2013 KOMABASAI EDITION

Theoretical Science Group

理論科学グループ

部報 305 号
— 駒場祭パンフレット号 —

目　次

展示企画

企画紹介 Bayesian Fighter

@hakatashi

あらすじ

一年の博多市と申します。駒場祭ということでそれっぽい展示企画を作ってきました。

スマホを傾けてベーゴマを操作し、相手のベーゴマを盤上から落とすゲームです。ブースに大きめの画面を用意し、来場者のお手持ちのスマホからあるウェブサイトにアクセスしてもらうと、スマホが傾き検知デバイスとして使用でき、ブースに設置された画面を見ながら遊べる、という仕組みになっています。

名前はいま考えました。元ネタはもちろん Bayesian Filter です。

ソースはたぶん github で公開します。 https://github.com/hakatashi?tab=repositories から探してください。

材料

- Node.js
- Socket.IO
- paper.js

からくり

多くのスマホブラウザで用意されている JS プロパティaccelerationIncludingGravity から重力加速度の値を取得し、用意した Node.js サーバーに Socket.IO による Socket 通信でリアルタイムに送信します。受け取ったデータを元にサーバーでベイの操作を行い、画面にその様子を出力します。(ちなみにモニタ側への情報も Socket 通信で送信しています)

感想

かねがね噂は聞いていましたが、Socket.IO の手軽さは本当に素晴らしいですね。Node.js も Socket.IO も初めて触ったのですが、かなりサクサクと実装することができました。

ちなみに一番苦労したのは AI の実装です。簡単そうに見えてこれが意外と手強い。

漁

phidnight

ゲームのタイトル

漁

制作

phidnight

ゲームの目的

あなたは漁師です。あなたの目的は、制限時間以内にできるだけ多く、そして美しく魚をとることです。

魚の取り方

まず、2 箇所に杭を立てます。杭は、杭を立てたい場所に立ち、ボタンを押すことで立てられます。
杭を 2 つ立てたあとは、もう一度杭を立てるボタンを押すことでプレイヤーと杭 2 つを結ぶ三角形の形に網を張ります。網を張ったときに網の内部にいた魚を捕まれられます。また、一度に複数の魚を捕まえると高得点が得られます。
網を張った後は杭が消滅するので新たに杭を立てに行きます。これを繰り返してできるだけ多く得点を獲得してください!

GomiHiroi (Garbage Collection の可視化)

zeptometer

やりたかったこと

scheme のごく単純なサブセットの処理系において GC が動作する様子を processing で可視化します。

楽しみ方

scheme のコードを入力します。
(◠‿◠)
ごみが回収される様を眺めます。
(’ ω’)
楽しい！！！
✌('ω'✌)三✌('ω')✌三(✌'ω')✌

背景

つくった理由

学科の課題で「c 言語で scheme 処理系を作れ」という課題があったためのんびり作っていたのですが折角だからこれを流用して面白いものが作れないかと思いこんなものを作った次第です。学科の同期の wasabiz 君は R7RS をフルサポートせんとする勢いで実装していますね。

ちなみにこの記事を書いている段階ではまだ GC すらまともに動いていないという状況です。ちなみに時前の GC を実装する以前は Boehm GC を使用していました。そういう進捗であるため駒場祭当日ややもするとこのプログラムを書いている最中かもしれませんがどうぞ生暖かい目で見守ってやってください。

Garbage Collection とは

Garbage Collection(直訳:ごみ集め) はプログラムが動的に確保したメモリ領域のうち、不要になったものを自動的に判断して解放する機構のことです。C 言語で言うと malloc で確保したメモリ領域を放置していても自動的に free してくれる機構です。便利ですね。

これだけの説明で GC が解説できているわけがないですが、よくある GC についての蘊蓄を並べたところで新規性が米粒ほどもないので割愛します。ご容赦ください。詳しいことを知りたい方は google 先生にお尋ねになることをお勧めします。

猶、GomiHiroi では GC のアルゴリズムとして mark and sweep を採用しています。理由はその方が可視化が面白そうだからというだけです。

scheme のサブセットについて

scheme はもともと言語仕様がかなりシンプルな言語ですが、主に実装と可視化を楽に済ませるために GC が活躍できる必要最小限の仕様にします。

- 型：empty-list, pair, symbol, number, bool, function
- 構文：quote, if, lambda, define, begin, set!
- ビルトイン関数：cons, car, cdr, 算術関数, 述語もろもろ
- クロージャをサポートする
- 継続とかマクロその他はサポートしない

ここまでくるとただの lisp 方言であると称した方がよいかもしれないですけどね…ここで確認しておくべきことは何を GC の対象にしておくべきか、ということです。この scheme のサブセットにおいて GC の対象となるのは

- 環境: 変数と値のペアを複数参照
- コンスセル (pair): 何らかの値を二つ参照
- クロージャ(function): 環境を参照

以上の三つです。よってこれらを観察すれば GC が可能です。これらのオブジェクトが形成するネットワーク構造を探索してマーキングをし、到達できなかったメモリ領域を解放するのが GomiHiroi における GC のお仕事となります。なおこの処理系の実装は c で行います。

可視化について

可視化には processing を使います。;;; zeptometer の文章はここで途切れている。

一般記事

ipc_bot の仕組み

原 将己 (qnighy)

概要

最近，@ipc_bot という Twitter Bot を作りました．命題を投げると証明可能かどうかを [**?**] の方法で判定し，証明可能なら証明図を出してくれます．かわいいです．

この bot の仕組みを紹介します．ちなみに全部直観主義論理の話です．

命題論理の論理式

命題論理の論理式を定義します．終端記号と非終端記号をあわせて 6 種類の文結合子があります．

原子論理式 $A, B, C, \ldots$ 具体的な内容は定まっていない論理式の最小構成要素，つまりまさしく**原子**的な命題です．対象レベルの変数だと思って下さい．名前が同じものは同じ論理式をあらわすというのが唯一の特徴です．名前は加算無限個以上あることとします．

含意 $A \to B$ 含意です．「A ならば B」を表します．古典的には「A でないか，B である」と同義ですが，直観主義論理だと区別があります．

選言 $A \lor B$ 選言です．「A または B」を表します．

連言 $A \land B$ 連言です．「A かつ B」を表します．

ボトム $\bot$ 矛盾です．矛盾は最強なので，矛盾を使う以外の方法で，矛盾を導くことはできません．逆に，矛盾を持っていれば何でもありの世界になります．選言の単位元です．

