ABNF in ACL2

Alessandro Coglio

Kestrel
Institute

ACL2 *Workshop 2017*

Augmented Backus-Naur Form is a formal context-free grammar notation that adds conveniences and makes slight modifications to Backus-Naur Form, e.g.:

numeric range terminal notations

case-insensitive string terminal notations

```
DIGIT = %x30-39

HEXDIG = DIGIT / "A" / "B" / "C" / "D" / "E" / "F"

h16 = 1*4HEXDIG

message-body = *OCTET

RWS = 1*( SP / HTAB )
```

repetition prefixes, min. (default 0) to max. (default ∞)

ABNF is specified by two RFCs (i.e. Internet standards).

Network Working Group — D. Crocker, Ed.
Request for Comments: 5234 — Brandenburg InternetWorking
STD: 68 — P. Overell
Obsoletes: 4234 — THUS plc.
Category: Standards Track — January 2008

Augmented BNF for Syntax Specifications: ABNF

Status of This Memo

This document specifies an Internet standards track protocol for the Internet community, and requests discussion and suggestions for improvements. Please refer to the current edition of the "Internet Official Protocol Standards" (STD 1) for the standardization state and status of this protocol. Distribution of this memo is unlimited.

Abstract

Internet technical specifications often need to define a formal syntax. Over the years, a modified version of Backus-Naur Form (BNF), called Augmented BNF (ABNF), has been popular among many Internet specifications. The current specification documents ABNF. It balances compactness and simplicity with reasonable representational power. The differences between standard BNF and ABNF involve naming rules, repetition, alternatives, order-independence, and value ranges. This specification also supplies additional rule definitions and encoding for a core lexical analyzer of the type common to several Internet specifications.

Crocker & Overell — Standards Track — [Page 1]

Internet Engineering Task Force (IETF) — P. Kyzivat
Request for Comments: 7405 — December 2014
Updates: 5234
Category: Standards Track
ISSN: 2070-1721

Case-Sensitive String Support in ABNF

Abstract

This document extends the base definition of ABNF (Augmented Backus-Naur Form) to include a way to specify US-ASCII string literals that are matched in a case-sensitive manner.

Status of This Memo

This is an Internet Standards Track document.

This document is a product of the Internet Engineering Task Force (IETF). It represents the consensus of the IETF community. It has received public review and has been approved for publication by the Internet Engineering Steering Group (IESG). Further information on Internet Standards is available in Section 2 of RFC 5741.

Information about the current status of this document, any errata, and how to provide feedback on it may be obtained at http://www.rfc-editor.org/info/rfc7405.

Copyright Notice

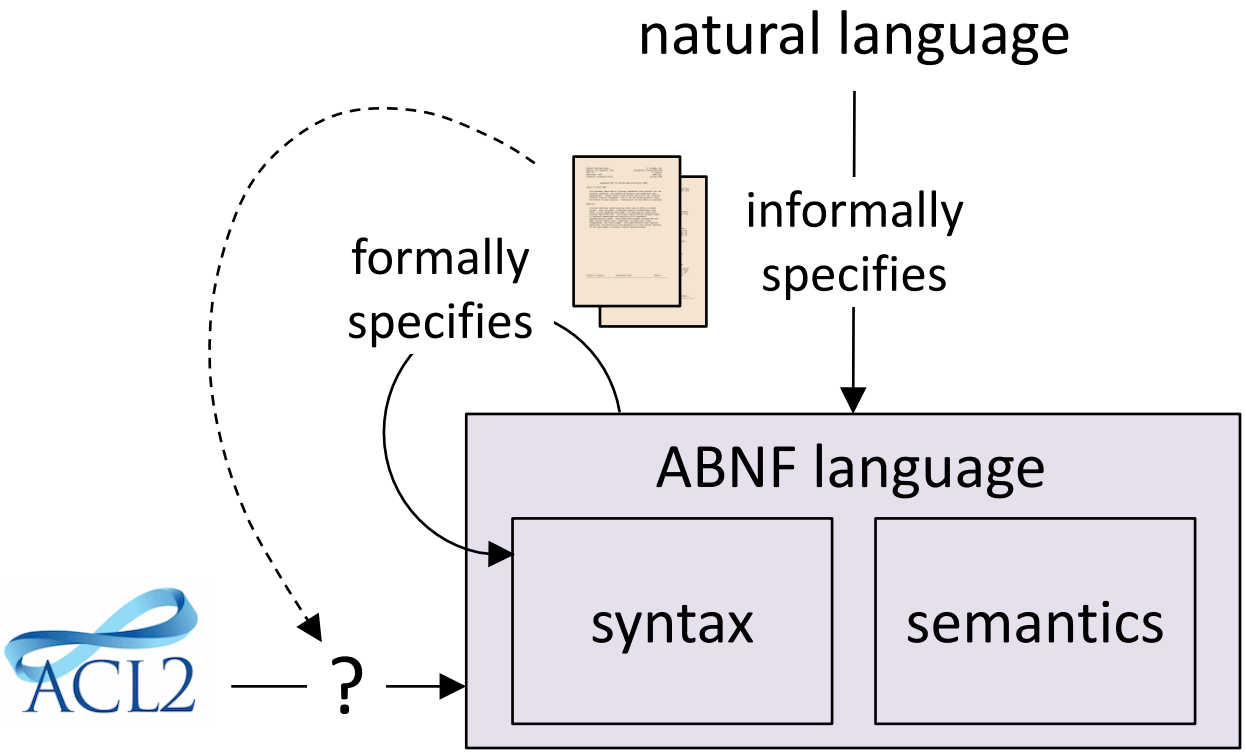
Copyright (c) 2014 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (http://trustee.ietf.org/license-info) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

Kyzivat — Standards Track — [Page 1]

The RFCs use natural language to informally specify ABNF.

