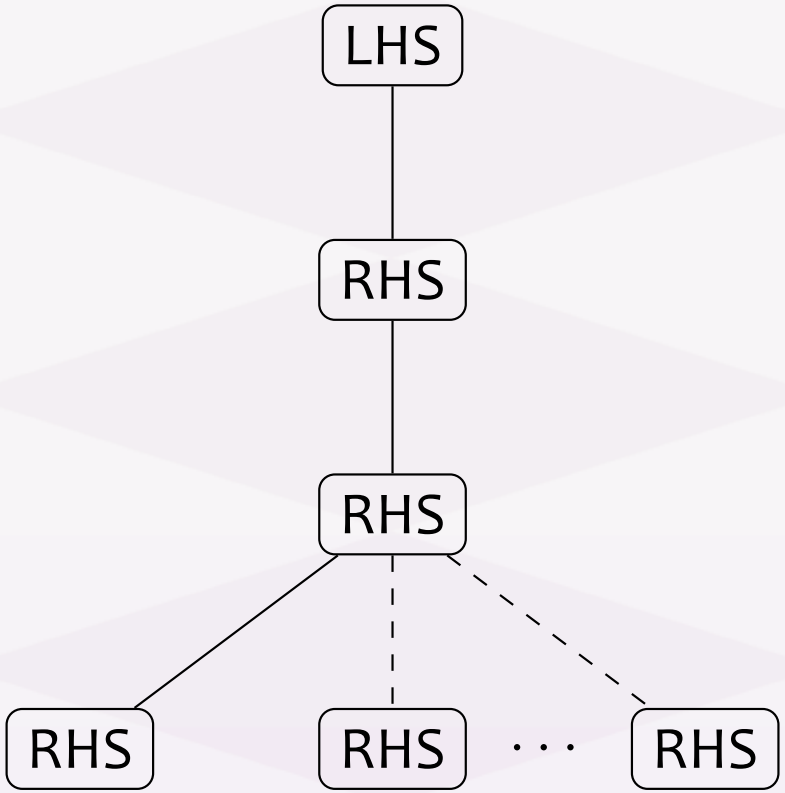




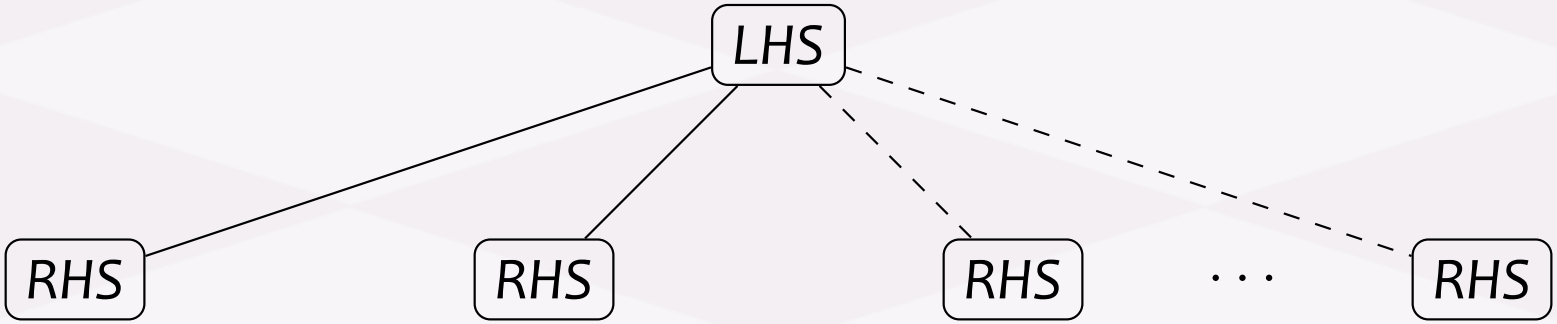
Det Abstrakte syntakstræ

- Vokser meget hurtigt
- Komprimere træet så godt som muligt
  - Fjerne unødvendige knuder (whitespace, kommentarer…)
  - Gør det videre arbejde med træet nemmere

Det Abstrakte syntakstræ



Det Abstrakte syntakstræ



Del 2

Kontekstuelle begrænsninger

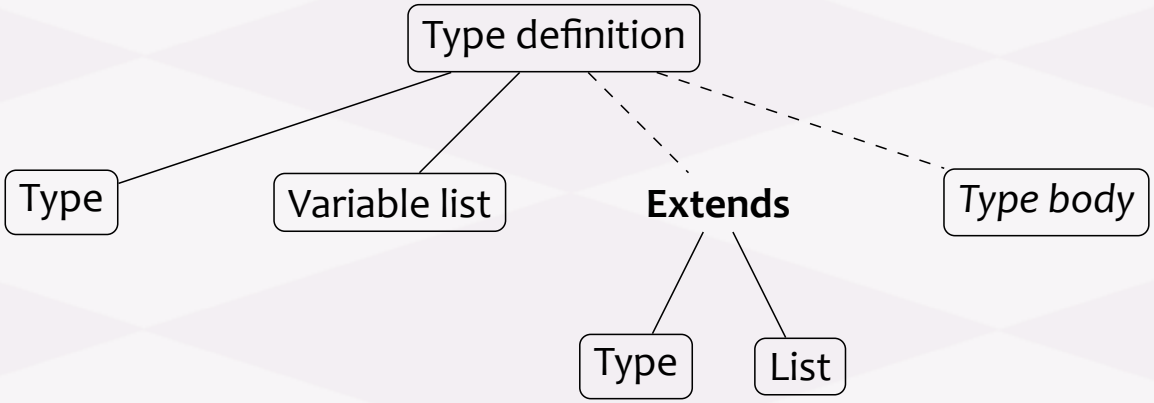
Kontekstuelle begrænsninger

- Hvad er kontekstuelle begrænsninger?
- Hvad kræver Junta?
- ScopeChecker

Typer

```
define foo = A[].bar
type A[]{
   define bar = 10
}
```

- Anvendte typer kan bindes til én og kun én erklæring
- Typer har de medlemmer, der tilgås