トップ $\top$ 常に真です．恒真は最弱なので，何をやっても駄目です．そのかわり，いつでも成立します．連言の単位元です．

実は De Morgan の法則のおかげで，連言とトップはなくても大して表現力は変わらないのですが，ここではこれらも存在すると思うことにします．

次の二つは 6 種類の文結合子の組み合わせで表現します．

否定　$\neg A$ 否定です．A ではないということです．古典論理だと含意よりも否定のほうが主役ですが，直観主義論理では話が違います．$\neg A$ は $A \to \bot$ の略記です．

同値　$A \Leftrightarrow B$ 同値です．A ならば B だし，B ならば A です．つまり，$(A \to B) \wedge (B \to A)$ の略記です．

自然演繹

命題が正しいかどうか (証明可能かどうか) を，推論規則によって定義します．上から下が導出できるというルールです．

$$\frac{\begin{array}{c}[A]\\ \vdots \\ B\end{array}}{A \to B}\ (\text{NJ-}{\to}\text{I}) \qquad \frac{A \to B \qquad A}{B}\ (\text{NJ-}{\to}\text{E})$$

$A \to B$ を証明するには，A を仮定して B を証明すればいいよというのが左側のルールです．$A \to B$ と A が両方証明できたら，B も正しいよ，というのが右側のルールです．

$$\frac{A \qquad B}{A \wedge B}\ (\text{NJ-}{\wedge}\text{I}) \qquad \frac{A \wedge B}{A}\ (\text{NJ-}{\wedge}\text{E1}) \qquad \frac{A \wedge B}{B}\ (\text{NJ-}{\wedge}\text{E2})$$

A と B が両方証明できることと，$A \wedge B$ が証明できることは同じだよねというルールです．

$$\frac{A}{A \vee B}\ (\text{NJ-}{\vee}\text{I1}) \qquad \frac{A}{A \vee B}\ (\text{NJ-}{\vee}\text{I2})$$

$$\frac{A \vee B \qquad \begin{array}{c}[A]\\ \vdots \\ C\end{array} \qquad \begin{array}{c}[B]\\ \vdots \\ C\end{array}}{C}\ (\text{NJ-}{\wedge}\text{E})$$

A と B のどちらかが証明できれば $A \vee B$ としてよいです．左側のルールはちょっと難しいことを言っています．$A \to C$ と $B \to C$ を両方証明できたら，$(A \vee B) \to C$ としてよいということです．

$$\frac{}{\top}\ (\text{NJ-}{\top}\text{I}) \qquad \frac{\bot}{A}\ (\text{NJ-}{\bot}\text{E})$$

トップはいつでも証明できます．矛盾 (ボトム) だったら何でもありです．

I という名前のついている規則は導入則といい，その命題を証明する方法を表します．E という名前のついている規則は除去則といい，その命題の証明があったときに何が導出できるかという使い方を表します．

以上で規則が定義できました．これを直観主義論理の自然演繹 (NJ) といいます．

規則を繋げて実際に証明するときは，次のように下から上に生える木のようになります．

$$\dfrac{\dfrac{[A\wedge(B\vee C)]}{B\vee C}\wedge_{E2} \qquad \dfrac{\dfrac{\dfrac{[A\wedge(B\vee C)]}{A}\wedge_{E1} \quad [B]}{A\wedge B}\wedge_I}{(A\wedge B)\vee C}\vee_{I1} \qquad \dfrac{[C]}{(A\wedge B)\vee C}\vee_{I2}}{\dfrac{(A\wedge B)\vee C}{A\wedge(B\vee C)\to(A\wedge B)\vee C}\to_I}\vee_E$$

仮定の明示

さきの証明図では，どの段階でどの仮定を使っていいのかがわかりづらいので，それを各段階で明示する書き方を用いることにします．

Γ と書いたら 0 個以上の証明の列だと思うことにします．$\Gamma\vdash G$ はそこから上の証明図中で Γ 中の仮定のみを用いることにより G が証明できるということを意味します．仮定の順番は自由に入れ替えてよいことにします．

$$\dfrac{}{\Gamma, A\vdash A}\quad\text{(NJ-Ax)}$$

これは仮定をもって証明とするという規則です．先ほどの図における $[A]$ に相当します．残りは全て先ほどの規則と対応します．

$$\dfrac{\Gamma, A\vdash B}{\Gamma\vdash A\to B}\quad\text{(NJ-}\to\text{I)} \qquad \dfrac{\Gamma\vdash A\to B \quad \Gamma\vdash A}{\Gamma\vdash B}\quad\text{(NJ-}\to\text{E)}$$

$$\dfrac{\Gamma\vdash A \quad \Gamma\vdash B}{\Gamma\vdash A\wedge B}\text{ (NJ-}\wedge\text{I)} \qquad \dfrac{\Gamma\vdash A\wedge B}{\Gamma\vdash A}\text{ (NJ-}\wedge\text{E1)} \qquad \dfrac{\Gamma\vdash A\wedge B}{\Gamma\vdash B}\text{ (NJ-}\wedge\text{E2)}$$

$$\dfrac{\Gamma\vdash A}{\Gamma\vdash A\vee B}\text{ (NJ-}\vee\text{I1)} \qquad \dfrac{\Gamma\vdash A}{\Gamma\vdash A\vee B}\text{ (NJ-}\vee\text{I2)} \qquad \dfrac{\Gamma\vdash A\vee B \quad \Gamma, A\vdash C \quad \Gamma, B\vdash C}{\Gamma\vdash C}\quad\text{(NJ-}\wedge\text{E)}$$

$$\frac{}{\Gamma \vdash \top}\ (\text{NJ-}\top\text{I}) \qquad \frac{\Gamma \vdash \bot}{\Gamma \vdash A}\ (\text{NJ-}\bot\text{E})$$

証明図は次のようになります．(長いので 2 つに分割しています)

$$\dfrac{\dfrac{\dfrac{\dfrac{}{A \wedge (B \vee C), B \vdash A \wedge (B \vee C)}\,Ax}{A \wedge (B \vee C), B \vdash A}\,\wedge_{E1} \quad \dfrac{}{A \wedge (B \vee C), B \vdash B}\,Ax}{A \wedge (B \vee C), B \vdash A \wedge B}\,\wedge_{I}}{A \wedge (B \vee C), B \vdash (A \wedge B) \vee C}\,\vee_{I1}$$

$$\dfrac{\dfrac{\dfrac{\dfrac{}{A \wedge (B \vee C) \vdash A \wedge (B \vee C)}\,Ax}{A \wedge (B \vee C) \vdash B \vee C}\,\wedge_{E2} \quad A \wedge (B \vee C), B \vdash (A \wedge B) \vee C \quad \dfrac{A \quad \dfrac{}{A \wedge (B \vee C), C \vdash C}\,Ax}{A \wedge (B \vee C), C \vdash (A \wedge B) \vee C}\,\vee_{I2}}{A \wedge (B \vee C) \vdash (A \wedge B) \vee C}\,\vee_{E}}{\vdash A \wedge (B \vee C) \to (A \wedge B) \vee C}\,\to_{I}$$

シークエント計算

命題が正しいかどうか，すなわちこのような証明図が存在するかどうかを判定する問題を考えます.「証明図は存在しない」ということをどう確認すればよいでしょうか.

