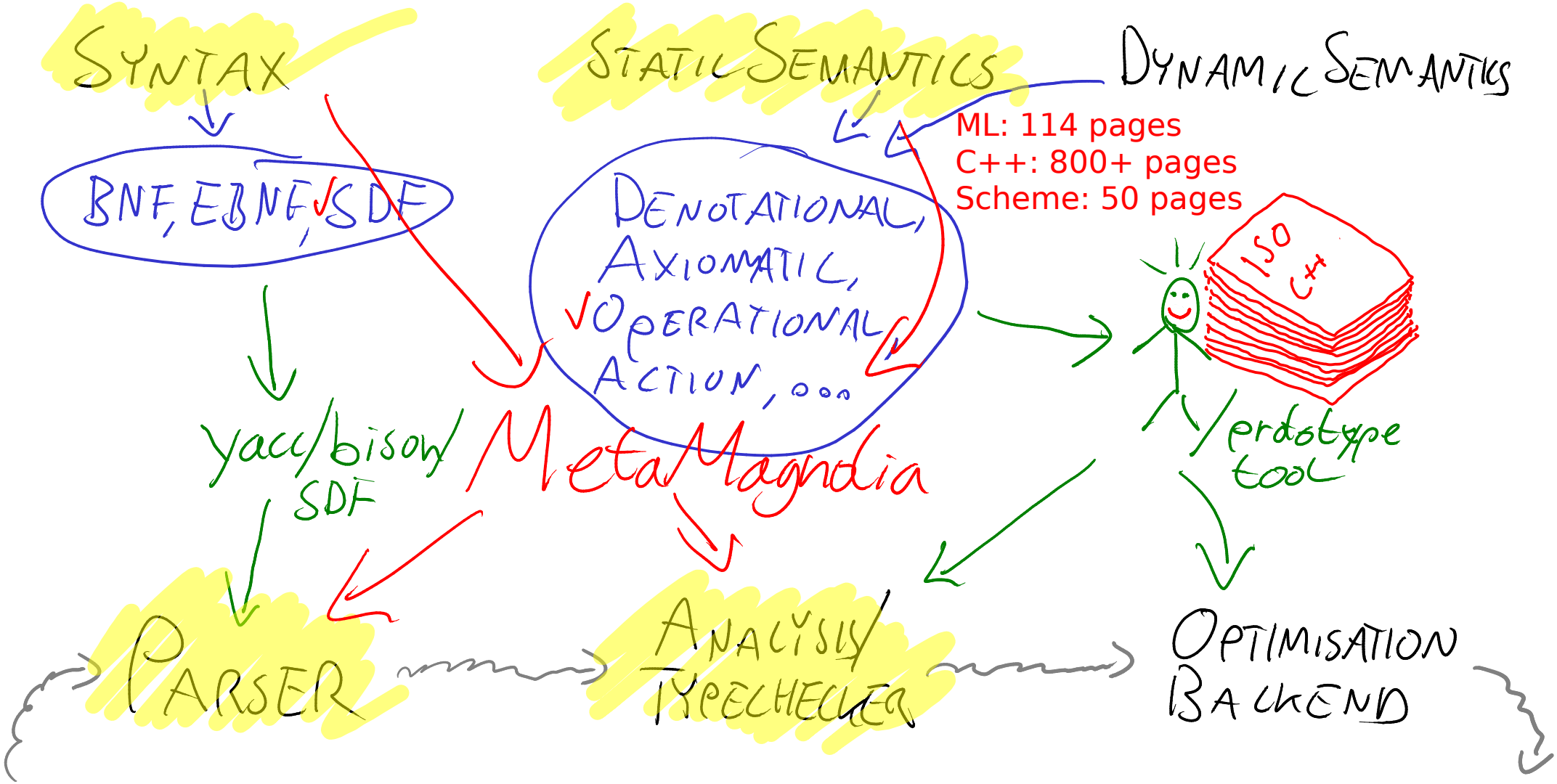
Language Description for Frontend Implementation

Anya Helene Bagge
Bergen Language Design Laboratory
Dept. of Informatics
University of Bergen

