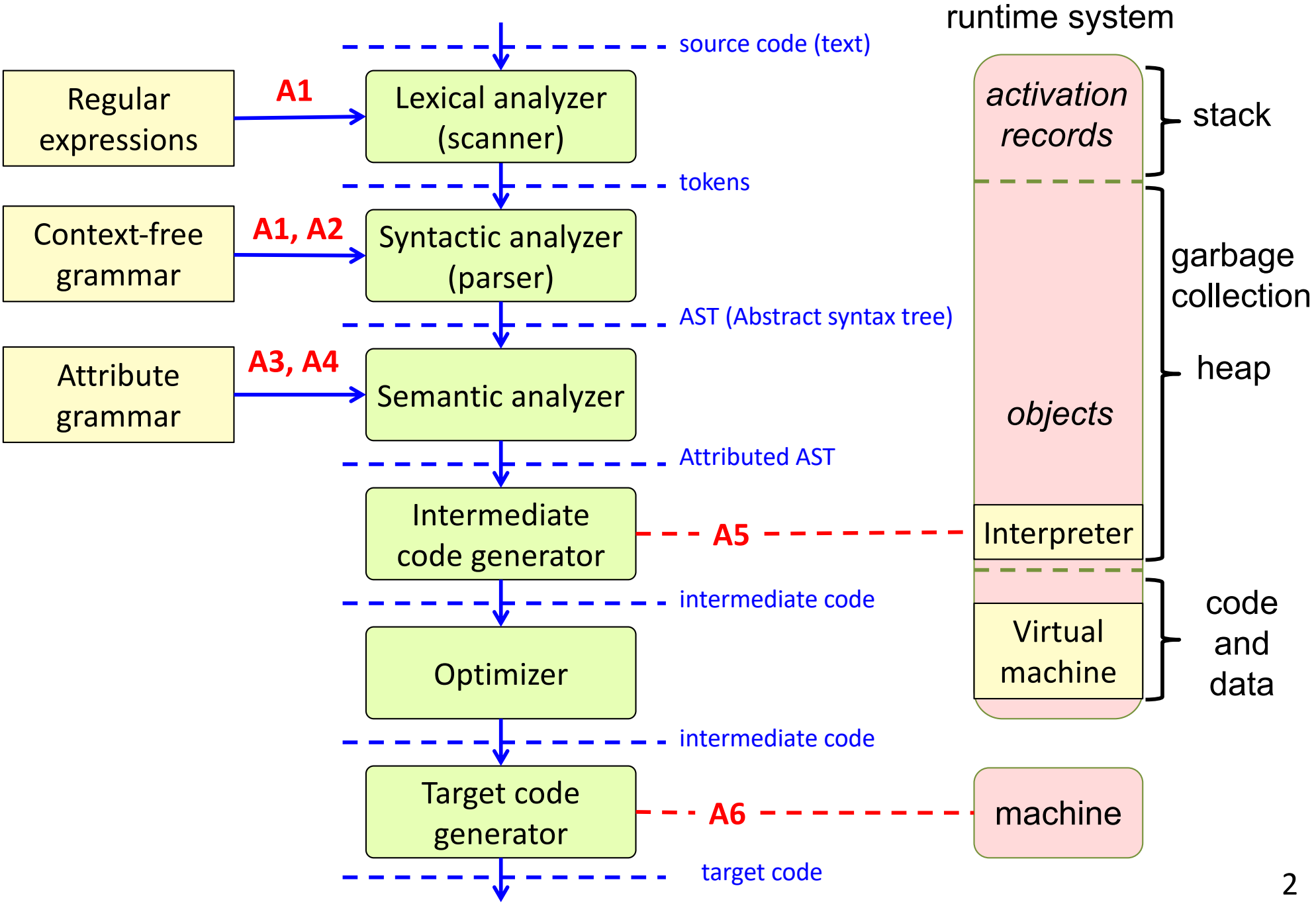
EDAN65: Compilers, Lecture 13

Review of important concepts

Görel Hedin

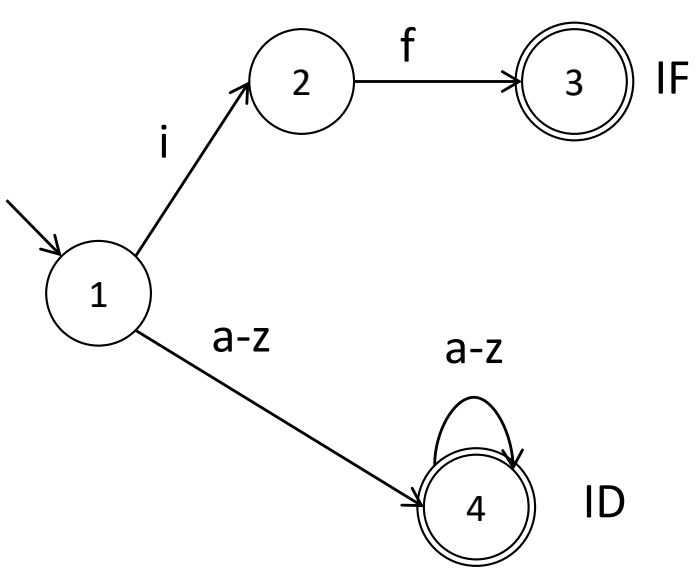
Revised: 2024-10-15

Course overview

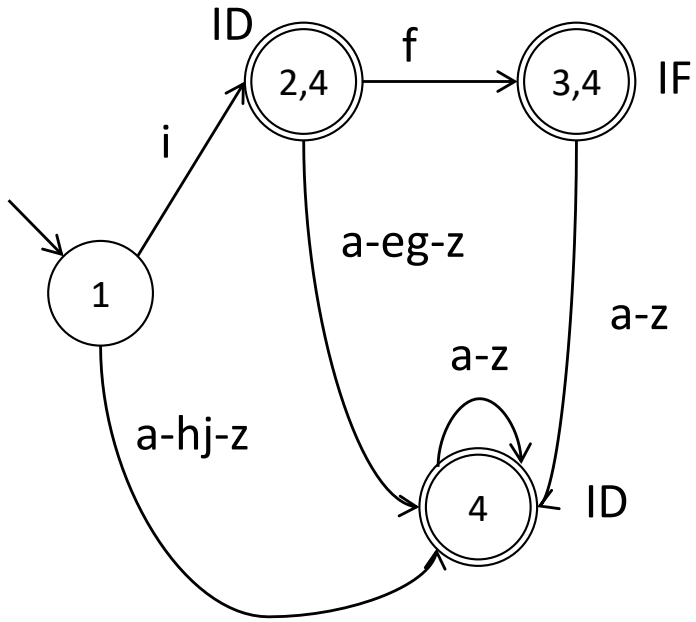


Regular expressions and scanning

NFA



DFA



Given some informal description, formulate a regular expression or automaton.
Translate between regular expressions, NFAs, DFAs.