Type definition
Type
Variable list
Extends
Type body
Type
List

Typer

```
define foo = A[].bar[3, 7]
type A[]{
   define bar = 10
}
```

- Tjekke korrespondance mellem formelle / aktuelle parametre

Typer

```
define foo = let $a = B[]
             in $a.bar
type A[]{
   define bar = 10
}
type B[]{
}
```

- Inferere en variabels type

Typer

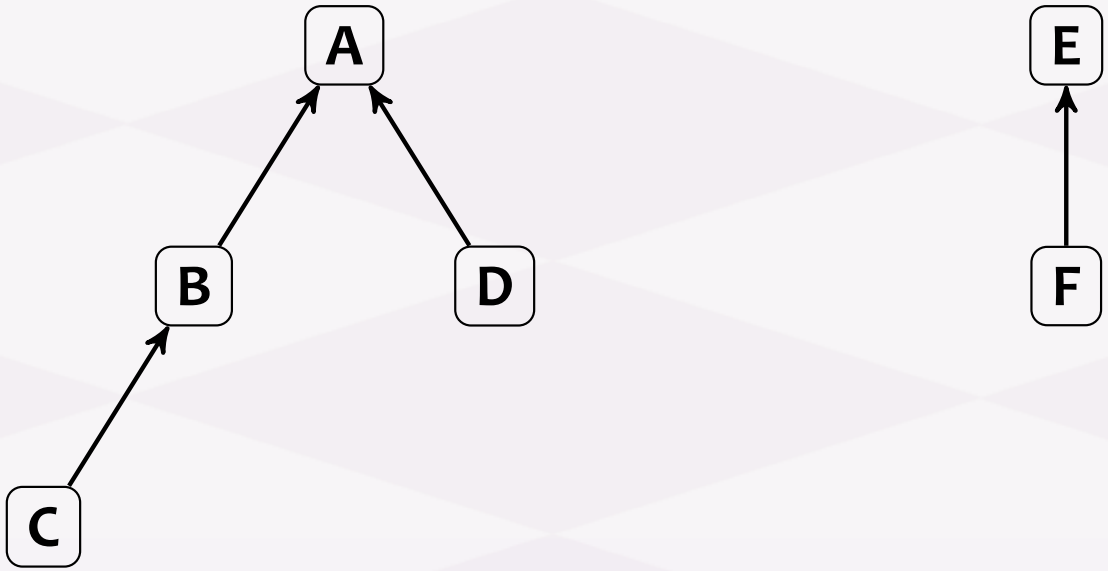
```
define someobject = A[1, 2]
type A[$a, $b]
type B[] extends A[1, 2]
```

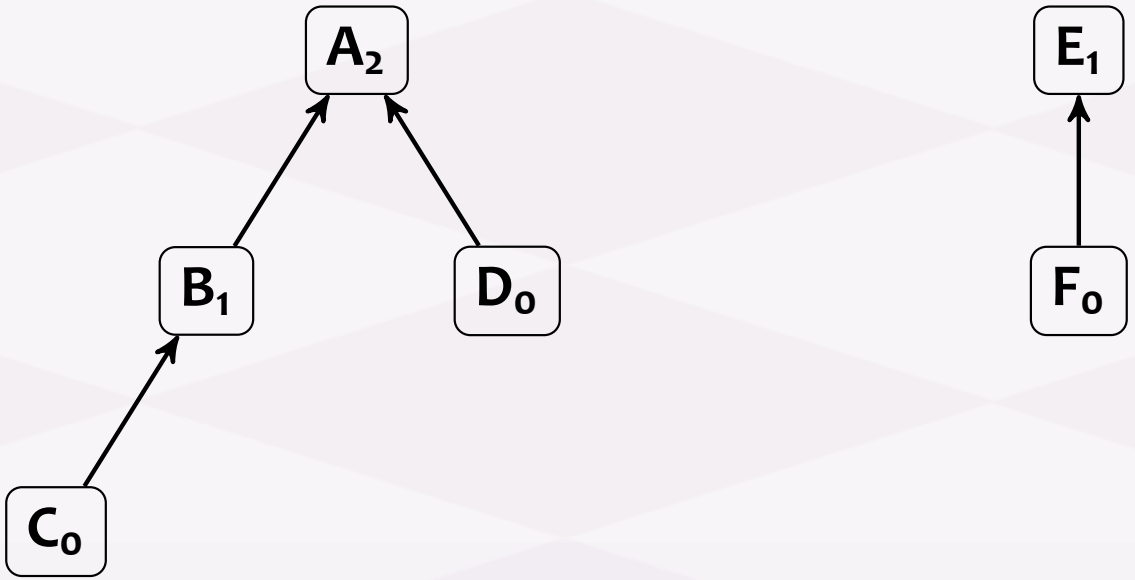
- Instantiering foregår med korrekt antal parametre
- Subtype kalder supertype med korrekt antal parametre
- Supertype eksisterer!

