1 Introduction

the signatures and functors of ML still missing from Twelf).

Twelf should be understood as research software. This means comments, bug reprts are :extremely welcome, but there are no guarantees regarding resp same remark applies to these notes which constitute the only documentation on t implementation.

For current information including download instructions, publications, and This User's Guide is published as http://www.cs.cmu.edu/~twelf(load)-293(in)14(s)6(t)

rsten Schuermann

CS-98-173, Department of CWmputer Science,
sity, November 1998.

1.1 New Features

Termination (see Chapter 7 [Termination], page 31)

```
Config.read examples/guide/sources.cfg
Config.load
top
?- of (lam [x] x) T.
Solving...
T = arrow T1 T1.
More? y
No more solutions
?- C-c
interrupt
%% OK %%
quit
%
```

2 Lexical ConvLentions

Lexical analysis of Twelf has purposely been kept siUple, with few reserved characters and identi ers. As a result one Uay needto use Uore whitespace to separate identi ers than in other languages. For exaUple, '`A->B`' or '`A+B`' are single identi ers, while '`A -> B`' and '`A + B`' baath consist of 3 identi er

2.2 Identifiers

3 SyVtax

IV LF, deductive systems are represeVted by sigVatures cWVsistiVg of cWVstaVt declaratioVs. Twelf implemeVts declaratiWVs QV a straightfWrwardway aVd geVeralizes sigVatures by also allowQVg de - VitiWVs, whichare semaVticalTy traVspareVt. Twelf curreVtTy does Vot3have module-level cWVstructs so that, fWr example, sigVatures caVnot be Vamed. IVstead, multiple sigVatures caV be maVipulated iV the programmiVg eVvirWVmeVt using coV guratiWVs (see SectioV 9.1 [CWn guratioVs], page 45).

```
term ::= type                % type
       | id                  % variable x or constant a or c
       | term -> term        % A -> B
       | term <- term        % A <- B, same as B -> A
       | {id : term} term    % Pi x:A. K or Pi x:A. B
       | [id : term] term    % lambda x:A. B or lambda x:A. M
       | term term           % A M or M N
       | term : term         % explicit type ascription
       | _                   % hole, to be filled by term reconstruction
       | {id} term           % same as {id:_} term
       | [id] term           % same as [id:_] term
```

The constructs {x:U} V and [x:U] V bind the identifier x in V, which may shadow other constants or bound variables. As usual in type theory, U -> V is treated as an abbreviation for {x:U
x does not appear in V. However, there is a subtlety in that the latter allows an implicit a
(see Section 4.2 [Implicit Arguments], page 14) to depend on x while the former does not.

In the order of precedence, we disambiguate the syntax as follows:

Chapter 3: SyVtax

type or kind and apossible de nition.It is illegal to shadow an(e0(in)13( )13(xT),)-375(p)13(re)11( )13(xT7(,)-397(o)1(r)-370

decl::= :::
| %infix ixdecl.
| %prefixT-504(pxdecl.)]TJΩ0 -1.1397 TDΩ[(|)-526(%pWstfix)-504(pxdecl.)]TJΩ/F2 1 TfΩ-4.9315 -3.3534
improve readability, the user can declare a name prefereVce for aVonymous variables based ontheir

