京都工芸繊維大学コンピュータ部

Lime 36

奉行挨拶

本年も学祭にあたり Lime を発行いたします。早いもので今年度が始まりましてから、もう半年が過ぎました。舵取りの資質に些か不安の残る出発ではありましたが、昨年に引き続き多くの新入部員を迎え、順風満帆に何もせずとも進んでいく勢いであります。人がいることの力強さを感じます。では最後までお付き合いいただけますよう。

編集を担当した小長谷氏に 10byte の感謝の言葉を。

平成 19 年 11 月 21 日

京都工芸繊維大学コンピュータ部部長　林　奉行

目 次

1	地上デジタル放送のあれこれ	2
2	EL ディスプレイについて	6
3	データ圧縮について	12
4	Windows Me について	17
5	Xlink Kai について	20
6	コピーガード	23
7	P2P について	31
8	Prolog プログラミングガイド	34
9	CPU を創る	41
10	ほのかとひよりの教えて HTML	48
11	はじめての構文解析器生成系	51

1　地上デジタル放送のあれこれ

情報工学課程　1回生　荒木　修

1.1　はじめに

地上デジタル放送が 2003 年の 12 月 1 日に三大都市圏で放送が開始されて、約 4 年が過ぎた。ふと通りかかった家電量販店では、ショーウィンドウ上に、今なおお高いプラズマテレビや液晶テレビが並べられ、「地上デジタル放送は、こんなにもきれいなものだぜ。」と過剰にアピールされている光景を多々見かける。地上デジタル放送では、アナログ放送にはない様々な機能があるらしい。でも、具体的には知らない。だから疑問を抱いた。「本当にいいものなのか。」だったら調べてみればいい。どうせアナログ放送は、映らなくなるんだ。地上デジタル放送のことを知っておいて損はない。

ということで調べてみることにしました。内容は、荒探しで調べたことを書いているので、内容が薄く、間違っている可能性が十二分にあります。どうかその点はご勘弁を。あと、あの長ったらしい名前を毎回書くのは大変なので、地デジと略して書きます。

1.2　地デジに変わる訳

地デジとは、正式名称が地上デジタルテレビジョン放送のことを指し、2003 年 12 月に関東・近畿・中京の都市圏から放送が開始され、2006 年 12 月には全国の県庁所在地で、放送が開始されています。さらに従来までのテレビジョン放送であるアナログ放送は、2011 年 7 月 24 日までに終了し、地デジに完全に移行することが決定されています。まずここでは、地上アナログ放送から地デジに、とってかわる必要性について述べます。

1.2.1　放送サービスの強化

とりあえず名前にもあるように、デジタルとアナログの違いが深く関係しています。深い技術的な内容は知りませんので、あくまで概要程度です。まずデジタルは、データを 0 と 1 のビット列で表現しています。いっぽうアナログはデータを連続する波のゆらぎによって表現しています。アナログは外部からの影響が受けやすく、アナログ放送の場合、ビルの反射波で映像によくゴーストが出たり、ノイズによって音声が乱れたりします。しかし、デジタルではデータを数値化しているため、ノイズによって多少誤差が入っても、中間値がないためエラー訂正によって元のデータに復元することができます。つまり放送をデジタル化すれば、映像や音声の乱れをほとんどなくすことが可能なわけです。さらにデジタル化によって、デジタルハイビジョン放送や、双方向サービスなど、アナログ放送に比べて、さまざまなサービスを可能とすることができます。

1.2.2 電波の有効利用

従来のアナログ放送では、VHF 帯と UHF 帯を使ったテレビジョン放送が提供されていました。VHF とは Very High Frequency の略で、超短波と呼ばれる 30MHz から 300MHz までの周波数帯のことを指します。UHF とは Ultra High Frequency の略で、極超短波と呼ばれる 300MHz から 3GHz までの周波数帯のことを指します。UHF は VHF より直進性が強く、より遠くまで電波を飛ばせるという長所がありますが、反面障害物に弱いという短所もあります。アナログ放送では、VHF 帯の 90MHz〜108MHz (1ch〜3ch) と 170MHz〜222MHz (4ch〜12ch) と UHF 帯の 470MHz〜770MHz (13ch〜62ch) を利用していましたが、周波数が近いもの同士で互いに干渉を受けやすく、受けなくするために、周波数の間隔を余計に空けなければいけない（VHF 帯のチャンネルの間が 1 つ飛んでいる理由）ため、膨大な電波の利用と伴って、過密状態という問題を抱えていました。しかし、地デジでは UHF 帯 470MHz〜710MHz(13ch〜52ch) だけを利用する。これはデジタル化により周波数が近いもの同士でも、互いに干渉を受けにくいので、チャンネル数を減らすことができるからである。さらに、あまった周波数の部分は他の利用用途に、有効利用できるというメリットも生まれます。

1.2.3 情報化の向上

家庭での主要なメディア媒体は、テレビです。また近年、さまざまなものがデジタル化されています。そこで、デジタル放送を提供することにより、テレビで誰でも手軽な操作で、より多くの情報を入手しやすくなるように目指しています。

1.3 地デジの仕組み

まずは、デジタル放送の仕組みについて、簡単に述べます。まず作られた映像信号と音声信号を MPEG-2Video と MPEG-2 Audio と呼ばれる符号化方式で圧縮されます。さらにこれを ES(Elementary Stream), そして PES(Packetized Elementary Stream) に変換されます。また番組情報やデータなどは Section と呼ばれる形式に変換されます。これらをまとめて、多重化処理をし、TS(Transport Stream) パケットと呼ばれる 188 バイトの伝送に適したものに分割されます。それらに、誤り訂正符号やインターリーブなど様々なものが付加・処理され、OFDM 形式にデジタル変調され電波どして送信されます。家庭では、これを復調し、誤り訂正などをしたあと、映像パケットと音声パケットを取り出し、映像信号・音声信号に戻されることによって、番組を楽しむことができます。

1.4 地デジのおもな特徴・機能

ここでは、地デジの主な特徴・機能を述べます。膨大な数の特徴や機能があるため、細かい部分やあまり重要そうでないものは、省いていますので、ご注意を

1.4.1 ハイビジョン放送とマルチ編成

MPEG-2 とデジタル化によりいままでのアナログ放送 1 チャンネル分でハイビジョン放送を楽しむことができます。またハイビジョン放送には 12 セグメント使用していますが、マルチ編成ではこれを 4 セグメント (標準画質) にすることにより、最大 3 チャンネルを同時に楽しむこともできます。

1.4.2 ゴーストの低減

アナログ放送では、建物による電波の反射の影響で、映像にゴーストが入ってしまいますが、地デジでは上記の説明を理由に、反射波やノイズの影響を受けにくく、ゴーストが入らない高品質の映像を見ることができます。

1.4.3 データ放送

番組放送と別にデータ放送というものを楽しむことができます。これは BML(Broadcast Markup Language) という言語を用いて作られています。BML は XHTML をベースにデジタル放送用に拡張したものです。受信者側では、BML を実行するために BML ブラウザが組み込まれていて、文字の大きさや表示場所の制御、情報の自動更新などを行うことができます。

1.4.4 双方向サービス

いままでにも、番組参加型の放送はありましたが、携帯やパソコンからなど別の通信機器を利用するタイプでした。しかし、地デジでは別途インターネット回線などを接続することによって、テレビ画面を操作して参加することを可能にしています。ただし、今現在双方向サービスに対応している番組は少なかったり、番組制作側のコストの問題など課題が残っているのが現状です。

1.4.5 EPG（電子番組ガイド)

番組表や番組情報のデータを電波の空いている部分を利用して、送信されています。地デジの EPG はテレビ局が各自でそれぞれ作っているため、更新頻度が早いというメリットがあります。

1.4.6 コピーワンス

地デジは、アナログ放送に比べてほぼ映像が劣化しないため、著作権保護の理由で放送映像の録画の制限をしています。CGMS(Copy Generation Management System) が使われており、「コピーフリー」「コピーワンス」「コピー禁止」の制御信号を付加できるようになっています。現在、地デジではコピーワンスという制御信号が送られており、これは 1 世代だけコピー可能、例えば HDD にコピーした場合、HDD から DVD-R にコピーができないことを指しています。ただし、「ムーブ」という機能もあり、これにより、録画した番組を HDD から DVD-R に移動すること (HDD の録画した番組は自動的に消去されます) ができます。

しかしながら、録画するレコーダーが CPRM に対応していないとムーブできなかったり、HDD から DVD-R に移すときの失敗によるデータ損失、利便性の欠如など様々な問題を出していて、現在再検討がなされています。

1.4.7 B-CAS カード

上記のコピー制御とは別に、地デジは、スクランブルがかけられており、見たりするのに解除するための鍵データが入っている B-CAS カードという IC カードが、必要になります。挿入されていないとテレビを見ることができません。B-CAS カードは地デジ対応の機器を購入したときに同梱しており、別途何らかの方法で手に入れる必要はないようです。

1.5 問題点

たしかにより高水準な放送を楽しめることは間違いないが、一方で様々な問題もあります。上に挙げた著作権と利便性の問題や、新たに地デジに対応したチューナー、テレビを買わなければいけないこと、さらに高機能がかえって操作を複雑にし、デジタル機器を苦手とする人から娯楽を奪ってしまうかもしれない問題などがあります。

1.6 おわりに

惹かれるところはある。でも、地デジといっても、いいことばっかりではなさそうだ。だが、まだ完全に移行するまで約 4 年ある。まだ、なにかが、変わるかもしれない。それまでは様子を見ておこうか。まだ、買い替えるには、テレビもチューナーも高すぎるしな。

とまあ、そんなことを思ってたりします。ここまで、こんないたらない文章を最後まで読んでもらってありがとうございます。本当はもっと詳しいことを書かなければいけないはずなんですが、自分の力量不足でこの程度の文章になってしまいました。すこしでも、テレビの買い替えなどに参考になれば、幸いです。

1.7 参考文献

- デジタル放送研究会:著「デジタル放送の技術とサービス」, 技術評論社, 2006 年
- 八木伸行/吉田俊郎/加井謙ニ郎:著「データ放送技術読本」, オーム社, 2002 年
- D-pa(社会法人地上デジタル放送推進委員会), http://www.dpa.or.jp/
- 総務省, http://www.soumu.go.jp/

2　ELディスプレイについて

情報工学課程 1 回生　出原真人

2.1　はじまりの前に

最近話題になっている有機 EL ディスプレイ。
今年 12 月にはソニーが 11 インチの有機 EL テレビを発売するらしい。
メディアに乗せられてなんとなく凄いような気がしてきたが…どんなものなのかよくわからない。と、いうわけで有機 EL のことを調べてみました。かなり無節操に調べていったので内容が浅めになってしまったような気がしますがご容赦を。寛大な心で生暖かく見守ってやってください。もし間違っていても責任は負いません、自己責任で GO です。
――それでは開幕。

2.2　基本事項

2.2.1　エレクトロルミネッセンス (Electro Luminescence)

EL とはエレクトロルミネッセンス (Electro Luminescence) を縮めたものです。物質が光、電子ビーム、電界などのエネルギーを受け取り、それを光として再放出する現象のことをルミネッセンス[1]といい、そのうち電圧をかけることによって放出される現象をエレクトロルミネッセンスといいます。

2.2.2　EL の分類

EL 素子には大きく分けて有機化合物[2]を使用している**有機 EL** [3]と無機化合物[4]を使用している**無機 EL** の 2 種類があります。さらに有機 EL は分子量によって分類されます。分子量が小さいものを低分子系、大きいものを高分子系といいます。

[1]光を照射したときに放出されるのをフォトルミネッセンス、電子ビームを照射したときのをカソードルミネッセンスと言う。
[2]炭素を含む化合物で簡単なものを除いたもの。
[3]原理が発光ダイオードに似ているので OLED(Organic LED) と呼ばれることもある。
[4]炭素を含まない水、石、鉱物など。炭素を含む簡単な化合物。

2.2.3 EL ディスプレイの特徴

EL ディスプレイの最大の特徴は**自発光デバイス**であることです。
それによって

- 視野角が大きい
- 応答速度大
- 高コントラスト

などの特性を持っています。ただし

- 寿命が短い (最近では急速に改善されてきている)

といった欠点も持っているのでそれを解決するために世界中で研究されています。ちなみに、磁気の影響を受けません。

2.3 EL の解説［→ side:有機 →］

2.3.1 有機 EL 素子の構造

有機 EL 素子は両側から電極で有機薄膜層をサンドイッチした構造になっています。有機薄膜層は発光層、電子輸送層、正孔輸送層の 3 層構造です。また、発光側の電極は光透過性のある透明電極である必要があります。
以上がすべての層を独立させた 5 層型の説明です。代表的な例として他に次の 3 つがあります。

- **単層型**：最もシンプルかつ理想的、ただし材料発見が難しい。
- **2 層型**：正孔輸送層を設けたタイプ。
- **3 層型**：正孔、電子ともに輸送層を設けたタイプ。

主として低分子系では多層構造が、高分子系では単層構造が多く採用されています。
電極などの役割・材料

- **陽極 (ITO)**：正孔の注入効率や表面抵抗の低さから ITO [5]を使うのが一般的。ガラス基板上にスパッタ蒸着などで形成する。
- **陰極**：不透明な金属材料が使えるので電子注入率の良いアルミやマグネシウム銀合金などを使う。
- **正孔注入層**：陽極と正孔輸送層を隣接させると注入率が落ちるので、この層を挟んで電極から正孔を効率よく引き出せるようにする。
- **正孔輸送層**：正孔の発光層への円滑な移動をサポートし、電子が移動してくるのを阻止する。
- **電子注入層**：陰極と電子輸送層を隣接させると注入率が落ちるので、この層を挟んで電極から電子を効率よく引き出せるようにする。
- **電子輸送層**：電子の発光層への円滑な移動をサポートし、正孔が移動してくるのを阻止する。

[5]Indium Tin Oxide:インジウムとスズの酸化物。

2.3.2 EL の発光

上の構造で電極の陽極に＋、陰極に－の直流電圧をかける、すると陽極、陰極からそれぞれ正孔、電子が注入されて対岸を目指します。このとき、正孔、および電子の振る舞いはかなり動きにくいものとなっているので輸送層は極めて薄いものが望ましいとされています。しかし、あまりに薄いとピンホールができやすく、有機薄膜間が短絡して動作不能を引き起こしてしまいます。通常、有機 EL ではこの厚さは数十～数百ｎｍです。
発光層にたどり着いた正孔と電子はキャリア再結合をします。これによって有機分子の電子エネルギーが基底状態から不安定な励起状態へと遷移します。励起状態は不安定なためすぐにエネルギーを放出して基底状態に戻ろうとします。このとき放出されるエネルギーが有機 EL の発光です。