Typer

```
type A[] extends B[]
type B[] extends C[]
type C[] extends A[]
```

- Ingen cyklisk nedarvning


A
B
D
C
E
F

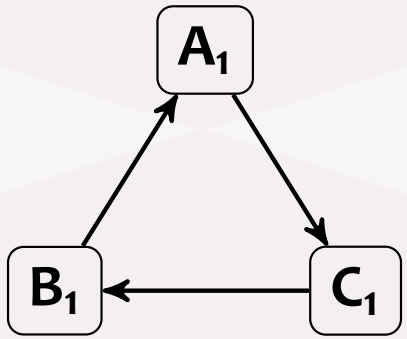

A_2
B_1
D_0
C_0
E_1
F_0

- Topologisk sorteret sekvens: C, D, F

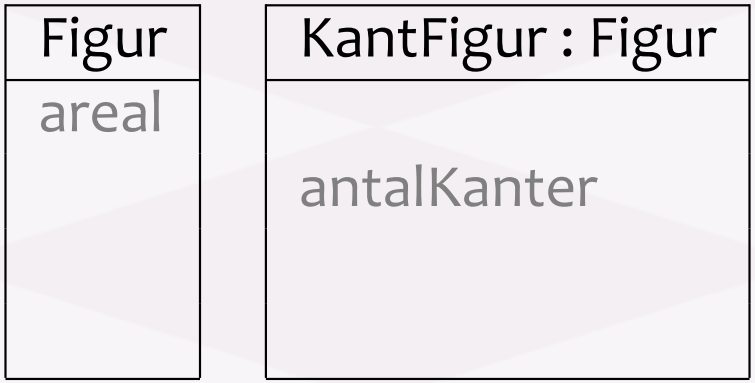
A_0

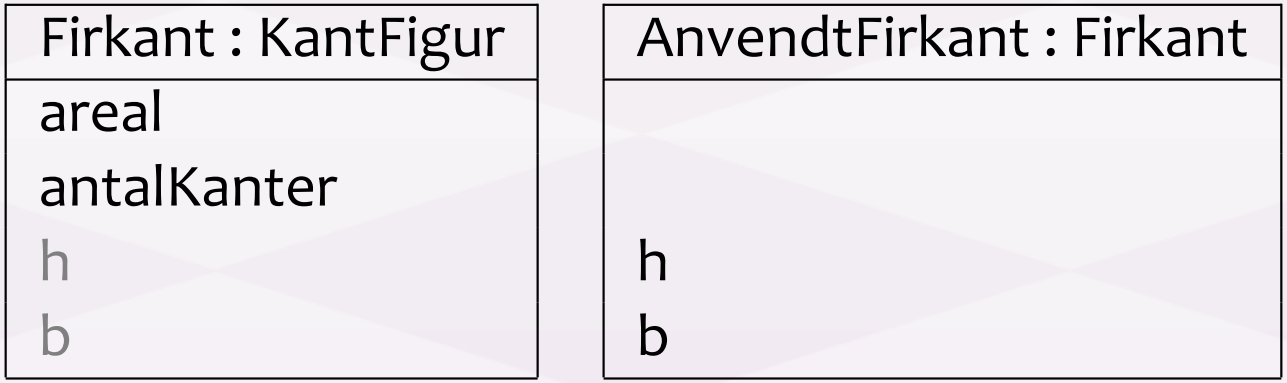
- Topologisk sorteret sekvens: C, D, F, B, E

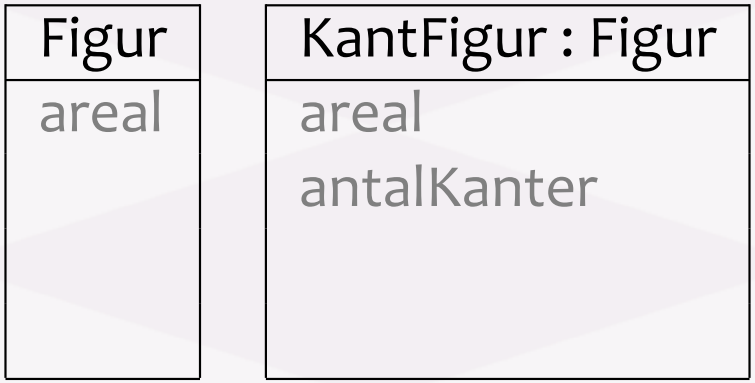
- Topologisk sorteret sekvens: C, D, F, B, E, A

