目次

第1章

λ力娘探索 2?

— @ark_golgo

1.1 プロローグ

「……私の 1 目勝ちじゃなイカ？」
「そのようじゃな。ヨセで細かくなったとは思っておったが、逆転には至らなかったようじゃ。」
「3 子で初勝利でゲソ！　はぁ。ふらふらでゲソ！」
「この半年で置石 2 つ分くらい強くなったようじゃの。いやはやたまげたわい。お主と打ち始めた当初は 4 子でも楽勝だったがの。」
　λカ娘は　C84 のころからある老人と囲碁を打つようになった。当初は 4 子のハンデ (段級位差にして 4 段級差に相当) をつけてもらってもまるで勝てなかったが、最近は 3 子でもいい勝負が出来るようになっていた。そして、今回 3 子で初勝利を収めた。
「やっぱり囲碁ソフトと打ちまくったのが良かったんじゃなイカ？」
「囲碁ソフト？　はて、儂が昔試してみた囲碁ソフトはあまりに弱すぎて話にならなかったが……。最近はお主の相手になるくらい強いソフトが出ているのかね? 」
「そうでゲソ。ここ数年で囲碁ソフトは急激に進歩したでゲソ。 最近は私より強くて、トッププロに 4 子で勝ったりしているでゲソ！」
「ほう、そうなのか。面白そうじゃの。儂も一つ買ってみるか。」
　さて、最近強くなったという囲碁ソフトについて、ここでλカ娘に解説してもらうことにしよう。旧来の囲碁ソフトとの比較を交えながら、強くなった理由を語ってもらおう。

作っておく)。そうすると、深さ 1 のところにノードがたくさん出来るでゲソ。そうなったら、以降のプレイアウトを行う際に、そのノードの勝率とプレイアウト数に注目するでゲソ。そして、『勝率が高く、まだプレイアウト数が少ない』ノード (候補手) を選んで、そこからまたプレイアウトをするでゲソ。これを何段も繰り返すと、モンテカルロとして有望なところは深いところまでノードが作られるでゲソ！ そして、上記の $\alpha-\beta$ 探索と同様の効果が得られるようになるでゲソ！ これがモンテカルロ木探索でゲソ。」

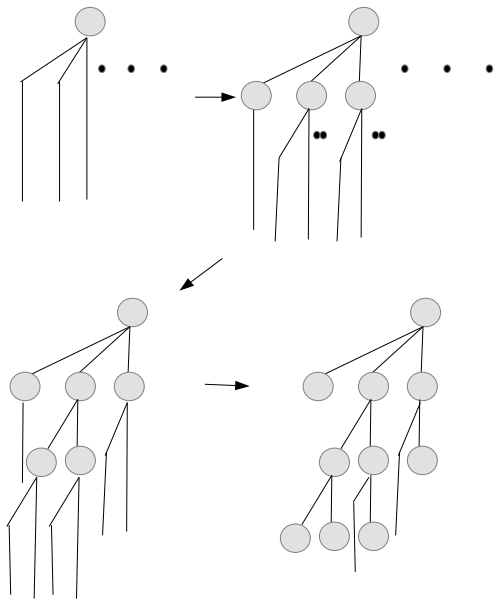

図 1.12: モンテカルロ木探索のイメージ。最初はルートノードしかないが、プレイアウトを繰り返すにしたがって、ノードが出来る。ノードが出来たら、有望そうな (勝率が高く、プレイアウト回数がまだ少ない) ノードを中心にプレイアウトすることになる。その結果、有望そうなところだけ、どんどんノードが深く作られ、木となる。

1.9 モンテカルロ木探索を用いた囲碁ソフトの現状

「モンテカルロ木探索を用いることで、囲碁ソフトは飛躍的に強くなったでゲソ！ 当初は 9 路盤でしか有効でないと思っていた人もいるようでゲソが、たくさんの工夫によって、19 路盤でもかなり強くなったでゲソ。現在最強の囲碁ソフトは『Zen』と『CrazyStone』だと言われているでゲソが、どちらもネット碁の KGS というところで 5d〜6d[*2] くらいにランクされているでゲソ。これは日本基準だとアマチュアの 7〜8 段くらいでゲソ。Zen は 4 子置いて (Zen 側が 4 段級差有利) トッププロの武宮プロに勝ったことがあるでゲソ。CrazyStone も 4 子置いてトッププロの石田プロに勝ったことがあるでゲソ！ また、Zen は 3 子置いて高校チャンピオンの大表さんに勝ったこともあるでゲソ。」