可能な全ての証明図を調べることができればよいのですが，自然演繹の規則を使うといくらでも大きい証明図が作れてしまいます．理由は名前に E がついている規則にあります．これらの「除去則」の代わりに「左導入則」を使うことにします.

$$\frac{}{\Gamma, A \vdash A}\ (\text{LJ-Ax})$$

$$\frac{\Gamma, A \to B \vdash A \qquad \Gamma, B \vdash G}{\Gamma, A \to B \vdash G}\ (\text{LJ-}\to\text{L}) \qquad \frac{\Gamma, A \vdash B}{\Gamma \vdash A \to B}\ (\text{LJ-}\to\text{R})$$

$$\frac{\Gamma, A, B \vdash G}{\Gamma, A \wedge B \vdash G}\ (\text{LJ-}\wedge\text{L}) \qquad \frac{\Gamma \vdash A \qquad \Gamma \vdash A}{\Gamma \vdash A \wedge B}\ (\text{LJ-}\wedge\text{R})$$

$$\frac{\Gamma, A \vdash G \qquad \Gamma, B \vdash G}{\Gamma, A \vee B \vdash G}\ (\text{LJ-}\vee\text{L}) \qquad \frac{\Gamma \vdash A}{\Gamma \vdash A \vee B}\ (\text{LJ-}\vee\text{R1}) \qquad \frac{\Gamma \vdash B}{\Gamma \vdash A \vee B}\ (\text{LJ-}\vee\text{R2})$$

$$\frac{}{\Gamma, \bot \vdash G}\ (\text{LJ-}\bot\text{L}) \qquad \frac{}{\Gamma \vdash \top}\ (\text{LJ-}\top\text{R})$$

この規則にしたがった証明は例えば次のようになります.

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{A, B \vdash A}\,Ax \quad \dfrac{}{A, B \vdash B}\,Ax}{A, B \vdash A \wedge B}\wedge_R}{A, B \vdash (A \wedge B) \vee C}\vee_{R1} \quad \dfrac{\dfrac{}{A, C \vdash C}\,Ax}{A, C \vdash (A \wedge B) \vee C}\vee_{R2}}{A, B \vee C \vdash (A \wedge B) \vee C}\vee_L}{A \wedge (B \vee C) \vdash (A \wedge B) \vee C}\wedge_L}{\vdash A \wedge (B \vee C) \to (A \wedge B) \vee C}\to_R$$

以上のルールは cut-free LJ と呼ばれていて，次のルール

$$\frac{\Gamma \vdash A \qquad \Delta, A \vdash G}{\Gamma, \Delta \vdash G}\ (\text{LJ-Cut})$$

を加えたものと証明能力が同じであることが知られています．証明は帰納法を適当に回すとできますが面倒なので省略します．さらに NJ とも証明能力が同じであることがすぐに言えます．こっちの証明のほうが簡単ですがやっぱり面倒なので省略します.

さらに，論理に詳しい人なら気づいているかもしれませんが，ここでは弱化規則と縮約規則

$$\frac{\Gamma \vdash G}{\Gamma, A \vdash G}\ (\text{LJ-Weak}) \qquad \frac{\Gamma, A, A \vdash G}{\Gamma, A \vdash G}\ (\text{LJ-Contr})$$

が存在しません．実は，カット規則と同じく，これらのルールを加えても証明能力が同じです．これにはトリックがあって，先ほどの連言や選言に関するルールを少しいじると，この性質は成立しなくなります．特に注目するべきは，次のルールです.

$$\frac{\Gamma, A \to B \vdash A \qquad \Gamma, B \vdash G}{\Gamma, A \to B \vdash G}\ (\text{LJ-}\to\text{L})$$

これはよく見ると導入則になっていないですね．しかも，$G = A$ とすると無限ループが作れることがわかります．したがってこれは駄目な規則です.

このルールのせいでうまく探索できないので，ルールをさらに変更します.

停止するシークエント計算

先ほどの $A \to B$ の左導入則を，A の形に応じて 5 種類 (普通は $\top$ を入れないので 4 種類です) にわけます．p は原子命題とします.

$$\frac{\Gamma, p, B \vdash G}{\Gamma, p, p \to B \vdash G}\ (\text{LJ-}\to P\text{L}) \qquad \frac{\Gamma, A_1 \to B, A_2 \to B \vdash G}{\Gamma, (A_1 \vee A_2) \to B \vdash G}\ (\text{LJ-}\to \vee\text{L})$$

$$\frac{\Gamma, A_1 \to (A_2 \to B) \vdash G}{\Gamma, (A_1 \wedge A_2) \to B \vdash G}\ (\text{LJ-}\to \wedge\text{L})$$

$$\frac{\Gamma, B \vdash G}{\Gamma, \top \to B \vdash G}\ (\text{LJ-}\to \top\text{L}) \qquad \frac{\Gamma, A_1, A_2 \to B \vdash A_2 \qquad \Gamma, B \vdash G}{\Gamma, (A_1 \to A_2) \to B \vdash G}\ (\text{LJ-}\to\to\text{L})$$

$\to_L$ 以外の規則は先ほどと同じとします．このとき，命題に適当な自然数の重み関数を与えると，常に下の重みのほうが上の重みより大きくなるようにできます．したがって，下から上に向かって，適用出来る推論規則をかたっぱしから試していけば証明可能か判定できるようになりました．

高速化

先ほどのルールでは，重みを指数的にとらないといけなかったので，証明探索の上界評価があまり嬉しくないです．そこで，次のような工夫をします．

まず，$A \vee B \to C$ の形の仮定を $A \to C$ と $B \to C$ に分解するとき，他に登場しない原子命題 q をはさんで $A \to q$, $B \to q$, $q \to C$ とします．すると，C が二回出てこないので嬉しいです．