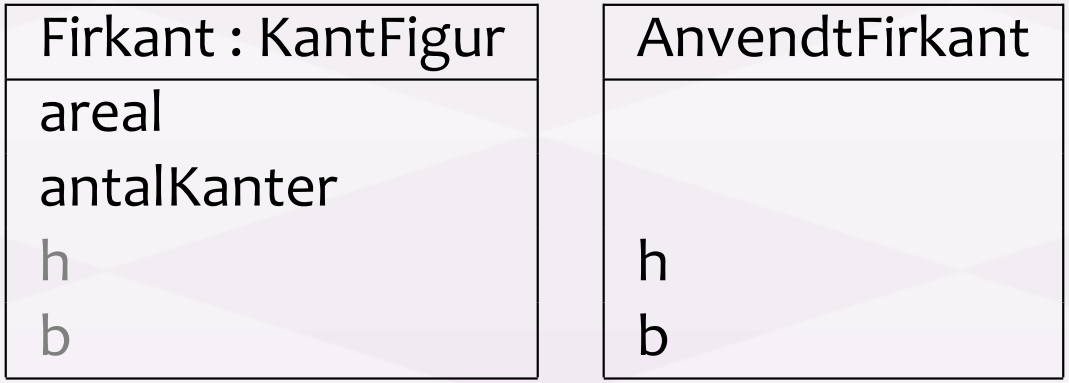


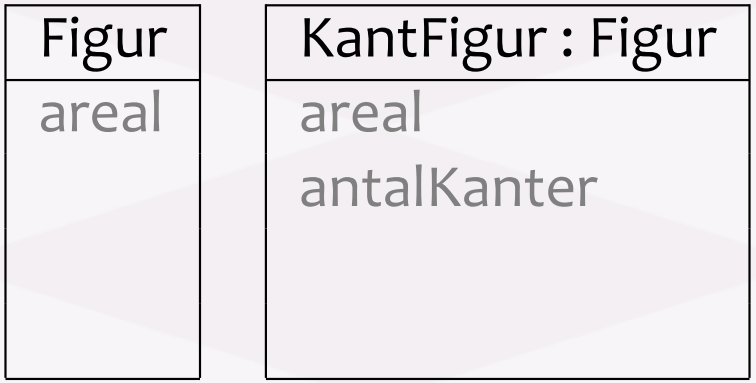
- Omvendt: A, E, B, F, D, C

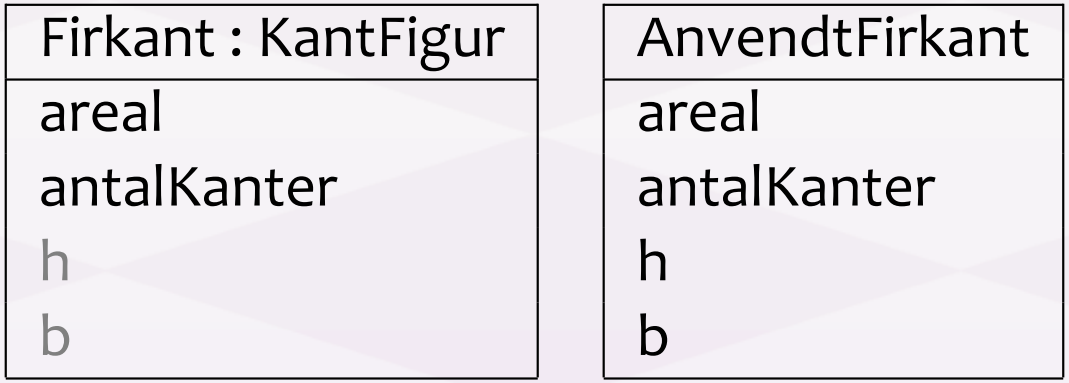

Figur
areal
KantFigur : Figur
antalKanter

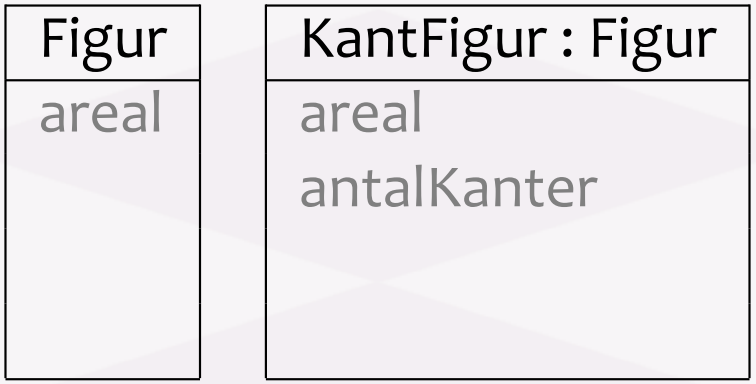

Firkant : KantFigur
areal
antalKanter
h
b
AnvendtFirkant : Firkant
h
b

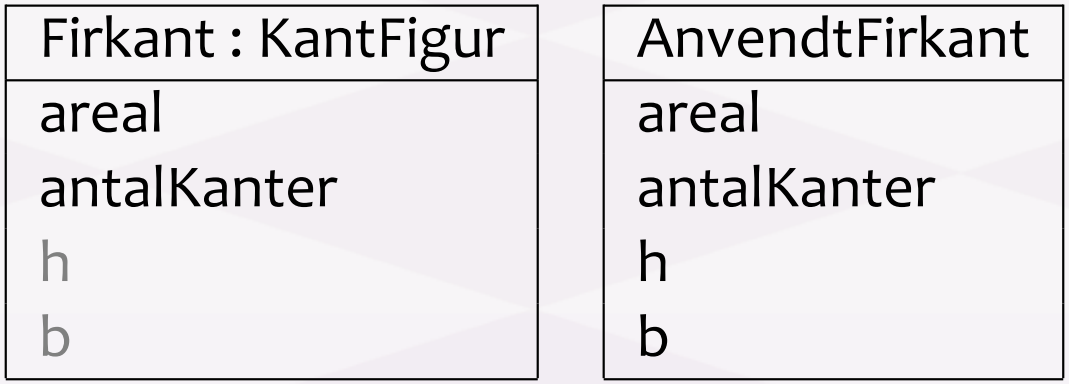

Figur
areal
KantFigur : Figur
areal
antalKanter


Firkant : KantFigur
areal
antalKanter
h
b
AnvendtFirkant
h
b


Figur
areal
KantFigur : Figur
areal
antalKanter


Firkant : KantFigur
areal
antalKanter
h
b
AnvendtFirkant
areal
antalKanter
h
b





`AnvendtFirkant[].areal` → ok

`Firkant[].h` → ikke tilladt

Abstrakte typer identificeres

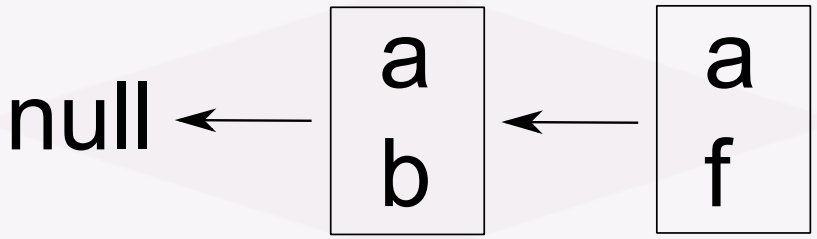
Variabler

```
let $a = 2, $b = 3
   in ...

add[$a, $b] = ...

#[$a, $b] => ...

type A[]{
   data $a
   data $b
   ...
}
```

- Åbner nyt scope
- Variabler tilføjes til nuværende scope

VARIABLER

```
let $a = 2, $b = 3 in
   let $a = 5, $f = 7 in
      $a + $b + $f
```



- Variabel erklæres, som allerede findes i aktivt scope = ScopeError

Del 3

Fortolker

Opbygning af fortolker

- Visitormønstret
- Valuetyper
  - Funktioner
  - Patterns
  - Typer
  - $\cdots \times 19$
- Symboltabel
- Game environment
- Interne hjælpeklasser og interfaces
- Optimeringer

Opbygning af fortolker

- Visitormønstret
- Valuetyper
  - Funktioner
  - Patterns
  - Typer
  - $\cdots \times 19$
- Symboltabel
- Game environment
- Interne hjælpeklasser og interfaces
- Optimeringer

Opbygning af fortolker

- Visitormønstret
- Valuetyper
  - Funktioner
  - Patterns
  - Typer
  - $\cdots \times 19$
- Symboltabel
- Game environment
- Interne hjælpeklasser og interfaces
- Optimeringer