2.3.3 蛍光とリン光

電子が励起状態から基底状態に戻るとき、2 通りの道筋があります。1 つは励起状態[6]から直接基底状態に戻る方法、そしてもう一つはエネルギーのやや安定した中間状態[7]を経由して基底状態へと戻る方法です。前者だと蛍光、後者だとリン光を発します。蛍光とリン光の割合は統計学的に 1:3 とされています。電気エネルギーを与えて光として取り出せる効率のことを内部量子効率といいますが、あくまで理論値で実際に取り出せる量の割合[8]は数年前まで 10 %程度でした。というのも蛍光しか発光を観測することができなかったからですが、最近ではリン光も観測できメーカーによっては内部量子効率 100 %近い値が出せるようです。

2.3.4 マルチフォトンエミッション素子

有機 EL はどこまでも続いていく → さらなる明るさを求めて
実は内部量子効率を上げる以外にも明るさを確保する手段はあります。それがマルチフォトンエミッション素子です。有機 EL をｎ個直列に並べて同一方向から光を取り出し、それを一つの有機 EL 素子とみたてるとします。すると、外部から観察した場合、フォトン数、光変換効率ともにｎ倍になります。全体での印加電圧もｎ倍になってしまいますが電流は従来と同じままでいけます。これがマルチフォトンエミッション素子の考え方です。各有機 EL 層を透過性のある材料で構成できれば直列につないだ有機 EL 素子を積層することで従来に比べて非常に明るい有機 EL パネルの製作が可能になります。

2.4 材料とそれぞれの利点 【低分子→←高分子】

2.4.1 材料の条件

- 発光効率が良いこと。

[6]エネルギーの大きい不安定な励起状態を一重項状態という。
[7]三重項状態ともいう。
[8]外部量子効率という。

- キャリアの輸送性が良いこと。
- 成膜性が良いこと (ピンホールのない均一な膜質)。

最近ではホスト材料とゲスト材料をドーパント (有色色素) として微量加えた組み合わせ方式、すなわちドーピング法が多く用いられています。

2.4.2 低分子系

材料の代表的なものとしてはアルミニウム錯体[9]があげられます。これは比較的電子移動度が大きく成膜性が良いのでひろく使われている。低分子系ではドーピング法によって発光の高効率化、長寿命化、低電圧動作化が図られています。

2.4.3 高分子系

高分子系材料の利点は低分子系と比べて比較的に簡単な塗布や印刷でできること、物理的強度が高いことなどです。よって低コスト化や大面積化が要求される TV 応用に向いている。特にインクジェット法での印刷は
【数ミクロン単位での制御ができる】【材料使用効率が高い】【フルカラー対応可能】
なことから有力視されている。

2.4.4 低分子系 vs 高分子系

ここで一度比較しておくことにする。

	【低分子系】	【高分子系】
【素子構造】	多層構造	単層構造
【製造方法】	真空蒸着	塗布、印刷
【材料コスト】	材料利用率が低くコスト高	ほぼ無駄がなく安い
【発光効率】	多層化で効率をあげられる	単層なので効率が落ちる

2.5 カラー化の方法

2.5.1 エリアカラー

有機 EL ディスプレイにはエリアカラーという製品があります。エリアごとに赤や青などの色が付いていて、色が変化することがない製品のことです。車載のカーオーディオなどに使用されています。不必要な部分をマスキングして色素材料を順次蒸着させてつくる。フルカラーのような精密さはいらないので比較的簡単にできます。

[9] トリスアルミニウム Alq:アルミニウムイオンと有機物の結合体。

2.5.2 3 色発光方式 (3 色塗り分け方式)

発光層の所定の位置に RGB3 色の発光領域が分割されて配置されているもの。発光材料は金属マスクを用いて順次シンク蒸着により発光膜として生成される。陰極側が共通で、RGB に対応した陽極が独立しています。高精細化、大型化が進むにつれアライメント制度が悪化し、高繊細なフルカラー化が難しくなる。

2.5.3 カラーフィルタ方式 (白色方式)

発光層は白色発光材料でのみできていて、カラーフィルタを通過させることにより RGB の 3 色に変化させます。製造工程が比較的容易なため、高精細化に有利とされている。しかしカラーフィルタで減衰が発生するので発光効率はおちてしまう。それをなんとかすると消費電力が増大してしまう。

2.5.4 色変換方式

発光層は青色発光材料のみでできていて、青色の場合はそのまま直接取り出し、赤、緑の場合はそれぞれ蛍光体を介して色変換して取り出す方式。色変換は青色のエネルギーで蛍光体を励起させて光を出させています。製造工程が容易でコストが安くできる可能性がありますが、外光で蛍光膜が励起してしまいコントラストが低下してしまうというデメリットも持っています。

2.6 駆動方式

駆動方式は液晶ディスプレイと基本的に同じでパッシブマトリクス方式とアクティブマトリクス方式が用いられる。カラー化技術に関しても同様であり 1 画素は RGB の 3 色の要素で構成され、加法混合でカラー化を実現します。

比較してみる

- 輝度：発光時間などを考慮するとパッシブマトリクス方式はアクティブマトリクス方式の 1/120 〜1/140 程になる。
- 消費電力：輝度を合わせようとすると、パッシブマトリクス方式はアクティブマトリクス方式の約 10 倍の電力が必要になる。
- 寿命：有機化合物なので、もし同じ材料で大電流を流すと寿命は大幅 (1/100 程度) に短くなってしまいます。

【輝度】	パッシブマトリクス方式＜アクティブマトリクス方式○
【大画面・高精細化】	パッシブマトリクス方式＜アクティブマトリクス方式○
【消費電力】	パッシブマトリクス方式＜アクティブマトリクス方式○
【寿命】	パッシブマトリクス方式＜アクティブマトリクス方式○
【コスト】	○パッシブマトリクス方式＞アクティブマトリクス方式

このような結果になったが、小型パネルならパッシブマトリクス方式でも十分にきれいな画質が得られるので用途に合わせて選択していくのが望ましい。

2.7 最後に液晶と比べてみる

・高輝度、高コントラスト
自発光デバイスなのでバックライト式の液晶に対して優勢であるが、液晶も改良[10]されてきている。
・視野角
有機 EL は視野角による制限を受けないが、最近の液晶は視野角が 170 度あるようなものも登場している。
・消費電力
バックライトがいらない分、有機 EL が優勢。発光効率次第で絶対的優位に立てる。
・寿命
液晶はバックライトの寿命だが有機 EL は輝度劣化そのものなので、液晶並みにしようとするとその分消費電力も UP して**寿命がさらに短くなる**のが問題になっている。
・外形寸法
有機 EL が非常に有利。基盤を含めディスプレイとして 1.5 mmが可能であり、曲げたり丸めたりもできる。
・応答スピード
有機 EL が圧倒的に有利であるが、液晶も実用的に問題ないレベルまで来ている。

有機 EL の敵は自分自身、 **輝度と寿命のトレードオフ**が最大の問題点だ。
色シフトの問題もあるが液晶と同程度には抑えられるようになっている。

2.8 オワリに

ここまでお付き合いいただいて本当にありがとうございました。本当は無期 EL や駆動方式、トップエミッション構造に SOLED などもう少し突っ込んでみたかったんですが…もう限界です。それにしても…なんでディスプレイにしたんでしょうね、EL シートとか EL 光源とかもあったのに。実用化されているものを考えればそれこそ星の数ほど、とまではいかないまでもたくさんありますね。まぁその辺は次の時にやる気が湧いてくればということで。
そういえば気になった商品が一つ、そろそろ発売するらしい「Optimus Maximus」という名前の有機 EL キーボード。キーが全て有機 EL ディスプレイになっていて、それぞれ好きな文字や絵を表示できるとか。値段はなんと約 1500 ドル。…高いです。
まぁそんな話は置いておくとして、ほんの少しでもお楽しみいただけたなら幸いです。

2.9 参考文献

- 伊吹順章, ディスプレイデバイス, 産業図書株式会社
- 吉野勝美, 有機 EL のはなし, 日刊工業新聞社
- 西久保晴彦, 最新ディスプレイ技術の基本と仕組み, 株式会社 秀和システム

[10] MVA 方式など。

3　データ圧縮について

情報工学課程　1回生　實 克洋

3.1　はじめに

「圧縮って何？」という人はあまりいないかも知れませんが端的に説明すると一定の手順にしたがって、データの意味を保ったまま、容量を削減する処理のことです。

ここではこの「圧縮」についていろいろと話していきたいと思います。

3.2　可逆圧縮と非可逆圧縮

データ圧縮には大きく分けて可逆圧縮と非可逆圧縮というものがあります。可逆圧縮とはデータの欠落がまったく起こらない圧縮方式のことで、解凍後は圧縮前のデータを完全に復元することができます。それに対して非可逆圧縮は多少のデータの欠落を許容する代わりに、劇的に圧縮効率を高めた圧縮方式のことで、解凍後は圧縮前のデータを完全に復元することはできません。

3.3　様々な圧縮方式

圧縮と一言に言っても圧縮する対象 (ファイルや静止画像、音声、動画など) によって圧縮方式が異なり、また用途 (高画質や高音質、圧縮効率重視など) によっても圧縮方式が異なるので、非常にたくさんの圧縮方式が存在するのです。また圧縮には単にデータ量を減らすため別のデータに変換するというものだけでなく、バイナリデータを対象とするデータ圧縮方式の中には、複数のファイルを 1 つにまとめて扱えるようにするアーカイブ機能を兼ね備えるものもあります。

3.4　データ圧縮の原理

では、圧縮とはどのような原理でデータ量を縮めているのだろうか。基本的には元データが持つ冗長性 (繰り返し性)、法則性 (予測可能性)、不要性などの性質を上手に利用してデータを圧縮するのです。ここでは簡単な圧縮法をいくつか紹介したいと思います。

3.4.1 冗長性 (繰り返し性) を利用した圧縮法

一番単純な圧縮法である連長符号化は、コードの冗長性を利用して元データをコードの繰り返し数とそのコードという 2 バイトの符号に変換します。

例えば「あああああいいいいううう」という文字列は' あ' が 5 つ、' い' が 4 つ、' う' が 3 つの順で並んでいるので、「あ 5 い 4 う 3」と圧縮することができます。これによって 12 バイトの元データを 6 バイトに圧縮することができました (説明の都合上、コードとして日本語を用いていますが、これは 1 バイトコードと考えてください。)

また 連長符号化はコードの冗長性を利用したものですが、コードそのものではなく、コードの組み合わせパターン (語句) の冗長性を利用したものもあります。

3.4.2 法則性を利用した圧縮法

法則性を利用した圧縮法には色々な記号を一定のビット数で符号化する通常の固定長符号ではなく、記号ごとにビット長が異なる可変長符号を用い、これは頻繁に出てくる記号は短いビット数で符号化し、あまり出てこない記号は長いビット数で符号化することにより、全体としてデータを圧縮する手法などがあります。

例えば「カエルピョコピョコミピョコピョコアワセテピョコピョコムピョコピョコ」という文字列が並んでいます。この文字列の文字数は 33 個で文字の種類は 12 種です。そして各文字の出現率は以下のようになっています。

ピ：8 個 (8/33)　ョ：8 個 (8/33)　コ：8 個 (8/33)　カ：1 個 (1/33)
エ：1 個 (1/33)　ル：1 個 (1/33)　ミ：1 個 (1/33)　ア：1 個 (1/33)
ワ：1 個 (1/33)　セ：1 個 (1/33)　テ：1 個 (1/33)　ム：1 個 (1/33)

この 12 種類の記号に対して、先ほど述べたように各文字に次のような可変長符号を割り当てていきます。

ピ＝ 0　　ョ＝ 1　　コ＝ 00　　カ＝ 01　　エ＝ 10　　ル＝ 11
ミ＝ 000　ア＝ 001　ワ＝ 010　セ＝ 011　テ＝ 100　ム＝ 101

これらの符号は可変長符号ですから、連続しますと符号と符号の区切りがわからなくなってしまいます。そこで符号の前に (符号長－１) を表す 2 ビットの整数を付け加えて、符号の区切りがわかるようにします。

ピ＝ 000　　ョ＝ 001　　コ＝ 0100　　カ＝ 0101　　エ＝ 0110
ル＝ 0111　　ミ＝ 10000　ア＝ 10001　ワ＝ 10010　セ＝ 10011
テ＝ 10100　ム＝ 10101

この符号を用いて元データ「カエルピョコピョコミピョコピョコアワセテピョコピョコムピョコピョコ」を符号化しますと、次のようになります。

0101 0110 0111 000 001 0100 000 001 0100 10000 000 001 0100 000 001 0100 10001 10010 10011 10100 000 001 0100 000 001 0100 10101 000 001 0100 000 001 0100

これによって符号化したデータは 122 ビットとなり、元データ 132 ビットに対して 10 ビットほど圧縮できました。

3.4.3 不要性を利用した圧縮法

不要性を利用した圧縮法はデータ中の不要なデータあるいは重要度の低いデータを削除してしまう手法で、たいていは元データを完全には復元できない非可逆的圧縮になります。

以上で圧縮法の例の説明を終わらせますが、上記で説明をした圧縮法はほんの一例で、実際に使われている圧縮法はもっと効率のよい符号化によって上記の物より圧縮率が高いものもたくさんあります。興味があればぜひ調べてみてください。

3.5 圧縮フォーマット (形式) の種類

最後に圧縮フォーマット (形式) をいろいろと簡単に紹介していきたいと思います。

3.5.1 ファイル圧縮

LHA

拡張子は「.lzh」、「.lha」などがあり、純日本産の圧縮形式で Windows 用の圧縮形式としては、Zip 形式と並んで広く利用されている形式である。日本においては「.lzh」のみが用いられるが、日本国外では「.lha」などが用いられることもあるそうです。

RAR

拡張子は「.rar」で lzh 形式や zip 形式より圧縮率が高く、圧縮ファイルを任意のサイズの複数のファイルに分割したり、分割されたファイルを統合して解凍したりできるので、ネット上の大容量ファイル交換に広く使われいます。

ZIP

拡張子は「.zip」で歴史が長く、利用率も高いことから世界的にもっとも広く使われている圧縮方法で事実上の世界標準とされている。

3.5.2 静止画像圧縮

GIF(Graphics Interchange Format)

拡張子は「.gif」で 256 色以下の画像を扱うことができる可逆圧縮形式のファイルフォーマットです。動画を保存できるアニメーション GIF や、透明色を指定して背景イメージと重ね合わせることができるトランスペアレント GIF、全体をダウンロードしなくてもイメージの確認ができるインターレース GIF などの拡張仕様がある。

JPEG(Joint Photographic Experts Group)

拡張子は「.jpeg」、「.jpg」、「.jpe」、「.jfif」、「.jfi」、「.jif」などがあり、一般的に非可逆圧縮の画像フォーマットとして知られているが、可逆圧縮形式もサポートしている。しかし、可逆圧縮は特許などの関係でほとんど利用されていないそうです。

PNG(Portable Network Graphics)