それから，$(A \to B) \to A$ の形の帰結をもつシークエントを特別扱いします．この文章ではこれを (B, A) と書くことにします．すると，次のような推論規則を作ることができます．p は原子命題，q は他に登場しない原子命題，G は普通の論理式または (B, A) 形の論理式とします．

$$\frac{}{\Gamma, A \vdash A}\ (\text{LJ-Ax1}) \qquad \frac{}{\Gamma, A \vdash (W, A)}\ (\text{LJ-Ax2})$$

$$\frac{\Gamma, A \vdash B}{\Gamma \vdash A \to B}\ (\text{LJ-}\to\text{R1}) \qquad \frac{\Gamma, A \vdash (W, B)}{\Gamma \vdash (W, A \to B)}\ (\text{LJ-}\to\text{R2})$$

$$\frac{\Gamma, A, B \vdash G}{\Gamma, A \wedge B \vdash G}\ (\text{LJ-}\wedge\text{L}) \qquad \frac{\Gamma \vdash A \qquad \Gamma \vdash A}{\Gamma \vdash A \wedge B}\ (\text{LJ-}\wedge\text{R1}) \qquad \frac{\Gamma \vdash (W, A) \qquad \Gamma \vdash (W, A)}{\Gamma \vdash (W, A \wedge B)}\ (\text{LJ-}\wedge\text{R2})$$

$$\frac{\Gamma, A \vdash G \qquad \Gamma, B \vdash G}{\Gamma, A \vee B \vdash G}\ (\text{LJ-}\vee\text{L}) \qquad \frac{\Gamma \vdash A}{\Gamma \vdash A \vee B}\ (\text{LJ-}\vee\text{R1}) \qquad \frac{\Gamma \vdash B}{\Gamma \vdash A \vee B}\ (\text{LJ-}\vee\text{R2})$$

$$\frac{\Gamma, B \to q, q \to W \vdash (q, A)}{\Gamma \vdash (W, A \vee B)}\ (\text{LJ-}\vee\text{R3}) \qquad \frac{\Gamma, A \to q, q \to W \vdash (q, B)}{\Gamma \vdash (W, A \vee B)}\ (\text{LJ-}\vee\text{R4})$$

$$\frac{}{\Gamma, \bot \vdash G}\ (\text{LJ-}\bot\text{L}) \qquad \frac{}{\Gamma \vdash \top}\ (\text{LJ-}\top\text{R1}) \qquad \frac{}{\Gamma \vdash (W, \top)}\ (\text{LJ-}\top\text{R2})$$

$$\frac{\Gamma, p, B \vdash G}{\Gamma, p, p \to B \vdash G}\ (\text{LJ-}\to P\text{L}) \qquad \frac{\Gamma, A_1 \to q, A_2 \to q, q \to B \vdash G}{\Gamma, (A_1 \vee A_2) \to B \vdash G}\ (\text{LJ-}\to \vee\text{L})$$

$$\frac{\Gamma, A_1 \to (A_2 \to B) \vdash G}{\Gamma, (A_1 \wedge A_2) \to B \vdash G}\ (\text{LJ-}\to \wedge\text{L}) \qquad \frac{\Gamma, A_1, \vdash (B, A_2) \qquad \Gamma, B \vdash C}{\Gamma, (A_1 \to A_2) \to B \vdash C}\ (\text{LJ-}\to\to\text{L1})$$

$$\frac{\Gamma, B \vdash G}{\Gamma, \top \to B \vdash G}\ (\text{LJ-}\to \top\text{L}) \qquad \frac{\Gamma, C \to W, A_1, \vdash (B, A_2) \qquad \Gamma, B \vdash (W, C)}{\Gamma, (A_1 \to A_2) \to B \vdash (W, C)}\ (\text{LJ-}\to\to\text{L2})$$

$\to$ の重みを 1, $\wedge$ の重みを 2, $\vee$ の重みを 3 として，命題やシークエントの重みは，その中に出現する論理記号の重みの総和とします．すると，全てのルールで重みが単調減少になるので，シークエントの重み n に対して証明図の深さは $O(n)$ であることがわかります．上手に実装すると，$O(n \log n)$ 空間で動くことがわかります．

ソルバの実装

$\to\to_L$ と $\vee_R$ を除く全てのルールは可逆的です．すなわち，そのルールを適用したらバックトラックする必要はありません．そこで，可逆的なルールを優先的に適用していくことにします．

あとは何か，ソースコードを見て下さい．

証明図の生成

この変なシークエント計算の証明図をそのままユーザーに見せるのはあまり嬉しくないので，証明図の生成をします．

最初に紹介した NJ の証明図を出力することにします．ここで，命題論理の NJ の証明図が単純型付きラムダ計算のラムダ項に対応することを使うと，次のような問題を解けばよいことがわかります．

$\Gamma \vdash A$ のシークエント計算による証明図が与えられた時，Γ を変数の型環境として，型 A を持つラムダ項をつくる．例えば，$A, A \to B \vdash B$ というシークエントがあったら，$x : A, y : A \to B \vdash yx : B$ という型判定は正しいので，項 yx を返す．

この問題は証明図の構造に関する帰納法で解けます．特に注意すべき点は 1 点だけで，先ほどの推論規則で新規に導入された原子命題 q は，ラムダ項を生成するときには，$A \vee B$ に置き換えるということです．q は他のどこにも出てこないので，このような置換を行っても問題ないです．規則によっては若干複雑なラムダ項を生成する必要がありますが，人間でも普通に書ける範囲内です．

これによって生成されるラムダ項は決して綺麗な証明図とは言えないので，次の 4 つの操作をします．

- β-簡約: $(\lambda x.\ M)N$ の形の項を，$M[x := N]$ に変換します．ただし，$M[x := N]$ とは，M 中に自由に出現する x を N で置き換える操作です．
- η-変換: $\lambda x.Mx$ の形の項であって，x が M に自由に出現しないものを，M に変換します．
- δ-簡約: 選言と連言について，コンストラクタで生成したものを直接デストラクタに渡している項を簡約します．
- 縮約: $\Gamma \vdash G$ の証明図中で $\Gamma \vdash G$ の証明図が出てきた場合，その間を縮約します．

1 から 3 はラムダ項としての意味を変えませんが，4 はラムダ項としての意味を変えます．ただし，証明図としての正しさは変わらないので今回は使ってよいです．

これらの操作によって，割りと綺麗な証明図ができます．実際の ipc_bot の実装では，もう少し変な簡略化の操作をしていますが，ここでは省略します．

フロントエンドの実装

証明器は OCaml で書かれています．OCaml コードの担当範囲は，論理式の構文解析から証明図の LaTeX ファイルとしての出力までです．

証明図は LaTeX ソースコードとして出力します．bussproofs と standalone を使っています．これを dvi に変換して，dvipng にかけることで png として出力します．

ここまでの一連の操作を，時間制限つきで行うためのシェルスクリプトがあります．

さらに，Twitter の mention を監視して，このシェルスクリプトを起動して結果を返す部分は Ruby で実装しています．

今後の展望

今後やりたいことはいくつかあります.