Opbygning af fortolker

- Visitormønstret
- Valuetyper
  - Funktioner
  - Patterns
  - Typer
  - $\cdots \times 19$
- Symboltabel
- Game environment
- Interne hjælpeklasser og interfaces
- Optimeringer

Opbygning af fortolker

- Visitormønstret
- Valuetyper
  - Funktioner
  - Patterns
  - Typer
  - $\cdots \times 19$
- Symboltabel
- Game environment
- Interne hjælpeklasser og interfaces
- Optimeringer

Opbygning af fortolker

- Visitormønstret
- Valuetyper
  - Funktioner
  - Patterns
  - Typer
  - $\cdots \times 19$
- Symboltabel
- Game environment
- Interne hjælpeklasser og interfaces
- Optimeringer

Symboltabel

- Brug
- Opslag og lagring
- Scopes

Symboltabel

- Brug
- Opslag og lagring
- Scopes

```
symbolTable.openScope()
symbolTable.closeScope()
symbolTable.addConstant(name, value)
symbolTable.addVariable(name, value)
symbolTable.addType(name, value)
symbolTable.getType(name)
symbolTable.getVariable(name)
symbolTable.getConstant(name)
symbolTable.getThis()
```

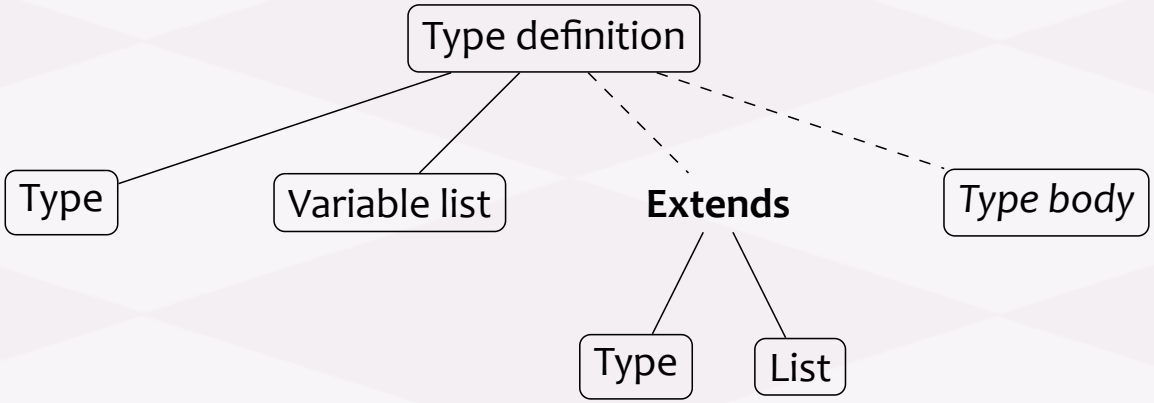
Symboltabel

- Brug
- Opslag og lagring
- Scopes

```
currentScope.addVariable(name, value)
currentScope.getVariable(name)
currentScope.getThis()
currentScope.getParent()
```

Typedefinitioner

AST



- Valgfri underknuder
- Håndtering af hver knude

Typedefinitioner

Kode

```
protected Value visitTypeDef(AstNode node) throws StandardError {
  TypeValue type;
  String name = node.getFirst().value;
  if (node.size() > 3) {
    type = new TypeValue(name, node.get(1), node.get(2).value, node.get(3));
  }
  else {
    type = new TypeValue(name, node.get(1));
  }
  if (node.getLast().type == AstNode.Type.TYPE_BODY) {
    for (AstNode defNode : node.getLast()) {
      if (defNode.type == Type.DATA_DEF) {
        type.addAttribute(defNode.getFirst().value, new Member(defNode));
      }
      else {
        type.addTypeMember(defNode.getFirst().value, new Member(defNode));
      }
    }
  }
  symbolTable.addType(name, type);

  return null;
}
```

Del 4

Indbyggede typer og konstanter

Programmørens omgivelser

- Formål: At implementere brugbare funktioner og typer
- Standard Environment
- Game Environment

Standard Environment

Globale konstanter

- `typeOf[]` er en funktion, som returnerer et givet objekts type
- `union[]` er en funktion, som returnerer foreningsmængden af to eller flere lister
- `true` og `false` er boolske konstanter

Standard Environment

Globale konstanter

- `typeOf[]` er en funktion, som returnerer et givet objekts type
- `union[]` er en funktion, som returnerer foreningsmængden af to eller flere lister
- `true` og `false` er boolske konstanter

Standard Environment

Globale konstanter

- `typeOf[]` er en funktion, som returnerer et givet objekts type
- `union[]` er en funktion, som returnerer foreningsmængden af to eller flere lister
- `true` og `false` er boolske konstanter

Standard Environment

Simple typer

- `Integer`: 32-bit heltal
- `Boolean`: Sandhedsværdi
- `String`: Unicode tekststreng

Standard Environment

Lister

- En ordnet liste af vilkårlige objekter: `[2, "hej", true]`
- Kan være tom: `[]`
- `.size` er listens størrelse
- Listen kan sorteres med `.sort[]`
- `.map[]` udfører en funktion på alle listens elementer
- `.filter[]` returnerer de elementer som opfylder et kriterium

Standard Environment

Lister

- En ordnet liste af vilkårlige objekter: `[2, "hej", true]`
- Kan være tom: `[]`
- `.size` er listens størrelse
- Listen kan sorteres med `.sort[]`
- `.map[]` udfører en funktion på alle listens elementer
- `.filter[]` returnerer de elementer som opfylder et kriterium

Standard Environment

Lister

- En ordnet liste af vilkårlige objekter: `[2, "hej", true]`
- Kan være tom: `[]`
- `.size` er listens størrelse
- Listen kan sorteres med `.sort[]`
- `.map[]` udfører en funktion på alle listens elementer
- `.filter[]` returnerer de elementer som opfylder et kriterium