拡張子は「.png」で 1 ピクセルあたりの情報量 (色深度) として 48 ビット (赤青緑それぞれ 16 ビットずつ) まで扱える、各ピクセル毎に透明度を指定できる可逆圧縮の画像ファイルフォーマットである。

3.5.3 音声圧縮

WMA(Windows Media Player)

拡張子は「.wma」、「.asf」で WMA は他の主要な圧縮方式と同様、人間の感じ取りにくい部分のデータを間引くことによって高い圧縮率を得る非可逆圧縮方式を採用している。音楽 CD 並みの音質を保ったまま約 1/22(64kbps) まで圧縮することが可能で、音質を犠牲にすればさらに高い圧縮率を得ることもできる。

MP3(MPEG Audio Layer-3)

拡張子は「.mp3」で MP3 は、音声データを極端な音質の劣化を伴わずに圧縮できるため、CD などの音源媒体からパーソナルコンピュータのハードディスクドライブに取り込む過程で広く普及して、現在では最も広く普及している音声圧縮方式の一つである。ちなみに音楽 CD 並の音質を保ったままデータ量を約 1/11(128kbps) に圧縮することができ、音質を犠牲にすればさらに高い圧縮率を得ることもできるそうです。

3.5.4 動画圧縮

MPEG(Moving Picture Experts Group)

画像の中の動く部分だけを検出し保存するなどしてデータを圧縮している。MPEG-1 から MPEG-4 までの各規格が定められていて MPEG-1 ではビデオ CD、MPEG-2 では放送や HDTV での使用を想定しているのに対して、MPEG-4 では低ビットレートでの使用にまで用途を拡大することを目標として規格化が開始されている。なお MPEG-3 は MPEG-2 に吸収する形でキャンセルされたそうです。

WMV(Windows Media Video)

拡張子は「.mwv」で動画圧縮標準の MPEG-4 を元に Microsoft 社が開発した動画形式。Microsoft Windows で標準対応（Windows Media Player で再生可能）であるため、PC 向けに広く普及している。またビットレートの割に画質が良く低ビットレートでも映像の破綻が少ない等の利点がある。

3.6 おわりに

以上でデータ圧縮についての基本的な説明を終わります。どうでしょう、普段よく見る拡張子だけどこういう長所があったんだと思う人もいれば、知っていることばっかりだったと思った人もいるでしょう。これを期に普段気にしていなかった圧縮形式について目を傾けていただけたら幸いです。最後まで読んでいただきありがとうございました。

4　Windows Meについて

情報工学課程　1回生　村上明男

4.1　はじめに

ある日、Windows Me を使っている時のことです。
「あれ？　動かない …」
30 分ほど放置していていると動作が停止してしまったのです。そんな Windows Me とはどのようなものなのでしょうか？

4.2　Windows Meとは？

4.2.1　Meの登場

2000 年 9 月 22 日、Microsoft 社は Windows Millenium Edition 日本語版を発売しました。Windows 98 Second Edition の後継となる Windows Me は 16bit コードを含んだ最後の 9x カーネルのＯＳとなりました。これよりも前の 2000 年 2 月 17 日には Windows 2000 が発売されています。Microsoft 社は Windows Me をインターネット、メール、ゲームを中心とした初級、中級ユーザー向けとし、Windows 2000 はビジネス向けとしました。

4.2.2　必要最低スペック

Windows Me の必要最低スペックは Pentium-150MHz+32Mbytes メモリ (**ムービーメーカー未使用時**)。一方、Windows 2000 Professional の必要最低スペックが Pentium-133MHz+32Mbytes メモリですから、Windows 2000 よりも Windows Me の方が高スペックを問われていることになります。ちなみに、Windows Me(ムービーメーカー使用時) の必要最低スペックは Pentium　-300MHz+64Mbytes メモリです。

4.2.3　Me独自の機能

- システムの復元ウィザード
- システムファイルの保護機能

- Windows ムービーメーカー
- Windows Imaging Acquisition(WIA) 対応
- インターネット接続共有やファイル/プリンタ共有を行うためのウィザード追加

当時ではムービーメーカーが有用な機能と言えます。

4.3 Me でのトラブル

4.3.1 動作が •••

Windows 98/98 SE から Me にアップグレードしたユーザーの多くが「動作が遅い」と感じたようです。皮肉にも、システムの復元ウィザードやシステムファイルの保護機能といったシステムの信頼性を向上させる機構がオーバーヘッドの原因ではないかとされているのです。

4.3.2 フリーズ

突然スクリーン全体が青くなり動作しなくなることがあります。また、複数のアプリケーションが動作している時に強制終了を求められることもあります。

4.3.3 その他

- 全角長音を含むコンピュータ名、ワークグループ名が文字化け
- スタンバイ状態から PS/2 キーボードを使ってレジュームすると、マウスが正常に動作しないことがある
- 「Windows の終了」メニューから、「スタンバイ」が消える

その他にもいろいろありますが、ここでは触れません。

4.4 原因は？

4.4.1 9x カーネル

Windows Me や Windows95/98 に搭載されているＯＳの基盤部分である 9x カーネルは、MS-DOS/Windows3.1 との互換性を維持しながらも 32bit アプリケーションを安定して動作させられるようにしたカーネルです。ここで問題となるのは、9x カーネルではシステムリソースが制限されていることです。

4.4.2 システムリソースの制限

システムリソースにはUSERリソースとGDIリソースとがあります。USERリソースはアプリケーションやダイアログ ボックス、ウインドウなどを使用したり、マウスやキーボードから入力するなど、ユーザーが行う操作などに使用されます。GDIリソースは画面上に表示するフォントやグラフィックス、アイコンの描画や画面表示の色数やサイズなどに使用されます。

Meではシステムリソースの残量をリソースメーターで確認できます。システムリソースの残量はUSERリソースとGDIリソースの残量の少ないほうを%表示し、これが0 %に達するとそれ以上アプリケーションやファイルを開くことはできなくなり、再起動する必要があります。また、16bitアプリケーションと32bitアプリケーションでは使う領域が異なります。後者の領域が6MBであるのに対し、前者の領域は384KBしかないため、主に16ビットアプリケーションを多数起動させた時にリソース不足に陥ってしまいます。また、MeはＯＳ自身がシステムリソースを多く占有してしまうのでリソース不足になりがちです。

4.4.3 16bitアプリケーション

32bitアプリケーションはそれぞれ独立したアドレス空間を持っていて、仮にクラッシュしたとしても他のアプリケーションには影響を与えることはありません。しかし、9xカーネルで16bitアプリケーションの場合は１つのアドレス空間や仮想DOSマシンを共有するので、16bitアプリケーションのどれか１つがフリーズすると他の16bitアプリケーションも停止してしまうことになります。

4.4.4 NTカーネル

Windows 2000/XPに搭載されているＯＳの基盤部分であるNTカーネルは完全に32bitで構成されていて、16bitアプリケーションにもそれぞれにアドレス空間が与えられることになりました。また、16bitアプリケーションに3MBの領域が与えられたので、システムリソース制限を気にすることもまずありません。つまり、Windows 2000はMeに比べて安定性が格段に高いと言えます。

4.5 おわりに

Windows Meは「不安定で不完全」でした。では、Windows Vistaは今後どうなるのでしょうか？

Vistaは数多くの問題を抱えています。アプリケーションの互換性やInternet Explorerの脆弱性などが指摘されています。一方で、ＵＩの刷新や検索機能・セキュリティの向上などの改善点もあります。

XPや2000と比較すると、今の段階ではVistaは未だ失敗の域から抜け出せていないのではないでしょうか。今後のMicrosoft社の動きが気になるところです。

まだまだ初心者の私の書いた文章を最後まで読んでくれてありがとうございました。

5 Xlink Kaiについて

情報工学課程　一回生　米井将二

5.1 Xlink Kaiとは？

ワイヤレス LAN 機能を持つ PSP や DS は通信協力・対戦ができますが、
機器同士が近距離に (PSP なら 10m 以内) にいないと通信が行えません。
この通信をネットを介して仮想的に接続している状況を共有し、遠隔地からサーバーを通してあたかも PSP 機器同士が近距離にあるかのようにするものです。
Xlink Kai はフリーソフトなので公式 HP(http://www.teamxlink.co.uk/?go=japanese) からダウンロードできます。

5.2 Xlink Kai導入の手順

1. 無線 LAN 機能付きの USB を用意
2. Xkink Kai の HP でアカウントを作る。
3. アカウント作るとメールがくるので、メールでアカウント確定を行います。
4. Xkink Kai Evolution　をインストール
5. いろいろと設定[1]
6. Xlink Kai 起動し、ログイン

5.3 トンネリングについて

Xlink Kai がどうやって PSP の遠距離通信を可能にしているのか？
それはトンネリングという技術を使っています。

[1]設定に関して詳しくは http://plusd.itmedia.co.jp/games/articles/0503/07/news071.html に載っています

5.4 トンネリングとは？

インターネットなどの公衆回線網上に、ある二点間を結ぶ閉じられた直結通信回線を確立することです。ネットワーク上に外部から遮断された見えない通り道を作るように見えることからトンネルと呼ばれるようになりました。

本来通信を行いたいプロトコルで記述されたパケットを、別のプロトコルの パケットで包んで[2]送り届けることにより通信を行います。パケットのカプセル化とその解除は トンネルの両端の機器[3]が自動で行うため、トンネルで結ばれた機器同士は途中の通信方式や経路を気にする必要がなく、あたかもトンネルの両端機器が直結しているように見えます。

会社の本社と支社の LAN 間接続など、プライベートなネットワークをインターネットを経由して接続する際に利用されることが多いため、実際のトンネリング機器やソフトウェアはパケットをカプセル化する際に暗号化を行うようです。

5.5 いろいろなトンネリング技術

トンネリング技術にはいろいろなものがあります。その中で一部を紹介してみようと思います。

- PPTP

  ポイント・ツー・ポイント・トンネリング・プロトコルの略称。
  マイクロソフトがアセンド・コミュニケーションズとともに設計した VPN[4](仮想施設網) プロトコルで、WINNT 4.0 で初めて提供されたそうです。

  基本的な考え方は、PPP のパケットを IP パケットに包み込んで、IP ネットワーク上のトンネルを通して運び、RAS サーバとの間で PPP 接続を張るという仕組みだそうです。リモート・アクセス VPN のために開発された技術で、マイクロソフトが中心となっているだけに、クライアント環境が充実しているのが利点。

- L2F

  シスコシステムズが開発した、ISP [5]経由のリモート・アクセス VPN 用トンネリングプロトコル。
  安定性が評価され、アメリカの ISP を中心に利用されてるようです。

- L2TP

  PPTP と L2F を基礎にルータ・ベンダなどの協力によって開発されたトンネリングプロトコル。基本的な考え方は PPTP と同じです。PPTP との違いは PPTP は IP ネットワークのみで利用可能ですが、L2FT は ATM やリレーフレーム・リレー上での利用も可能です。また、PPTP は

[2]カプセル化といいます
[3]PC やルータなどトンネリング機能を持った機器は様々です
[4]公衆回線をあたかも専用回線のように利用できるサービス
[5]インターネット接続業者

トンネル構築の際の認証確認手段を持ちませんが、L2TP はこの機能を持つようです。さらに、L2TP では一つの仮想トンネルで複数のユーザー・セッションをやりとりできます。

5.6 Xlink Kai の問題点

Xlink を起動していると急にブルースクリーンになって通信が終わってしまうという状況が頻繁にあるようです。
Team Xlink(Xlink Kai の開発者) が planetex という会社と提携して Xlink Kai βが出てるので、それをインストールすればブルースクリーン発生の可能性が下がるようですが、あくまで「下がる」なのでいつ落ちるかの恐怖にはさらされます。
また、ラグの問題、Xlink はたまにラグが恐ろしいことになるようです。(通称「鬼ラグ」) 多くの Xlink ユーザーがいる MHP では「モンスターが急にワープした」「通信してる仲間が空中に浮いていた」等の話もあります。ファイヤーウォールを無効にすれば、多少は改善されますがセキュリティ的に大丈夫なの？と心配になります。
また通信を行う前に集まるアリーナという場所があるのですが、最近はユーザーが増えすぎて回線が重くなったり酷い時はサーバークラッシュが起きているようです。さすがにこれは Team Xlink や planetex がサーバー追加するなどの対応してくれないとどうしようもありません。

5.7 最後に

ここまで僕の Xlink Kai やトンネリングについてよくわかっているのかわからないような駄文にお付き合いいただきありがとうございました。
近々、この Xlink Kai を僕の PC に導入しようと思ってたので、今回の lime のついでに調べようと思った次第です。
とりあえず、Win2000 で動いてくれることはわかったので安心。しかし、CPU が Celeron1.8GH なのでちょっと恐怖です。(Skype を起動するだけで CPU 使用率 60 %近くにいきます) まぁ、多分動いてはくれるだろうと思います。いつ落ちるかはわかりませんけど。

また,Xlink Kai に関して調べていると PSP 開発環境について知ることができました。PSPSDK というのですが、これで PSP の自作アプリを作れたりするそうです。そのうちこれで遊んでみれたら楽しいと思います。以上で lime を終了します。ありがとうございました。

5.8 参考文献

- IT 辞書:e-words(http://e-words.jp/)
- Xlink Kai 公式サイト日本語翻訳 (http://www.teamxlink.co.uk/?go=japanese)
- RBB(http://dictionary.rbbtoday.com/Details/term1827.html)

6　コピーガード

情報工学課程　１回生　米谷健吾

はじまり

　いざと言うときに備えて、CD や DVD のデータをバックアップすることは一応、著作権法でも例外的に認められています。
しかし、それを譲渡、販売、配付することは禁止されています。例えば、ＤＶＤを借りてきて、パソコンでコピーして、このデータをファイル共有ソフト[1]で交換したりするのはもちろん犯罪行為です。
デジタルの著作物は、質の劣化なしに簡単にコピーやアレンジすることが可能です。
権利者は権利を守るためにコピープロテクトを著作物に施します。
しかし、このコピープロテクトを回避するためにプロテクト破りの技術がすぐに開発されてしまいます。
実際、三つ売っていたら一つはコピー商品とまで言われるほどにコピー商品は出回っています。
このため、著作権法で定められる技術的保護手段によって、コピープロテクトを故意に外すことは違法になります。
しかしそれでもなお、プロテクト技術とプロテクト破りの技術とが「いたちごっこ」を繰り広げるという状況が今もまだ続いています。
今回はゲームに使用されるプロテクト、ProRing、SafeDisc、Alpha-ROM の三つを紹介します。
出てくる用語は各説明の「用語」で確認してください。

6.1　SafeDisc とは

6.2　内容

　米国の Macrovision 社が開発したパソコン向け CD-ROM 用プロテクト。
エラーセクタを断続的に配置して、チェックプログラムが本物の CD-ROM かどうかを見分けるため、セクタを正しく再現できていない場合は起動することができません。
これは CD を焼く場合、普通のライティングソフトは専らエラーセクタを無視するか補正してなくそ