- 他の形式の証明図の出力，具体的にはシークエント計算とヒルベルト算術による証明図の出力.
- 古典論理の問題も解けるようにする．また，解けなかったときに反例を出力する.
- 線形論理の問題も解けるようにする
- 指数を含む線形論理や，古典の一階述語論理などの，解けないものに対しても，発見的なソルバを提供する.
- SAT ソルバを使うことでソルバを高速化する.
- $\Longleftrightarrow$ を含む論理式に対する前処理によってソルバを高速化する.
- 問題を生成する．現在も実装済みだが，質が悪いので，生成エンジンを作りなおす.

ソースコード

ソースコードは `https://github.com/qnighy/ipc_solver/` で入手可能です.

Java8 機能紹介

koba-e964

はじめに

1. この記事は Java7 までの Java の知識を仮定します. とくにインタフェースの仕組み, 匿名クラスなどは理解しておく必要があります.

2. 紙面および締め切りの都合上, この記事は Java8 で新しく導入された機能 (ラムダ式, メソッド参照, ストリーム,...) のうち, ラムダ式などに関連した機能だけを紹介します.

3. Java8 はまだ正式版がリリースされていません. そのためこの記事に書かれていることは将来変更される可能性があります. Java8 を使いたい方は Java8 Early Access を使用しましょう.

Java8 で新しく導入された機能について

デフォルトメソッド (インタフェース)

インタフェースは Java7 までは抽象メソッドしか持てなかったが, Java8 からはデフォルトメソッド (あらかじめ定義されているメソッド) を持てるようになった.
インタフェースを実装するクラス側ではデフォルトメソッドをオーバーライドしてもよいし, しなくてもよい.
(例)
従来からあった `java.lang.Iterable` には, 以下のようなデフォルトメソッドが追加されている
(`java.util.function.Consumer<T>`と `java.util.Spliterator<T>`は Java8 で新たに定義されたインタフェースである. また, デフォルトメソッドには `default` というキーワードが付いている):

```
package java.lang;

public interface Iterable<T> {
  public abstract java.util.Iterator<T> iterator();
```

```
  default void forEach(java.util.function.Consumer<? super T>){
    /* ... */
  }
  default java.util.Spliterator<T> spliterator() {
    /* ... */
  }
}
```

この機能の利点として, 互換性を失わずにインタフェースに新たなメソッドを定義できるという点が挙げられる. (上の `Iterable<T>`の例を用いると,Java7 までは `iterator()` メソッドしか定義されていないので, このインタフェースを実装するクラスは `iterator()` を実装していることしか保証されず, 別の抽象メソッドを追加するとエラーが起こるおそれがある.)

ラムダ式

ラムダ式とは

ラムダ式とは, 関数を名前を付けずに定義したもの (匿名関数) である. Java7 までは, 関数そのものを別の関数に渡すには, 匿名クラスを作って渡す方法 (`new Runnable(){public void run(){/**/}}` などの書き方で作る) しかなかったが,Java8 のラムダ式によってより直接的な記述ができるようになる.

文法

```
(Integer x,String y)->{
  int z=x+y.length();
  return z;
};
```

のような書き方をする.

ラムダ式のキャスト

多くのプログラミング言語と 違い, **ラムダ式単体には値を渡すことはできない**ので, 後述する **functional interface** 型にキャストする必要がある. キャストのやり方は主に 3 通り存在する. (注:以下で `IntUnaryOperator` は抽象メソッド `int applyAsInt(int value)` を持つ functional interface であるとする (標準ライブラリの `java.util.function.IntUnaryOperator`))

(1) 変数に代入

```
IntUnaryOperator temp=(int x)->{int v=x*x;return v+2};
temp.applyAsInt(20131123);//引数に 20131123 を与えて呼び出す
```

(2) 関数の引数として与える

```
void invoker(IntUnaryOperator arg){
  for(int i=0;i<10;i++){
    int result=arg.applyAsInt(i);
    System.out.println(result);
  }
}
void caller(){
  invoker((int x)->x*x);//int の値を 2 乗する関数を invoker に与える
}
```

(3) 明示的にキャストする

```
((IntUnaryOperator)x->x*x).applyAsInt(20131124);
```

functional interface

定義

Functional interface(機能インタフェース (この訳はどうにかならないかなと思ってしまいます.「関数インタフェース」の方が自然.)) とは,Java8 で導入された概念で, **抽象メソッドを 1 つだけ持つインタフェース**のことを指す. 一つだけ持つ抽象メソッドと同じシグニチャ(signature) を持つラムダ式を, その型にキャストすることができる.
例:インタフェース `java.util.function.Function<T,R>`は, 以下のように定義されている

```
public interface Function<T, R> {
  public abstract R apply(T);
  public <V> Function<V, R> compose(Function<? super V, ? extends T>);
  public <V> Function<T, V> andThen(Function<? super R, ? extends V>);
  public static <T> Function<T, T> identity();
}
```

唯一つの抽象メソッド `apply(T)` を持つので,`Function` は functional interface である. ここで, ラムダ式 `(Object o)->o.toString()` を考えると, この式は Object 引数を一つとり,String 値を返す関数を表しているので, `Function<Object,String>`にキャストすることができる:

```
Function<Object,String> ts=(Object o)->o.toString();
```

functional interface 型にキャストしたときに, 唯一つの抽象メソッドにはそのラムダ式が割り当てられるので, 例えば上の場合では, `ts.apply((Object)"test")` とした場合には, ラムダ式に `(Object)"test"`を渡して得られた戻り値が `apply()` の戻り値になる.