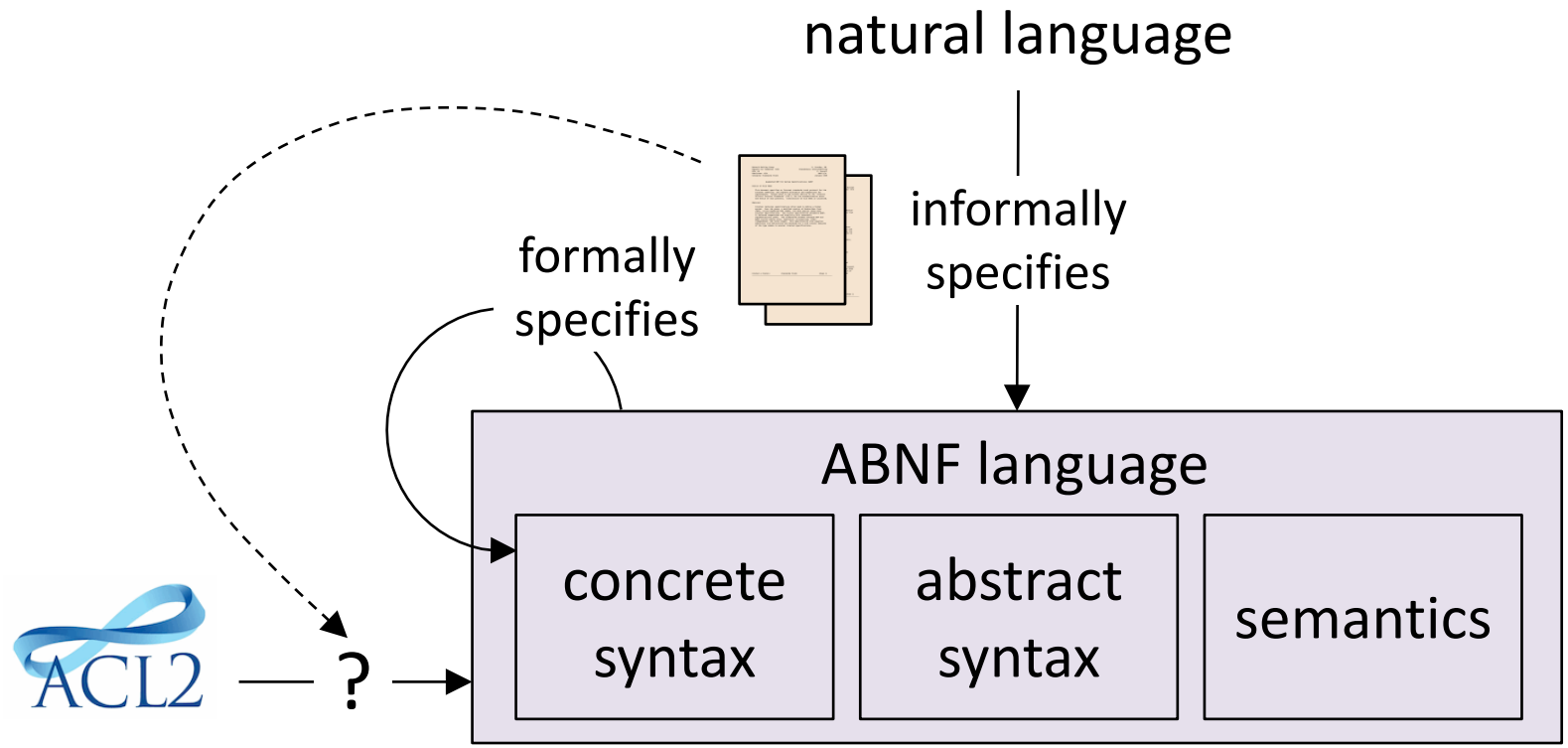
And they use ABNF to formally specify the syntax of ABNF.



How to formally specify ABNF in ACL2, faithfully to the RFCs, including the meta-circular formal syntax specification?

The RFCs use natural language to informally specify ABNF.

And they use ABNF to formally specify the syntax of ABNF.



How to formally specify ABNF in ACL2, faithfully to the RFCs, including the meta-circular formal syntax specification?

Formalize an abstract syntax of ABNF in ACL2, based on the ABNF grammar rules that define the concrete syntax of ABNF, e.g.:

```
alternation = concatenation
              *(*c-wsp "/" *c-wsp concatenation)
concatenation = repetition *(1*c-wsp repetition)
repetition = [repeat] element
element = rulename / group / ...
group = "(" *c-wsp alternation *c-wsp ")"
...
```

```
(fty::deftypes alt/conc/rep/elem
  (fty::deflist alternation :elt-type concatenation)
  (fty::deflist concatenation :elt-type repetition)
  (fty::defprod repetition
    ((range repeat-range) (element element)))
  (fty::deftagsum element
    (:rulename ((get rulename)))
    (:group ((get alternation)))
    ...)
  ...)
```

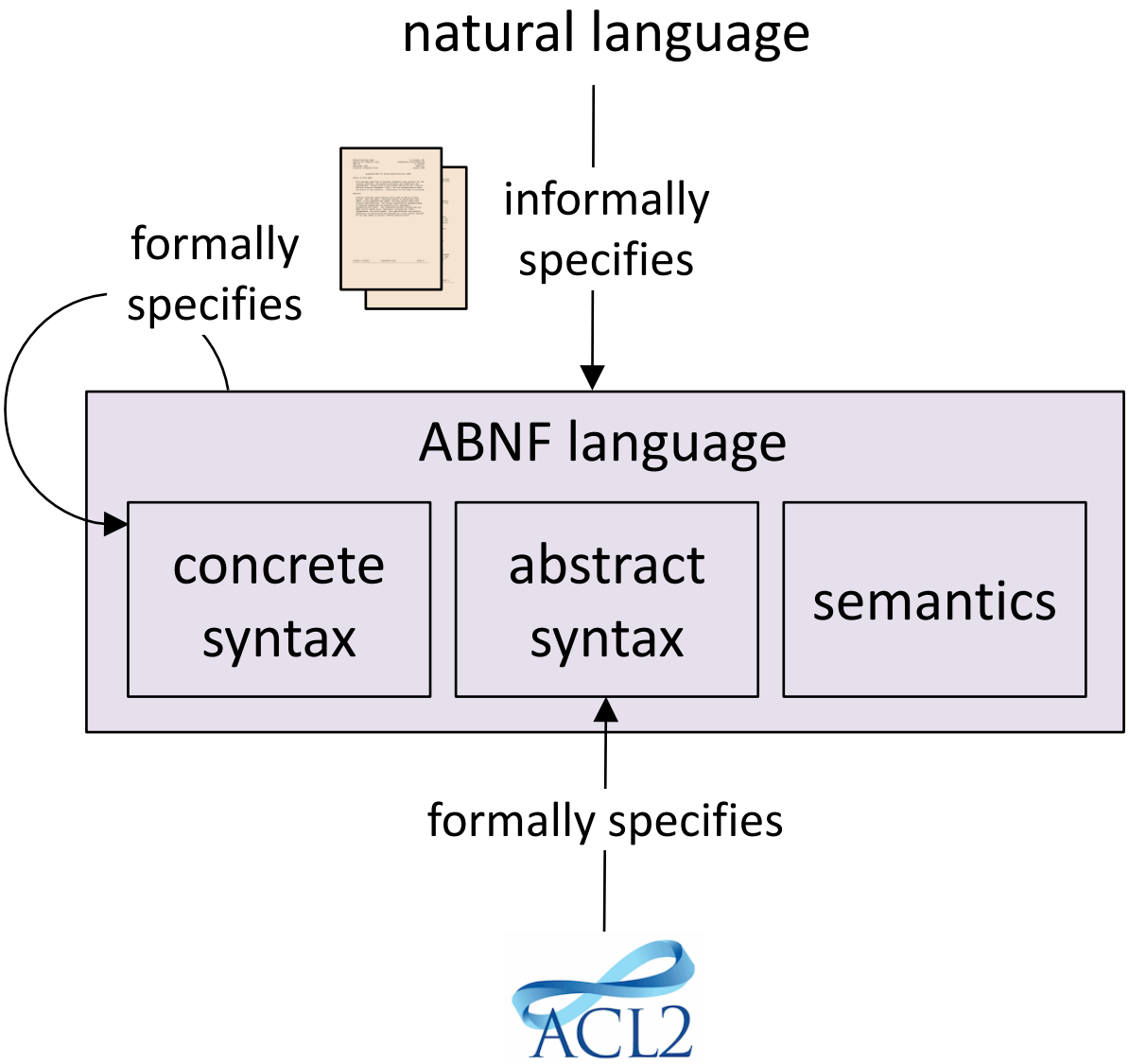
The start nonterminal of the ABNF grammar of ABNF is `rulelist`.

```
rulelist = 1*( rule / (*c-wsp c-nl) )
```

