

Mjølnér System: Grammars

Mjølnér Informatics Report

February 2002

Copyright © 1990–2002 [Mjølnér Informatics](#).

All rights reserved.

No part of this document may be copied or distributed
without the prior written permission of Mjølnér Informatics

Table of Contents

Beta Grammar.....	1
Option.....	1
Rule.....	1
Attribute.....	5
Index.....	5
A.....	6
B.....	6
C.....	6
D.....	6
E.....	6
F.....	6
G.....	6
I.....	6
L.....	6
M.....	6
N.....	6
O.....	7
P.....	7
R.....	7
S.....	7
T.....	7
U.....	7
V.....	7
X.....	7
 Metagrammar Grammar.....	 8
Option.....	8
Rule.....	8
Attribute.....	9
Index.....	9
A.....	9
C.....	9
D.....	9
E.....	9
G.....	9
L.....	10
N.....	10
O.....	10
P.....	10
S.....	10
T.....	10
 Prettyprint Grammar.....	 11
option.....	11
rule.....	11
Attribute.....	11
Index.....	12
A.....	12
B.....	12
C.....	12
D.....	12

Table of Contents

I.....	12
L.....	12
N.....	12
P.....	12
T.....	12
Property Grammar.....	13
Option.....	13
Rule.....	13
Index.....	14
B.....	14
D.....	14
I.....	15
L.....	15
M.....	15
N.....	15
O.....	15
P.....	15
R.....	15
S.....	15
T.....	15
V.....	15

Beta Grammar

```
contractioncategories
  MainPart Attributes Imperatives
--- beta : aGrammar : metagrammar ---
```

Grammar beta :

Option

```
version = 9
comBegin = '('*
comEnd = '*)'
stringChar = '\'
splitString = true
suffix = '.bet'
```

Rule

```
<BetaForm> ::= | <DescriptorForm>
              | <AttributesForm>
              ;
<DescriptorForm> ::= <ObjectDescriptor>
                    ;
<AttributesForm> ::= <Attributes>
                    ;
<ObjectDescriptor> ::= <PrefixOpt> <MainPart>
                      ;
<MainPart> ::= '(#' <Attributes> <ActionPart> '#)'
             ;
<Attributes> ::=+ <AttributeDeclOpt> ';' ;
<PrefixOpt> ::= ? <Prefix> ;
<Prefix> ::= <AttributeDenotation> ;
<AttributeDeclOpt> ::= ? <AttributeDecl> ;
<AttributeDecl> ::= | <PatternDecl>
                    | <SimpleDecl>
                    | <RepetitionDecl>
                    | <VirtualDecl>
                    | <BindingDecl>
                    | <FinalDecl>
                    | <ErrorDecl>
                    ;
<PatternDecl> ::= <Names> ':' <ObjectDescriptor> ;
<SimpleDecl> ::= <Names> ':' <referenceSpecification> ;
<RepetitionDecl> ::= <Names> ':' '[' <index> ']' <referenceSpecification> ;
<VirtualDecl> ::= <Names> ':' '<' <ObjectSpecification> ;
<BindingDecl> ::= <Names> ':' ':' '<' <ObjectSpecification> ;
<FinalDecl> ::= <Names> ':' ':' <ObjectSpecification> ;
<VariablePattern> ::= '##' <AttributeDenotation> ;
```