[*2] KGS では対戦成績によって段級位がつく。5d というのは 5 段ということ。ただし、日本のアマチュアの 5 段よりも評価が厳しいと言われている。なお、日本と同様に級もあり、例えば 1 級なら 1k となる。

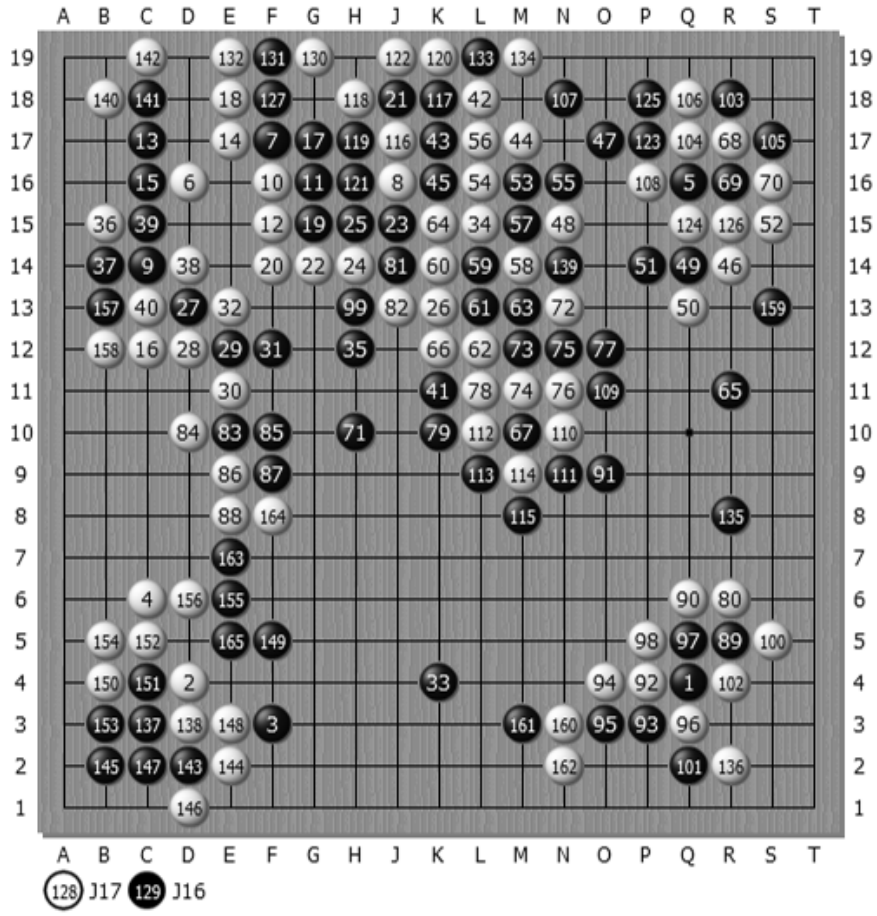

図 1.13: 囲碁ソフト、トップアマに完敗!

第2章

僕のカノジョはスナッチャー

— @master_q

2.1 川のほとり

『下におろして、垂直にまっすぐ持ち上げて敵を打ち抜くんでゲソ。ここ、両脇が大事なんでゲソ。そーして、打つべし、打つべし、打つべし、ぐるーんでゲソー!』
「なんなの、それ？」
『スナッチを制する者は、世界を制すでゲソ。最初のターゲットを決めるのが難しいんでゲソ』
「またサバゲーの話？」
『ん？ 何やってんでゲソ？』
「型推論の宿題」
『なんで大学でやんないんでゲソ？』
「かっこう悪いから」
『ワシのもやってくれなイカー』
「Dvorak 配列のままだよ」
『ぬ？ おぬしもスナッチやればいいじゃなイカ。なんでいつも ThinkPad 持ってるんでゲソ？』
「自分こそ、なんでいつもここに来るんだよ。」
『それはでゲソね、…あれ、なんででゲソ？』
「お気楽だね。イカ墨くさい」
『出してないでゲソ』
「あのさ、なんでいつもこんなこと…」
『人間の指はおもしろいでゲソー』
「ん？」
『ワシはこうしてないと溢れちゃうじゃなイカ』
「溢れちゃうって、どうなっちゃうんだ？」
『……きっと…すごいことになるでゲソ……』