LDTA 2010

Language Specification



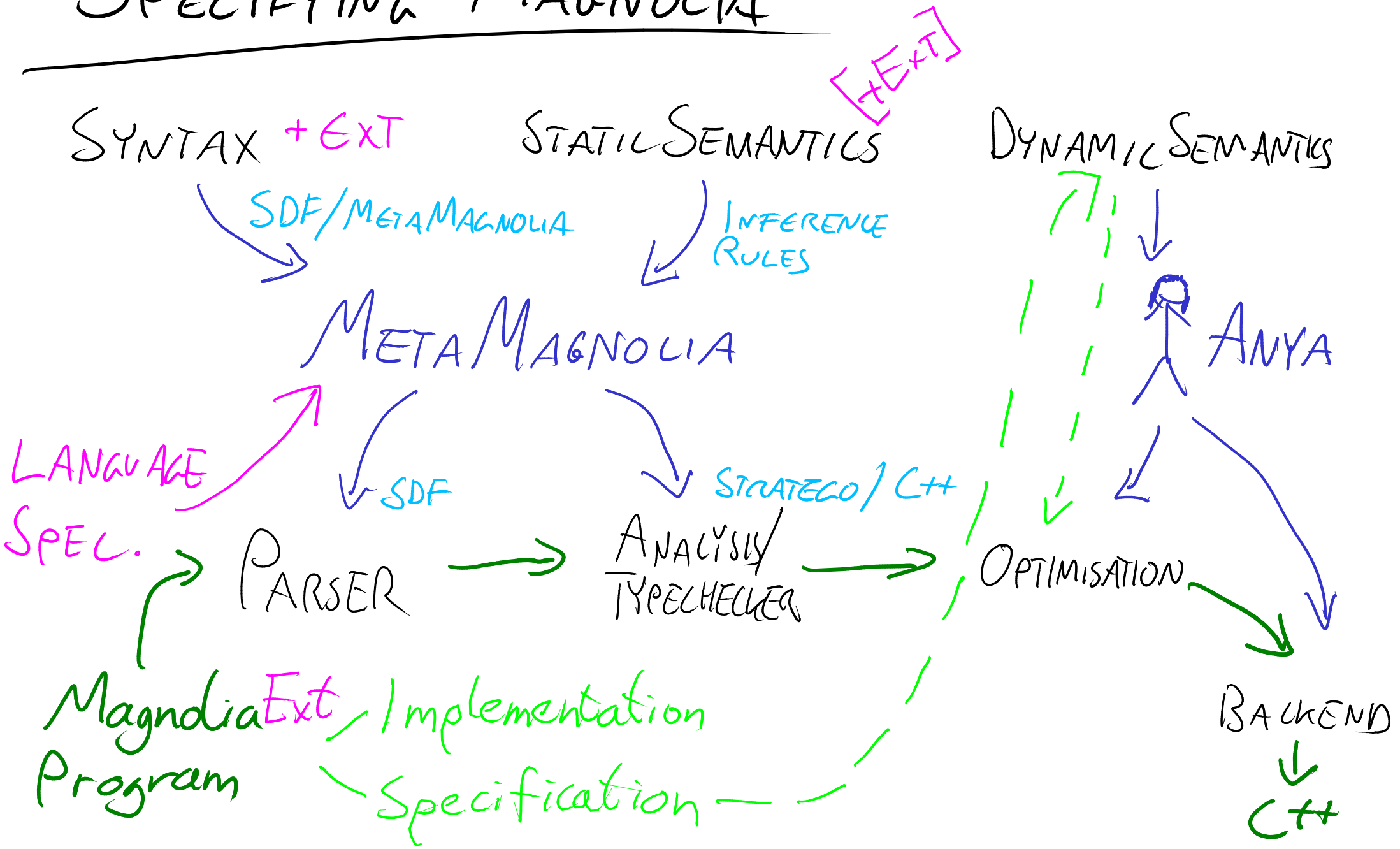
The Magnolia Language

- Static Typing, No Inference (typechecking; easy!)
- Overloading (name/overload resolution)
- Implicit Declarations [LOTA09]
- Algebrail Specification (concepts/typeclasses + axioms)
- Similar to C++ (well, somewhat...)

Experimental!