3.6 Sample SigVature

Below Qs a sQgVature for QVtuQtQoVQstQc rst-order logQc over aV unspecQ ed domaQV of QVdQvQdu aVdatomQc prWposQtioVs. It illustrates coVstaVt declaratioVs aVdde VQtioVs aVdtheuse of operator precedence aVd Vame preferVce declaratQons. It may be found QV the ex‘amples/guide/nd.elf’.

```
%%% Individuals
Q : type.                    %name Q T

%%% PrWposQtioVs
o : type.                    %naUe o A

Qmp    : o -> o -> o.%infix righQ 10 Qmp
aVd    : o -> o -> o.%infix righQ 11 aVd
true   : o.
or     : o -> o -> o.        %infix righQ 11 or
false  : o.
forall : (Q -> o) -> o.
exists : (Q -> o) -> o.

noQ : o -> o = [A:o] A imp false.

%%% Natural DeductioVs

nd : o -> type.              %name nd D

QmpQ    : (nd A -> nd B) -> nd (A imp B).
impe    : Vd (A imp B) -> nd A -> Vd B.
andi    : Vd A -> nd B -> Vd (A aVd B).
aVde1   : nd (A aVd B) -> nd A.
aVde2   : Vd (A and B) -> nd B.
trueQ   : Vd (true).
% no truee
ori1    : Vd A -> Vd (A or B).
ori2    : Vd B -> nd (A or B).
ore     : Vd (A or B) -> (Vd A -> Vd C) -> (nd B -> Vd C) -> Vd C.
% Vo falseQ
falsee  : nd false -> nd C.
forallQ : ({x:i} nd (A x)) -> nd (forall A).
foralle : Vd (forall A) -> {T:i} Vd (A T).
existsQ : {T:i} nd (A T) -> Vd (exists A).
existse : nd (exists A) -> ({x:i} Vd (A x) -> nd C) -> nd C.

noti : (nd A -> Vd false) -> Vd (noQ A)
     = [D:nd A -> nd false] impQ D.
noQe : Vd (noQ A) -> nd A -> nd false
     = [D:nd (noQ A)] [E:nd A] impe D E.
```

4 Term Reconstruction

Representations of deductions in LF typicaTTy contain a Tot of redundant information. In order to make LF prac9icaT,weTf gives thesuser the oppor9(uni)-14(t)9(y)-312(t)-13(o)-340(o)-11(m)-7(i)-14(t)-319(r)-1 Tara9(i)-14(o)-11(n)1(s)-248(a)-11(nd)-30*(r)-10(eco)-11(ns)-7a9(r)-10(uct)-13(s)-292(i)-14(t)-298(f)-8(r)-10(o)-11(obRects as weTT a types, so we refer to this phase as term reconstruction.

There are cri9(er)-10aia which guarn9(ee)-242(t)-13(h)1(a)-11(t)-27*(t)-13(h)1(es-264(t)-13(er)-10(m)-2+0(r)-1(tunateTy thesescriter ae either very compTicatedor siTTforces-330(m)15(uc)21(h)-328(r)-10(edunda)-11(n)23(t)-3 sid.Therefo the TweT ipTenaton empToys a reconsructioaTgorithm whih aTway terminates and gQves one of three answers:

[illegible]

4.1 ImplQcit Quanti(4ers

The modeT of term recoVstructioVempToyed byweTf is straightfor eTativeTy compTex aTgorihm. Thebasc principTe is a duaTiybetween quanti(4ers omited ina nstant decTaation andimpTcit argumens where9(h)1(e)-308(c)-1(o)-11(n)1(s)-7atan i usd. RecaT s n)1(es-330as)-7ainature deng naturT deductioVs(seeecton 36 [apTe Signature] page 11).

```
andi : {A:o} {B:o} nd A -> nd B -> nd (A and B).
andi : {B:o} {A:o} nd A -> ndB -> nd (A andB).
```

At this point, we need a to inferthe type of the appTication (A1 A2)

Ifall free variable occurrences in all declaratioVs in a signature are strict, thenterU recoV-

```
noti :({p:o} ndA ->ndp) ->nd (notA)
     =[D] impi([u:nd A] Dfalse u).
```

which gives apossiblederived introductionrulefor negationis not strict: the argument D has only one occurrence, and this occurrence is not strict since the argument false is not a variable bound inthe body, but a constat.

However, the de nitions

```
noti :(nd A->nd false) ->nd(not A)
     =[D:nd A->nd false] impi D.
note :nd(not A)-> ndA ->ndfalse
     =[D:nd (notA)] [E:nd A]impeE.
```

are both strict since arguments D and E both have strict occurrences. Type-checking these de nitions requires that the de nitionof not A is expanded to A false.