[1] winMX,winny 等、ネットワークを通じて複数対複数で、共有されているファイルのやりとりを行う仕組みを持つソフトウェア

うとするのを利用したものです。
また ATIP [2]の情報を参照し、
それが CD-R であると判明した場合に不正ディスクとして起動不可能にした ATIP 参照型の SafeDisc も存在します。
やっていることは、CD を読み取ると SafeDisc のシステムが動き、エラーセクタを通ったかチェックします。
もし既定のセクタのアドレスにてエラーがあったら無視して次のアドレスへ向かいます。
しかしもしエラーがない、つまりエラーセクタが存在しない場合、偽物と考えて読み込みを拒絶します。
これによって不正なコピーをしても読み込むことができないようにしています。

6.3 イタチごっこ

上の説明で分かってもらえるかもしれませんが、エラーセクタの中身はほぼなんでもいいと考えてもよさそうです。[3]
そこにあればいいのですから
ならば、読み込みエラーを無視しつつ、エラーセクタを書き込むかセクタのアドレスを覚えておくようにすれば、同じものが複製可能ということになります。
もちろん SafeDisk 事態も進化し続けているので簡単にはいきません、しかし読み込みソフトとドライブの性能によっては問題にならないことも多いです。
現在最新と言われているヴァージョンの SafeDisc でも仮想化することは比較的容易であり、ドライブの種類によっては複製もできる。
SafeDisk はエラーセクタ部分をチェックするので、リードエラーを無視かつエラーセクタのアドレスを覚える事ができれば回避できるということです。
現にフリーで出ているあるソフトを使えば複製可能だというから驚きです。
SafeDisk は常に開発する側と複製する側のイタチごっことなっています。

6.4 ver

さっきも書いたようにイタチごっこがさかん（？）な SafeDisc はかなりの ver が出ています。
出てくる用語は次の「用語」で確認してください。

- ver1.0 最初のモノ、エラーセクタの存在をチェックする。回避にドライブの能力がかなりかかわってくるが誤爆も多い。やり方次第で複製可能。

[2]その CD-R メディアの製造元・メディアの種類などの情報が記載されている部分
[3]もちろんあとでウィークセクタなるものが登場しますが

- ver2.0 ver1.0 に加え変化しやすいことから書き込みが至難なウィークセクタと呼ばれるセクタをチェック。ＥＦM変調のバグを利用している。つまり読めなかったエラーセクタにプラス読めるエラーセクタを混ぜたと考えてください。

- ver2.4 フィリップス製のＬＳＩのバグを利用し SafeDisk2 より復調が難しいウィークセクタパターンを使用。また ver2.4 から ATIP によるチェックも行っている。DVD-ROM ドライブで読むとメディアチェックができず仮想化されやすい。

- ver2.7 今までのタイプと違いチェックするのがインストール時と起動時（今までは起動時のみ）になった。いまだ CD/DVD-ROM ドライブで読み込むとメディアチェックをしない。またこれ以降のヴァージョンでは daemontool が種類によっては使えなくなる。

- ver2.8 ver2.51 [4]までの機能に加えトラック１プリギャップ（LBA マイナス領域）チェックが加わり、「CloneCD」[5]を使えなくした。

- ver2.9 プレクスター、ライトン（性能の高い CD-ROM ドライブの一つ）対策用などを強化。ウィークセクタの再現も非常に困難になる。もはやバックアップが確実に可能なドライブがかなり少なくなっている。

- ver3.0 SafeDisc の DVD-ROM 対応版。DVD ± R/RW であるかどうかを調べるため DVD-ROM 化が可能なドライブが必要。

- ver3.1 ウィークセクタの再現がさらに困難になっている。日立 LG の数機種や、I-O DATA DVR-ABH8 ぐらいしか書き込みできない。

- ver3.2 複製フリーソフトである CloneCD 等がインストールされている場合、マスターディスクであっても、起動は絶対に不可能。レジストリまで完全に削除しなければならない。

とかなりの研鑽を積み上げてきている。だがしかしいまだ複製根絶には至らず、あまり強力なものをかけると誤爆する。
ver3.2 でも複製できるソフトが存在し、今だイタチごっこは続きそうです。

[4]ver2.4 のウィークセクタをより書き込みにくいものにした
[5]フリーの CD 書き込みソフト。SafeDisc 回避を行うことが容易

用語

6.5 セクタって？

ディスク状の記憶装置 (HD とか FD とか CD とか) における最小の記録単位。
円盤状の記憶装置は、円盤をまず年輪のように同心円状に区切って分割します。
このとき分割された各部分を「トラック」といいます。
さらに円盤を放射線状に区切ることによって、「トラック」をさらに分割した領域が「セクタ」です。
通常はいくつかの「セクタ」をまとめて「クラスタ」といい、「クラスタ」の単位で記憶管理をします。

6.6 エラーセクタって？

エラー（失敗した）セクタ（記憶領域）と考えれば分かりやすいかと思います。
バッドセクタ、不良セクタとも。
物理的に読めない部分（セクタ）のことです。

6.7 ウィークセクタって？

CD への情報の記録はピットとランドの境界を 1 それ以外では 0 としていますが、
CD のピックアップが認識できるピットの解像度の関係で、1 のチャネルが連続することはできず最低 0 が二回はつづかねばなりません。
なので EFM 変調（8bit のデータを 14bit のデータに変換すること）により 1 のデータが連続するパターンを、0 を含むパターンに変換して記録しなければならないのです。
この EFM 変調を通常使用されないパターンで変換することによって、多くのＣＤ－Ｒは書けない (エラーになってしまう) ようにしてプロテクトを作っています。
変換された読めないセクタをウィークセクタといいます。
複製しようとするなら特定のドライブを使用しなければなりません。

6.8 誤爆って？

きつく条件付けなどをしてプロテクトを掛けたりすると、本物までをニセモノと誤認し読み取れなくなってしまうこと。
DiscGuard [6]などは誤爆が多すぎて開発者自ら解除パッチを配布するほどでした。

[6]主要な実行ファイルの暗号化とディスクに対する電子署名の埋め込みによるプロテクト、基本バックアップは不可能といわれる

6.9 Alpha-Rom とは

6.10 内容

韓国の SETTEC 社が 2002 年 2 月に開発したパソコン向け CD-ROM 用プロテクト。
違法コピーの氾濫状況は日本の比ではないといわれる韓国で、プロテクトシェアの占有率 70 %[7]を誇り、
日本のＰＣゲームに導入された当時、あらゆるライティングソフトが敗北し、最強のプロテクトと呼ばれました。
基本的には順読みではコピー不可能な重複セクタがあり、逆読みにより本物 (マスターディスク) かを判別するプロテクトです。
Alpha-Rom の基本的特徴としては

- イメージが作成できない。
- プロテクト解除用パッチが使用できない。
- データのディスアセンブルができない。
- 誤爆率が低い

ことで、イメージが作成できないと複製はほぼ不可能、解除パッチがあてられないなら外すことがままならず、データのディスアセンブルができないと CD の解析ができません。
よって複製不可能というプロテクトでした。
読み込みの際無視されてしまう重複セクタを逆から読むことによって判別し、それが書き込まれていれば本物と見なします。
なぜ読み込めないのかは「用語」参照

6.11 イタチごっこ

たしかに入ってきた当時は最強と言われましたが、
CD Manipulator を皮切りに無理やり吸い出したり書き換えたりして、別のソフトウェアで対応され、神話はもろくも崩壊します。
誤爆率の低さに関しても一部の松下製品とパイオニア製など韓国ではあまり使われていなかったドライブが誤爆するなどして、最強とは言い難くなりました。
逆読みできるソフトも出てきたことから、回避も楽になりました。
けれども細かい改良を繰り返し、やはり強いプロテクトという地位は変わりません。
これも開発者側と複製側のイタチごっこが続いていくでしょう。

[7]SETTEC 社公表のデータ

6.12 ver

プロテクトはイタチごっこの毎日です。
が、それほど多くのヴァージョンが出ていないことからも、このプロテクトの強度がうかがえます。

- ver1.2 日本に入ってきた最初の ver、二重セクタのプロテクトがかけられている。初期版であったため、正規ディスクでも認識しない誤爆現象が見られた。
- ver2.0 ver1.2 では二重セクタだったのが三重化され、リッピングが困難になった。しかしファイル共有ソフトが出てきたことでコピーツールが出回り始めた。このことでリッピング後の扱いは容易となる。
- ver3.1 通称 Alpha-ROM DVD ともいわれる。DVD-ROM ソフトに対応する。重複セクターは三重のままだが、エディスクのエラー訂正機能が向上したこともあり、再現が難しい。Windows9x 系の OS から一部のファイルにアクセスできない不具合がある。

最強と銘打ってはいましたが、ファイル共有等に出回るコピーツールや NoCD 化 patch は強力で、daemontool を使うのならば問題なく複製品を使えることが少なくないです。
焼くことに関しても逆読みできるソフト等を使って再現してしまえることがあり、解析ツールによる回避も可能。
こちらもまだまだイタチごっこは続きそうです。

用語

6.13 重複セクタって?

各セクタには番号がふってあり、番号順に並べて記録されます。
各セクタには 2048 バイトのデータが入り、データを読むときはセクタの番号を指定して読み込むことで目的のデータを取り出します。
このとき複数回同じ番号で書き込みするとどうなるでしょうか？
例えば 1 2 3 4 4 5 6 と順番を振った場合、読み込んだ時 4 の後は 5 に飛ぼうとして、結果二個目の 4 を飛ばしてしまうことになります。
つまり普通に前から読んで言った時、二個目の 4 は認識されないのです。
仮に逆から読んだ場合、4 のあとは 3 なので今度は一個目の 4 が読まれず、二個目の 4 が読まれます。
つまり、順読み逆読みの両方をすることによってはじめて全てのデータが読み込める、といったものです。

6.14 二重セクタって？

とあるセクタ群に
1 2 3 4 5 6 4 5 6 7 8 9 番号を割り振って、かつ三つづつに
A B B' C と内容が書いてあるとします。
順に読んだ時は A B C と読んでいきます。
次に逆から読むと C B' A と読んでいきます。
後は比較すると A B B' C とデータが取り出せます。
なので、逆読みすれば吸い出しは可能です。
また、アドレスが間違っていると判断し ABB' と読んで、C を読まないドライブのために、B と B' の間にエラーセクタを挟み込むといった手法もとられています。

6.15 三重セクタって？

二重セクタに加えて
とあるセクタ群に
1 2 3 4 5 6 4 5 6 4 5 6 7 8 9 番号を割り振って、かつ三つづつに
A B B' B C と内容が書いてあるとします。
これならば逆から読んでも A B C と読むのでデータが吸い出される心配がありません。
しかし、多重セクタの全体の事として次のセクタを読むまでの時間を計られてしまうと、どこが重複しているかなどが判別されてしまうといった弱点もあります。

6.16 ProRing とは

結構廃れつつあるプロテクトです。簡単に紹介しておきましょう。

6.17 簡単に

COTRIVE(イーディーコン) 社が開発した RingPROTECH の進化系。
コンピュータソフトウェア倫理機構が公認している技術です。
RingPROTECH はリングプロテクト、バームクーヘンと呼ばれるもので、名の通りディスクの記憶面にリング状の筋が入っているのが特徴です。
この筋は無信号のセクタ（つまりはエラーセクタ）で視覚的なコピー抑制効果をねらっています。
また、エラーと認識させることでコピーガードの役割を果たしています。
ただし、無信号のセクタを無視して読み取れば、時間はかかるものの複製が簡単に出来てしまいます。
ProRing は似たようなものですが、１本設ける毎に約 40MB を消費するリング状の無信号部分を複数

配置することで仮想ドライブ対策も行われています。
またプロテクション CD-ROM と PCR(プロテクトチェックルーチン) と呼ばれる DLL の組み合わせで、プロテクション CD-ROM をキーディスクとして利用することが出来ます。
ただし、40MB 分容量が削られることや、誤動作はほとんど無いが若干コピーされやすいことがデメリットです。

6.18 まとめ

以上三種類のプロテクトを上げてみたわけですが、見ての通り完全なプロテクトにはなりません。
人が作ったものが人に破られるのは仕方ないこととはいえ、やはりイタチごっこは終わりそうにありません。
しかし、その過程が結果として技術向上に貢献していると考えれば少しは救われるような気がしないでもないです。
後はひとりひとりの意識次第。
まだまだ追いかけっこは続きそうですが。

注意として書いておくと、プロテクトを知るということは裏を返せば破れるということです。
しかし故意にプロテクトを外すことはれっきとした犯罪です。
知ることは義務を負うことです。
知識の悪用はやめましょう。

最後に、
私のような者の取るに足らない文章を最後まで読んでいただきありがとうございました。
皆様の知的欲求を少しでも満たせたなら幸いです。

7　P2P について

情報工学課程　一回生　吉田　暁彦

7.1　P2P とは？

まず P2P について簡単に説明すると、P2P とはネットワーク形態の一つで、一般的に知られているサーバというものを持たず、ネットワーク上のコンピュータに対してサーバの役割を分散させ、かつそのコンピュータはクライアントの役割もできるコンピュータの集合を指すものである。

7.2　P2P の必要性

ユーザ情報の管理や、スムーズな通信経路を確保するために通信事業者はサーバ設備や交換機を設置する。そこで安定性と冗長化のために設備を二重化したり、回線の品質を向上させるには巨額な費用が要り、多大なコストがかかる。しかしそれでもこれまでの方法だと、一旦何らかの原因で一部のサーバ設備がダウンしてしまうとネットワーク全体へ大きな影響を与えてしまう要因になる。たくさんのユーザが同時に回線を利用するなどでトラフィックが据えて、ネットワークが混雑して渋滞すると、サーバ等ではエラーが増え、データの再送によってますます遅延が発生する。この場合にはこれをどのように分散させるかがネットワークの構築においてポイントになる。そこで P2P の技術を利用すれば、このような問題を次の二点の要領で解決することができる。

・サーバ設備自体を無くし、トラフィックを分散させて通信障害の原因を無くす・それまでサーバ上で一括管理されていたデータ情報は各クライアントに分散保持させる

上記の二つの項目のうち前者は、サーバがある場合の様に個々のコンピュータ性能や回線の容量がそれほど要求されない。という利点があり、後者は、処理を比較的秘密裏で行える。隠匿性が高い。という利点がある。ここで、サーバのないネットワーク上でどのように設定やユーザ管理を行うか、それが重要な問題である。特にビジネスに利用する場合、ネットワーク上のユーザや設定を監視することは大切だ。またファイル共有ネットワークの法的問題。これはリスクとも言えるであろう。

7.3　P2P の利用

このサービスは性質上どちらかというと企業向けである。そして、このような企業向けのサービスにおける他のメリットは、冗長化であることだ。またサーバが無いので設備や管理工数の削減が期待でき、新たなネットアプリケーション事業を立ち上げる際に最小限の設備と工数で始められる。よって事業がうまくいか中っても投資した設備が無駄にならないのでリスクが低い。