In the abstract syntax, an ABNF grammar is a value of type `rulelist`.

```
(fty::deflist rulelist :elt-type rule)
```

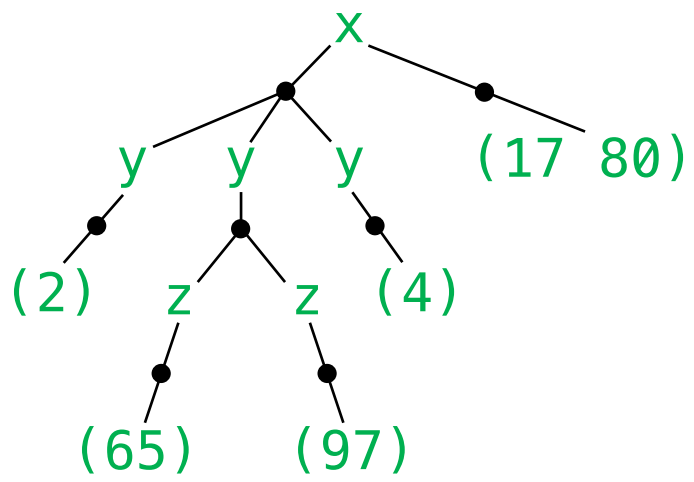
These values (grammars) can be operated upon in the ACL2 logic, e.g. to check their well-formedness and to compose them.


natural language
informally specifies
formally specifies
ABNF language
concrete syntax
abstract syntax
semantics
formally specifies
ACL2

Formalize a semantics of ABNF in terms of matching relations between parse trees and (abstract) syntactic entities, e.g.:

```
x = *y %d17.80
y = 2z / %d1-4
z = "a"
```

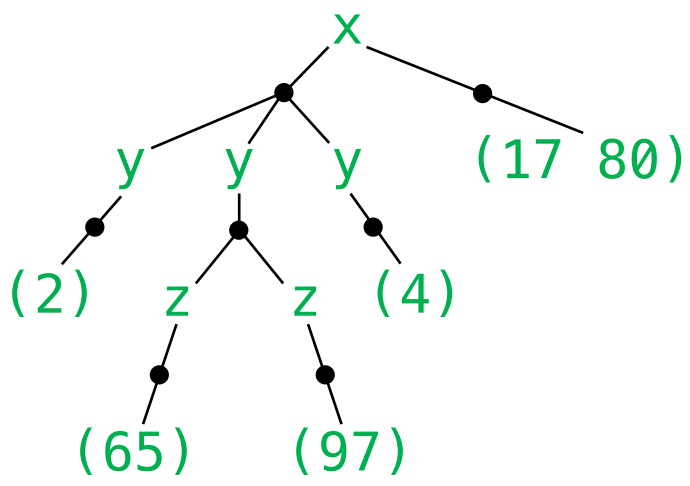
a grammar



branches are organized into lists (for concatenations) of lists (for repetitions)

a parse tree that matches x

Formalize a semantics of ABNF in terms of matching relations between parse trees and (abstract) syntactic entities, e.g.:



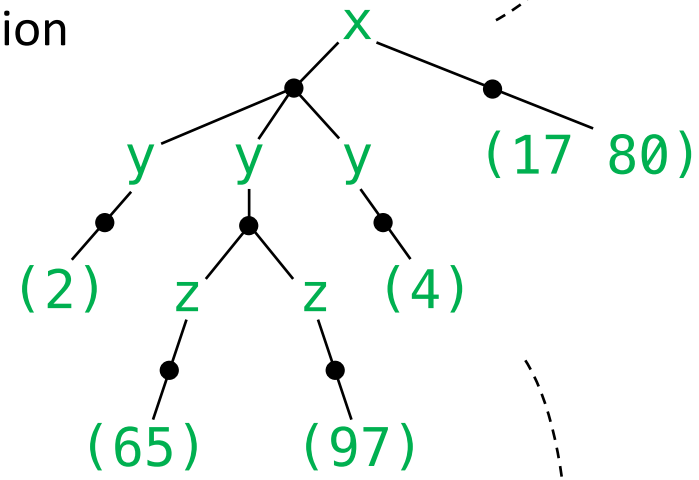
type of parse trees

```
(fty::deftypes trees
  (fty::deftagsum tree
    (:leafterm ((get nat-list)))
    (:leafrule ((get rulename)))
    (:nonleaf ((rulename? maybe-rulename)
               (branches tree-list-list))))
  (fty::deflist tree-list :elt-type tree)
(fty::deflist tree-list-list :elt-type tree-list))
```

Formalize a semantics of ABNF in terms of matching relations between parse trees and (abstract) syntactic entities, e.g.:

```
(tree-match-element-p tree element rules)
```