すごいことなんてない。ただ当たり前のことしか起こらない。僕達の大学から見えるオンボロなビル、ソフトハウス–メタセピのオフィスができたとき、プログラマたちは大騒ぎだった。毎日決まった時刻に鳴り出すスナッチのメロディーが、僕にはなんだか不吉なサイレンのように聞こえる。それはうっすらと広がり、世界を覆っていく。

2.2 カノジョのこと

僕のカノジョはスナッチャーだ。スナッチャーというのは他の生物の一部を置換して自分の体内に取り込むエイリアンの名前だ。いつから彼等がこの世界に現われたのかはわかっていない。とにかく現在では僕たち人間とスナッチャーが同居して暮していた。

僕が大学で授業を受けていると、カノジョが突然声をかけてきた。どうも授業の宿題が解けないらしい。僕は他人と会話するのが苦手なのでおっくうだったが、簡単に鍵となるアイデアを伝えた。次の日もその次の日もカノジョは僕に声をかけてきた。そのうち、僕達は毎週いっしょにハイキングにいくようになった。毎週どこにハイキングに行くのか、実のところそのネタを考えるのも僕にはおっくうだった。紙と鉛筆がそれまで僕の友達だったから。

カノジョはスナッチャーなので、ときおり僕の一部をスナッチしたくなるようだ。どこが好かれているのかわからないが、カノジョは僕のことをお気に入りだ。もちろん僕だってスナッチされるのは御免だ。カノジョだって僕を完全にスナッチしてカノジョ自身の中に飲み込んでしまったら、僕という存在がなくなってしまう。一緒にハイキングに行く相手がいなくなる。そんなのつまらないと思っているんだろう。今のところ、カノジョは僕のことを本気でスナッチしてこようとは思わないようだ。

カノジョのお気に入りのタブレットは Nexus 7 だ。このタブレットは Android OS で動いていて、Android NDK を使えば C 言語だけでアプリケーションを書くこともできる。カノジョはスピード狂だったから、Android のアプリはいつも C 言語で書いていた。でも僕は C 言語よりも Haskell の方が気に入っている。型の強い言語というのは設計時の安心感が段違いだからだ。せっかくなので、Haskell で面白いアプリをカノジョのために書いてあげたいと最近は思うようになった。