Standard Environment

Lister

- En ordnet liste af vilkårlige objekter: `[2, "hej", true]`
- Kan være tom: `[]`
- `.size` er listens størrelse
- Listen kan sorteres med `.sort[]`
- `.map[]` udfører en funktion på alle listens elementer
- `.filter[]` returnerer de elementer som opfylder et kriterium

Standard Environment

Lister

- En ordnet liste af vilkårlige objekter: `[2, "hej", true]`
- Kan være tom: `[]`
- `.size` er listens størrelse
- Listen kan sorteres med `.sort[]`
- `.map[]` udfører en funktion på alle listens elementer
- `.filter[]` returnerer de elementer som opfylder et kriterium

Standard Environment

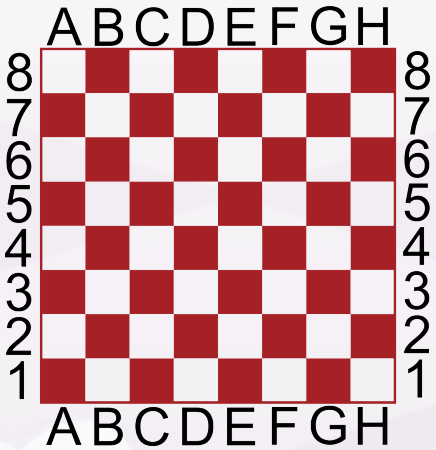
Lister

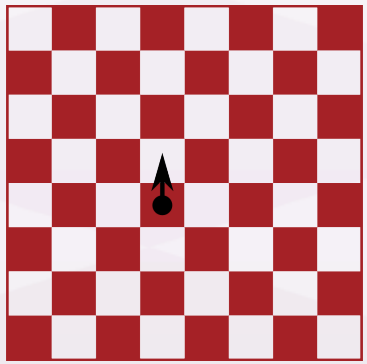
- En ordnet liste af vilkårlige objekter: `[2, "hej", true]`
- Kan være tom: `[]`
- `.size` er listens størrelse
- Listen kan sorteres med `.sort[]`
- `.map[]` udfører en funktion på alle listens elementer
- `.filter[]` returnerer de elementer som opfylder et kriterium

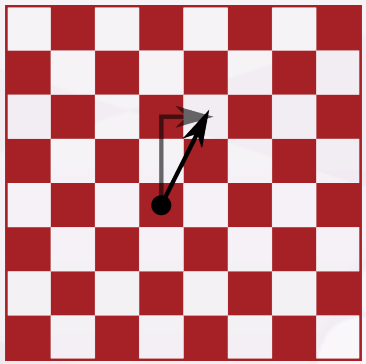
Standard Environment

Simple brætspilsrelaterede typer

- `Coordinate` er en vektor som repræsenterer et felt på et bræt: `C5` og `M27`
- `Direction` er en vektor som repræsenterer et flyt: `n`, `nw` og `n + ne`
- `Pattern` er et mønster



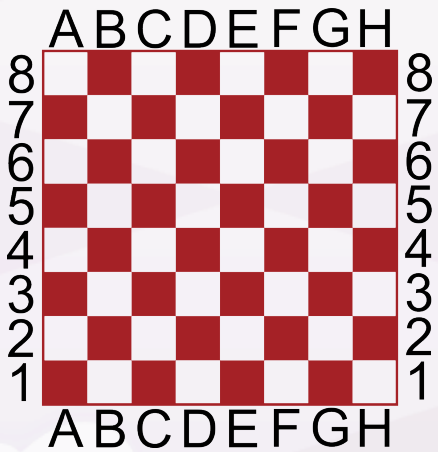


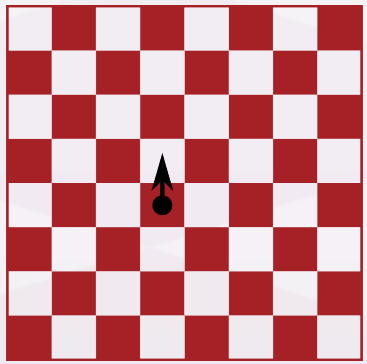


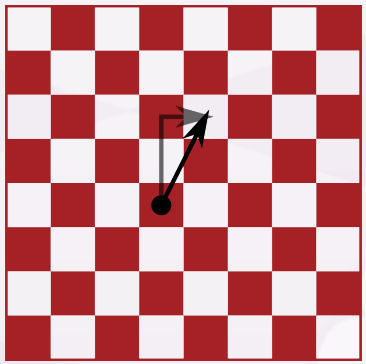
Standard Environment

Simple brætspilsrelaterede typer

- `Coordinate` er en vektor som repræsenterer et felt på et bræt: `C5` og `M27`
- `Direction` er en vektor som repræsenterer et flyt: `n`, `nw` og `n + ne`
- `Pattern` er et mønster



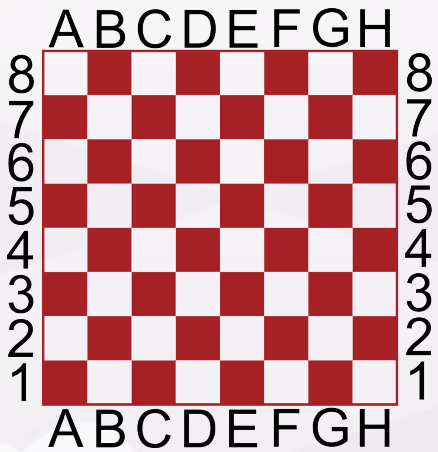


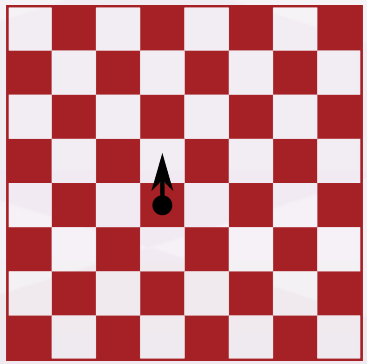


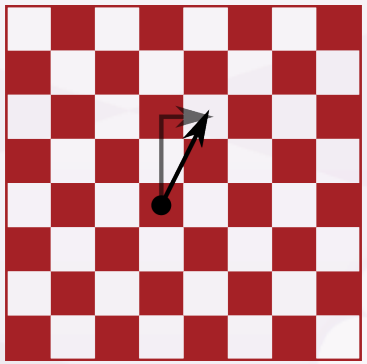
Standard Environment

Simple brætspilsrelaterede typer

- `Coordinate` er en vektor som repræsenterer et felt på et bræt: `C5` og `M27`
- `Direction` er en vektor som repræsenterer et flyt: `n`, `nw` og `n + ne`
- `Pattern` er et mønster