Know how to combine automata representing tokens.
Know how to handle rule priority and longest match.

Context-free grammars

```
Exp -> Exp "+" Exp
Exp -> Exp "*" Exp
Exp -> INT
```

Given some informal description, formulate a language as a context-free grammar.

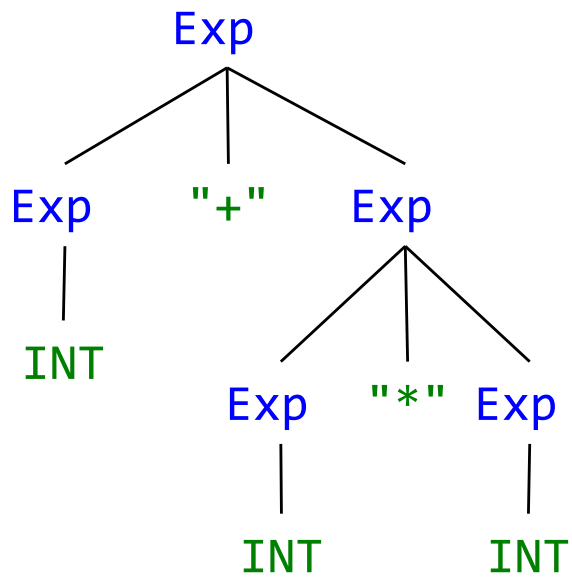
The elements of a context-free grammar G=(N, T, P, S):
nonterminal symbols, terminal symbols, productions, start symbol