7.4 P2P の欠点

しかしここまで殆どメリットばかりを述べてきたが、実際は P2P にはデメリットやリスクも多い。その代表となるのが、先ほど述べたファイル共有ネットワークの法的問題である。Ｐ２Ｐネットワークで最も一般的に共有されているファイルはｍｐ３ファイルと Ｄｉｖ Ｘコーデックを使った AVI ファイルである。このような利用の実態から、P2P ネットワーク上のファイル共有が大手のレコード会社や映画関連企業などのビジネスに対する重大な脅威となっていると考えられている。例えば、一つの例を挙げると、Napster というものがある。今回説明する Napster は現在運用されているシステムではなく、過去不正コピーの温床として音楽関連団体から圧力をかけられたシステムを指す。Napster では P2P とクライアント=サーバーモデルを使うので Hybrid-P2P モデルと言われている。まず Napster では MP3 という音楽ファイルを交換するコミュニティを作る。その通信手順を説明すると、

1) 利用者は Napster のサーバに接続する 2) 利用者は利用者が持っているコンテンツ (ここでは MP3 ファイルのこと）の情報（ファイル名やコンテンツのデータ量等）を Napster サーバに送信する。3) 利用者は自分の欲しいコンテンツのキーワードを Napster に送信し、それに合ったコンテンツ情報が利用者側に送信される 4) 利用者は受信したコンテンツ情報を元に，そのコンテンツを持っている他の利用者と直接通信を行い，コンテンツのダウンロードを行う

ここで注目されるのは Napster に渡されるのはコンテンツそのものではなく、コンテンツ情報（ファイル名等）だけということである。コンテンツ情報を Napster サーバが扱うので処理は比較的軽い。仮にサーバでコンテンツを保管したとすれば、とんでもない大きな容量のストレージ（ハードディスク）と大容量の回線が必要となる。Napster はこの原理を用いてわずか１０台程度のサーバで数千万人とも言われた利用者間のファイル共有を可能にした。クライアント=サーバモデルでは処理がサーバに集中するため、このような大人数でのファイル共有はかなり困難になるはずである。Napster が流行した原因はいろいろなことが考えられる。

・無料で音楽が交換できる・大きなコミュニティに成長したため、非常にマイナーな曲を手に入れることが可能・MP3(音楽フォーマットの一つで小さいデータ量で高品質の曲が再生可能）によって長時間の曲を送信することが可能・ADSL のような帯域の大きい高速回線が登場した

一つだけ注目する点があるとすれば、Napster の登場以前には違法な MP ３コンテンツをサーバ配信する事が流行した点である。ところが、サーバ=クライアント方式では、そのような違法コンテンツを配信するサーバはすぐに分かってしまったため、すぐに配信停止等を著作権団体から要請され、違法コンテンツ配信がそれほど規模が大きくなることが小さかった。ゆえに CD 業界に非常に大きな打撃を加える事は少なかった。しかし、無料で音楽が共有できることは CD 業界やアーティストにとって大きな脅威になっている。その一つの理由として、Napster が音楽コンテンツダウンロードの一種のポータルサイトとなってしまったからである。たとえ、サーバ=クライアント型で違法な音楽配信をしても、その配信するコンテンツ数は限られる。すなわち、ユーザが欲しいコンテンツがそのサイトにあるとは限らないのである。ところが、 Napster のように大きなコミュニティが形成されると、大抵のコンテンツはダウンロード可能になる。そのため、「とりあえず Napster に行こう」というユーザが増え、コミュニティーがさらに巨大化し、そのためにファイル共有数とコンテンツダウンロード数が激増したのである。

P2P ネットワークは潜在的な脅威として RIAA や MPAA などの業界団体によって標的にされていた。それらから攻撃を受けることによって P2P ネットワークはより攻撃を受けにくい仕組みに姿を変

えていった。そして P2P ネットワークを運営している仕組みの攻撃が困難になると、P2P ネットワークの利用者がレコード会社や映画関連企業の攻撃の対象になった。不特定多数に広く公開されるタイプの P2P ネットワークは役割を終え、ファイルを受け取る側は送る側が誰かを特定できないような、閉鎖的で暗号化されたものが取って代わると考えられている。例えば、別の秘匿性の高い接続手段として、ピュア P2P 型モデル上の各ノードがその場限りのワイヤレスネットワーク上の近隣のデバイスと接続するといった形態が考えられている。

7.5 最後に

とかく著作権侵害と結びつけて語られがちな技術であるが、技術自体が本来的に違法性や反社会性を持っているわけではなく、グループウェアの情報共有システムとして活用したり、ソフトウェアの配布やコンテンツ配信に応用したりといった「平和利用」を模索する動きも活発化しつつある。また、P2P においてファイル共有ネットワーク問題は有名で結び付けられているが、それを除いた問題としてまだいくつかデメリットやリスクがある。それは、クライアント自身がサーバの役割も果たすということは、そのコンピュータが作動していない限りそのサーバが使えない、という状況である。サーバとして使用しないコンピュータが常に稼働している確率は少ないのでサーバとしての役割が不完全である。実際、現状としては P2P の評判は良くなく、特定のサービスにしか適応されないが、上記の「平和利用」が実現すれば、飛躍的に便利なツールとなるのではないかと思われる。

P2P の説明を終えた今、皆さんも割と普段利用していることに気付くかもしれない。有名なところでスカイプなどが P2P を利用したサービスである。このように自分が使っていたものでも知らないことは多々あるもので、それを知ってみるのは面白いことだとこの本から気づいていただければ幸いである。

8 Prologプログラミングガイド

06121023 電子システム工学課程　小宮山敦史

8.1 初めに

エキサイティングな Prolog (プロログ：Programming in Logic) プログラミング言語の世界へようこそ。Prolog は、関係を定義する論理型言語です。Prolog のプログラムでは、パターンマッチによって、プログラムを実行したり、リストを処理したりします。

プログラマが記述するプログラムは、事実を定義することによって表され、それらについて質問することによって動作します。この記事では、swi-prolog [1]という Prplog を用います。他の処理系を用いる場合は適宜読み替えてください。Prolog のデータは項と呼ばれ、項は、定数、変数、構造からなっています。定数は、英大文字以外から始まる文字列またはシングルクォート (') で囲まれた文字列であるアトムか、整数か浮動小数点数です。変数は、英大文字から始まる文字列であり、構造は、関数子とその引数となる複数の項から作られます。

今、大量の専門用語が出てきて戸惑っているかもしれません。でも大丈夫です。これらの言葉を知らなくても、それらを直感的に理解できれば、プログラムを書くことができますし[2]、この記事は、それを目標としているのです。

8.2 サンプル

この記事には、細かい指示と説明がついていて、今すぐ Prolog をやってみることができます[3]。実際にプログラミングをやってみながら、基本的なルールが覚えられるようになっているのです[4]。このガイドにはあなたへの指示があります。各ステップの項目は**太字**で書かれており、それについて説明が書かれています。あなたのセクションに書かれている指示と説明を、声に出して読むようにしてください。

Prolog の仕組みは難しくないので、自分だけでプログラミングできるようになるまで時間はかからないはずです。

---

[1] http://www.swi-prolog.org/。
[2] 他人との会話では困りますが。
[3] 処理系も付いていたら、ほんとに「今すぐ」なんだけど。
[4] 大抵の入門書はそうですね。

8.3 プログラミングの準備

swi-prolog と、エディタをインストールする。

繰り返しになりますが、上段のように太字で書かれた文章は、あなたへの指示を意味します。

インストーラで拡張子を設定する項目では.swi を選択[5]します。エディタは使い慣れたものがあればそれでいいでしょう。

エディタを起動する。

保存時の拡張子は先ほど設定したものにします。こうすると、そのファイルをダブルクリックすれば swi-prolog が起動します。

プレイガイドを声に出して読みながら進める。

そこに書かれている指示に従いながら、「あなたの第一ターン」を声に出して読んでください。ルールを覚えることがサンプルプログラムの目的なので、わからないことや納得できないことがあったら、わかっている人に聞いてみてください。途中いくらかの場面で、画面がどうなっているかを表す図が示されます。その図と実際を見比べて何か間違ったことはなかったかを確認してください。

8.4 プログラミング開始

あなたの第一ターン

(何がどうなっているのか体で覚えるために、指示と説明の部分を声に出して読むようにしてください)

このターン、あなたは事実の書き方と、それをプログラムとして実行する方法を学びます。

ファイル dango.swi に、

```
dango.
club(curry).
club(chorus).
```

と記述し、保存する。

これにより、団子、カレー部、合唱部が存在することが定義されます。つまり、「.」(ドット) で 1 つの事実を区切ります。しかし、これだけでは不足です。swi-prolog にこれを教えなければなりません。

そのファイルをダブルクリックして、swi-prolog を起動する。

図??のようなウィンドウが出てきます。以後、ダブルクリックして起動することを単に起動する、ということにします。

[5].pl や.pro は不都合があるそうで、.swi がお勧めらしいです。

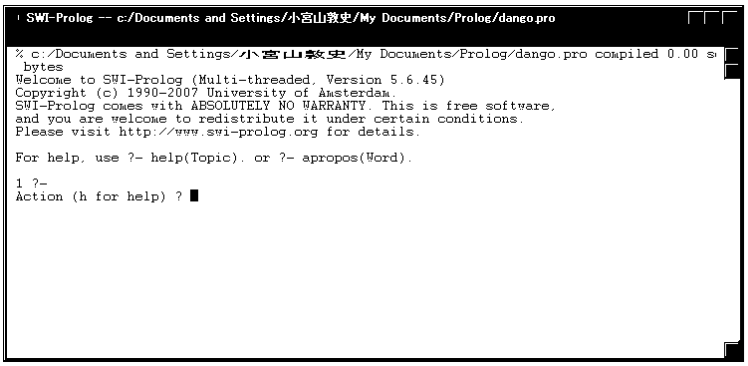

図 8.1: 起動画面

dango.[ret] と入力する。([ret] はエンターキーを押す。)

```
1 ?- dango.

Yes
2 ?-
```

このようになるはずです。これは、dango が正しいかを尋ね、prolog はそれが正しいと答えています。

club(chorus).[ret] と入力する。
今度は合唱部があるか尋ねてみます。同様に Yes と返事されるはずです。

club(play).[ret] と入力する。

```
3 ?- club(play).

No
4 ?-
```

これは「演劇部がある」という事実がないので、prolog は違うといってきたのです。

あなたの第 2 ターン

このターン、 あなたは変数を使って、複数の答えを持つ質問をすることを学びます。

club(X).[ret] と入力する。

swi-prolog では、半角英大文字から始まる文字列を変数として扱います。使い方は、何を当てはめてもよい場合に変数を用いて表します。今回は、「何部があるか」と尋ねました。ちなみに、先ほどのウィンドウを閉じてしまっていた場合、再度ダブルクリックで起動してください。

```
4 ?- club(X).

X = curry
```

カレー部があることがわかります。しかし、他にも部活はあります。それを調べるにはどうすればいいのでしょうか。

; を入力する。

解が他にあるとき、「;」[6]を入力すると prolog は別解を探します。これをバックトラックといいます。

```
4 ?- club(X).

X = curry ;

X = chorus
5 ?-
```

合唱部があることもわかりました。これでもう他の解がないので入力待ちに戻りました。
また、このときに、; の代わりに [ret] を入力すると別解を求めず、入力待ちに戻ります。

```
5 ?- club(X).

X = curry
6 ?-
```

Prolog は質問に答える場合、質問と一致する事実がないか調べます。このことを パターンマッチング といいます。このパターンマッチングによって変数の値が決まるのです。

あなたの第 3 ターン

[6]セミコロンです。念のため。

このターン、あなたは事実を追加して、パターンマッチと規則について学びます。

ファイル dango.swi に、

```
member(curry , misawa).
member(curry , kabayama).
member(chorus , sugisaka).
buin(chorus , nishina).
like(curry , X):-buin(curry , X).
like(piano , nishina).
```

を追加し、保存する。

カレー部の部員と合唱部の部員を定義し、カレー部はカレーが好きであるということにしました。ここで新たに出てきた「,」は対象を書き並べるためのもので、「:-」は、事実をまとめたもの[7]である規則を作る演算子です。

$$head : -goal_1, goal_2, ..., goal_N.$$

という形を取って、1〜N までの全ての goal が成り立てば、head も成り立つということを意味します。ここでは、「カレー部はカレーが好きである」ということに規則が使われています。

そのファイルから swi-prolog を起動する。

このとき、前のウィンドウが残ったままになります。もういらないので、消しても構いません[8]。

like(curry , misawa).[ret] と入力する。

三沢君がカレー好きだとは書いていないのですが、前述のように、カレー部員はカレー好きであり、三沢君はカレー部員なのでカレーが好きであるということになります。

この動作を調べてみます。まず、規則「like(curry , X):-member(curry , X)」の頭部「like(curry , X)」の X が misawa にセットされます。そして、「buin(curry , X)」の X は、頭部の X と一致し、misawa がセットされます。そして、「member(curry , misawa).」を調べ、成り立つのです。

最初、変数 X は値を定まっておらず、自由変数と呼ばれます。自由変数は全てのデータにパターンマッチします。

buin(Club , Person).[ret] と入力し、途中は全て; を入力する。

変数を複数使うこともできます。この場合、部員を全て求めることができます。

[7] しかし、今回はまとめず、1 つだけですが。
[8] halt. などを入力してもいいし、右上の × ボタンでもいいでしょう。

```
2 ?- member(Culb , Person).

Culb = curry,
Person = misawa ;

Culb = curry,
Person = kabayama ;

Culb = chorus,
Person = sugisaka ;

Culb = chorus,
Person = nishina
3 ?-
```

like(Z , Z).[ret] と入力する。

まず、「like(curry , X)」を発見し、Z を curry にします。これと質問により、「like(curry , curry)」を調べることになります。「member(curry , curry)」を調べます。部員に一致するものがないので失敗します。そこで、「like(piano , nishina)」を見つけ、Z を piano に変更して調べます。しかし、「like(piano , piano)」はないので失敗し、ほかに like がないので失敗となります。こうして No と返事します。

あなたの第 4 ターン

このターン、あなたは数を扱うことにより、プログラミングしている気になれるでしょう。

X is 1 - 2 + 3 * 4 ^ 5.[ret] と入力する。

X=3071 となります。swi-prolog では、四則演算+-*/のほかにも累乗を計算する^があります。そして、演算結果を変数に代入するには is を使います。このとき、数の前後の空白はなくても構いませんが、is の前後は必要です[9]。

ファイル fact.swi に、

```
fact(0 , 1).
fact(X , Rslt):-
    X > 0 , X1 is X - 1 ,
    fact(X1 , Rslt1) , Rslt is X * Rslt1.
```