Beta Grammar

```
<referenceSpecification> ::= | <StaticItem>
                          | <DynamicItem>
                          | <StaticComponent>
                          | <DynamicComponent>
                          | <VariablePattern>
                          ;

<StaticItem> ::= '@' <ObjectSpecification>;

<DynamicItem> ::= '^' <AttributeDenotation>;

<StaticComponent> ::= '@' '|' <ObjectSpecification>;

<DynamicComponent> ::= '^' '|' <AttributeDenotation>;

<ObjectSpecification> ::= | <ObjectDescriptor>
                          | <AttributeDenotation>
                          ;

<Index> ::= | <SimpleIndex>
           | <NamedIndex>
           ;

<NamedIndex> ::= <NameDcl> ':' <Evaluation>;

<ActionPart> ::= <EnterPartOpt> <DoPartOpt> <ExitPartOpt>;

<EnterPartOpt> ::= ? <EnterPart>;
<DoPartOpt> ::= ? <DoPart>;
<ExitPartOpt> ::= ? <ExitPart>;

<EnterPart> ::= 'enter' <Evaluation>;
<DoPart> ::= 'do' <Imperatives>;
<ExitPart> ::= 'exit' <Evaluation>;

<Imperatives> ::=+ <ImpOpt> ';' ;
<ImpOpt> ::= ? <Imp>;

<Imp> ::= | <LabelledImp>
          | <ForImp>
          | <SimpleIfImp>
          | <GeneralIfImp>
          | <LeaveImp>
          | <RestartImp>
          | <InnerImp>
          | <SuspendImp>
          | <Evaluation>
          | <CodeImp>
          | <errorImp>;

<LabelledImp> ::= <NameDcl> ':' <Imp>;

<ForImp> ::= '(' 'for' <Index> 'repeat' <Imperatives> 'for' ')';

<GeneralIfImp> ::= '(' 'if' <Evaluation> <Alternatives> <ElsePartOpt> 'if' ')';

<SimpleIfImp> ::= '(' 'if' <Evaluation> 'then' <Imperatives> <ElsePartOpt> 'if' ')';

<LeaveImp> ::= 'leave' <NameApl>;

<RestartImp> ::= 'restart' <NameApl>;

<InnerImp> ::= 'inner' <NameAplOpt>;

<NameAplOpt> ::= ? <NameApl> ;
```

Beta Grammar

```

<SuspendImp> ::= 'suspend' ;

<CodeImp> ::= '(' 'code' <CodeItems> 'code' ')';

<Alternatives> ::=+ <Alternative> ;
<Alternative> ::= <Selections> 'then' <Imperatives>;

<Selections> ::=+ <Selection> ;
<Selection> ::=| <CaseSelection> ;

<CaseSelection> ::= '//' <evaluation>;

<ElsePartOpt> ::=? <ElsePart>;
<ElsePart> ::= 'else' <Imperatives>;

<Evaluations> ::=+ <Evaluation> ',';

<Evaluation> ::=| <Expression>
                | <AssignmentEvaluation>;

<AssignmentEvaluation> ::= <Evaluation> '->' <Transaction>;

<Transaction> ::=| <ObjectEvaluation>
                  | <ComputedObjectEvaluation>
                  | <ObjectReference>
                  | <EvalList>
                  | <StructureReference>
                  | <Primitive>
                  | <Address>
                  ;

<ObjectEvaluation> ::=| <InsertedItem>
                       | <reference>
                       ;

<Reference> ::=| <ObjectDenotation>
                | <DynamicObjectGeneration>
                ;

<DynamicObjectGeneration> ::=| <DynamicItemGeneration>
                              | <DynamicComponentGeneration>
                              ;

<InsertedItem> ::= <ObjectDescriptor> ;
<ObjectDenotation> ::= <AttributeDenotation> ;
<ComputedObjectEvaluation> ::= <ObjectEvaluation> '!';
<ObjectReference> ::= <Reference> '[';
<StructureReference> ::= <AttributeDenotation> '##';
<EvalList> ::= '(' <Evaluations> ')';
<DynamicItemGeneration> ::= '&' <ObjectSpecification>;
<DynamicComponentGeneration> ::= '&' '|' <ObjectSpecification>;
<Primitive> ::= 'tos' <SimpleEntry>;
<Address> ::= '@@' <AttributeDenotation>;

<AttributeDenotation> ::=| <NameApl>
                        | <Remote>
                        | <ComputedRemote>
                        | <Indexed>
                        | <ThisObject>
                        | <RemotePrimitive>
                        ;

<Remote> ::= <AttributeDenotation> '.' <NameApl>;
<ComputedRemote> ::= '(' <Evaluations> ')' '.' <NameApl> ;
<Indexed> ::= <AttributeDenotation> '[' <Evaluation> ']';
<ThisObject> ::= 'this' '(' <NameApl> ')';
<RemotePrimitive> ::= <AttributeDenotation> '.%' <NameApl>;

<Expression> ::=| <RelationalExp> | <SimpleExp> ;