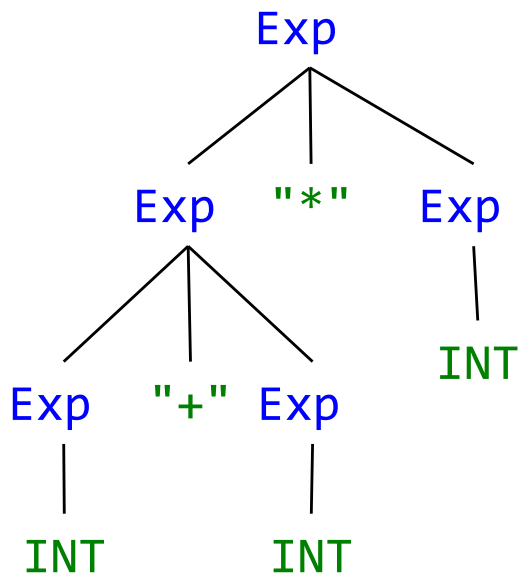
Understand what the language defined by a grammar is.
Understand the difference between regular expressions and context-free grammars.

```
Exp
=> Exp "+" Exp
=> INT "+" Exp
...
```

Derivations. How to prove that a sentence belongs to a language.
Parse trees. How a parse tree corresponds to a derivation.

Ambiguities in context-free grammars





Understand what it means for a grammar to be ambiguous.
Understand what it means for two grammars to be equivalent.

Be able to prove that a grammar is ambiguous for common ambiguities (expressions and dangling else).

Be able to transform an ambiguous grammar to an equivalent unambiguous grammar, for common ambiguities.

Notations for context-free grammars

```
A -> B d e C f
A -> g A
```

Canonical form

```
C -> D a b | b E F | a C
```

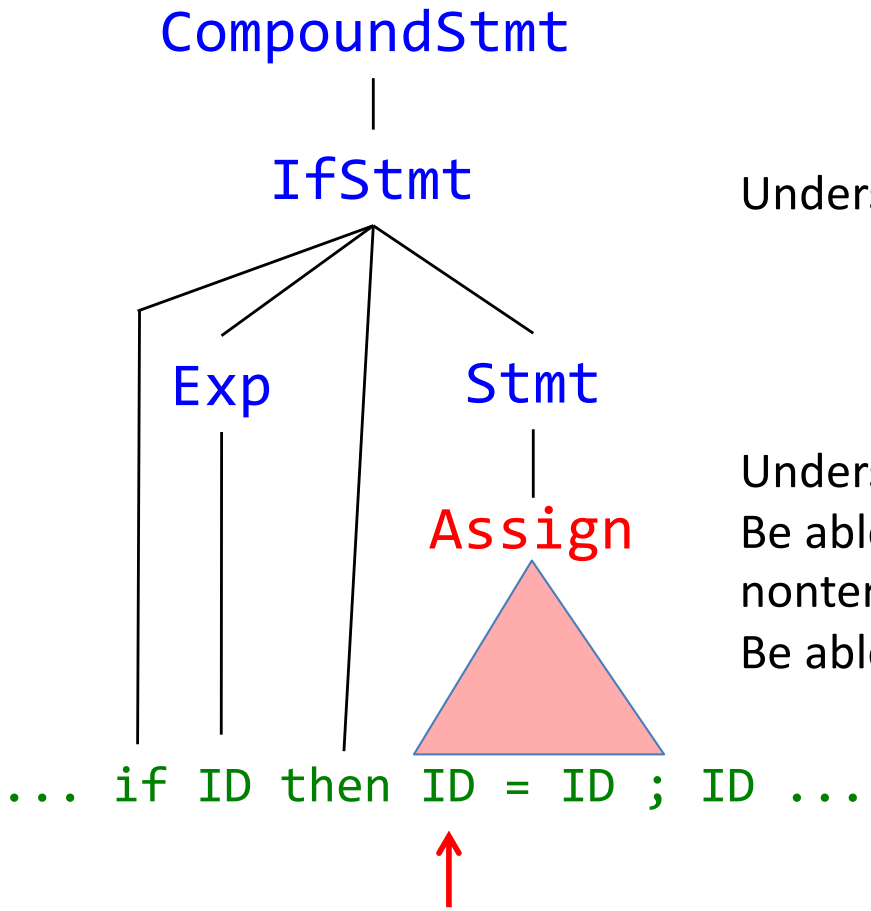
BNF

```
G -> H* i | (d E)+ F | [d C]
```

EBNF

Be able to formulate grammars on canonical form, as BNF, and EBNF
Be able to translate between these forms.