標準ライブラリ

`java.util.function` パッケージに新たに追加されたインタフェースで, 引数 1 個の関数はすべてカバーされる (詳しくは http://download.java.net/jdk8/docs/api/java/util/function/package-summary.html). それぞれのインタフェースに定義されている抽象メソッド (のシグニチャ) は以下の通りである:

`Function<T,R>` : `R apply(T t)`
`Predicate<T>` : `boolean test(T t)`
`Supplier<T>` : `T get()`
`Consumer<T>` : `void accept(T t)`
`IntFunction<R>` : `R apply(int value)`
`ToIntFunction<T>` : `int applyAsInt(T value)`
`IntUnaryOperator` : `int applyAsInt(int value)`
(このほかにも引数 2 個の関数のために `BiFunction<T,U,R>`などが用意されている)

また, 既存のインタフェースも抽象メソッドが 1 つであれば functional interface とみなされる:

`java.lang.AutoCloseable` : `void close() throws Exception`
`java.lang.Comparable<T>` : `int compare(T o)`
`java.lang.Readable` : `int read(java.nio.CharBuffer cb)`
`java.lang.Runnable` : `void run()`
`java.util.Comparator<T>` : `int compare(T o1,T o2)`

注釈 (annotation)

`java.lang.FunctionalInterface` という注釈が用意されている. これを付けたインタフェースが functional interface でない場合はエラーになるが, functional interface として扱いたいインタフェースにこれを付けなくてもエラーにならない (`@Override` などと同じ扱い).

応用編

ラムダ式を使って書けるコードは, 少し長くなるがラムダ式を使わなくとも書ける. ラムダ式の利点は, 記述量が圧倒的に少なくなることである. (匿名クラスの宣言, オーバーライドするメソッドのシグニチャを書かなくてよい点が大きい.)

数列ジェネレータジェネレータ

漸化式をラムダ式で与え, それに対して数列を生成するオブジェクトを作る.

コード例

```
import java.util.function.*;
import java.util.*;
public class Sequence{
  private ToIntFunction<int[]> recur; //recurrence relation
  private int[] buf;
  /*the number of integers which were already output*/
  private long count;
  public Sequence(ToIntFunction<int[]> recur,int[] initial){
    this.recur=Objects.requireNonNull(recur);
    Objects.requireNonNull(initial);
    this.buf=Arrays.copyOf(initial,initial.length);
    this.count=0L;
  }
  public int numberOfArguments(){
    return buf.length;
  }
  public int next(){
    if(count<buf.length){
      count++;
      return buf[(int)count-1];
    }
    //count>=buf.length, generate new number
    int[] arg=new int[buf.length];
    for(int i=0,n=arg.length,k=(int)(count%n);i<n;i++,k++,k%=n){
      arg[i]=buf[k];
    }
    int next=recur.applyAsInt(arg);
    buf[(int)(count%arg.length)]=next;
    count++;
    return next;
  }
  public static void main(String[] args){
    Sequence fib=new Sequence(x->x[0]+x[1],new int[]{0,1});
```

```
    //Sequence の利用者が記述すべきコードが非常に短いことに注意
    //fibonacci:a_{n+2}=a_{n}+a_{n+1},a_{0,1}=0,1
    for(int i=0;i<20;i++){
      System.out.println(fib.next());
    }
  }
}
```

Sequence の利用者は, 漸化式と初項を与えるだけで良い. 上のコードでは両方合わせて一行で済んでいる.

あとがき

まあ実際に使ってみるとよいと思います. 使ってみると問題点も見つかりますし. (型変数にプリミティブ型を指定できないので, 例えば int→long の関数型を示すときに Function<int,long> と書けずに IntToLongFunction と書かなければならない, とか)

ご紹介したコードは
https://github.com/koba-e964/experiment/tree/master/Java8EA
に置いてあります. 良かったらのぞいてみてください.

ゴシック体 3 分ロゴタイプ

@hakatashi

あらすじ

気がついたら提出締め切り 1 時間前だった。目の前の原稿は一文字も埋まっていない。
博多市は目の前が真っ暗になった。

概要

という訳で、モダンゴシックフォントを使用した簡単ロゴタイプの作り方を解説します。
ロゴタイプに普通のゴシック体を使うのはかっこ悪い、けどロゴタイプ用の書体は馬鹿みたいに高い！(参考価格:「ロゴ G」52500 円) そんなデザイナーのための簡単レクチャーです。
ベースのフォントには**モダン**ゴシック体を使用してください。新ゴやロダンあたりが最適です。

解説

博多市の強み

ベースとなるフォントを用意します。今回は新ゴとロゴラインを使用しました。

博多市の強み

若干ツメます。

博多市の強み

目標のボディ領域がわかりやすいように色分けします。

尖っている部分をボディ領域の外側に伸ばしていきます。

文字全体をボディ領域でカットします。

左右の平たくなっていない部分を適当にカットします。

文字全体を外側に引き伸ばしていきます。

細かい部分を調整します。

最後にカーニングを調整します。以上。

あとがき

「博多市の強み」ってなんだよ意味わかんない

理学部数学科「計算数学 I」 マシン組立実習の紹介

Hideaki Hosaka

計算数学 I とは

東大理学部数学科 3 年生の授業のひとつに「計算数学 I」という科目があります. 数学科というと数学の授業ばっかりしてそうなイメージがありますが, この授業はちょっと違って, 「コンピュータに親しむ」ことを目標としています.

その計算数学 I の中でも目玉の実習が「コンピュータ組立実習」です. この実習は 2 年に 1 度行われ, 文字通り, パーツからコンピュータを組み立てます. ちょうど今年は組立実習を行う年でしたので, 4 月の中頃に受講生のみなさんが頑張ってコンピュータを組み立てていました.

そして, この記事を書いている著者は今年の実習で用いるパーツの選定をしていました. そこで今回 TSG の部報の紙面を借りて, 授業の舞台裏をご紹介したいと思います. 授業自体の詳細については、授業の公式ウェブサイト `http://ks.ms.u-tokyo.ac.jp/`をご覧ください。

パーツ選定のコンセプト

コンピュータ組立実習は授業の一環として行われるので, 当然のことながらいくつか制約があります. より具体的には

- 予算は 1 台あたり 15 万円程度
- 実習のやりやすいパーツ構成にする
- 面白いマシンを作る

という条件がつきます. 最初の 2 つは当然という感じですね. 最後の「面白いマシンを作れ」というのは, 先生からの指示です. こういう指示が来たので, 面白いマシンを組むために色々と頭をひねることとなりました. その結果 3 つのコンセプトが出てきたので, まずそれをご紹介したいと思います[1].

[1]なお, パーツ選定を行った時期は 2012 年の 11 月, ちょうど 1 年くらい前のことです. 適宜補足を入れはしますが, 分かる人は当時の状況を思い浮かべながら読んで頂けると幸いです.