```

Beta Grammar

```

<RelationalExp> ::= | <EqExp> | <LtExp> | <LeExp>
                  | <GtExp> | <GeExp> | <NeExp>
                  ;

<SimpleExp>      ::= | <AddExp> | <SignedTerm> | <Term> ;

<AddExp>         ::= | <PlusExp> | <MinusExp> | <OrExp> | <XorExp>;

<SignedTerm>    ::= | <unaryPlusExp> | <unaryMinusexp>;

<Term>          ::= | <MulExp> | <Factor> ;

<MulExp>        ::= | <TimesExp> | <RealDivExp> | <IntDivExp>
                  | <ModExp> | <AndExp> | <PrimitiveExp> ;

<EqExp> ::= <Operand1:SimpleExp> '=' <Operand2:SimpleExp>;
<LtExp> ::= <Operand1:SimpleExp> '<' <Operand2:SimpleExp>;
<LeExp> ::= <Operand1:SimpleExp> '<=' <Operand2:SimpleExp>;
<GtExp> ::= <Operand1:SimpleExp> '>' <Operand2:SimpleExp>;
<GeExp> ::= <Operand1:SimpleExp> '>=' <Operand2:SimpleExp>;
<NeExp> ::= <Operand1:SimpleExp> '<>' <Operand2:SimpleExp>;

<PlusExp>      ::= <SimpleExp> '+' <Term>;
<MinusExp>     ::= <SimpleExp> '-' <Term>;
<OrExp>        ::= <SimpleExp> 'or' <Term>;
<XorExp>       ::= <SimpleExp> 'xor' <Term>;

<unaryPlusExp> ::= '+' <Term>;
<unaryMinusExp> ::= '-' <Term>;

<TimesExp>     ::= <Term> '*' <Factor>;
<RealDivExp>   ::= <Term> '/' <Factor>;
<IntDivExp>    ::= <Term> 'div' <Factor>;
<ModExp>       ::= <Term> 'mod' <Factor>;
<AndExp>       ::= <Term> 'and' <Factor>;

<PrimitiveExp> ::= <Term> '%' <NameApl> <Factor> ;

<Factor>       ::= | <TextConst>
                  | <IntegerConst>
                  | <NotExp>
                  | <NoneExp>
                  | <RepetitionSlice>
                  | <Transaction>
                  | <UnaryPrimitiveExp>
                  ;

<RepetitionSlice> ::= <AttributeDenotation>
                    '[' <Low:Evaluation> ':' <High:Evaluation> ']' ;

<notExp>         ::= 'not' <factor>;
<noneExp>        ::= 'none' ;
<UnaryPrimitiveExp> ::= '%' <NameApl> <factor>;

<Names> ::=+ <NameDcl> ',' ;
<NameDcl> ::= <NameDecl>;
<NameApl> ::= <NameApl>;

<SimpleEntry> ::= ? <TextConst>;
<TextConst>   ::= <String>;
<IntegerConst> ::= <Const>;

<SimpleIndex> ::= <Evaluation>;

<CodeItems> ::=+ <CodeItem> ',' ;
<CodeItem>   ::= | <CodeString> | <CodeConst>;
<CodeString> ::= <String>;

```

```

<CodeConst> ::= <Const>;

(* now for the errorproductions *)
<ErrorDecl> ::= Error;
<ErrorImp>  ::= Error

```

Attribute

```

<ObjectSpecification> : 0
<Attributes> : 0

<DescriptorForm> : 18
<AttributesForm> : 18
<ObjectDescriptor> : 8
<MainPart> : 2
<DoPart> : 2
<ForImp> : 2
<repetitionDecl> : 2
<LabelledImp> : 2
<nameDcl> : 2
<nameApl> : 4
<bindingDecl> : 2
<FinalDecl> : 2
<InsertedItem> : 2
<ObjectDenotation> : 2
<ComputedObjectEvaluation> : 2
<RepetitionSlice>:2
<ObjectReference> : 2
<EvalList> : 2
<Address> : 2
<Primitive> : 2
<DynamicItemGeneration> : 2
<DynamicComponentGeneration> : 2

<EqExp> : 2
<LtExp> : 2
<LeExp> : 2
<GtExp> : 2
<GeExp> : 2
<NeExp> : 2
<PlusExp> : 2
<MinusExp> : 2
<OrExp> : 2
<XorExp> : 2
<MulExp> : 2
<TimesExp> : 2
<RealDivExp> : 2
<IntDivExp> : 2
<ModExp> : 2
<AndExp> : 2

<EnterPart> : 2
<ExitPart> : 2

```

Index

The entries in the alphabetic index consists of all left-sides in the grammar. The small table of letters below links directly to the section of identifiers starting with the corresponding letters.

