PIDE backend implementation

PIDE protocol layers (1)

Bidirectional byte-channel:

- pure byte streams
- block-buffering
- high throughput
- Unix: named pipes; Windows: TCP socket; not stdin/stdout

Message chunks:

- explicit length indication
- block-oriented I/O

Text encoding and character positions:

- reconcile ASCII, ISO-Latin-1, UTF-8, UTF-16
- unify Unix / Windows line-endings
- occasional readjustment of positions

PIDE protocol layers (2)

YXML transfer syntax:

- markup trees over plain text
- simple and robust transfer syntax
- easy upgrade of text-based application

XML/ML data representation

- canonical encoding / decoding of ML-like datatypes
- combinator library for each participating language, e.g. OCaml:

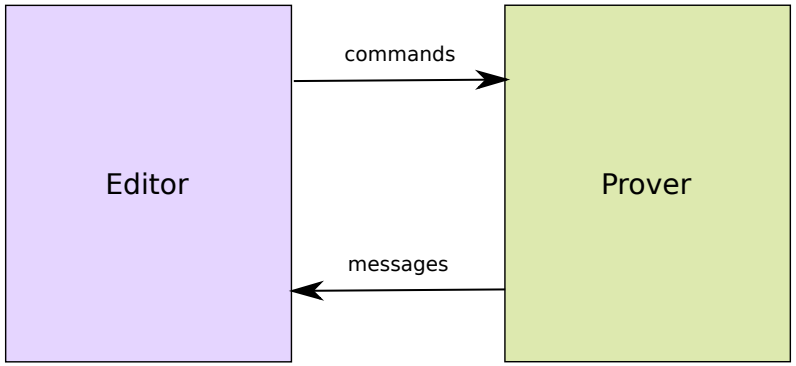
```
type 'a Encode.t = 'a -> XML.tree list
Encode.string: string Encode.t
Encode.pair: 'a Encode.t -> 'b Encode.t -> ('a * 'b) Encode.t
Encode.list: 'a Encode.t -> 'a list Encode.t
```

- untyped data representation of typed data
- typed conversion functions

Protocol functions

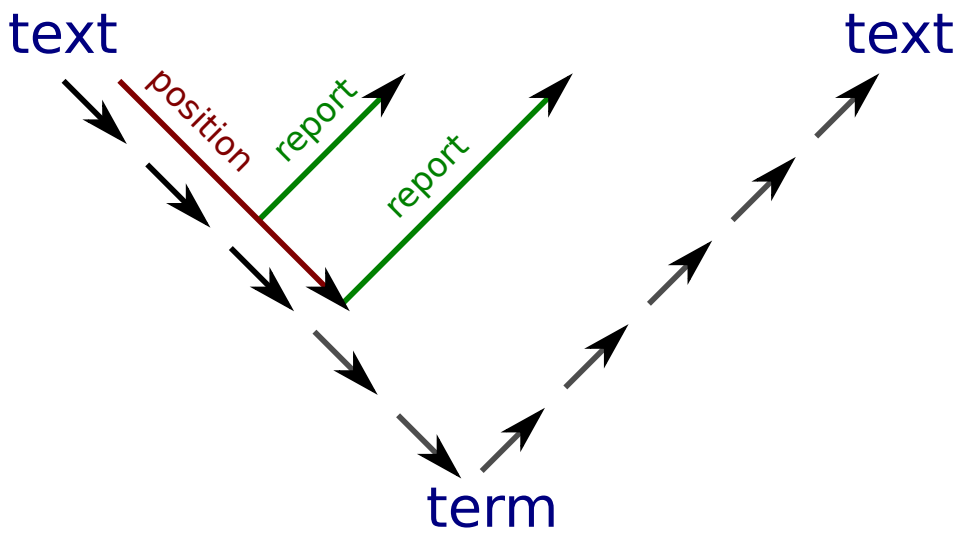
- `type protocol_command = name -> input -> unit`
- `type protocol_message = name -> output -> unit`
- outermost state of protocol handlers on each side (pure values)
- asynchronous streaming in each direction

$\longrightarrow$ editor and prover as stream-procession functions



Markup reports

Problem: round-trip through several sophisticated syntax layers
Solution: execution trace with markup reports



Document snapshots

Approximation and convergence:

1. text T, markup M, edits ΔT
2. apply edits: $T' = T + \Delta T$ (immediately in editor)
3. formal processing of T': ΔM after time Δt (eventually in prover)
4. temporary approximation (immediately in editor):
   $\tilde{M} = \mathit{revert}\ \Delta T; \mathit{retrieve}\ M; \mathit{convert}\ \Delta T$
5. convergence after time Δt (eventually in editor):
   $M' = M + \Delta M$