Standard Environment

Specielle typer

- `Type` repræsenterer en typeværdi, dette er f.eks. resultatet når man skriver navnet på en type.
- `Function` repræsenterer en funktionsværdi, dette er resultatet når man skriver navnet på en funktion. Eller et lambda-udtryk:
- `#[$a, $b] => $a + $b`

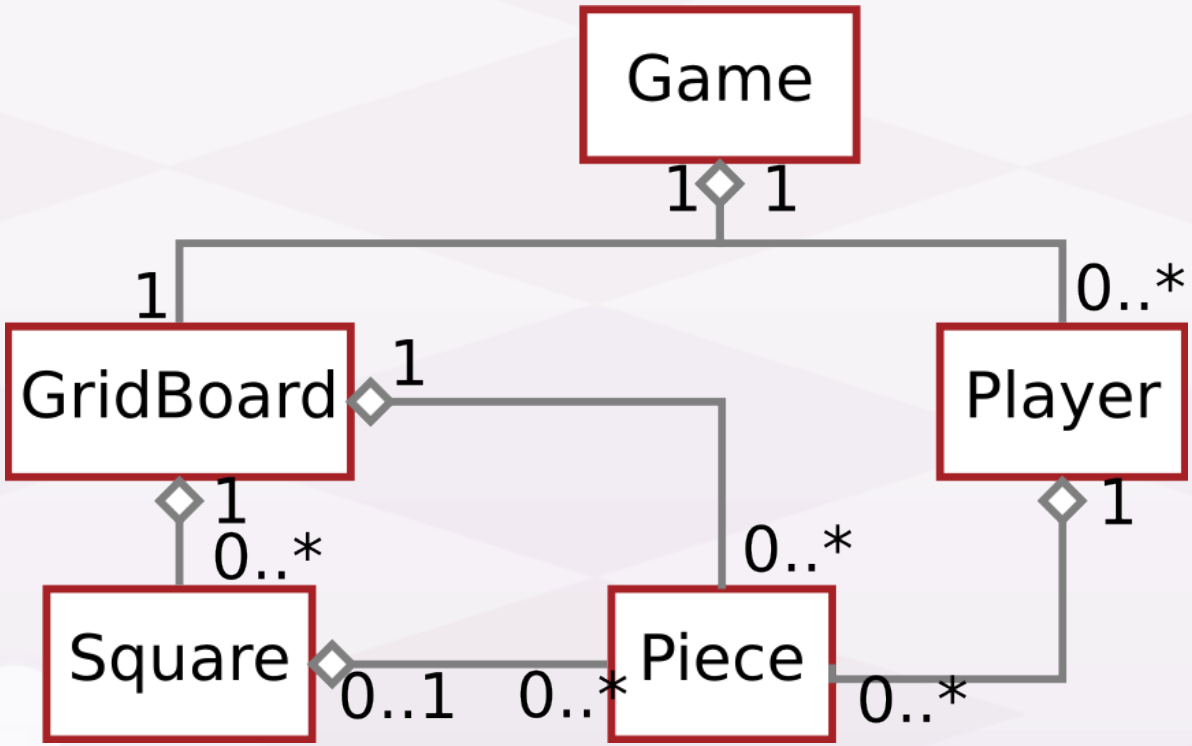
Standard Environment

Specielle typer

- `Type` repræsenterer en typeværdi, dette er f.eks. resultatet når man skriver navnet på en type.
- `Function` repræsenterer en funktionsværdi, dette er resultatet når man skriver navnet på en funktion. Eller et lambda-udtryk:
- `#[$a, $b] => $a + $b`

Game Environment

- Et klassehierarki til beskrivelse af brætspil.
- `Game`-typen repræsenterer f.eks. et brætspil. Man nedarver fra `Game` for at implementere sit brætspil.



Game Environment

Actions

- Håndtering af tilstandsændringer i brætspil, dvs. træk.
- F.eks. returner `Player`-typens `actions[]`-metode en liste af `Action`-objekter, dvs. en liste af mulige træk.
- `AddAction`, `RemoveAction` og `MoveAction` tillader manipulering af brikker på brættet: Tilføj, fjern, flyt.
- `ActionSequence` tillader kombinering af flere actions.

Game Environment

Actions

- Håndtering af tilstandsændringer i brætspil, dvs. træk.
- F.eks. returner `Player`-typens `actions[]`-metode en liste af `Action`-objekter, dvs. en liste af mulige træk.
- `AddAction`, `RemoveAction` og `MoveAction` tillader manipulering af brikker på brættet: Tilføj, fjern, flyt.
- `ActionSequence` tillader kombinering af flere actions.

Game Environment

Actions

- Håndtering af tilstandsændringer i brætspil, dvs. træk.
- F.eks. returner `Player`-typens `actions[]`-metode en liste af `Action`-objekter, dvs. en liste af mulige træk.
- `AddAction`, `RemoveAction` og `MoveAction` tillader manipulering af brikker på brættet: Tilføj, fjern, flyt.
- `ActionSequence` tillader kombinering af flere actions.

Game Environment

Actions

- Håndtering af tilstandsændringer i brætspil, dvs. træk.
- F.eks. returner `Player`-typens `actions[]`-metode en liste af `Action`-objekter, dvs. en liste af mulige træk.
- `AddAction`, `RemoveAction` og `MoveAction` tillader manipulering af brikker på brættet: Tilføj, fjern, flyt.
- `ActionSequence` tillader kombinering af flere actions.