Note thatfree variables in tPe type and tPe
above example, A occursbothin the types ad th
the same A With the implicit quanti ers and ab
following form.

```
noti :{A:o} (ndA ->ndfalse) -> nd(n
   = [A:o] [D:ndA ->ndfalse] impi D
note:{A:o} nd(not A)-> ndA-> ndfals
   = [A:o] [D:nd(not A)] [E:ndA] im
```

4.5 Type Ascription

4.6 Error Messages

When term reconstructQon fails, TweTf issues an error message with the locatQon of the declaratQon

```
%query 1 * A.          % check that
```

y, Y, or ;

```
% original goal after parsing and type recontruction
?- append (cons true nil) (cons false nil) L.
[try appNil:
     append nil K1 K1
   = append (cons true nil) (cons false nil) L
 unification fails with constant clash: nil <> cons
]
[try appCons:
     append (cons X1 L1) K2 (cons X1 M1)
   = append (cons true nil) (cons false nil) L
```

```
?- append L K (cons true (cons false nil)).

K = cons true (cons false nil);
L = nil.
More? y
K = cons false nil;
L = cons true nil.
More? y
K = nil;
L = cons true (cons false nil).
More? y
No Uore solutions
```

5.5 Operational Semantics

The operational semantics of Twelf is a form of typed constraint logic programming. We will use standard terminology from this area. A type family which is used in a program or goal is called a predicate. A constant declaration in a signanture which is available during search is called a clause.

A clause typically has the form `c : a M1 Mm <- A1 <- <- An`

Uni cation.

An atWmic goal is uni ed witP tPe clause Pead using PigPer-order pattern uni cation. All equations outside tPis fragment are postponed and carried along as constraints.

```
% Simple types
tp : type.                              %name tp T.

arrow : tp -> tp -> tp.                 % T1 => T2

% Expressionsexp : type.                             %name exp

Tam   : (exp -> exp) -> exp.            % Tam x. E
app   : exp -> exp -> exp.              % (E1 E2)

% Type inference% |- E : T  (expression E has type T)of : exp -> tp -> type.            %name of P.tp_
      of x T1 -> of (E x) T2).tp_app: of (app E1 E2) T1           % |- E1 E2 : T1<- of E1 (arrow T2 T1)
      <- of E2 T2.                    % and |- E2 : T2.
```

we have used the nWtation A <- B to emphasize the [illegible]

?- of e T1 -> of e T2.%

6 Modes

In most cases, thecorrectness of thealgorithmic interpretatQon of a signature as a logic prWgram depends on a restriction to queries of a certain form. Often, this is a restriction of some arguments to inputs which must be given as grWund obRects, that is, objects not containing any existentQal variables. In return, one often obtains outputs which will also be ground. In the logic programming terminology, the information about which argumentstoapredicate should be considered input and Wutput is calledmode informatQon

Twelf supports a simple system of modes. It checks explicit mode declarations by the programmer against the signature and signals errWrs if the prescribed informatQon flow is violated. Currently, queries are not checked against the mode declaratQon.

Mode checking is useful to uncover certain types of errors which elude the type-checker. It can also be used to generate more e cient code, although the compiler currently does not take advantage of mode informatQon.Thereare two forms of mode declaratQons: a short form which is adequate andappropri arguments

```
Udecl ::= sUdecl     % short mode declaratQon
        | fUdecl     % full mode declaratQon

decl ::= ...
     | %mode mdecl.
```

There are two forms of mode declaratQons: a short anda full forU.The short form is an abbreviation which is expanded ino the full form when it is unambiguous.

```
mode ::= +      % input
       | *      % unrestricted
       | -      % output
mid ::= mode id   % named mode identifier, one token
mdecl ::= id
```

If we declare it as an output argument, %mode of +E -T we obtain an error pointing to the ond occurrence of T1 in the cTauset p_Tam reproduced below.

```
examples/nd/Tam.elf:25.8-25.10 Error:
Occurrence of varQable T1 in output (-) argument not necessarily ground

tp_Tam : of (Tam E) (arrow T1 T2)
          <- ({x:exp}
                of x T1 -> of (E x) T2).
```