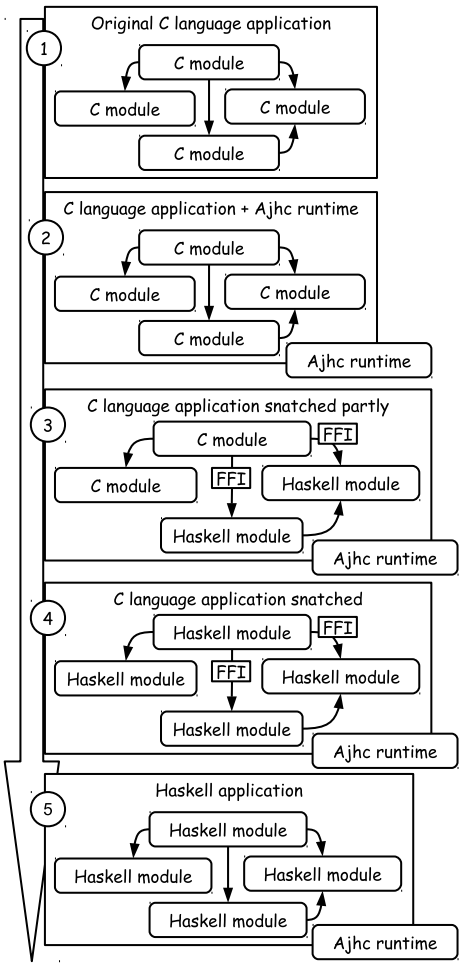

図 2.1: Snatch-driven development の概要

2.3 Snatch-driven development

ところが Android NDK を使っても Haskell で Android アプリケーションを書くのは簡単ではない。たしかに Android NDK を使えば POSIX API を使った C 言語プログラミングができる。そのため GHC を使ったクロス開発が可能だ。しかし GHC だとバイナリサイズが巨大になる傾向がある。簡単なロジックでも数十 MB ものコードサイズになってしまうこともある。また、Android はアクティビティという少しかわったしくみでプログラムの状態管理をしている。このようなドメイン固有の作法を守りつつゼロからアプリケーションを設計することは簡単ではないのだ。もし C 言語のまっとうな実装があれば、そのコードの設計を推測して Haskell で書き直すこともできる。しかしプログラムの規模が大きくなるにつれてこの作業は難しくなる。というのも”C 言語のコードから設計を推測する” ことが完全に完了しないと” 設計を Haskell で書き下す” ことができないからだ。当然この作業が完全に完了しないと動作確認はできない。小さなユニットテストはできるが、そのユニットテストが設計から正しく意図されたものであるかも自信が持てない。この流れを少しずつ進行させる方法はないだ

ろうか？

ところで、ソフトウェアには “Snatch-driven development” という開発手法がある。Ajhc という Haskell コンパイラを組み合わせて使えば、C 言語の設計を動作可能な状態をたもったまま Haskell で書き直すことが可能だ。また Ajhc には大変小さなコードサイズのバイナリを吐くという特性もある。以前からこの手法に興味があった。カノジョへプレゼントする Android アプリを作るのに、C 言語で書かれた簡単なアプリケーションを Haskell でスナッチした後、その Haskell の設計を成長させるような方法を取ることはできないだろうか？ Android アプリを C 言語から Haskell で書き直すことは、スナッチ設計を身に付ける上で良いトレーニングになるだろう。

スナッチのやり方を簡単に解説しておこう (図 2.1)。スナッチは多くの場合以下のステップを取る。下のステップの番号はそのまま図 2.1 の番号にそのまま対応している。

Step 1 まず既存の動作可能な C 言語のソースコードを用意する。この C 言語ソースコードは実行可能、もしくはテスト可能であればなんでも良い。

Step 2 その C 言語ソースコードにとりあえず Ajhc コンパイラのランタイム部分だけを埋め込む。この状態でまず実行/テストしてみて問題ないことを確認する。

Step 3 その後、プログラム内のモジュールの内 Haskell 化できそうな箇所を選んで設計置換をする。この時、Haskell と C 言語の間は FFI を使って接続する。つまり Haskell の型を使わず C 言語の ABI でモジュール間を接続する。

Step 4 この設計置換をくりかえしてプログラム全体を Haskell 化する。

Step 5 全体を Haskell 化しおわった状態で動くことを確認できたら、最後に FFI を取り去って Haskell module 同士を型によるインターフェイスで接続する。

これで元の C 言語の設計がそのままの形をたもったまま Haskell 化できたことになる。

2.4 Android NDK とそのサンプルアプリケーション

さっそく Android アプリスナッチの準備をはじめよう。何を用意すればいいだろうか？

- Android NDK 開発環境とその概要理解
- スナッチする Android NDK サンプルソースコード

こんなところだろうか。順番に手をつけていこう。

2.4.1 Android NDK 開発環境

Android NDK [*1] というのは先にも言った通り Android のアプリケーションを C 言語で書くための開発環境だ。詳細ははぶくが Android は通常の UNIX ライク OS と異なり、プロセスではなくプロセスの上に構築されたアクティビティというしくみでアプリケーションを管理している。通常はこのアクティビティは Java 言語で設計することになっている。特殊なライフサイクルがあるので生の C 言語では扱いにくいのだ。ところが NativeActivity というしくみを使う