matching relation



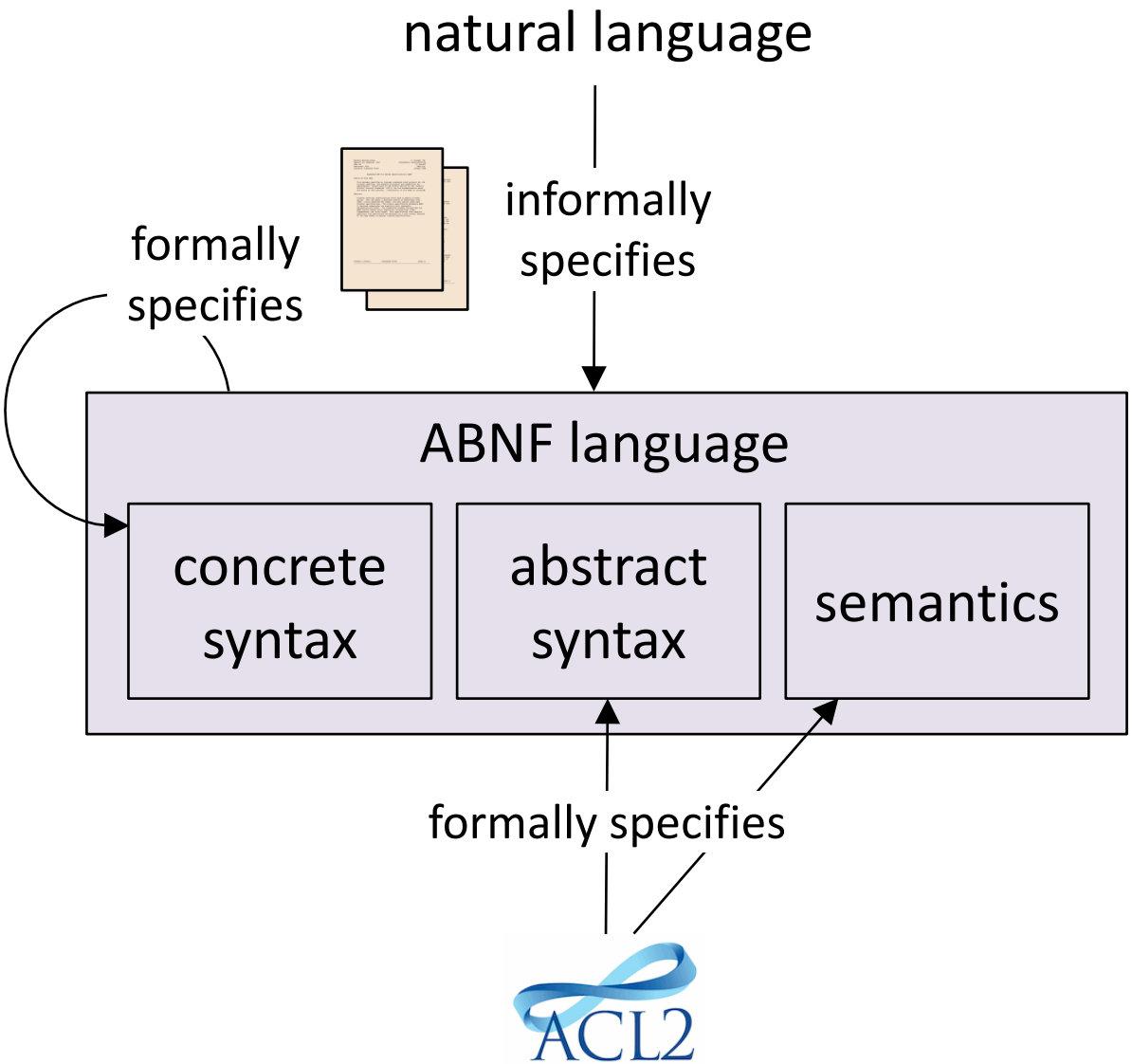

```
x
```

```
x = *y %d17.80
y = 2z / %d1-4
z = "a"
```

```
(equal (tree->string tree) string)
```

string at the leaves of the tree

```
(2 65 97 4 17 80)
```


natural language
informally specifies
formally specifies
ABNF language
concrete syntax
abstract syntax
semantics
formally specifies
ACL2

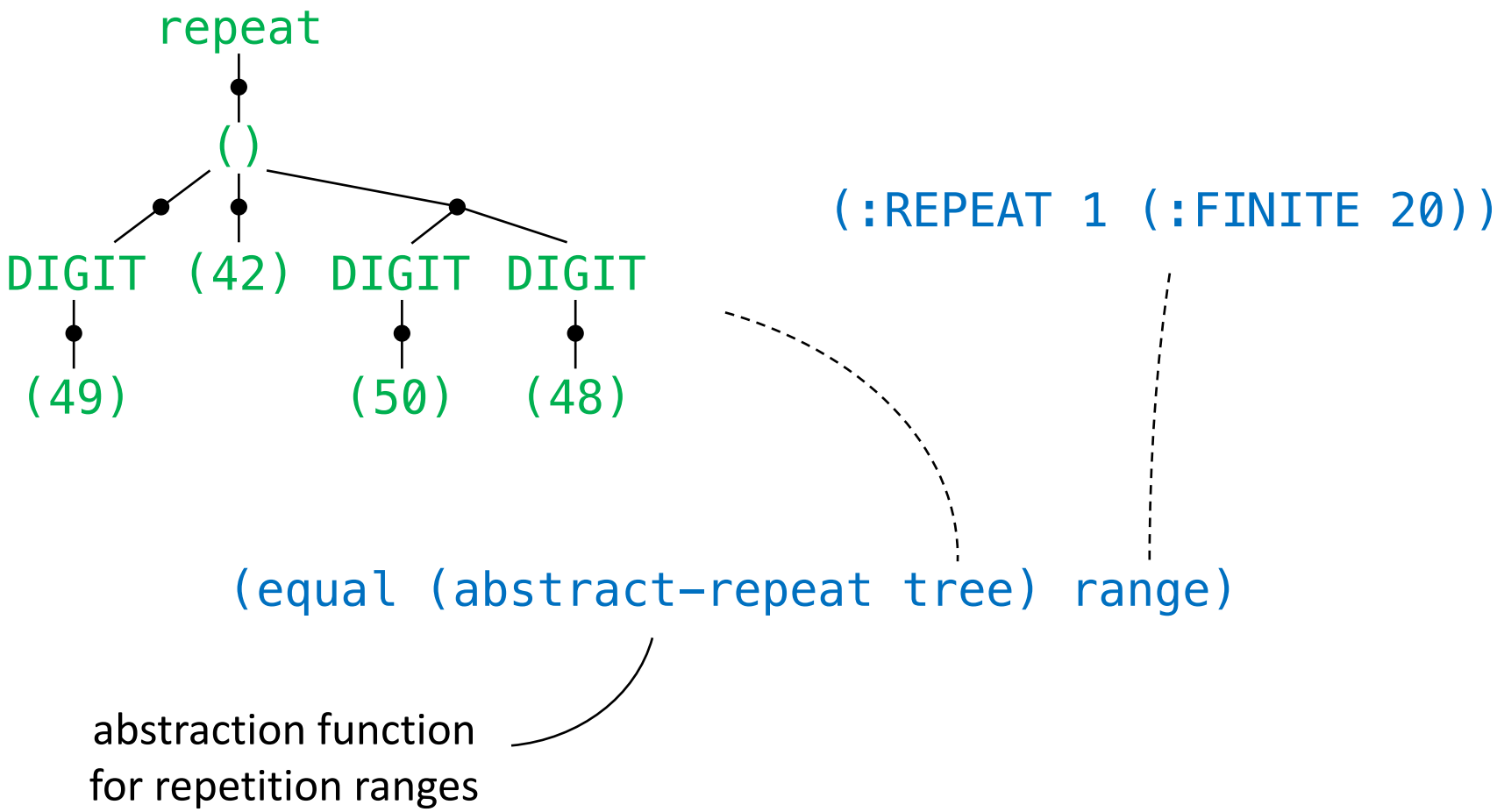
Formalize the concrete syntax of ABNF by transcribing the ABNF grammar rules of ABNF “in abstract syntax”, thus breaking the meta-circularity, e.g.:

```
num-val = "%" (bin-val / dec-val / hex-val)

(def-rule-const *num-val*
  (/_ "%" (!_ (/_ *bin-val*)
              (/_ *dec-val*)
              (/_ *hex-val*))))
```