LL Parsing



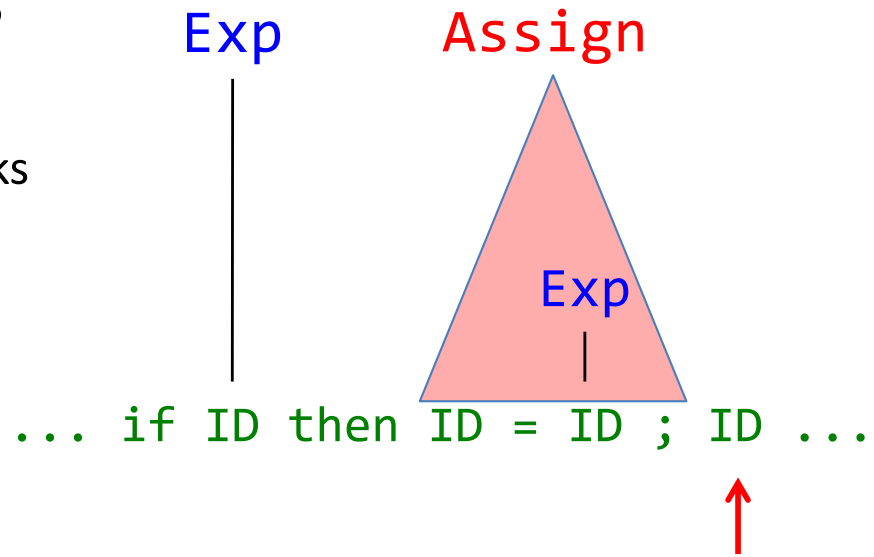
Understand the main idea of how LL parsing works

Understand the concepts of Nullable, FIRST, and FOLLOW.
Be able to compute them for a given nonterminal in a grammar.
Be able to construct an LL(1) table.

Be able to construct a recursive-descent parser for a given LL(1) grammar.

Understand what left recursion and common prefix mean.
Understand why grammars with these properties are not LL(1).
Be able to transform such a grammar to an equivalent LL(1) grammar.

LR parsing



Understand the main idea of how LR parsing works with shift, reduce, and accept actions.

Be able to show which actions an LR parser takes when parsing a specific example.

Understand why LR is more powerful than LL.

Understand what an LR item is.
Understand what an LR state is.

```
E -> E • "+" E     ?
E -> E "*" E •     "+"
```

Understand what an LR conflict is.
For a given LR conflict, be able to construct a program that would expose the conflict (for a simple case).

I will **not** ask you to construct an LR DFA or an LR parsing table on the exam (even for small examples, the DFAs/tables become very big).

Abstract grammars

Abstract grammar

```
abstract Stmt;
IfStmt : Stmt ::= Expr Stmt;
Assignment : Stmt ::= IdUse Expr;
IdUse : Expr ::= <ID:String>;
```

Understand the difference between an abstract grammar and a context-free grammar.

Understand how an abstract grammar corresponds to an object-oriented model.
Be able to design an abstract grammar for a simple language, corresponding to a good object-oriented model.

Be able to design a high-level (possibly ambiguous) concrete grammar that is very close to an abstract grammar.