Del 5

Game Abstraction Layer

GameAbstractionLayer klasse

```
1  private GameEnvironment env = new GameEnvironment();
2  private Interpreter interpreter = new Interpreter(env);
3
4  public GameAbstractionLayer(InputStream input) throws Error {
5    Scanner s = new Scanner(input);
6    LinkedList<Token> tokens = new LinkedList<Token>();
7    Token ts;
8    while ((ts = s.scan()).type != Token.Type.EOF) {
9      tokens.add(ts);
10   }
11   Parser p = new Parser();
12   AstNode ast = p.parse(tokens);
13   ScopeChecker scopeChecker = new ScopeChecker();
14   scopeChecker.visit(ast);
15   interpreter.visit(ast);
16 }
```

API

- Et interface for hver type i Game Environment
- For hvert interface, en wrapper som implementerer det

Game interface

Game	interface
players	getPlayers
board	getBoard
title	getTitle
history	getHistory
applyAction	applyAction
nextTurn	nextTurn
	getActions

Del 6

Simulator

Hensigt

- Grafisk interface til GAL
  - Skal kunne bruges af almindelige brugere
- Gøre det let at teste og spille spil

Widgets

- Model til at styre input og visualisering
- Organiseret i et hierarki
- Indkapsler opførsel

Widgets


Junta Simulator - Chess
FPS: 59
Chess
Player: White
Moves E2-)E4;
Moves A7-)A5;

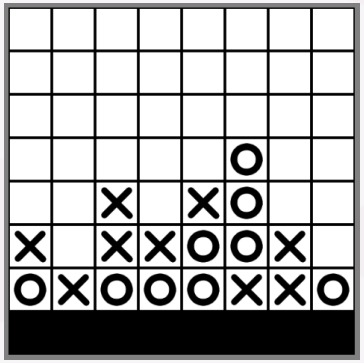
Widgets

```
1  final public void draw(){
2    g.translate( getX(), getY() );
3
4    g.setClip( absX, absY, getWidth(), getHeight() );
5    handleDraw( g );
6    g.clearClip();
7
8    for( Widget o : widgets )
9      o.draw( g, absX + o.getX(), absY + o.getY() );
10
11   g.translate( -getX(), -getY() );
12 }
```

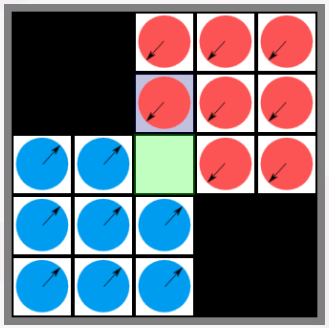
Del 7

Præsentation af spil, evaluering af krav og perspektivering

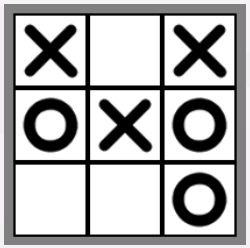
Spil



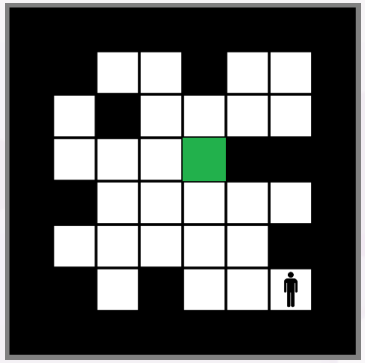
Figur : Connect four



Figur : Kents spil



Figur : Kryds og bolle



Figur : Ice

Connect four

```
type ConnectFour[] extends Game["Connect Four"] {
  define players = [
    ConnectPlayer[Crosses, "Crosses"],
    ConnectPlayer[Noughts, "Noughts"]
  ]
  define initialBoard = GridBoard[8, 8]
    .setSquaresAt[Bottom[],
                  [A1, B1, C1, D1, E1, F1, G1, H1]]
}
type ConnectPlayer[$pieceType, $name] extends Player[$name] {
  define winCondition[$gameState] =
    $gameState.findSquares[/friend (n friend)3 | (e friend)3 | (nw friend)3 |
                           (ne friend)3/].size != 0
  define tieCondition[$gameState] = $gameState.board.isFull
  define actions[$gameState] = addActions[$pieceType[/this/],
                                          $gameState.findSquares[/empty s !empty/]]
}
type Crosses[$owner] extends Piece[$owner]
type Noughts[$owner] extends Piece[$owner]
type Bottom[] extends Square[] {
  define isEmpty = false
}
```

Evaluering af krav

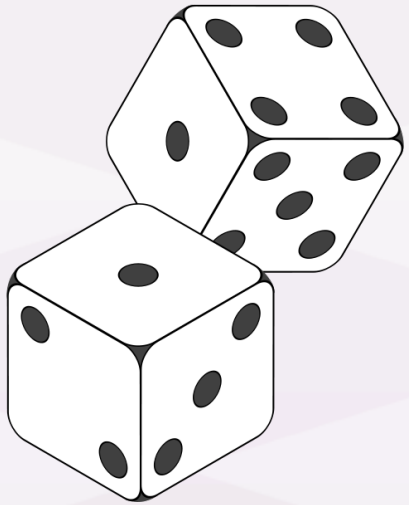
- Programmeringssproget kan bruges til at programmere brætspil i
- Det skal være muligt at implementere skak inklusiv dets specielle regler
- Det skal være muligt at lave brætspil på relativt få linjers kode
- Junta brætspil skal kunne spilles i en simulator

Evaluering af krav

- Det skal være muligt at spille spillene over netværk
- Programmeringssproget må ikke være en udvidelse af et eksisterende programmeringssprog
- Brætspillene skal være spilbare på forskellige platforme

Perspektivering

- Tilfældige værdier
- Et stærkere action-system



Perspektivering

- Forskellige brættyper

Tak for opmærksomheden

Pause tid!