A B C D E F G I L M N O P R S T U V X

A

<ActionPart<AddExp<Address<Alternative<AndExp<Assignment<AttributeDecl<AttributeDeclDenotation<A

B

<BetaForm <BindingDecl

C

<CaseSelection<CodeConst<CodeItem<CodeItems<CodeString<ComputedObjectEvaluation<Computed

D

<DescriptorForm<DoPart<DoPartOpt<DynamicComponent<DynamicCodeDynamicGeneralDynamicGeneralObject

E

<ElsePart<ElsePartOpt<EnterPart<EnterDeclOpt<ExpEvalList<Evaluation<Evaluations <ExitPart<ExitPartOpt<Expression

F

<Factor <FinalDecl <ForImp

G

<GeExp <GeneralImp <GtExp

I

<Imp<Imperatives<ImpOpt <Index<Indexed<InnerImp <InsertedItem<IntDivExp<IntegerConst

L

<LabelledImp<LeaveImp <LeExp<LtExp

M

<MainPart<MinusExp <ModExp<MulExp

N

<NameApl<NameAplOpt<NameDecl<NamedIndex<Names<NeExp <noneExp<notExp

O

<ObjectDenotation<ObjectDescription<ObjectEvaluation<ObjectReference<ObjectSpecification<OrExp

P

<PatternDecl<PlusExp <Prefix<PrefixOpt <Primitive<PrimitiveExp

R

<RealDivExp<Reference<reference<ResidualExp<Remote<RemotePrimitive<RepetitionDecl<RepetitionSlice<Restart

S

<Selection<Selections<SignedTerm<SimpleDecl<SimpleExp<SimpleIndex<SimpleIndexDecl<StaticItem<Structure

T

<Term<TextConst <ThisObject<TimesExp <Transaction

U

<unaryMinusExp <unaryPlusExp <UnaryPrimitiveExp

V

<VariablePattern <VirtualDecl

X

<XorExp

Metagrammar Grammar

```
--- metagrammar : Agrammar : metagrammar ---  
Grammar metagrammar :
```

Option

```
version      = 5  
suffix=     '.gram'
```

```
BobsOption = '32,34'  
comBegin   = '(*'  
comEnd     = '*)'  
stringChar = '\\'
```