[9]X とつなげば Xis という変数に、1 とつなげば is1 という定数として扱われてしまいます。

と記述し、保存する。

階乗を定義します。3 行目にある X1 is X - 1 ですが、他のプログラミング言語のように X is X - 1 とはできません。X の値はわかっているので、その値に、それより 1 小さい値を代入することはできません。そのため、新しい変数 X1 を使って続きを計算します。

ちなみに、階乗の定義は

$$\begin{cases} 0! &= 1 \\ n! &= n*(n-1)! \end{cases}$$

です。

fact(10 , X).[ret] と入力する。

10 の階乗は 3628800 です。X = 3628800 と表示され、正しい答えがでるはずです。

今度は自力でやってみよう

このプログラミングの続きは、あなたが自分だけでやってみよう。

仮に試してみる。

swi-prolog に慣れるための、いわば練習では、「これをこう書いたらどうなるの？」と試しに実行してみることはズルいことでもなんでもありません。どういう結果になるのか、また、どういうエラーになるのか、ためてみるといいでしょう。

プレイガイドを使う。

ルールを思い出すために、必要ならサンプルを見直してください。どうしても分からないことがあったら、参考文献などをあさってみてください。

それでは、楽しい prolog を！

8.5 参考文献

- 著:MakotoHiroi **お気楽 Prolog プログラミング** *2000〜2005*
  `http://www.geocities.jp/m_hiroi/prolog/index.html`
- 著:安部憲広 **Prolog プログラミング入門** *1985* 共立出版株式会社
- 著:LeonSterling and EhudShapiro **The Art of Prolog** *1994* The MIT Press

9 CPUを創る

電子システム工学課程　小長谷拓・楠健也

9.1 はじめに

きっかけは、”CPU の創りかた”という本でした。ご存知の方もおられるかもしれません。なにしろ、表紙に特徴のある？　本なので。(どうでもいい話ですが) 最初はその本を真似て、4bitCPU を作ろうと計画していました。しかし、OB の方からいろいろご指摘をいただいて、”つくるコンピュータ”という本を参考にすることになりました。この本には ATOM-8 という自作コンピュータが紹介されています。そのコンピュータは 8bit 構成なのですが、演算は 1bit ずつ 8 回に分けて行います。それを 8bit いっきに演算するように改造したようなものが、今回私たちが作ろうとしている CPU です。

9.2 仕様

ACC(アキュミュレータ) は 8bit。ALU(Arithmetic and Logic Unit) を用い、加算、減算などのほかに、AND、NOT の論理演算ができます。プログラムはスイッチ式の ROM ではなく、メモリにストアします。また、ACC、プログラムカウンタの値を LED で表示します。メモリは 62256 というものを使用します。62256 は 256KB の SRAM ですが、実際に使用するのはたったの 64B だけです。リニアなアドレス空間は 32B です。RAM なので、CPU の電源を切ってしまえば、記憶したものは全て消えてしまいます。プログラムはあらかじめスイッチで入力しておき、全ての命令をメモリに入れ終わった後プログラムを実行します。ACC、プログラム・カウンタ、instruction register,address register の内容を表示する LED もつけます。それと、どのぐらいのクロック周波数までなら正常に動くのかはわかりませんが、自作 CPU の場合、ロジック IC の周波数特性以上に CPU 回路のインダクタンス、キャパシタンスが大きくかかわってくることでしょう。

9.3 構造

今回製作する CPU は以下のような構造をしています。

メモリ メモリにプログラムや変数を書き込みます。メモリの読み出し、書き込み時に、データを入出力する端子は同じなので、データがぶつからないような工夫が必要です。

バッファ 論理関数としては、データをそのまま通すだけの部分です。つまり、$f(x) = x$　ということです。メモリからのデータと ACC からのデータがぶつからないようにするために、3state buffer

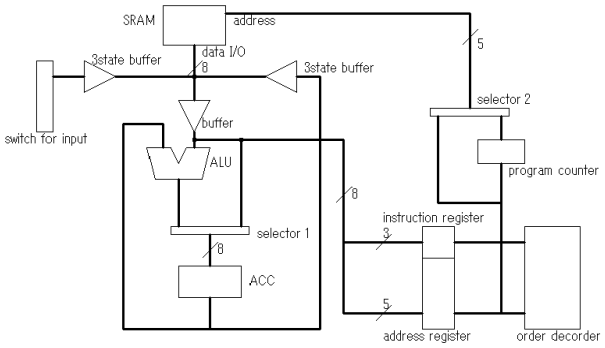

図 9.1: CPU の構造

を使用しています。3state buffer の出力は、H,L 以外にどこにもつながっていない状態 (ハイ・インピーダンス) を作ることができます。また、メモリのファンアウトが心配なので、ALU とメモリの間にバッファを挟みました。

ALU ALU で論理演算 (AND,NOT,OR など) や算術演算 (加算、減算など) をさせます。ALU には入力 A,B があり、それぞれ 8bit(4bit の ALU を 2 個使います) 分あります。つまり、A plus B,A minus B,A & B などができるということです。A しか使わない演算もあり、例えば、A plus A などができます。

セレクタ 2 つの 8bit データから 1 つを選びます。selector1 では、ALU からのデータまたはメモリからのデータのどちらを ACC に書き込むかを切り替えるものです。selector2 は address レジスタまたはプログラムカウンタのどちらのデータをメモリのアドレス端子に送るかを切り替えるものです。

ACC ALU で計算した結果を一時的に記憶しておくものです。ACC からのデータは再び ALU に送ることも、メモリに書き込むこともできます。ALU はメモリからのデータと ACC からのデータを使い、演算します。

instruction register と adress register メモリから読み出した命令、アドレスを保持しておくレジスタです。address register のデータを使って、メモリ内の変数にアクセスしたり、ジャンプします。

プログラム・カウンタ 命令はメモリの 0 番地から順番に書き込みます。プログラムを実行するとき、メモリを順番に読み出すためにプログラムカウンタによってアドレスが順次 +1 されるようになっています。

オーダ・デコーダ instruction register にはいっている命令を解析し、ALU やセレクタを制御します。

シーケンサ メモリ、ACC、プログラム・カウンタなどに順序に従ってパルスを送ります。その出力を図 2、図 3 に示します。マスタクロックはクロックジェネレータがつくったパルスです。通常は図 2 の動作を繰り返します。メモリへの書き込み時だけは図 3 の動作をします。

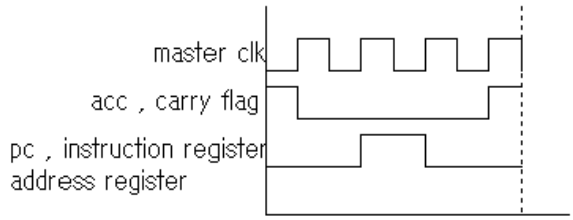

図 9.2: シーケンサの出力 1

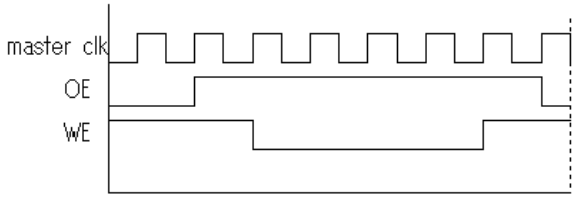

図 9.3: シーケンサの出力 2

クロック・ジェネレータ 文字通り、クロックを作り出します。できるだけきれいなクロックを作る必要があります。クロックが乱れていると、意図しないところで ACC が動作したり、メモリに変な値が書き込まれるということがあるかもしれません。

入力用スイッチ メモリにプログラムを書き込むときに使います。スイッチは 8 個必要です。ここで使うスイッチはトグルスイッチにしておきます。

9.4 使用している部品について

使用している主な部品について説明します。まず、CPU の中核、ALU には 74LS181 というロジック IC を 2 個使用しています。なかなか入手の難しい IC であるそうですが、大阪日本橋、東京秋葉原ならおそらく手に入ります。CMOS の 74HC181 はほとんど手に入らないかもしれません。この ALU は 4bit なので、2 個使って 8bit をいっきに演算できるようにしています。74LS181 のピン配置、ファンクションは図 2、図 3 のとおり。s0,s1,s2,s3,M という入力でどの演算を行うか選択できます。下位ビットからのキャリー入力、上位ビットへのキャリー出力もできます。このキャリーの入力、出力を用いて左ローテート命令を実現しています。この命令を実行するとき、ALU は A plus A という演算をしています。

メモリは 62256 という 256KB の SRAM を使用します。ACC,instruction register,address register には 74LS273 という、D フリップフロップが 8 個入った IC を使用します。プログラムカウンタ、シーケンサの一部には 74LS161 を使用します。ロジック IC やメモリには 5V の電圧が必要なので、7805 という IC を使用します。7805 は安定化された 5V を出力する、レギュレータ IC と呼ばれるものです。

入力する電圧にもよりますが、1A の電流容量を持っています。入力電圧 7V 程度から使用可能です。クロックは 74LS14(NOT) と CR で作ります。C と R の定数を変えることで、クロック周波数を変えることができます。この CPU の元のクロック周波数は 50 から 100Hz 程度に設定しました。しかし、それをカウンタで分周しているので、実際は遅くなります。また、手動クロック？ということもできます。

ところで、CMOS である 74HC シリーズと TTL の 74LS シリーズを混ぜて使うのはよくないということが、ある本に書いてありました。原因はスレッショルド電圧の違いです。74LS181 が手に入ったので、ほぼ全てのロジック IC は 74LS にしました。74HC に比べて、74LS は消費電力が非常に大きいそうで、本当は全部 74HC にしたかったのですが。74LS を多量に使うとなると、電源容量も気にしなければなりません。しかし、どうなのでしょう？　大学の実験資料には混在させてもよいという趣旨で書いてありましたが。

CPU のリセットなどを行う際に、プッシュスイッチを使います。プッシュスイッチを使う場合注意しなければならないのはチャタリングです。チャタリングとは、スイッチを押した瞬間、スイッチの ON,OFF がばたつく現象です。これを取り除くために、CR による積分回路、または、RS フリップフロップを使います。この CPU では簡単のため、積分回路を使います。

ロジック IC の周辺の電源ラインには、IC1 個または 2 個ごとにバイパスコンデンサを接続します。バイパスコンデンサには、積層セラミックコンデンサを使うのがよさそうです。これがない場合、ノイズで IC が誤作動する可能性があります。電源ラインは高い周波数においても、低インピーダンスにすることが大切です。

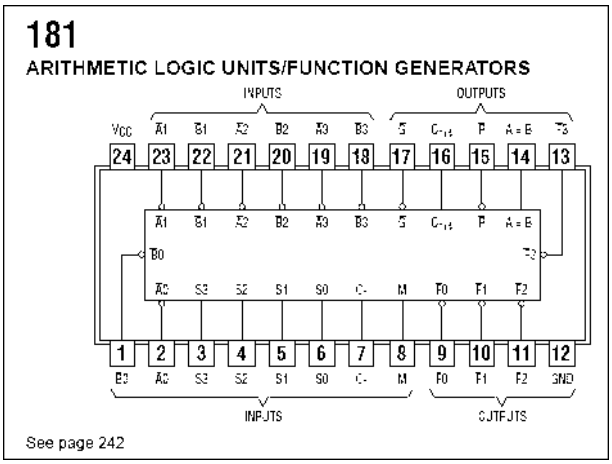

図 9.4: ALU のピン配置

9.5 命令一覧

この CPU は 2 種類の命令の形式を持っています。ALU には演算されるデータを入れるために、A,B という 2 つの入力があると書きました。演算には A,B 両方を使って演算するもの (パターン A)、例え

FUNCTION TABLE (ACTIVE HIGH)

SELECTION				ACTIVE-HIGH DATA		
				M = H	M = L; ARITHMETIC OPERATIONS	
S3	S2	S1	S0	LOGIC FUNCTION	$\overline{C}n$ = H (no carry)	$\overline{C}n$ = L (with carry)
L	L	L	L	F = A	F = A	F = A PLUS 1
L	L	L	H	F = $\overline{A+B}$	F = A + B	F = (A + B) PLUS 1
L	L	H	L	F = $\overline{AB}$	F = A + $\overline{B}$	F = (A + $\overline{B}$) PLUS 1
L	L	H	H	F = 0	F = MINUS 1 (2's COMPL)	F = 0
L	H	L	L	F = $\overline{AB}$	F = A PLUS $A\overline{B}$	F = A PLUS $A\overline{B}$ PLUS 1
L	H	L	H	F = $\overline{B}$	F = (A + B) PLUS $A\overline{B}$	F = (A + B) PLUS $A\overline{B}$ PLUS 1
L	H	H	L	F = A ⊕ B	F = A MINUS B MINUS 1	F = A MINUS B
L	H	H	H	F = $A\overline{B}$	F = $A\overline{B}$ MINUS 1	F = $A\overline{B}$
H	L	L	L	F = $\overline{A}$ + B	F = A PLUS AB	F = A PLUS AB PLUS 1
H	L	L	H	F = $\overline{A \oplus B}$	F = A PLUS B	F = A PLUS B PLUS 1
H	L	H	L	F = B	F = (A + $\overline{B}$) PLUS AB	F = (A + $\overline{B}$) PLUS AB PLUS 1
H	L	H	H	F = AB	F = AB MINUS 1	F = AB
H	H	L	L	F = 1	F = A PLUS A*	F = A PLUS A PLUS 1
H	H	L	H	F = A + $\overline{B}$	F = (A + B) PLUS A	F = (A + B) PLUS A PLUS 1
H	H	H	L	F = A + B	F = (A + $\overline{B}$) PLUS A	F = (A + $\overline{B}$) PLUS A PLUS 1
H	H	H	H	F = A	F = A MINUS 1	F = A

図 9.5: ALU のファンクション

ば　A plus B,A minus B　などと、A のデータのみ使うもの (パターン B)、例えば　A plus A,NOT A などがあります。この CPU の場合は、入力 A は ACC の値の事を入力 B はメモリから読み出してくる値のことを指します。

また、address register は 5bit、instruction register は 3bit です。

パターン A の場合、instruction register に入っている命令を解析し、ALU やセレクタを制御して address register にある値を用いてメモリから値を読み出してきます。パターン B の場合は特殊で instruction register の値が「１１１」のときと決めておき、そのときは address register に入っている値を命令として解析し ALU やセレクタを制御します。address 値を必要としない命令のときに address register の値を命令として使うことにより、使用できる命令数を増やしています。以下に命令一覧を示します。

パターン A	
Add	ACC と入力 B を加える
Sub	ACC から入力 B を引く
Read	指定したアドレスにある値を読み出す
Write	ACC の値を指定したアドレスに書き込む
Jump	carry がセットされていたら、指定したアドレスへジャンプする
Add carry	carry を加えて
AND	ACC と入力 B の論理積をとる
パターン B	
0	ACC の値を 0 にする
NOT	ACC の値の否定をとる
Rotate left	ACC の値を 1bit 左ローテートする
AplusA	ACC の値をを 2 倍する (左シフト)
Increment	ACC の値を 1 増やす
Decrement	ACC の値を 1 減らす
Set carry	carry をセットする