specially crafted and named functions and macros

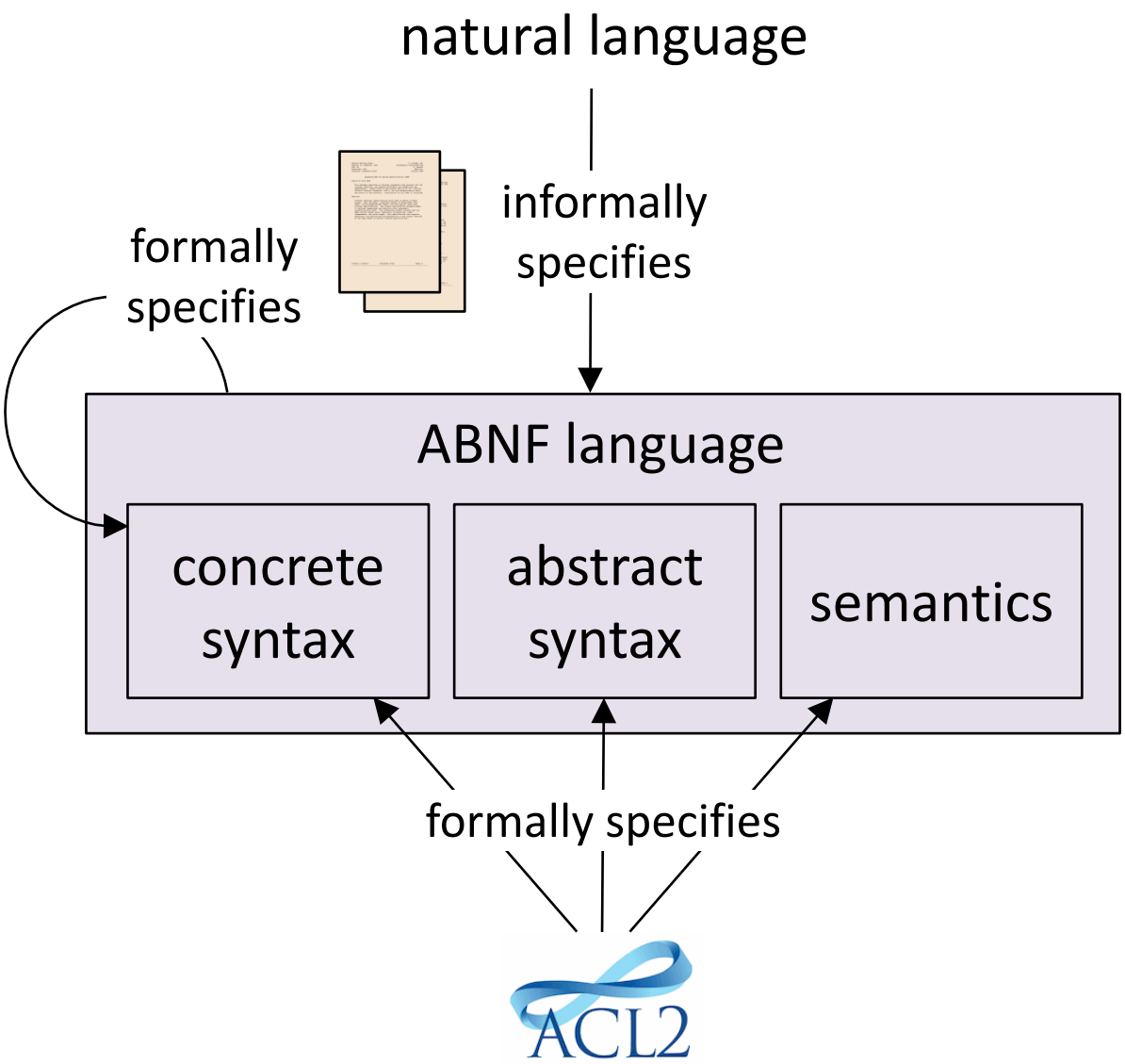
```
group = "(" *c-wsp alternation *c-wsp ")"

(def-rule-const *group*
  (/_ "(" (*_ *c-wsp*) *alternation* (*_ *c-wsp*) ")"))
```

Formalize the relationship between concrete and abstract syntax via abstraction functions from parse trees of ABNF grammars to corresponding abstract syntactic entities, e.g.:



Formalize the relationship between concrete and abstract syntax via abstraction functions from parse trees of ABNF grammars to corresponding abstract syntactic entities, e.g.:

abstraction function
for lists of rules


natural language
informally specifies
formally specifies
ABNF language
concrete syntax
abstract syntax
semantics
formally specifies
ACL2

ABNF is used in several Internet syntax specifications, e.g. HTTP.

Internet Engineering Task Force (IETF)
Request for Comments: 7230
Obsoletes: 2145, 2616
Updates: 2817, 2818
Category: Standards Track
ISSN: 2070-1721

R. Fielding, Ed.
Adobe
J. Reschke, Ed.
greenbytes
June 2014

Hypertext Transfer Protocol (HTTP/1.1): Message Syntax and Routing

Abstract

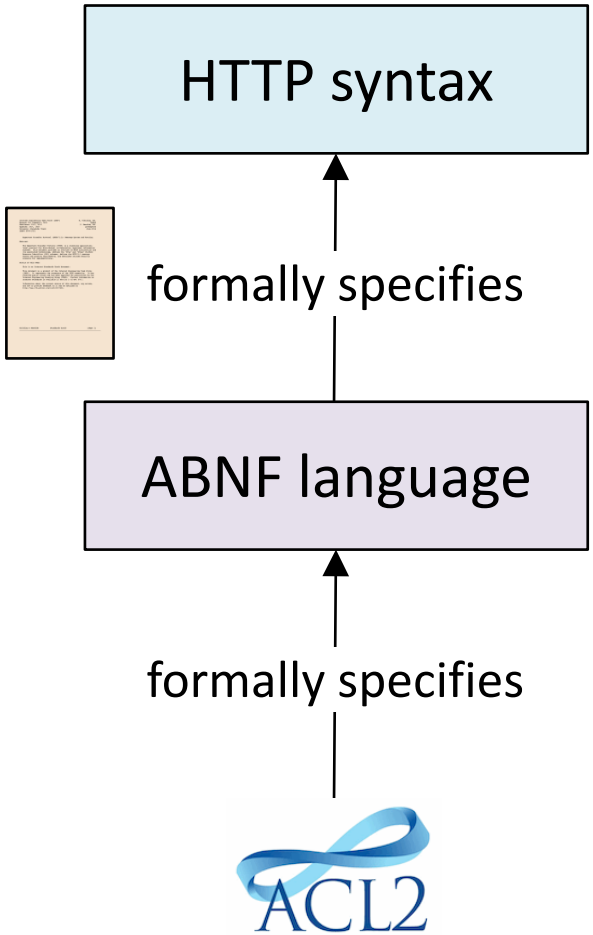
The Hypertext Transfer Protocol (HTTP) is a stateless application-level protocol for distributed, collaborative, hypertext information systems. This document provides an overview of HTTP architecture and its associated terminology, defines the "http" and "https" Uniform Resource Identifier (URI) schemes, defines the HTTP/1.1 message syntax and parsing requirements, and describes related security concerns for implementations.