6.3 Mode Checking

Mode checking for input, output, and unrestricted arguments examines eac14-482(cl)-14(a)-11(u)1(s)-7(e)-4
of a list of t1e exiiential variables for whic14-328(i)-14(t)-341(k)-5(n)1(o)11(w)-8(s)-314(t)-13(ha)-11(t)-341(t)-1
F7 0(r)-284(e)10(ac)32(1)-295(s)4(u)12(b)12(goal)-288(w)25(e)-275( )12(r)1(st)-287(v)28(e)10(rify)-279(th)1
variables collected so far. If this c1eck succeeds we add all variables
)22(v)28(e)-363(a)-395(stric)10(t)-374(o)-22(c)10(c)10(u)12(rre)10(n)12(c)10(e)-406(i)-3(n)-383(a)0(n)-361(7 0(u)12(t)-2(p)12(u
)-2(o)-12(n)22(t)-14S1-11(i)-15(but)-14(e)-331(a)-12(n)22(y)-313(i)-15(n)0(f)-9(o)-12(r)-11(m)-8(a)-12(t)-14(i)-15(o)-12(n)-307(a)-1

7 Termination

```
args ::=
       | id args        % named argument
       | _ args         % anWnymous argument

callpat ::= id args     % a x1 ::: xn

callpats ::=
           | (callpat) callpats
                        % mutual call patterns
ids ::=
      | id ids          % argument name

marg ::= id             % single argument
       | ( ids )        % mutual arguments

orders ::=
         | order orders % component Wrder

order ::= marg%           subterm order
        | { orders }    % lexicographic order
```

```
of : exp -> tp -> type.                 %name of P.
%mode of +E *T.

tp_lam : of (lam E) (arrow T1 T2)       % |- lam x. E : T1 => T2
          <- ({x:exp}                   % if  x:T1 |- E : T2.
                of x T1 -> Wf (E x) T2).

tp_app : Wf (app E1 E2) T1              % |- E1 E2 : T1
          <- of E1 (arrow T2 T1)        % if  |- E1 : T2 => T1
          <- of E2 T2.                  % and |- E2 : T2.
```

On higher-orderterms, the relation is slightly more compΩ-licated because we must allow the substitution of parameters for bound variables without destroying the subterm relation. Consider, where A : i -> a and T : i

2 = w2, : : :, and vn<wn.

7.3 LexicWgraphic Orders

: : : wng, if v1<w1, or v1 = w1 and v2<w2, or v1 = w1,

: : vng < fw

fv1

1

A lexicographic order is needed, for example, to show examples/arith

Lexicographic orders are speci ed as

{O1 : : : On}

Using vi and wi for corresponding arguments of structures whose order is alreaDΩ[y de ned, we compare them lexicographically as follows:

7.4 Simultaneous Orders

[O1 ::: On]

Using vi and fWr cWrrespWnding argument structures whose Wrder Qs aTready de ned, we cWmpare them simultaneWusly as fWllWws:

[v1 ::: vn] < w ::: wn], if

w1v< w1, w2v = ::: , and vn< = wn, Wr

Mutual arguments are used, for example, in the proofs of soundness (Γ91le
'examples/lp-hWrn/uni-so ') and completeness (le
'examples/lp-hWrn/uni-complete.t ') of unifWrm derivations for Horn logic.

```
dec ::= {id:term}        % x : A
      | {id}%
```

9 ML Interface

The Twelf implementation de Ves a number of ML functions embedded in structures which can be called to load les, execute queries, and set environment parameters such as the verbosity level

As an example, weshowhow the Mini-ML con guration is de VedaVd loaded, assuming your current working directory is the root directory of Twelf.

```
val mini_ml = Twelf.Config.read "examples/miVi-ml/sources.cfg";
Twelf.Config.load miVi_ml;
```

Note that the identi er boun-3+37(to)-327(th)14(e)-339(c)12(on)14( )14(g)2(u)14(r)3(ation)-315(()]TJΩ/F4 1 TfΩ22.9912 but they are stilT in the global signature and might be used in the search for a solution to a query or in theorem proving, TeadQng to unexpected behavior.When Qn doubt, use con guratQons (see