9.6 動作

ほとんどの場合、CPU は以下の手順を踏んで動作します。

- まず、プログラム・カウンタの番地をメモリに伝え、メモリにその番地のデータを出力させます。
- このとき、instruction register,address register にメモリの出力を記憶させます。
- そして、プログラム・カウンタの番地を 1 つ進めます。
- instruction register にある命令を解析します。
- address register の番地をメモリに伝え、メモリにその番地のデータを出力させます。ただし、上で説明した、パターン B の命令は address register の内容も命令として扱います。よって、address register の番地をメモリに伝えることはしません。
- 出力されたデータは ALU に伝えられ、計算が行われます。
- 計算結果は ACC に記憶されます。

9.7 最後に

私たちの CPU は、ユニバーサル基板上に組み立てられます。しかし、やはり配線の数が多いですね … 。このような回路を組み立てるのは初めてです。詳しくは、展示してある CPU をご覧ください (展示までには完成させたい)。

私たち、まだまだ素人ですので、わけのわからないことを言っているところが多々あったかもしれません。また、書いているうちに、かなり電子回路よりの説明が多くなってしまった気がします。そして、残念ながら回路図など、詳細まで説明することはできませんでした (もっと早くから取り掛かるべきだった？)。この原稿を書いている時点では、まだ CPU は製作途中です。このような複雑な回路は組みあがっても、一発で正確に動いてくれることは珍しいようです。製作中にもよくミスに気づいて修正することがあります。完成が楽しみです。そういえば、この CPU の名前がまだ決まっていません。

9.8 参考文献

- つくるコンピュータ　トランジスタ技術編集部　編
- CPU の創りかた　渡波　郁　著
- TEXAS INSTRUMENTS Digital Logic Pocket Data Book

10　ほのかとひよりの教えてHTML

情報工学課程　東川　知生

10.1　はじめに

webページを作成しようと思い立ち、webページ作成法に関するものを調べていると、よく「HTML」「XHTML」「CSS」といった単語を目にすると思います。そこで、今回は、webページの作成がよく分からないと言う人の為に、web作成の上で、よく聞くor目にする単語の簡単な説明をしようと思います。

10.2　「HTML」と「XHTML」

まずは、「HTML」と「XHTML」の説明をしたいと思います。

「HTML」と「XHTML」の根底は同じものです。どちらもwebページ用の文書を記述するための言語です。一番の特色は作成した文書から文書へとリンクする機能がついていることです。これを利用することで、インターネット上ではページからページへとジャンプしていき、色々なサイトに行けるのです。

では、なぜ「XHTML」という似たものが出てきたのでしょうか？その背景には、技術の進歩が隠されています。「HTML」では情報を扱う言語としての限界が指摘されるようになったのです。そこで、時代に合った規格の「XML」という**拡張性・汎用性**に優れた言語を元に「HTML」に改変を加え「XHTML」にしたのです。

ただ、単に**拡張性・汎用性**と言っても、分りにくいので例を挙げます。

- アプリケーションの**データ形式を問わずに**データのやり取りを行える。
- 必要なデータだけを取り出して**再利用**、また、**新しい要素を定義**してデータを処理する。

つまり、「XHTML」は「HTML」よりも扱える範囲が広くなったと認識してもらえればいいと思います。

10.3 「CSS」

次に、「CSS」の説明をしたいと思います。先ほどの「HTML」では触れなかったのですが、「HTML」はあくまでも**文書の構造を示す言語**です。時代が進むにつれて、「HTML」の機能を用いて見栄えを**無理に**装飾するようになりました。（現在、「HTML」の仕様を協議決定する団体は、そのような見栄えを装飾する機能を「HTML」で記述することを**非推奨**としています。

そこで、登場するのが文書から分離した「見栄えを指定する部分」を定義する概念です。それを**スタイルシート**と言います。「CSS」はそのスタイルシートの一種です。そして、一般的に「HTML」文書や「XHTML」文書に適応させる場合に使われる方法です。

ここから「CSS」本体の説明からは少し外れますが、「CSS」の各要素が持っている**「ボックス」**という四角の領域について説明していきたいと思います。

ボックスには内側から順に**「内容領域」「パディング」「ボーダー」「マージン」**から構成されています。以下は、**「ボックス」**を構成する各部分の説明です。ついでに、**「背景色」**と**「背景画像」**にも触れています。

- **「内容領域」「背景色」**や**背景画像**はこの部分にも適用されます。
- **「パディング」「内容領域」**と**「ボーダー」**の間の余白域です。**「背景色」**や**背景画像**はこの部分にも適用されます。
- **「ボーダー」**要素の周りに表示される枠線で、**「パディング」**の外側に設定されます。**「背景色」**や**背景画像**はこの部分にも適用されます。
- **「マージン」「ボーダー」**の外側に設定される余白域です。**「背景色」**や**背景画像**はこの部分には**適用されません**。

また、イメージとしては、**「ボックス」**という土台に**「背景色」**が乗っかって、順に**「背景画像」「ボーダー」「内容領域」**が乗っかると捉えてください。つまり背景色に「黄色」を、背景画像を「海の絵」に設定すると その要素のボックスには**「海の絵」**が表示されます。（つまり、**「黄色」**は上塗りされる。）

10.4 「HTML」の基本タグ

ここでは、全部を紹介することは出来ませんが、一番基本となりそうなタグ (要素) を紹介したいと思います。**「文書の構造定義」「見出し／段落／改行の設定」「リンクの設定」「リストの表示」「画像の表示」「テーブルの表示」**の６つです。一般的な各ブラウザ（Internet Explorer、Firefox、Netscape、Safari、Opera）で反映されるタグを選びました。つまり、独自のブラウザでのみ拡張されているものは含んでおらず、どのブラウザでも反映されるものです。）最後に、文書宣言にも触れます。尚、記述方式は《XHTML の場合》で書きます。理由は、この書式だと「HTML」で書いた場合でも問題が無いからです。（「HTML」で出来ても、「XHTML」では出来ない書法が存在します。）

「文書の構造定義」

〈html〉～〈/html〉…文書の最初と最後に定義。

〈head〉～〈/head〉…文書に関連する情報を定義。ここで書いた内容はブラウザには表示されません。

〈title〉～〈/title〉…ブラウザの最上部のタイトルを定義。これは、〈head〉タグに書きますが、ブラウザに表示されます。
〈body〉～〈/body〉…実際にブラウザに表示される内容。

「見出し／段落／改行の設定」
〈h ●〉～〈/h ●〉…※●には 1～6 の数字のどれかを入れる。文字フォントが数字の大きさを表します。１が一番大きく、６が一番小さいです。
〈p〉～〈/p〉…タグで囲った内容を１段落とする。
〈br/〉…このタグで改行する。
「リンクの設定」
〈a href= ”☆ ”〉～〈/a〉…※☆にはそのリンク先の URL を入れます。
「リストの表示」
〈ul〉〈li〉～〈/li〉〈/ul〉…〈ul〉タグで囲んだ箇所が順不同のリストになります。〈li〉タグでリストの項目を表示します。
「画像の表示」
〈img src= ”☆ ” alt= ”■ ”/〉…※☆には画像の URL を入れます。■には画像が何らかの障害で表示されなかった時の代わりのテキストを書き込みます。
「テーブルの表示」
〈table border= ”● ”〉～〈/table〉…※●にはピクセル数を記述。囲んだ範囲が表であることを定義します。
〈tr〉～〈/tr〉…囲んだ範囲が横一列の行を定義します。
〈td〉～〈/td〉…囲んだ範囲がセルを定義します。
「XHTML」文書宣言について…「XHTML」文書を宣言する場合には、「XML」宣言と名前空間と言語コードを設定しないといけません。これでは具体的に何を書けばいいかが分らないので、下に例を出しますので、その通りに記述してもらえれば、文書宣言は出来ます。
（例）
〈?￥xml version= ”1.0 ” encording= ”Shift_JIS ”?〉
〈!DOCTYPE html PUBLIC “ -//W3C//DTD XHTML 1.0 Frameset//EN ”
”http://www.w3.org/TR/xhtml1/DTD/xhtml1-frameset.dtd ”〉
〈html xmlns= ”http://www.w3.org/1999/xhtml ” xml:lang= ”ja ” lang= ”ja ”〉

10.5 おわりに

色々と説明の至らぬところがあると思いますが、web 作成のイメージだけでも掴んでいただければ幸いです。ここまで読んでくださって有難う御座います。

※参考書物

- 「HTML タグ辞典第６版 XHTML 対応」著：株式会社アンク
- 「スタイルシート辞典第４版」著：株式会社アンク

11　はじめての構文解析器生成系

情報工学課程 2 回生　湯浅信吾

prologue

　終わりまで残り 65 規則。
そこで立ち尽くす。
「はぁ」
ため息と共に空を仰ぐ。
その先に C の文法の終わりがあった。
誰が好んで、こんな文法を考えたのか。
妙に複雑な規則が、悪魔のように伸びていた。
「はぁ…」
別のため息。俺のよりかは小さく、短かった。
隣を見てみる。
そこに同じように立ち尽くす女生徒が居た。
「C 言語は、好きですか」
いや、俺に訊いているのではなかった。
「わたしはとってもとっても好きです。
　でも、構文解析器を作るには…多すぎるんです。
　生成規則とか、production rule とか、ぜんぶ。
　…ぜんぶ、多すぎるんです。」
たどたどしく、ひとり言を続ける。
「それでも、この言語が好きでいられますか」
「わたしは…」
「作ればいいだろ」
「えっ…？」
少女が驚いて、俺の顔を見る。
「構文解析器生成系とか、parser generator とかを
　作ればいいだけだろ。あんたにとっての構文解析器は
　手書きだけなのか？　違うだろ」
そう。
何も知らなかった無垢な頃。
誰にでもある。
「ほら、作ろうぜ」

俺たちは作り始める。
　　へぼい、へぼい構文解析器生成系を

11.1　はじめに

　これは「坂道の下で出会った少年が少女が、逆境を乗り越えて構文解析器を完成させる、発売日がやたらと伸びる物語」ではありません。タイトルに『はじめての』がついているからといって、アレな趣味 の人向けな文章

でもありません。

これはコンパイラの講義だけ受けて、コンパイラを作り始め、気がつけば構文解析器生成系だけになり、面白い方がいいという大義名分の下、Common Lisp のマクロとして構文解析器生成系を作った記録を書いたものです。本物の構文解析を勉強したい人は内容を信じないように。この文章を鵜呑みにして人前で恥をかいても当方は一切責任を負いません。

11.2 概要

11.2.1 構文解析器

構文解析とは、文がどのような生成規則から生成されたか調べることです。その構文解析を行うプログラムを**構文解析器**といいます。この説明から分かるとおり、あまり詳しい説明を書くつもりはありません (笑) 非常に長くなりそうなので正直、書いてられません。

11.2.2 構文解析器生成系

構文解析器は全て人が作ろうとすると非常に手間が掛かるので、人は言語の文法だけを入力し、それを元にプログラムに構文解析器を作らせます。そのプログラムを**構文解析器生成系**といいます。

例えば、構文解析器生成系の一つ **yacc** は、文法を記述した**ファイルを読み込み**、構文解析を行う C 言語の関数を記述した**ファイルを生成**します。下は、yacc への入力の例です。

```
%{
  #include <stdio.h>
%}
%union {
  double val;
}
%token <val> NUM
%type <val> expr
%%
expr: NUM
     | expr '+' expr { $$ = $1 + $3; }
     | expr '-' expr { $$ = $1 - $3; }
     | expr '*' expr { $$ = $1 * $3; }
     | expr '/' expr { $$ = $1 / $3; }
     ;
%%
```

11.2.3 開発に至った経緯

始め、C のコンパイラを作ろうと思ったんです。ただし、yacc などの既成の構文解析器生成系は使用せずに、上向き構文解析を行う構文解析器を手書きで作ろうと考えました。

しかし、すぐに量が多すぎて無理だと気づき、構文解析器生成系から自分で書こうと思い直しました。

yacc と同じような構文解析器生成系を作ろうとすると、読み込ませる文法ファイルの構文解析を行う構文解析器を作る必要があります。つまり、**「構文解析器を生成する構文解析器生成系を作るために構文解析器を作る必要がある」**のです。そもそも、構文解析器を作りたいのに、構文解析器を自分で書くのはいやだろうと思い、これは避けることにしました。

どうにか楽をできないかと考え、思いついた手段が Lisp でした。文法を **S 式として記述**し、read すれば、煩雑な処理を書く必要がなく、また、構文解析器を生成する処理も Lisp のマクロとして記述すれば、ファイルを出力する必要がなく、構文解析器を使用するソースを**直接コンパイルするだけ**で済みます。

つまり、次のように書くと、関数 parser が構文解析を行うようなものを考えたんです。

```
(setf (symbol-function 'parser)
      (make-parser
        (E (E '+ T) (+ $1 $3)
           (T) ($1))
        (T (T '* F) (* $1 $3)
           (F) ($1))
        (F ('num) ($1))))
```

自分で言うのもなんですが、結構かっこいいと思いました。けど、似たようなもの[1]を既に何処かの外人さんが作ってました。正直ショックでしたが、二番煎じであろうとも自分で書くことにしました。

11.3 LR 構文解析

yacc は出力として LALR 構文解析を行う関数を吐くとのことだったので、私もそれに対抗して LALR 構文解析を行う関数を吐くプログラムを書くことにしました。

LALR 構文解析は Kunuth 先生によって提案された LR 文法に制限を加えた LALR 文法に対する構文解析です。まず、LALR 構文解析の前に、その基礎となる LR(1) 構文解析の説明をします。

11.3.1 概略

LR 構文解析は原始プログラムを左から右へ、字句単位で走査し、構文解析を行います。

A → xyz

という生成規則があり、x を読み終え、これから y を読み込もうとしているという状態を

A → x·yz

と表します。さらに、このとき z の後に読み込まれる文字が k だと分かっていると、これを

[A → x·yz, k]

と表し、これを **LR(1) 項**と呼ぶことにします。しかしこのとき、y が非終端記号 B であり、

B → uvw

という生成規則があれば、これから y を読み込もうとしている状態であると同時に、これから u を読み込もうとしている状態であるとも考えられます。すなわち、

[A → x·yz, k]
[B → ·uvw, z]
(ただし、z は終端記号とする)

この二つをあわせたものを一つの状態と考える必要があります。複数の LR(1) 項の集合から成り、一つの状態を表すものを **LR(1) 状態** (LR(1) 集成) と呼ぶことにします。LR(1) 状態を求めるには、後述する LR(1) 項の閉包を求めます。