Status of This Memo

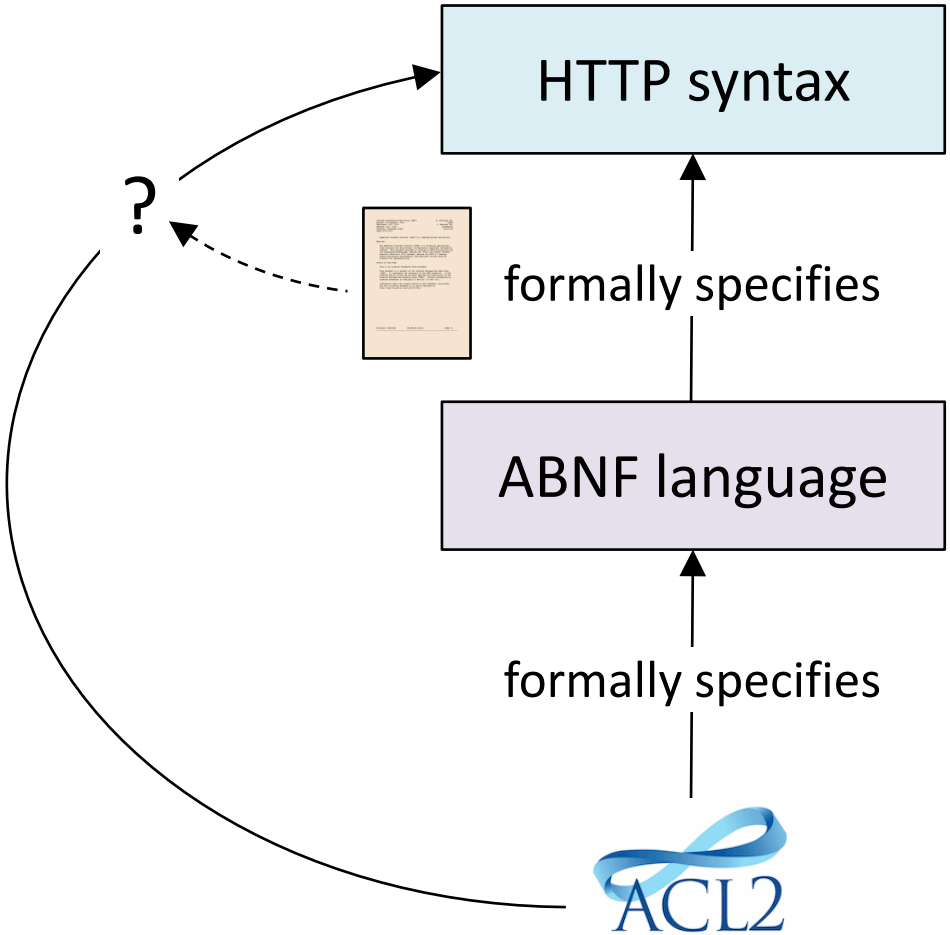
This is an Internet Standards Track document.

This document is a product of the Internet Engineering Task Force (IETF). It represents the consensus of the IETF community. It has received public review and has been approved for publication by the Internet Engineering Steering Group (IESG). Further information on Internet Standards is available in Section 2 of RFC 5741.

ABNF is used in several Internet syntax specifications, e.g. HTTP.

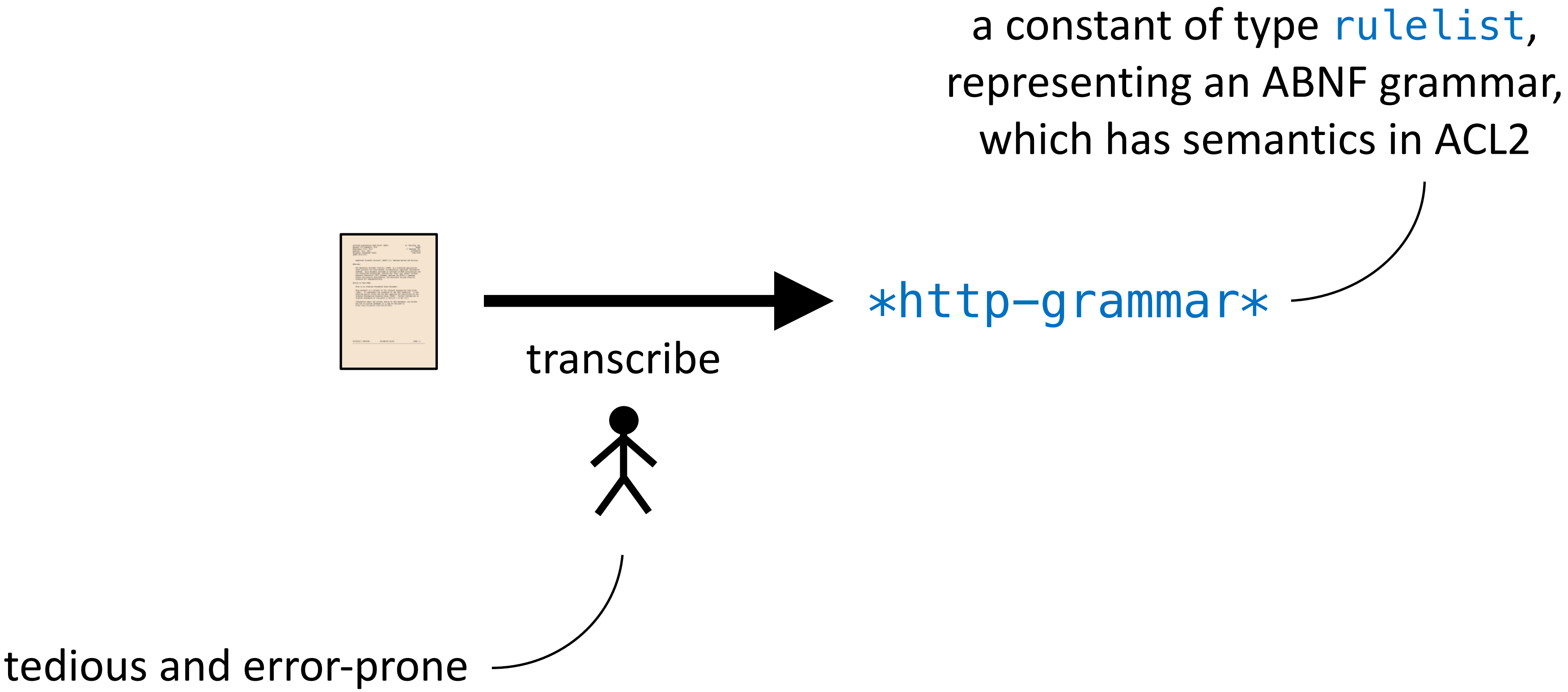


How to formally specify the HTTP syntax in ACL2, faithfully to the RFC?



How to formally specify the HTTP syntax in ACL2, faithfully to the RFC?