Rule

```
<AGrammar> ::= 'Grammar' <GrammarName> ':' <OptionOp>  
            'Rule' <ProductionList> <AttributeOp>;  
<GrammarName> ::= <NameDecl>;  
<ProductionList> ::=+ <Prod> ';' ;  
  
<Prod> ::= |<Alternation> |<Constructor> |<Lst>  
          |<Opt> |<Dummy> |<ErrorProd>;  
  
<LeftSide> ::= '<' <SynDeclName> '>';  
  
<Alternation> ::= <LeftSide> '::|' <SynCatList>;  
<SynCatList> ::=+ <SynCat> '|';  
  
<Constructor> ::= <LeftSide> '::=' <ConsElemList>;  
<ConsElemList> ::=+ <ConsElem>;  
<ConsElem> ::= |<TaggedSyn> | <SynCat> | <Term> | <ErrorSpec>;  
<TaggedSyn> ::= '<' <TagName> ':' <SynName> '>';  
<SynCat> ::= '<' <SynName> '>';  
<ErrorSpec> ::= 'error';  
  
<Lst> ::= |<ListOne> | <ListZero>;  
<ListOne> ::= <LeftSide> '::+' <SynCat> <TermOp>;  
<ListZero> ::= <LeftSide> '::*' <SynCat> <TermOp>;  
<TermOp> ::= ? <Term>;  
  
<Opt> ::= <LeftSide> '::?' <SynCat>;  
  
<Dummy> ::= <LeftSide> '::' <SynCat>;  
  
<SynName> ::= <NameAppl>;  
<TagName> ::= <NameDecl>;  
<SynDeclName> ::= <NameDecl>;  
<Term> ::= <String>;  
  
<OptionOp> ::= ? <OptionPart>;  
<OptionPart> ::= 'option' <optionList>;  
<optionList> ::=+ <optionElement>;  
<optionElement> ::= <optionName> '=' <optionSpecification>;  
<optionSpecification> ::= |<singleOption> | <optionSpecList>;  
<optionSpecList> ::= '(' <optionSpecList> ')';  
<optionSpecList> ::=+ <singleOption>;  
  
<singleOption> ::= |<optionName> | <optionConst>  
                  |<optionString> | <optionError>;
```

```

<optionName> ::= <NameAppl>;
<optionConst> ::= <Const>;
<optionString> ::= <String>;

<AttributeOp> ::? <AttributePart>;
<AttributePart> ::= 'attribute' <attriblist>;
<AttribList> ::= * <Attrib>;

<Attrib> ::= <SynCat> ':' <NoOfAttributes>;
<NoOfAttributes> ::= <const>;
<errorProd> ::= Error;
<optionError> ::= Error

```

Attribute

```

<LeftSide> : 2
<SynName> : 1

<Prod> : 0
<ConsElem> : 0
<AGrammar> : 0
<TaggedSyn> : 0
<SynCat> : 0
<Term> : 0

```

Index

The entries in the alphabetic index consists of all left-sides in the grammar. The small table of letters below links directly to the section of identifiers starting with the corresponding letters.

A C D E G L N O P S T

A

[<AGrammar<Alternation](#) [<Attrib<AttribList](#) [<AttributeOp<AttributePart](#)

C

[<ConsElem](#) [<ConsElemList](#) [<Constructor](#)

D

[<Dummy](#)

E

[<errorProd](#) [<ErrorSpec](#)

G

[<GrammarName](#)

L

<LeftSide<ListOne <ListZero<Lst

N

<NoOfAttributes

O

<Opt<optionConst<optionElement<optionList<optionName<OptionGroup<OptionPairification<optionSpecList<opt

P

<Prod <ProductionList

S

<singleOption<SynCat <SynCatList<SynDeclName <SynName

T

<TaggedSyn<TagName <Term<TermOp

Prettyprint Grammar

```
-- prettyprint : Agrammar : metagrammar --  
Grammar prettyprint:
```

option

```
suffix='.pgram'  
bobsoptions = '25, 32, 34'  
combegins  = '(*'  
comEnd     = '*)'  
stringChar = '\\'
```

rule

```
<PrettyPrint> ::= 'PrettyPrintScheme' <SchemeName:nameDecl>  
                'for' <GrammarName:nameDecl> ':' <ProductionList>;  
<ProductionList> ::= * <Production> ';' ;  
  
<Production> ::= | <Constructor> | <ListProd> ;  
<Constructor> ::= <ProductionName:nameAppl> '=' <Stream:ItemList>;  
<ListProd> ::= <ProductionName:nameAppl> '=' '(' <ListSpec> ')';  
  
<ItemList> ::= * <Item>;  
<Item> ::= | <Terminal> | <NonTerm> | <Break> | <Block>  
          | <CommentPlace>;  
  
<Terminal> ::= | <DefaultTerm> | <AltTerm> ;  
<DefaultTerm> ::= 'T' ':' <TerminalNo:const>;  
<AltTerm> ::= <AlternativeTerminal:String> ;  
  
<NonTerm> ::= 'N' ':' <NonTerminalNo:const>;  
  
<Break> ::= | <DefaultBreak> | <AltBreak> ;  
<DefaultBreak> ::= '$$';  
<AltBreak> ::= '$' <Space:const> ',' <Indention:const>;  
  
<Block> ::= '[' <BlockType> <ItemList> ']';  
  
(* comments must only be specified after terminals! *)  
<CommentPlace> ::= '*';  
  
<ListSpec> ::= <Beginning:ItemList>  
              '{' <BlockType> <Separator:ItemList> '  
              <Ending:ItemList> ;  
  
<BlockType> ::= | <Consistent> | <InConsistent> ;  
<Consistent> ::= 'c';  
<InConsistent> ::= 'i'
```