Windows 8 とタッチパネルディスプレイの導入

2012 年 10 月 26 日に, Windows 8 発売という大ニュースがありました. この Windows 8 については操作感を中心にネガティブな評価が続出していました[2]が, とりあえず「新しいもんだし使うしかないだろ」と, 脊髄反射で Windows 8 を採用しました.

そして Windows 8 と言えばタッチパネルがウリですから, 是非とも実習でタッチパネル操作をしたいと思うのが人情です. ちょうどディスプレイの老朽化が進んでいた[3]ので, その刷新も兼ねて, タッチパネルディスプレイを買うことにしました.

光学ドライブ不採用と USB3.0 メモリの利用

続いて決めたのは, 光学ドライブの不採用です. 一昔前は OS のインストールというと DVD を使うのが普通でしたが, DVD の転送速度は遅く[4], これまでの実習でも OS インストール時の待ち時間は無駄に使われていました. そこで今回は OS のインストールに USB メモリを使うことにしました.

この背景には USB3.0 の普及があります. いま “USB” というときの USB は大体 USB2.0 という規格に対応するもので, USB3.0 はその進化版にあたります. パーツ選定をした当時は「USB3.0 が徐々に市場に広まりつつある」といった状況でそれなりに商品のラインナップもあり, なにより「USB2.0 の 10 倍の転送スピード」が魅力的だったので, USB3.0 対応の USB メモリから高速に OS インストールを行うことを企てました.

PCI-Express 接続の SSD

せっかくインストールに USB3.0 対応のメモリを使うのだから, OS のインストール先には早い SSD を選ぼうと思いました. そこで目を付けたのが PCI-Express 接続の SSD です. 大抵のコンピュータでは中を開けると SSD が SATA ケーブルでマザーボードに接続されています. しかし SATA ケーブルだと, 最も早いものでも転送速度 750MB/秒が限界です[5]. そこで PCI-Express 接続で直にマザーボードと SSD をつなぐことにしました[6].

[2] と言っても Windows 7 より良くなった点もあります. たとえば動作が軽く, 起動時間はかなり短縮されました.
[3] 正確には覚えていませんが, 少なくとも耐用年数を大分超過して使われていました.
[4] 頑張っても 20MB/s くらいしか出ません。
[5] ここで念頭に置いているのは SATA6G のことですが, 2013 年になって, より新しい SATA の規格が出ています
[6] 後述しますが, この判断で罠にハマりました.

パーツ選びの詳細

CPU とメモリ

CPU 選定にあたり, まず最初に秋葉原での価格調査結果[7]を見ました. その結果, お手頃価格帯の CPU として Intel Core-i7 3930K と AMD FX-8350 が候補に浮上しました. コア数は FX-8350 の方が多かったのですが, ベンチマークテスト[8]の結果 Core-i7 3930 が勝っているようでしたので, こちらを選びました.

メモリについては深く考えず, 32GB 積むことに決めました. 「何に使うんだ」という声が聞こえてきそうですが, 8GB のメモリが 1 本 3000 円で買えてしまう時代でしたので, ケチケチしないことにしました.

SSD

今回のパーツ選定のキモになったのが SSD です. 前述の通り PCI-Express 接続のものを採用することを目論んで kakaku.com を調べたところ, 手の届く価格帯に唯一 OCZ 社の RVD3-FHPX4-120G があり, これを採用することにしました. 転送速度は読み込み 1GB/s を公称しており, 文句なしです[9].

グラフィックボード

デュアルディスプレイをする可能性を念頭に置き, 今回取引したお店の方に「HDMI 端子と DVI 端子を 1 つずつ搭載する nVidia チップ搭載の製品で, 1 万円程度のもの」を提示してもらいました. nVidia 製チップを選んだ理由は「GPU プログラミングをするなら, nVidia チップを使う人が多い」というアドバイスをもらったからです. その結果 Leadtek WinFast GT 630 4096MB SDDR3 を使うことになりました.

もし GPU プログラミングをしたいという要求があれば上位のグラフィックボードを選んでいたところですが, 残念なことに数学科内では GPU プログラミングをする人がほとんどいませんでした. 加えて震災後からずっと節電が叫ばれていたこともあり, 無駄に GPU パワーを求めることはしませんでした.

---

[7] http://ascii.jp/elem/000/000/746/746101/

[8] http://pc.watch.impress.co.jp/docs/topic/feature/20111114_490745.html
http://pc.watch.impress.co.jp/docs/topic/feature/20121023_567804.html

[9] しかしこの SSD は Windows 用にしかドライバが用意されていないという罠がありました. その結果, 同期の K 君に迷惑をかけ, 挙げ句 Linux 実習用に追加の SSD を買うことになってしまいました. もっときちんと調査すべきだったと反省しています.

マザーボード

ここまでのパーツ選定の結果, マザーボードは

- Core-i7 3930K を載せられる
- PCI-Express 16x, 4x のスロットを 1 本ずつ搭載している

という条件を満たさなければいけないことが分かります. 当時は Core-i7 3930K が出たばかりで, この 2 条件だけでマザーボードは ASUS Rampage IV GENE だけに絞られてしまいました.

USB メモリ

とりあえず USB3.0 対応メモリを使えば転送速度が早いというのは分かっていました. しかし落ち着いてネット上を調べてみると, USB3.0 対応メモリの中にも早い物と遅い物があることが分かりました. たとえばグリーンハウス社の GH-UFD3-*J シリーズが「読み込み 150MB/s の高速転送を実現」とうたっている[10]ことから, その辺の適当な USB3.0 メモリでは 150MB/s は実現できないことが分かります.

その後調べを進めてみたところ, SanDisk 社の SDCZ80 という製品が読み込み 190MB/s を実現していることが分かりました. ただし SDCZ80 の 64GB 版には不良品が多いという口コミがあったので, これの 32GB 版を採用しました.

ディスプレイ

ディスプレイについては「Windows 8 をタッチパネルで使う」「デュアルディスプレイにも対応する」という目的に適うものを探し回りました. また古いディスプレイが 15 インチ程度の大きさしかなかったので, 机に入るギリギリの大きさの商品を選ぶことも狙いました. その結果, グリーンハウス社の GH-JTJ223GSHK を採用することになりました. なんとただのタッチパネルではなく, マルチタッチ対応です. これで Windows 8 が一層楽しめます[11]ね !

