A Type-Based Approach to Divide-And-Conquer Recursion in Coq

Artifact for POPL'23 paper "A Type-Based Approach to Divide-And-Conquer Recursion in Coq". This README is available in both Markdown and PDF format.

Installation

This artifact may be fully built and installed either **manually** or via **Docker**.

Which installation should I use?

Docker installation is faster & easier – however, we are not entirely sure how a docker installation will integrate with your favorite IDE. Installation via Docker will build the artifact in entirety and allow you to interact with it via shell. You may need to perform a manual installation if you want to interact with proofs incrementally, i.e., with a goal buffer and all of your favorite tools.

In short, **we offer Docker installation to guarantee the successful delivery of our artifact**, but you may find you want to perform a manual installation in order to interface with the artifact more completely.

Docker Installation

First, install Docker Desktop. Then, run (in this directory):

```
docker build -t dc-image .
docker run -dt --name dc-container -v $(pwd):/home/coq/dc/ dc-image
```

The first command builds a docker image with precisely the desired Coq version and dependencies installed. The second command boots up that image as a container with a **shared drive** between this directory and `/home/coq/dc` within the container.

We can now build the development in the docker container via:

```
docker exec -it dc-container sh make-all
```

This will build all of the coq files & documentation. Documentation is placed in `./html`.

Docker Help & FAQ

Once the container is built and running, you can shell into it via

```
docker exec -ti dc-container sh
```

If commands such as `coqtop` do not work, you may need to export all `opam` environment variables to the shell session via

```
eval $(opam env)
```

once inside the container. Instructions for artifact evaluation in this file will be given under the presumption that you have an active shell session in this docker container (or, locally, if you have chosen manual installation).

If you ever need to wipe the whole thing and start over, run

```
docker container stop dc-container;
docker container rm dc-container;
docker volume rm dc;
```

Manual Installation

Manual installation instruction can be found here.

Additional Artifact Description

Documentation

Included in ./html/ is html documentation generated by coqdoc. You can view a table of contents by opening ./html/toc.html in the browser of your choice.

Documentation can be regenerated by running `make html`.

Code Snippets & Figures Index

Below we have indexed most code found in the paper by (sub)section and location in this codebase.

§3

Section	Figure	Location	Description
§3.1	Figure 1	WordsBy.hs	The `WordsBy` function in Haskell.
§3.2	Figure 2	WordsByWf.v	Using `Program` to generate a well-founded version of `wordsBy`.
§3.2	Figure 3	WordsByEq.v	Using `Equations` to generate a well-founded version of `wordsBy`.
§3.6		SmallerList.v	A custom ordering introduced directly on the `list` data type.

§4

Section	Figure	Location	Description
§4.1		List.v	The `List` data type as a signature functor & definition of `lengthAlg`.
§4.2		Kinds.v	The `KAlg` kind.
§4.2		Dc.v	Definition of our divide and conquer interface: the types `SAlg`, `FoldT`, `Alg`, and `Dc`, and functions `inDc`, `fold`, `sfold`, and `out`.

§5

Section	Figure	Location	Description
§5.1.1	Figure 7	Span.v	The `Span` (and helper `spanr`) function.
§5.1.2	Figure 8	WordsBy.v	The `WordsBy` function.
§5.1.2	Figure 9	mapThrough.hs	The `WordsBy` function.
§5.2.1	Figure 10	MapThrough.v	The `mapThrough` combinator.
§5.2.2	Figure 11	Rle.hs	`mapThrough` and run-length encoding in Haskell.
§5.2.2	Figure 12	Rle.v	Run-length encoding.
§5.2.2	Figure 13	MapThroughWf.v	Using `Program` to generate a well-founded version of `mapThrough`.
§5.3	Figures 14 & 15	Matcher.v	Harper's matcher.
§5.4	Figures 16 & 17	Mergesort.v	`mergesort` implemented using `split`.
§5.5	Figures 18 & 19	Quicksort.v	`quicksort` implemented using `partition`.

§6

Section	Figure	Location	Description
§6.1	Figure 20	Mu.v	Derivation of retractive-positive recursive types.
§6.2	Figure 21	Dc.v	Definition of `promote`.

§7

Section	Figure	Location	Description
§7		Dci.v	Derivation of Divide & Conquer induction.
§7		Rle.v	The decoding property for `rle`.
§7.1	Figure 22	SpanPfs/	Formulations of three lemmas about span.
§7.2	Figure 23	MotivePres.v	Carrier for proving that `Split` preserves motives.
§7.3	Figure 24	Forall.v	Motive used to avoid noncanoicity problems.