- Evolving Design
- Language Variants
- Extensions

Specifying Magnolia



MetaMagnolia Specifications

Syntax

Core

Static Semantics

Annotated Tree

- Defined in SDF or MetaMagnolia

```
syntax mix Magnolia/Core
         + Magnolia/BaseExt
```

- Sugar Rules to/from Core

```
if:  +«if e then s1* end»
  <-> <if e then s1* else end>
```

- Pretty-Print by Example

```
«if e then
  s1*
else
  s2*
end»
```

- Pattern Abstraction

```
patterns
  cond(c,t,e)
```

- Inference Rules

```
      e ==> e'    isPredicate(e')
  s1* ==> s1'*    s2* ==> s2'*
-------------------------------------
    «if e then s1* else s2* end»
 ==> <if e' then s1'* else s2'* end>
```

Sugar Example

```
language module Magnolia

syntax mix Magnolia/Core + Magnolia/BaseExt

syntax

 +« n » -> < n:? >

  « e1 o e2 »  <->  < _o_(«e1», «e2») >

  « o e1 »     <->  < o_(«e1») >

  « e[es*] »   <->  < index(«e», «es*») >

 +« if e then s1* end »   <->  < if «e» then «s1*» else end >

// C++-like syntax:

 +« if(e) s1 » {prefer}     ->  < if «e» then «s1» else end >       // amb
 +« if(e) s1 else s2 »      ->  < if «e» then «s1» else «s2» end >
 +« while(e) s »            ->  < while «e» do «s» end >
```

QUOTING:

sugared

<< input language >>

< output language >

Core

+:

New Syntax

Generated in SDF

WHAT HAPPENS:

- TRANSFORMATION CODE (Stratego / C++)
- BANANA SAFETY
  - COMPLETE TRANSFORMATION
  - TERMINATION
  - EFFICIENCY

implicit recursion

[Andersen Brabrand '89]

Defining Static Semantics

SOS / MSOS / I-MSOS

~~Attribute grammars~~

~~Transformation rules~~

$$\frac{f : T \to U \Rightarrow f' \qquad e : T \Rightarrow e'}{f(e) \Rightarrow f'(e')}$$

Abstraction

Modularity [MSOS] — pattern abstraction
— constructs

Implicit [I-MSOS] — handling inessential features

(similar to MSDF)

Example

construct let statement:

  syntax

    «let var n : t = e; in s* end»

  static semantics

$$\frac{e \Longrightarrow e' : t' \qquad s^* \;\texttt{==declare(<var n : t' = \_;>)=>}\; s'^*}{\texttt{«let var n : ? = e; in s* end»} \Longrightarrow \texttt{<let var n : t' = e'; in s'* end>}}$$

Operations

Use

$$\rho \Rightarrow \rho'$$

$$\rho \xrightarrow{op} \rho'$$

$$\rho \xrightarrow{op(\ldots)} \rho'$$

$$op(\rho, \ldots) = \rho'$$

Definition

$$\frac{\rho_1 \ldots \rho_n}{\rho \Rightarrow \rho'}$$

$$\frac{\rho_1 \ldots \rho_n}{\rho \xrightarrow{op()} \rho'}$$

OR: EXTERNAL DEF (WITH AXIOMS)

Example

construct if statement:

  static semantics

$$\frac{\text{isPredicate}(\text{«e»})}{\text{«if e then s1* else s2* end»} \Longrightarrow \text{<if «e» then «s1*» else «s2*» end>}}$$

construct while statement:

  static semantics

  «while e do s* end» ==> <while «e» do «s*» end>

Overload Resolution

construct call statement:

  syntax

    «call p(as*);»

  static semantics

```
          as* ==> as'*:ts'*
    p --?-callable(ts'*)-best-unique-> p'
    ------------------------------------
      «call p(as*);» ==> <call p'(as'*);>
```

CALLABILITY

```
operator callable?:

                compatible(t1*, t2*) = x
    ----------------------------------------------
    procedure p(t1*); --callable?(t2*)-> p weight x


operator compatible?:

   compatible?(t, t) = 0

   not(equal(t1, t2)) nameOf(t1)=n  n(?:t1) ==> (_, t2)
   ------------------------------------------------------
           compatible?(t1, t2) = -10

operator compatible = reduce(compatible?, 0)
```

Handling Errors

Need all information

Avoid clutter!

Implicit prop of context info

Attach error to „successful" result

Difference between

not applicable

semantic error

Conclusion — MetaMagnolia

- Syntax
  - composition (with SDF)
  - extension
  - desugaring
- Semantics — static
  - modules, abstraction, ADTs
  - error handling
  - implicit propagation/recursion
  - deal with overloading!
- Use — Magnolia
  - ? — need new name...

Check out our new Language Design Lab:

http://bldl.ii.uib.no/

we're looking for PhD students!

Photos from last night:

http://www.ii.uib.no/~anya/photos/ldta-2010/