Attribute

```
<Constructor> : 1  
<ListProd> : 1  
<DefaultTerm> : 2  
<AltTerm> : 2  
<NonTerm> : 2
```


Property Grammar

```
-- property : aGrammar : metagrammar ---
```

```
Grammar property :
```

Option

```
version = 4
comBegin = '('
comEnd = ')'
splitOnFiles = 1
stringChar = '\\'
suffix = '.prop'
```

Rule

```
<Properties> ::= <PropertyList> ;

<PropertyList> ::=+ <PropertyOpt> ';' ;

<PropertyOpt> ::=? <Property> ;

<Property> ::= | <ORIGIN>
                | <INCLUDE>
                | <BODY>
                | <MDBODY>
                | <OBJFILE>
                | <LIBFILE>
                | <LINKOPT>
                | <BETARUN>
                | <BUILD>
                | <MAKE>
                | <RESOURCE>
                | <LIBDEF>
                | <LIBITEM>
                | <ON>
                | <OFF>
                | <Other>;

<ORIGIN> ::= 'ORIGIN' <TextConst> ;

<INCLUDE> ::= 'INCLUDE' <StringList> ;

<BODY> ::= 'BODY' <StringList> ;

<MDBODY> ::= 'MDBODY' <MachineSpecificationList> ;

<OBJFILE> ::= 'OBJFILE' <MachineSpecificationList> ;

<LIBFILE> ::= 'LIBFILE' <MachineSpecificationList> ;

<LINKOPT> ::= 'LINKOPT' <MachineSpecificationList> ;

<BETARUN> ::= 'BETARUN' <MachineSpecificationList> ;

<MAKE> ::= 'MAKE' <MachineSpecificationList> ;

<BUILD> ::= 'BUILD' <MachineSpecificationList> ;
```



```

<RESOURCE> ::= 'RESOURCE' <MachineSpecificationList> ;

(*<LIBDEF> ::= 'LIB_DEF' <Name:TextConst> <Location:TextConst>;
  this gives qua error in sif (setsyncatno) *)

<LIBDEF> ::= 'LIB_DEF' <StringList>;

<LIBITEM> ::= 'LIB_ITEM' <Name:TextConst>;

<ON> ::= 'ON' <IntegerList>;

<OFF> ::= 'OFF' <IntegerList>;

<StringList>::* <TextConst> ;

<IntegerList>::+ <IntegerConst> ;

<MachineSpecificationList>::+ <MachineSpecification>;

<MachineSpecification> ::= <Machine> <StringList>;

<Machine> ::| <NameApl> | <Default> ;

<Default> ::= 'default' ;

<Other> ::= <NameDcl> <PropertyValueList> ;

<PropertyValueList> ::* <PropertyValue> ;

<PropertyValue> ::= <Value> ;

<Value> ::| <NameDcl> | <IntegerConst> | <TextConst> ;

<NameDcl> ::= <NameDecl>;

<NameApl> ::= <NameAppl>;

<TextConst> ::= <String>;

<IntegerConst> ::= <Const>

```

Index

The entries in the alphabetic index consists of all left-sides in the grammar. The small table of letters below links directly to the section of identifiers starting with the corresponding letters.

BDILMNOPRSTV

B

[<BETARUN](#)

[<BODY](#)

[<BUILD](#)

D

[<Default](#)

I

<INCLUDE <IntegerConst <IntegerList

L

<LIBDEF<LIBFILE <LIBITEM<LINKOPT

M

<Machine<MachineSpecification <MachineSpecificationList<MAKE<MDBODY

N

<NameApl <NameDcl

O

<OBJFILE<OFF <ON<ORIGIN <Other

P

<Properties<Property <PropertyList<PropertyOpt <PropertyValue<PropertyValueList

R

<RESOURCE

S

<StringList

T

<TextConst

V

<Value