また、[A→xyz·, k] という項は、k が読まれたときに、既に読み込んだ xyz を A に還元することを表す状態です。これを還元状態と呼び、それをはっきりと区別するために記号「#」を最後につけることにします。すなわち、

[A → xyz·#, k]

のように表します。

[1] CL-Yacc http://www.pps.jussieu.fr/ jch/software/cl-yacc/

11.3.2 LR(1) 項の閉包

LR(1) 項の集合 I の閉包 (closure(I)) を次のように定義します。

1. closure(I)=I
2. [A→α·Bβ, a]∈closure(I) に対して、B→γ という生成規則があれば、b∈First(βa) なる全ての b について、[B→·γ, b] を closure(I) に加える
3. 上記 2 を新たに加えるものがなくなるまで繰り返す

ただし、First(A) は A の先頭の終端記号になりうるものの集合です。

先ほど言ったように、この閉包が LR 構文解析における一つの状態と考えられ、これを LR(1) 状態と呼びます。

11.3.3 状態の遷移

LR(1) 項 [A→α·Xβ, a] を含む状態で、X を読み込んだとき、[A→αX·β, a] を含む状態に遷移すると考えられます。状態 I で記号 X を読み込むことによって遷移する先を GOTO(I, X) と表し、GOTO を次のように定義します。

$$GOTO(I, X) = closure([A \to \alpha X \cdot \beta, a] \mid [A \to \alpha \cdot X\beta, a] \in I)$$

文法 G=(V, T, P, S) について、取りうる状態を全て求めるには、入力の終端を表す記号「$」を導入して、$G' = (V, T \cup \{\$\}, P \cup \{S' \to S\$\}, S')$ を考え、次のアルゴリズムを実行します。

1. C={closure({$[S' \to \cdot S, \$]$})}
2. 任意の I∈C と X∈V に対して、GOTO(I, X) が空集合でなく、C に入っていなければ、それを C に加える
3. 新たな集合が C に付け加えられなくなるまで上記 2 を繰り返す

これにより、全ての状態及び、遷移を求めることができます。

11.3.4 構文解析表の作成

ある状態 I で、入力 X を受け取ったときにとる動作を表にして記述したものを**構文解析表**といいます。この表が出来上がれば、それをプログラムに直すのは比較的簡単です。

スタックの先頭の状態が I_j で、次の入力記号が a であるときの動作を、A[I_j, a] で表し、入力と遷移先の状態 (I_j とする) をスタックに積む (シフトする) ことを sj と表すと、LR(1) 構文解析表は次のように作れます。

1. 初期状態は $S' \to \cdot S\$$ を含む状態とする
2. GOTO(I_i, a) = I_j ならば、A[I_i, a] = sj とする
3. $[A \to \alpha \cdot \#, a] \in I_i$ ならば A[I_i, a] = {$A \to \alpha$ で還元 } とする
4. $S' \to S \cdot \$ \in I_i$ なら A[I_i, a] = { 受理 } とする

LR(1) 構文解析は非常に広い範囲の言語の構文解析を行うことができますが、構文解析表が大きくなってしまう (=プログラムが大きくなってしまう) という欠点を持っています。

11.4 LALR 構文解析

LR(1) 項 $[A \to x \cdot y, a]$ の、a を先読み符号、$A \to x \cdot y$ を核といいます。二つの LR(1) 状態 I_i と Ij があり、I_i に含まれる項の核の集合と、I_j に含まれる項の核の集合とが等しいとき、I_i と I_j を合併し、一つの状態にしたものを **LALR(1)** 状態といいます。合併できない LR(1) 状態は、それがそのまま LALR(1) 状態となります。

LALR(1) 構文解析は LR(1) 構文解析より扱える言語の幅が狭いですが、LR(1) 構文解析に比べ、構文解析表が小さくなるという利点があります。

11.5 LALR 構文解析の実装

さて、長い長い前置きを終えてここからようやく具体的な作り方を説明していきます。これは大半が**思いつき**で作ったものなので、決して効率がよいものではないのでご了承ください。

11.5.1 LALR 項の表現

LALR 項 $[A \to x \cdot yz, k]$ を表現する場合を考えます。
核で使用している構文規則を

```
(A x y z)
```

というリストとして表現します。終端記号はクォートし、非終端記号と区別します[2]。単純に、構文規則の頭部、本体部を並べたリストです。
核はこのリストに対する参照で、上の場合、

```
(y z)
```

と表現します。これは構文規則を表すリストに対する参照であり、そのコピーではありません[3]。記号を読み進める (”·”を右に進める) ことは、リストの CDR を取ることに対応します。
先読み記号の集合[4]は、ただのリストとして表現します。上の場合は、

```
(k)
```

となります。
これに還元するときの処理も加え[5]、

```
((構文規則 還元時の処理) 核 先読み記号の集合)
```

というリストで LALR 項を表現します。

11.5.2 First 集合

まず、First(X) を求めることを考えます。この時点で構文規則は上で定めた形式で持っているものとします。最初、これは簡単だろうと思って 30 分ほどで作ったんですが、出来上がったものは、$E \to E + T$ のように再帰的な定義があると見事に無限ループに陥るという悲しいものでした。
この失敗を乗り越えて作ったものは、最初予想していたよりかなり複雑で、最初に書いたソースの 2 倍以上の長さとなってしまいました。
First(X) の X は記号の列ですが、まず X が単一の非終端記号のものを全て求め、記憶しておきます。単一の非終端記号 X に対する First(X) を求めるアルゴリズムの概要は次の通りです。

1. rule := 全規則のリスト, heads := rule から頭部を集めたリスト とする
2. rule の中から $A \to a\beta$(a は終端記号) と $A \to \varepsilon$ を見つけ、First(A) に a、ε をセット
3. 上記 2 の条件に当てはまらなかった規則を集めたリストを作り、それを改めて rule とし、heads も再設定する
4. $A_0 \ldots A_n$ が heads に含まれず、$B \to A_0 \ldots An\alpha\beta$ があり、$\forall i(\varepsilon \in First(A_i))$ かつ α が終端記号、もしくは $\exists i(\varepsilon \notin First(A_i))$ ならば First(B) に要素をセット[6]
5. 規則が一つでも見つかれば 3 へ
6. 規則が $B \to \ldots B \ldots$ であるとき、B を A_{n+1} として (heads に含まれないものと扱い)4 を試す
7. 規則が一つでも見つかれば 3 へ
8. エラー (文法が不正)

rule が空になれば終了で、C が求める状態です。

[2]つまりこの例の場合、x, y, z は全て非終端記号ということになります。
[3]コピーを使わずに実態に対する参照にすることにより、d 核を eq で比較することができます。
[4]同じ核を持ち、異なる先読み記号を持つ項は先読み記号を合併し、一つの項にします。
[5]今思えば、全ての項に還元時の処理を持たせる必要はなかったのですが、構文規則と還元時の処理を常に組として扱っていたのでこの様な仕様となりました。
[6]具体的に書くと長くなるので省略させていただきます。

11.5.3 閉包

次に closure(I) を求めることを考えてみます。この閉包を求める処理が、かなり速度を左右するとは思うのですが、全く速度を考えずに作ったので、無駄が多いかもしれません。しかし、大事なのは構文解析器を生成する速さではなく、生成された構文解析器の速さなんです[7]！

closure(I) は同じ I に対して同じ状態を返す必要があるので、I と closure(I) の組を保持しておく必要があります。私は連想リストとして保持しました。また、既に求めた状態と同じ核を持ち、先読み記号が異なる場合は、既にある状態に先読み記号を加える必要があります。

closure のアルゴリズムを簡単に書くと

1. stack := LALR 項の集合 (引数), C := 空リスト とする
2. stack から 1 つ項を取り出し、**既に C に入っていないか確認**
3. C に入っていなければ項を C に追加
4. 項の核の先頭 (”·”の次の文字) が非終端文字なら、**追加すべき項の集合**を求め stack に積む

stack が空になれば終了です。

項が既に C に入っていないか確認するアルゴリズムは

1. C に同じ核が入っていなければ、まだ C に入っていない
2. 同じ核を持つ項の先読み記号に含まれない先読み記号を持っていれば、まだ C に入っていない
3. 上記 2 条件を満たせなければすでに C に入っている

というものです。

追加すべき項の集合を求めるアルゴリズムは

1. 項を $[A \to \alpha \cdot B\beta, a]$ とする
2. b := First(βa) を求める
3. 文法から B を頭部に持つ規則を全て見つけてリストにつなぐ
4. 3 で作ったリストを元に、先読み記号 b を持つ LALR 項の集合 (=リスト) を作り、それを返す

簡単に closure(I) を求めるアルゴリズムを書きましたが、実際は LALR 項は先読み記号の**集合**を持つので、もう少し複雑なものになります。

11.5.4 GOTO

次は、GOTO(I, X) を求めるアルゴリズムを考えますが、楽をするため、GOTO(I) とすると、I からたどり着くことのできる全ての遷移を求め、この時点で構文解析表の遷移の箇所 (GOTO 表) を作ってしまう。

各状態は記号をキーとする遷移のための連想リスト、還元のための連想リストを持ち[8]、また、これらの連想リストも状態をキーとする連想リストから取り出せるようにします。

構文解析表を求める GOTO(I) のアルゴリズムは以下の通りです。

1. I から項を一つ取り出す
2. 項の核が $A \to \beta \cdot \#$ なら、1 に戻る
3. 項の核が $A \to \alpha \cdot \beta\gamma$ なら、同じ形の項 (”·”の次が β の項) を集めたリストを J とし、それ以外の項を改めて I とする
4. J の各項の”·”を 1 つ進め、 $target_\beta = closure(J)$ とする
5. (元の形の)I の遷移のための連想リストに、(β $target_\beta$) を加える
6. GOTO($target_\beta$) を求める
7. I が空なら終了。そうでなければ 1 に戻る

closure 同様に、GOTO も同じ引数に対しては同じ値を返す (2 回同じ計算をしない) 必要があります。

[7] ただし、それも速いようには見えませんが (笑)

[8] この連想リストが構文解析表の行に相当します。

11.5.5 還元

次に、遷移の箇所が出来上がった構文解析表に還元を加えます。既に全ての状態が求まっているので還元を見つけるためのアルゴリズムは非常に簡単です。全ての状態 I に対して、reduce(I) を呼び出します。

reduce(I) の定義は以下の通りです。

1. I から項を一つ取り出す
2. 項が $[A \to \beta \cdot \#, a]$ でなければ、1 に戻る
3. I の還元のための連想リストに (a (構文規則 還元時の処理)) を加える
4. I が空なら終了。そうでなければ 1 に戻る

実際は、項が持つのは先読み記号の集合なのでもう少し複雑です。

これで構文解析表が出来上がりました。

11.5.6 コード生成

さて、最後に構文解析表を元にコードを吐き出します。吐き出すコードは次のような形をします。

```
(lambda (input)
  (let ((stack 初期状態) (head (pop input)))
    (labels (iter (state)
                  (if 終了条件
                      戻り値
                      (case state
                        ((状態 1)
                         (case head
                           ((記号 1) シフト or 還元)
                           ((記号 2) シフト or 還元)
                           ...
                           (otherwise (error "Parse Error!"))))
                        ((状態 2)
                          ...)))))))
```

非常に読みづらいですが、生成されたコードを人間が読む必要はないので問題ないでしょう。実際はさらに、input や stack というアトムも gensym が吐き出す意味のない名前になります。

処理の概要としては、まずスタックの先頭の状態で場合分けし、その後、入力の先頭の記号で処理を決めます。記号に対する処理は、シフトか還元の二種類だけです。

シフトの場合、遷移先を取り出し、スタックに入力と遷移先の状態を積み、引数に遷移先を付けて再帰します。還元の場合、let で $\$1 \ldots \$n$ という変数にスタックに詰まれた値を束縛し、構文規則とセットにしていた**還元時の処理**をそのまま吐きます。

こんなもので動いていいのかと思いきや、動いてしまいました。世の中は不思議で一杯です。

11.6 構文解析器生成系は完成したものの……

完成した構文解析器生成系に簡単な四則演算を行う文法 (冒頭に示したものです) でうまく動くのを確認したら、B.W. カーニハン/D.M. リッチー著「プログラミング言語 C」(共立出版) の付録 A に載っている、C 言語の文法を試してみました。

結果は……まあ、最終的には多分[9]うまくいったんですが、想像以上に面倒でした。何が面倒かといったら、まず全ての構文規則を入力するという作業が物凄く大変でした。Prologue に書いてあるとおりです。今まで C 言語の文法は単純だと思っていましたが、もっと単純でいいとおもいました。ようやく入力が終わったと思ったら、文

[9]字句解析器を作ってないので、テストデータを幾つか直接入れただけなので、本当に正しいかは分かりません。

法にタイプミスがあったため、うまく動かなかったり、微妙に間違っていてもある程度は動いてしまうので、一つのミスを見つけるのに数時間を費やしてしまいました。
　なによりも、ここで私は力尽きてしまったため、最初は**コンパイラを作る**という目標だったのに、構文解析器生成系で満足してしまいました。なんてこった。今度、またコンパイラを作ろうと思ったときは素直に yacc を使おうかと思ってしまいました (笑)

epilogue

「もし…
　構文解析器生成系に人と同じように、意志や心があるとして…
　そして、それを使う人たちが構文解析器を作らせようって…
　そんな思いで、いるとしたら…
　私の構文解析器が完成したという奇跡も、そんな構文解析器生成系のしわざかもしれないです」
いや…奇跡はこれからたくさん起こるのだろう。
そんな気がしていた。
「でも、それは奇跡じゃないですよね
　構文解析器生成系を大好きな人が、構文解析器生成系を作り…
　構文解析器を好きな人が、構文解析器生成系に構文解析器を作らす…
　そんな、誰にでもある感情から生まれるものです
　私の構文解析器だけじゃないです
　どんな構文解析器だって、そうです
　わたしたちは構文解析器を愛して、構文解析器生成系に育まれてるんです
　そう思います」
「なぁ…
　構文解析器生成系は、大きな家族か」
「はい。構文解析器生成系も構文解析器も人も、みんな家族です」
「そっか…」
「だんご大家族です」
ああ…そうか。
そうだったのか。

ずっとずっと歌っていよう。
今日からの思い出を…
この構文解析器と…
この構文解析器生成系のために。

　―完―

編集後記

lime Vol.36 はいかがでしたか？ コンピュータ部の部員はこのようなことに興味を持ち、日々勉強しています。lime はコンピュータ部の部員それぞれが、興味をもったことについて調べたり、作ったことを記したものと申し上げればよいのでしょうか。私も、これから部員の原稿をじっくり読んでみようと思います。

私は、lime の編集があり、松ヶ崎祭の展示物の準備もあり、大変でした。実は、この文章を書いている時点では、私の作品の一部は完成率 0 パーセントのものがあります。

来年の lime をご期待ください。

平成 19 年 11 月 21 日
編集担当　小長谷　拓