Transcribe the ABNF grammar rules of HTTP in abstract (ABNF) syntax, as done with the ABNF grammar rules of ABNF?



a constant of type `rulelist`, representing an ABNF grammar, which has semantics in ACL2

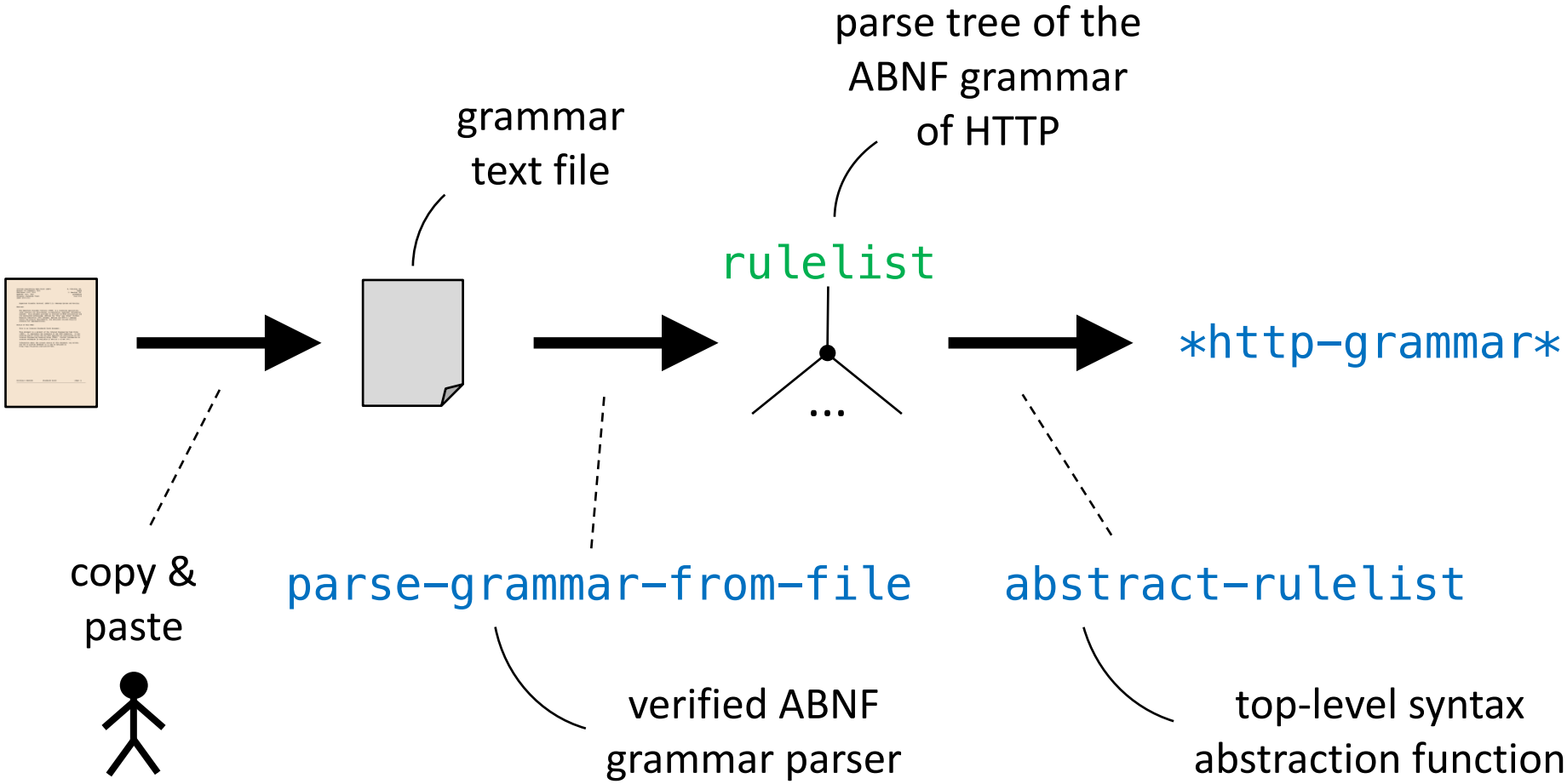
transcribe

`*http-grammar*`

tedious and error-prone

How to formally specify the HTTP syntax in ACL2, faithfully to the RFC?

Write and use a verified parser of ABNF grammars.



The verified ABNF grammar parser consists of 85 functions.

The ABNF grammar of ABNF is mostly LL(1), with three LL(2) rules, two LL(*) rules, and one (non-critically) ambiguous rule.

The parser is implemented as recursive descent with backtracking, using the *Seq* macros from the Community Books.

The parser correctness proof consists of soundness and completeness.

Soundness: if the parser succeeds, the output parse tree matches `rulelist` and its `tree->string` is the input string.

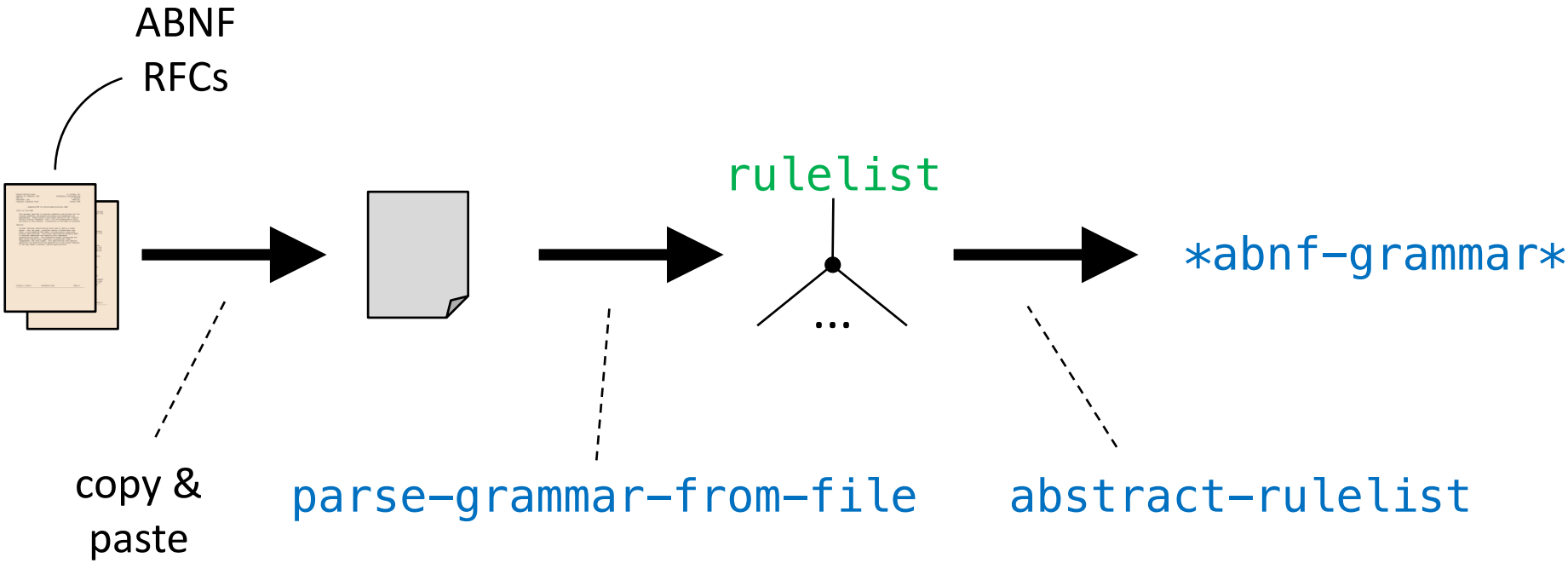
Completeness: running the parser on the `tree->string` of a parse tree that matches `rulelist` and that (necessarily) satisfies certain disambiguating restrictions, succeeds and yields that parse tree.