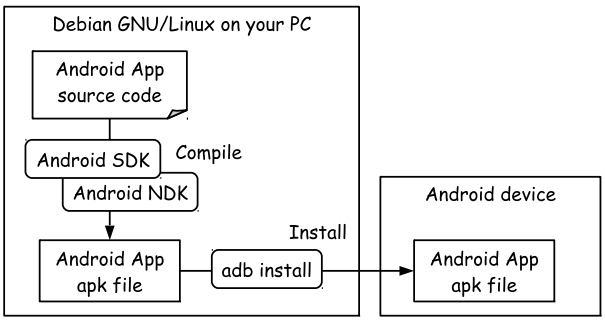

図 2.2: Android NDK アプリのクロス

[*1] http://developer.android.com/tools/sdk/ndk/index.html

と、このアクティビティを C 言語のみで設計することが可能になる。しかし Android フレームワークから渡されるイベントにすばやく応答するために、Android NDK プログラミングでは POSIX スレッドを使う必要がある。もちろん誰もが知っている通り、POSIX スレッドを使ったプログラミングは大変難しい。デッドロックを防止するために非常に注意深い設計が求められることなどがその理由だ。

そこで、Android NDK ではこの POSIX スレッドを使ったアクティビティ制御のラッパーとして android_native_app_glue を提供している。android_native_app_glue はアクティビティのライフサイクルを C 言語でも書きやすくしてくれるミドルウェアのようなものだ。android_native_app_glue を使えば設計者は POSIX スレッドを意識することなく NativeActivity アプリケーションを設計できる。

Android NDK 開発環境の設定に入る前に、開発環境のおおざっぱなイメージをつかんでおこう (図 2.2)。通常のアプリケーションとは違い、Android 上のアプリケーションは Android OS 上ではなく別の PC 上でコンパイルする。ぼくは Debian GNU/Linux を使っているので、Intel アーキティクチャ上の Debian に Android SDK と Android NDK をインストールして、ARM アーキティクチャの Android OS 向けのアプリケーションパッケージである apk ファイルをビルドできる環境を整える必要がある。apk ファイルができあがったら Android SDK に付属している adb というコマンドを使って apk ファイルを Android デバイスにインストールする。これで Android デバイスのランチャー画面から自作の Android アプリケーションのアイコンが見えるようになるはずだ。

僕はさっそく Android NDK 開発環境をととのえるために愛用の ThinkPad を開いた。[*2] まず必要なファイルをそろえよう。以下のファイルをダウンロードする。

- Android SDK (adt-bundle-linux-x86_64-20130917.zip) [*3]
- Android NDK (android-ndk-r9-linux-x86_64.tar.bz2) [*4]

そして Linux 用の JVM をインストールした後、Android SDK/NDK のインストールを行なう。

```
$ cd $HOME
$ unzip -x adt-bundle-linux-x86_64-20130917.zip
$ mv adt-bundle-linux-x86_64-20130917/sdk $HOME/android-sdk
$ rm -f adt-bundle-linux-x86_64-20130917
$ export PATH=$PATH:$HOME/android-sdk/tools:$HOME/android-sdk/platform-tools
$ tar xf android-ndk-r9-linux-x86_64.tar.bz2
$ mv android-ndk-r9 android-ndk
$ export PATH=$PATH:$HOME/android-ndk
$ sudo apt-get install openjdk-7-jdk ant lib32z1-dev lib32stdc++6
$ android update sdk --no-ui --force
```

Android SDK/NDK のセットアップが終わったので、なにかサンプルアプリケーションをビルドしてみよう。さっき展開した Android NDK の中に native-activity という NativeActivity を使ったサンプルアプリケーションが含まれている。どうやら非常に規模の小さいアプリケーションのようだ。今回はこの native-activity をスナッチしよう。さっそくスナッチの前にこのサンプルをビルドして apk ファイルを作ってみよう。

[*2] Debian GNU/Linux amd64 sid 2013/11/01 時点。GHC のバージョンは 7.6.3。Ajhc のバージョンは 0.8.0.9。

[*3] http://developer.android.com/sdk/index.html

[*4] http://developer.android.com/tools/sdk/ndk/index.html