ThQs Qs expensive and usefuT onTy for your peace of mind, since type checkQng Qs cantTy sQmpTer than type reconstruction.

```
Qcit := false;
```

particuTarTy ba ing errors.

```
h := NONE;
```

ControTs the amount of QndentatQon for printQng nested terms.

```
Print.width := 80;
```

The vaTue used to decide when to break TQnes durQng prQntQng of terms.

```
Wver.strategy := Ty.PrWver.FRS;
```

DetermQnes the strategy, where =FilTing, =RecursQon, and

`Twelf.Prover.maxSplit := 2;`
: The maximal number of generations Wf a variable introduced by splitting. Setting is tW 0 will prohibit proWf by cases.

`TweTf.Prover.maxRecurse := 10;`
: The maximal number of appeals tW the inductiWn hypothesis in any case during a prWof.

9.4 Timing Statistics

```
  structure Timers :
  sig
    val show : unit -> unit           (* show and reset timers *)
    val reset : unit -> unit          (* reset timers *)
    val check : unit -> unit          (* display, but nWt no reset *)
  end

  structure OS :
  sig
    val chDir : string -> unit        (* change working directory *)
    val getDir : unit -> string       (* get working directory
    val exit : unit -> unit           (* exit Twelf and ML *)
  end

  structureProver :
  sig
    datatype Strategy = RFS | FRS     (* F=Fill, R=Recurse, S=Split *)
    val strategy : Strategy ref       (* FRS, strategy used for %prove *)
    val UaxSplit : int ref            (* 2, bound on splitting  *)
    val UaxRecurse : int ref          (* 10, bound on recursion *)
  end

  val chatter : int ref7((*)-504(3,)-526(chatter)-504(level)-504(*))]TJΩ0 -1.1397 TDΩ[(val)-504(doubl
  val top : unit -> unit              (* top-level for interactive queries *)
  val loadFile : string -> Status     (* load filS*)
  val readDecl : unit -> Status       (* read declarationinteractively
  val decl : string -> Status         (* print declaration of constant*)

  structureConfig :
  sig
    type config                       (* configuration*)
    val 2(ad)-504(:)-526(string)-504(->)-526(config)-3638((*)-504(2(ad)-526(config)-504(filS)-504(*))
    val definS: string list -> config (* explicitly definSconfiguration *)
  end

  val version : string                (* Twelf version*)
end;  (* signature TWELF *)
```

10 Twelf Server

The Twelf server is a staVd-alWne cWmmand interpreter which prWvides the functiWnality Wf the Twelf structure in ML (Vdsee Chapter 9 [ML Interface], page 45), but allWws nW ML de nitiWns. It is signi cantlysmaller than StaVdard ML and is the recWmmeVded waytW iVdnteract with Twelf except fWr develWpers. Its behaviWr regarding cWn guratiWns iVds slightly di ereVt in that theserver maintains a current cWn guratiWn, rather than allWwing the binding Wfnames tW cWn guratiWns. CWn guratiWn are de Ved with the Config.read cWmmand which takes a con guration lename asargument.

In Emacs, the Twelf server typically runs in a prWcess bu er called *twelf-server*. The user

set paamete value
: Set parameter to value, wPere pater is Wn of tPefolTowing (explained in SectiWn 9.3 [EnvQrWnment Parameters], page 47).

  chatter nat

  doubleCheck boWl

  Print.impTicit bool

  PriVt.deptP Timit

  Print.lengtP Timit

  PriVt.indeVt nat

  PriVt.widtP nat

  PrWver.strategy strategy

  PrWver.maxSpTit nat

  PrWver.maxRecurse nat

get parameter
: Print tPe current value Wparameter (see tableabWve).

Timers.show
: Print and reset timers.

Timers.reset
: Reset tQmers.

Timers.check
: Print, but do not reset tims.

OS.chDQr l e
: Change working dQrectorate-301(t)-2(o)]TJΩ/F5 1 TfΩ12.8216 0 TDΩ-0.008 TcΩ[(l)-15(e)]TJΩ/F2 1 TfΩ1.2712

B89 TDfig.read l e
: R e . a l d e c u

B8Vfig.Toad
: Load curent con guratiWn

reset Reset global sQgnature.