The correctness proof consists of 424 theorems, organized into several inter-dependent categories.

Completeness is much more laborious to prove than soundness.

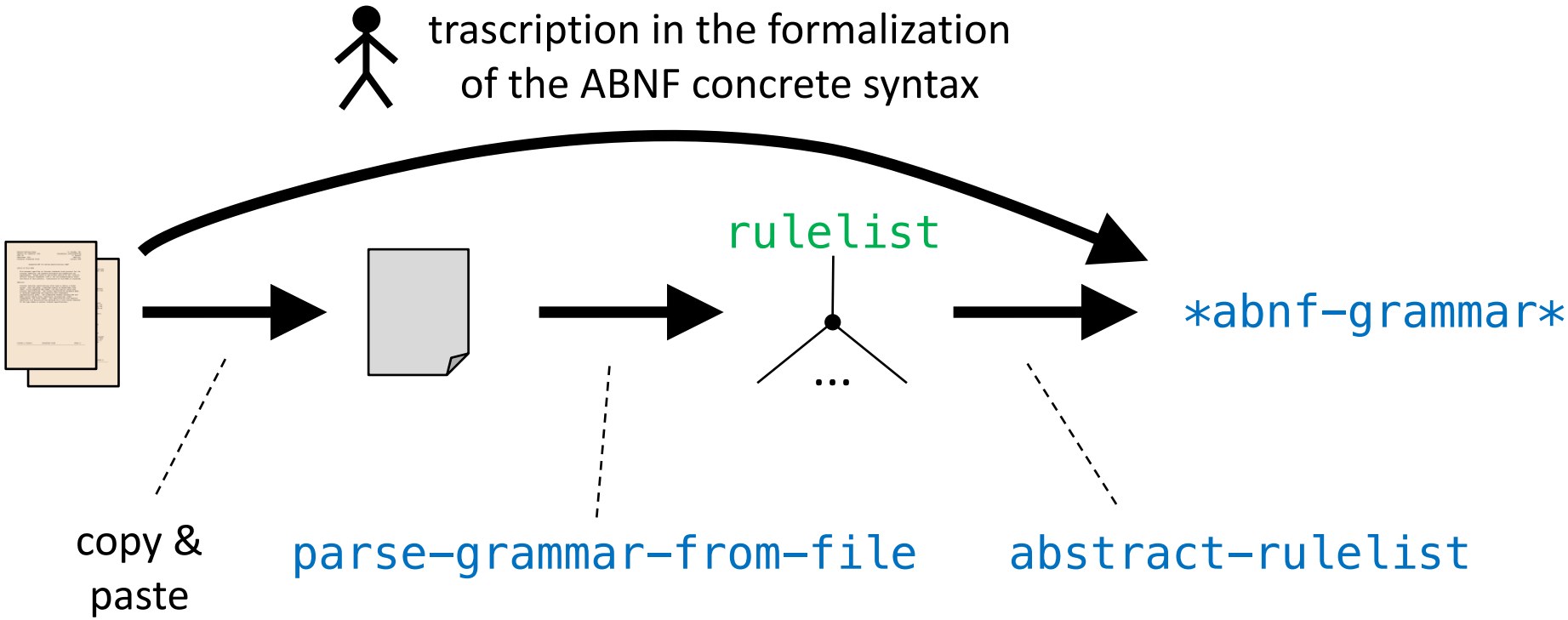
These proof techniques should be more generally applicable.

The ABNF grammar parser can be run on the ABNF grammar of ABNF.



The ABNF grammar parser can be run on the ABNF grammar of ABNF.

This provides a validation of the ABNF concrete syntax formalization.



For more information:
"ABNF in ACL2", Technical Report (http://www.kestrel.edu/~coglio)

ABNF in ACL2

Alessandro Coglio

Kestrel Institute, Palo Alto, California, USA
http://www.kestrel.edu/~coglio

Technical Report, April 2017

Abstract. Augmented Backus-Naur Form (ABNF) is a standardized formal grammar notation used in several Internet syntax specifications. This paper describes (i) a formalization of the syntax and semantics of the ABNF notation and (ii) a verified parser that turns ABNF grammar text into a formal representation usable in declarative specifications of parsers of ABNF-specified languages. This work has been developed in the ACL2 theorem prover.

1 Problem, Contribution, and Related Work

Augmented Backus-Naur Form (ABNF) is a standardized [12,18] formal grammar notation used in several Internet syntax specifications [8,13,21,16,11,10]. Since inadequate parsing may enable security exploits such as HTTP request

For much more information:
‘ABNF’ topic in the ACL2+Books manual

ABCDEFGHIJKLM NOPQRSTUVWXYZ Jump to append Search files

- Top
  - ACL2
  - Books
  - Boolean-reasoning
  - Debugging
  - Documentation
  - Hardware-verification
  - Interfacing-tools
  - Macro-libraries
  - Math
  - Projects
  - Proof-automation
  - Software-verification
  - Std

ACL2::kestrel-books

Abnf

[books]/kestrel/abnf/top.lisp

ABNF Package

A library for ABNF (Augmented Backus-Naur Form).

ABNF is a standardized formal grammar notation used in several Internet syntax specifications, e.g. URI, HTTP, IMF, SMTP, IMAP, and JSON. ABNF is specified by RFC 5234 and RFC 7405; the latter updates two portions of the former. The syntax of ABNF is specified in ABNF itself.

This ACL2 library provides:

- A formalization of the syntax and semantics of the ABNF notation.
- A verified parser that turns ABNF grammar text (e.g. from the HTTP RFC) into a formal representation suitable for formal specification (e.g. for HTTP parsing).
- Executable operations on ABNF grammars, e.g. to check their well-formedness and to compose them.

In the documentation of this library, we append dotted section and subsection numbers to ‘RFC’ to refer to the corresponding sections and subsections of the result of updating RFC 5234 as specified by RFC 7405. For example, ‘RFC.3’ refers to Section 3 of RFC 5234. As another example, ‘RFC.2.3’ refers to the result of updating Section 2.3 of RFC 5234 as specified in Section 2.1 of RFC 7405.

Subtopics

Abstract-syntax
Abstract syntax of ABNF.