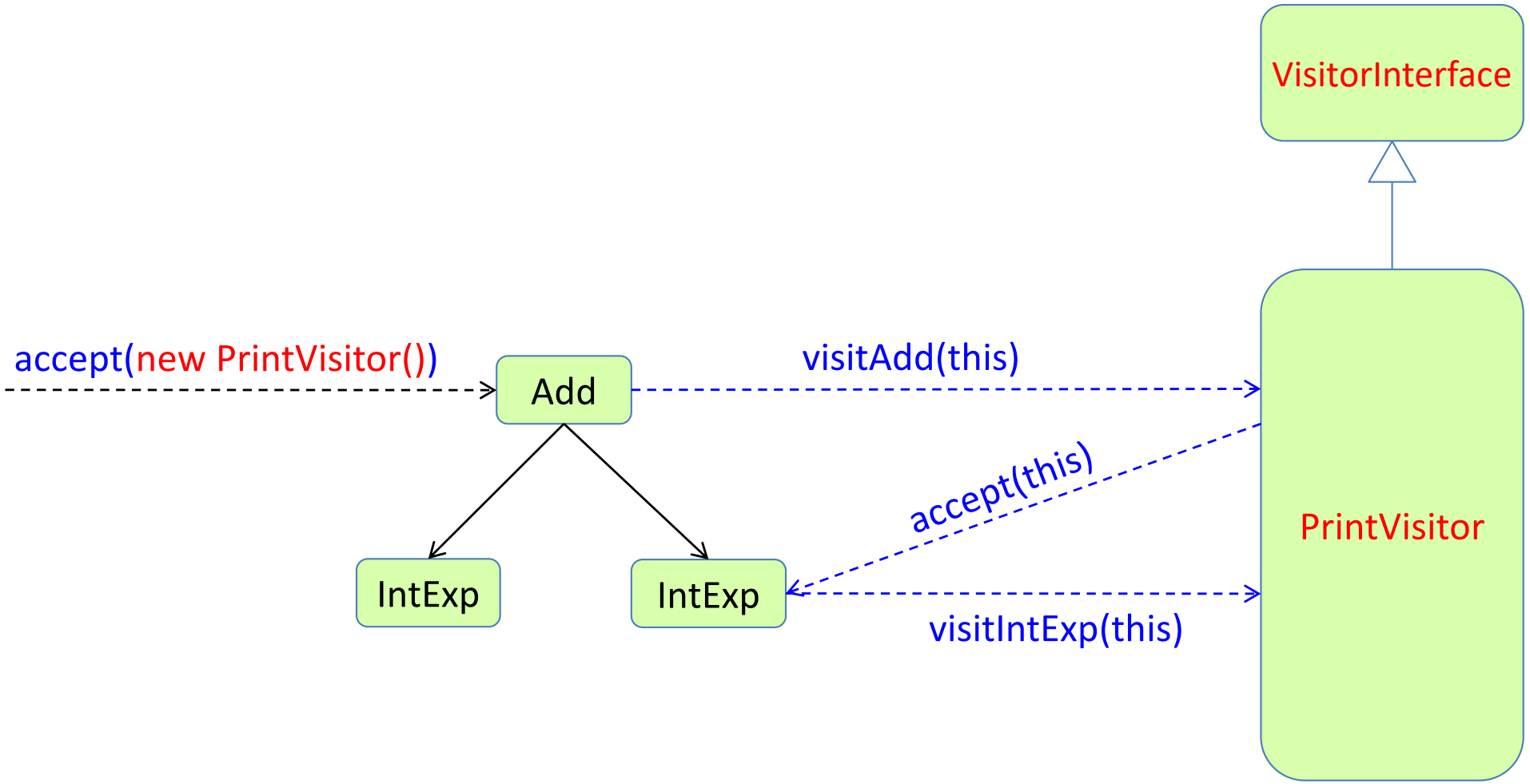
Static aspects with intertype declarations

```
aspect Evaluator {
  abstract int Expr.value();
  int Add.value() { return getLeft().value() + getRight().value(); }
  int Sub.value() { return getLeft().value() – getRight().value(); }
  int IntExpr.value() { return String.parseInt(getINT()); }
}
```

Understand how static aspect-oriented programming with inter-type declarations works.

Be able to program problems using methods declared in aspects.