ケース, 電源とクーラー

最後に残ったのはケースと電源ですが, 電源は不安定でさえなければ良く, ケースについてもこだわりませんでした. 適当に検索したら GIGABYTE 社の GZ-M1 ZGM1B9R が microATX 規格で, かつフロントパネルに USB3.0 ポートを搭載していたので, これに決定しました. また CPU クーラーには Intel の純正品を選び, 電源は適当に玄人志向 KRPW-L4-600W にしました. 手抜きの理由は著者の知識不足によるものです.

[10] http://www.green-house.co.jp/products/pc/usbmemory/hispeed/gh-ufd3-_j/

[11] Windows 8 の機能を全て使うには 10 点マルチタッチ対応製品が必要らしいですが, さすがに予算の関係で諦めました.

実習中のトラブル

上記のパーツを使って, 2013 年度の夏学期の授業で組立実習を行いました. 実習自体については時間がかかる学生さんはいたものの, パーツ初期不良と思われるトラブルを除き, 全員が組立実習を完成させることができました. 実習の流れは普通のマシン組立と何ら変わらず, 新マシンの性能に文句がないのは明らかですので, 特筆すべきトラブルだけをご紹介します.

謎の再起不能

Windows 8 のインストールが終了したマシンが何故か起動不能になるというトラブルが, ポツリポツリと見られました. この原因は未だに分かっておらず, また再現条件も確認できていません. 非常に怖いですが, 安定なマシンはいつまで経っても安定なようなのでそのまま使い続けました. 何だったのでしょう.

ただ Windows 8 を USB3.0 対応メモリからインストールしていたおかげで, OS 再インストールにかかる時間はたった 10 分で済みました. 「OS 壊れても入れ直せばいいよね」とあっさり言える環境を作っておいて, 良かったです.

Linux インストールについて

計算数学 I の授業の後半では Linux を使うのですが, あろうことか, 今回導入した SSD に Linux 用ドライバが提供されていませんでした. 結局のところ SATA 接続の SSD を各マシンに 1 個ずつ追加して凌いだのですが, SSD 追加購入の決断に至るまで, 同期の K 君をはじめ関係者のみなさんに大変なご迷惑をかけてしまいました. ごめんなさい. パーツ選定の段階で気付くべきでした.

UEFI と secure boot

一昔前に PC 組立をやったことがある人は, 必ず BIOS のお世話になっていました. しかし最近は UEFI というものが使われており, かつて BIOS でやっていた Windows インストール前のセッティングにマウスが使えたりします. 時代は変わったものですね.

…で, UEFI になって便利になるだけなら良かったのですが, UEFI のせいで Linux インストール時につまずくことがありました. 著者は未だに UEFI が何たるかを理解できていないので, どうしてトラブルが起きるのかもよく分かっていません. 誰か教えてください. また PC 起動時にウイルス等が立ち上がるのを防ぐ secure boot という機能がありますが, これも何かトラブルの原因になってた気がします.

総括など

実習を終えての感想

マシン組立について一通りのことを紹介したので, ここで実習を終えてみての感想を述べたいと思います.

真っ先に思うのが, K 君をはじめとする TA, 職員さんや先生の力強いサポートがあって良かったということです. 著者の力では解決できないことを回りの人が助けてくれるのは, 大変ありがたかったです[12].

それから, 組立実習は何度やっても楽しいものですが, 今回のは過去のどの回よりも楽しかったです. その理由は, パーツ選定コンセプトのところで述べたように Windows 8, USB3.0, PCI-Express 接続の SSD という新技術たちを一斉に投入できたからだと思います. 「自分が次に PC を買う時はこんな感じで使うのかな」というイメージを思い描くことができました. ただ, ちょっと先走りすぎたという感もあります. SATA ケーブルの転送速度がボトルネックになることは 2012 年中から指摘されていましたが, PCI-Express 接続の SSD は 2013 年夏になってようやく Samsung が出荷を開始するという状況なので, 計算数学 I への投入は時期尚早でした. また新技術を一斉に投入したせいで, トラブルの原因特定などにも悩まされました. でも, チャレンジして良かったか悪かったかと聞かれたら, 確信を持って「良かった」と答えられます.

将来について

最後に, 今回の実習で組み上がったマシンを触った後に「将来の PC はどうなるんだろう」と思いを馳せてみました. 先生からの「面白いマシンを作れ」という指示に今回はそれなりに答えられたと思いますが, 果たして 2 年後の実習を今回以上に面白くするにはどうしたら良いのでしょう?

ここ最近, 世間の嗜好は明らかに「スマートなもの」にシフトしています. デスクトップ PC よりはラップトップ PC が好まれ, 性能よりむしろ軽さ, 電池の持ちや静かさが重視されます. そういう意味では, 古めかしいデスクトップ PC を組み立てるだけでも受講生には珍しい体験になるかもしれません. しかし PC の性能も徐々に上がるし Windows 8 はタッチパネルを意識して作られているし, きっと操作感については様々な PC で等質化が進むはずです. そうしたときに PC 組立実習は「デスクトップ PC の物珍しさ」以外にどんな面白さを実現できるのでしょうか?

次の組立実習をする TA たちが突き当たるであろうこの問題をどうすればいいのか, 著者にはまだアイデアがありません[13]. 良いアイデアをお持ちの方は, ぜひ TSG の駒場祭企画で著者を捕まえてください. また読者のみなさんの中に理学部数学科に在籍している/将来進学しようと思っている人がいたら、ぜひ計算数学 I で遊びましょう!

[12] 逆に考えれば, 著者が好き勝手やって回りに問題をばらまいてたことになりますね…orz

[13] ただ Windows 8 を USB メモリに入れて持ち運ぶ “Windows To Go” が面白い可能性はあると思っています.

編集後記

★ TSG の展示にお越しくださり、ありがとうございます。
★ 今回は原稿の募集が遅く締め切りが早めだったので、部員の皆さんは大変だったと思います。
★ 私が編集する部報としては最後になると思います。
★ 展示作品、および記事の方を楽しんでいただけたら幸いです。
★ TSG にお越しいただいた方に, 今一度礼を重ねて失礼します。

理論科学グループ　部報 第 305 号

2013 年 11 月 14 日　発行
発行者　村瀬唯斗
編集者　小林弘季
発行所　理論科学グループ
〒 153–0041　東京都目黒区駒場 3–8–1
東京大学教養学部内学生会館 305
Telephone: 03–5454–4343

理論科学グループ部報 第 305 号
── 駒場祭パンフレット号 ──
2013 年 11 月 14 日

THEORETICAL SCIENCE GROUP