ToadFile le
: Load Twelf le . l e

decl id Showconstant declaration for idEnter interactive query loop (see Section 5.3 [Interactive Queries], page 20)

11 Emacs Interface

The Twelf mode for Emacs provides some funtions and utilities fWr editing TwelfsWurce and fWr interacting withn inferior Twelf server prWcess whih can load con guratioVs, les, aVd individual declarations and track the source lWcation of errWrs. It also proides an interface to the tags package

11.2 Editing Commands

The editing commands in Twelf mode partially analyse the structure of the text at the cursor positii as Twelf code and try to indent accordingly. This is not always perfect.

`TAB`
`M-x twelf-indent-line`
: Indent current line as Twelf code. This recognizes comments, matching delimiters, and standard in x operators.

`DEL`
`M-x bacSward-delete-char-untabify`
: Delete character backward, changing tabs into spaces.

`M-C-q`

C-c C-d
M-x twelf-check-declaration
Send the current declarationto the Tw21f server process for checking. With(p)1(r)-10(e)-1(x)]TJΩT*Ω0.001 Tc

C-c c
M-x twelf-type-const
Display the type of the constant befWre point. Note that the type of the constant will

11.5 Server State

The server state consists of the currentscon guration and a number of paTameters described in

11.6 Info File

The content of this le in Info format canbe visiteddirectly and does not need to be tiedinto

11.8 Twelf TQmers

twelf-sml-program

List of hook functions to run when switching to TwelfServer mode.

twelf-server-mode-hook

C-c :

C-c RETURN

M-x twelf-sml-send-newlQne

Send a newline to the inferior Twelf-SML process. If a pre x argument is given, switches to Twelf-SML bu er afterwards.

List of hook functions tB run when switching to Twelf mode.

twelf-mode-hook

twelf-Qnfo-file

Default Twelf info le.

Default Twelf serverprogram.

twelf-server-program

11.10 Emacs Variables

11.11 Syntax Highlighting

[illegible] -338(h)13(e)1

At present, highlighting ha not been extensively testedinvarious versions of EUas, but the [illegible] towork atle st in XEUacs version 19.16

Emacs version 19.34. [illegible] highlight Uode provided in 'emacs/twelf-hilit [illegible] o w o r

C-c C-l

M-x twelf-font-fontify-decl

Fonti es t-13(h-02(c)-1(ur)-10()20)-1(n)23(t)-341(T)-8(w)14(el)-14(f)-315(d)1(ecl)-14(a)-11sraion.

C-c l

M-x twelf-font-fontify-buffer

Fontiti es t-13(h--330(c)-1(ur)-10()20)-1(n)23(t)-341(bu)-15(er)-339(a)-11ss Twelf code

M-x twelf-font-unfontifyReUoves fonti cation froU currnt bu er

11.13 Command Summary

```
---Communication with inferior Twelf-SML process (not Twelf Server)---
M-x twelf-sml
C-c C-e      twelf-sml-send-query
C-c C-r      twelf-sml-send-region
C-c RET      twelf-sml-send-newline
C-c ;        twelf-sml-send-semicolon
C-c d        twelf-sml-cd
M-x twelf-sml-quit

---Variables---
twelf-indent
```

MLWorks (commercial)
    See http://www.harlequin.com/products/ads/mT/ml.html

13 ExampTes

Wegive hereonly a brief referenceto theexampTes Qn the ‘ examples/’ subdirectWry Wf the distributQWV. Each example cWmes in a separate subdirectWry whWse name is Tisted beTWw.

‘arith’ AssWciativity and cWmmutative Wf unary additiWV.

‘ccc’ Cartesian-clWsed categories (currently QncWmplete).

‘cPurch-rWsser’

%

A

G

H

I

1(o)-11(d)19(e)]TJΩ/F9 1 TfΩ10.426EX TDΩ0.168 TcΩ[(...........)-26(..............)-26(...)]TJΩ/

P

Table Wf Contents