Visitors



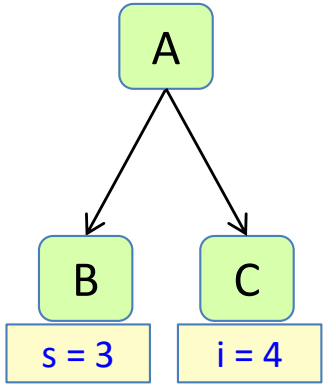
Understand how the visitor pattern works.

Be able to program problems using visitors.

Reference attribute grammars

```
syn int B.s();
eq B.s() = 3;

inh int C.i();
eq  A.getC().i() = getB().s() + 1;
```



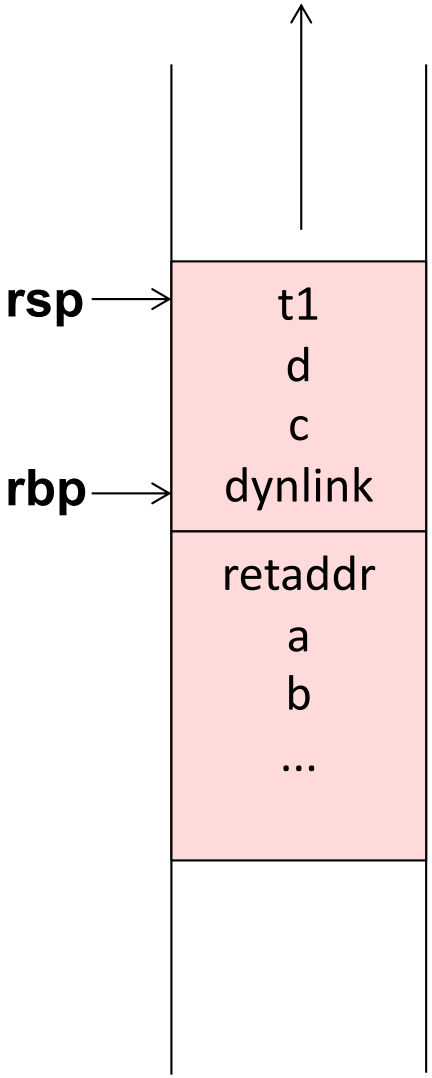
Understand how synthesized and inherited attributes work.
Understand reference attributes and parameterized attributes.
Understand also NTAs, circular, and collection attributes.

Given an attribute grammar and an example AST, be able to compute attribute values.

Be able to program problems using reference attribute grammars
without using getParent or instanceof.

You **don't** have to memorize details in the JastAdd syntax –
copies of the JastAdd reference manual will be available at the exam.

Runtime systems and code generation



Understand how an activation stack works, with frame pointer, stack pointer, and dynamic link.

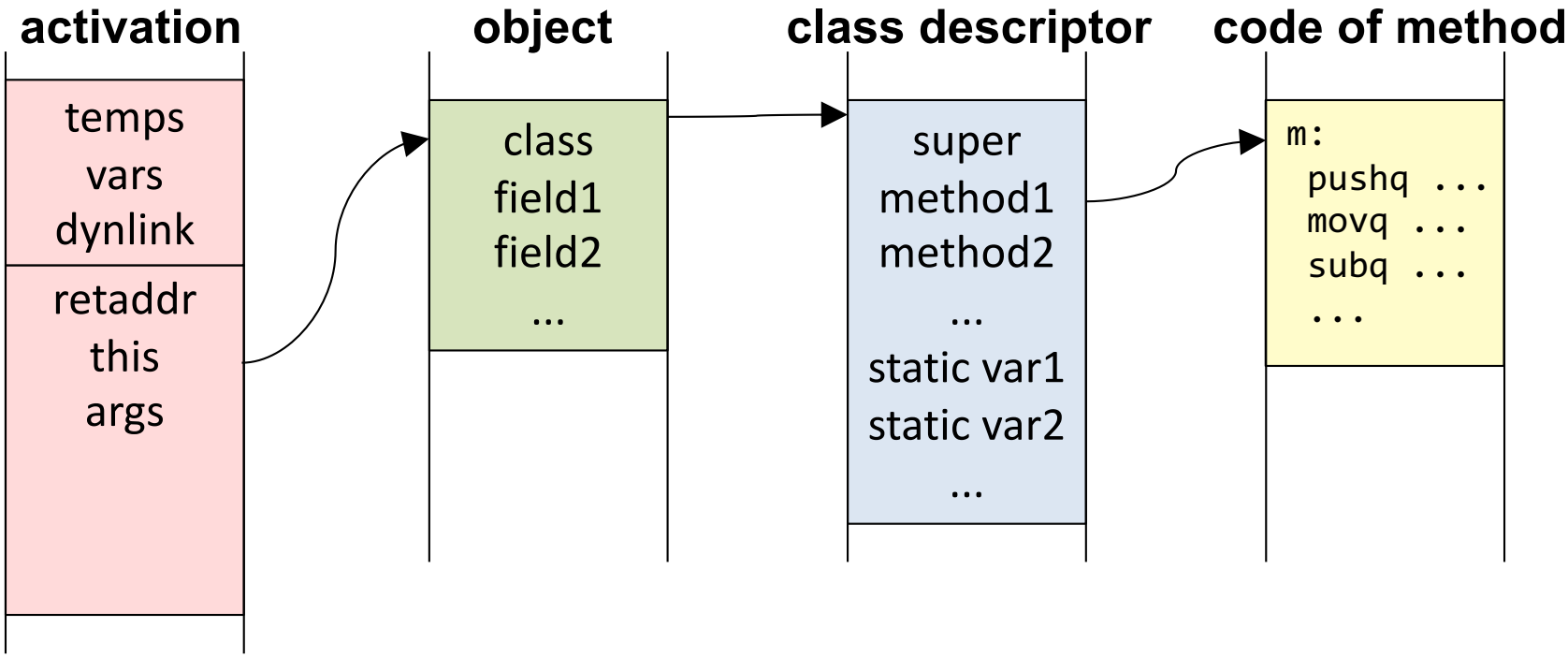
Understand simple calling conventions, like passing arguments on the stack and the return value in a register.

Be able to write down x86 assembly code for procedure calls, procedure activation and returns, and simple computations like loops, assignments, expression evaluation, etc.

Be able to draw the contents of the stack for a given execution point in a program.

You **don't** have to memorize x86 instructions –
the cheat sheet will be available at the exam.

Runtime systems for object-oriented languages



Understand the role of the static link ("this" pointer).
Understand how prefixing works for single inheritance of fields.
Understand how vtables work for single inheritance dynamic dispatch.

Have an overall understanding of how dynamic adaptive compilation works with inline call caches and PICs.

Have an overall understanding of how different garbage collection algorithms work, with mark-sweep, copying, generational, and reference-counting, and what their advantages and drawbacks are.

You **don't** have to memorize how to handle multiple inheritance with virtual tables.

Preparing for the exam

Read slides, book, papers, lab descriptions.
Do the quizzes and the exercises.
E13 contains exercises for many parts of the course.
Study old exams.

Ask at the **forum** if you have questions!

Don't forget to **sign up** for the exam by Sunday Oct 20.
(Not needed for PhD students – contact me instead)

You need to **finish the assignments/labs** before the exam.

Catch-up lab session (for showing A5):

- Friday Oct 18, 9:30-11:30, E:2116

Catch-up lab session (for showing A6):

- Tuesday Oct 22, 13:00-15:00, E:Gamma

The exam: Kårhuset: Gasque. Tuesday Oct 29, 14:00-19:00

Continued studies and work

- EDAN70: Project in Computer Science, lp2
- EDAP15: Program analysis, lp3, spring 2025
- EDAN75: Optimizing Compilers, lp3, spring 2025

- Compiler related master thesis projects in industry (ARKAD Nov 12-13)
  - Modelon AB, Lund
  - Cognibotics AB, Lund
  - ABB, Malmö
  - Arm, Lund
  - Axis, Lund
  - Neo4J, Malmö
  - ...
- Master thesis projects at the CS department

- NOTE! New routines for MSc projects at the CS department
  - You need a partner (no solo projects). See canvas page for partnering up
  - Deadline for signing up for spring MSc projects: Dec 1
  - For more info, see: https://cs.lth.se/examensarbete/

- Student internships abroad, e.g., Google Software Engineering internships (https://careers.google.com/students/engineering-and-technical-internships/)
- ...