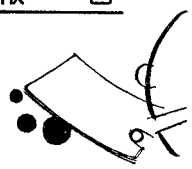
報　告



パネル討論会

Common Lisp†

パネリスト

井田　昌之[1]，竹内　郁雄[2]，萩谷　昌巳[3]
安村　通晃[4]，湯浅　太一[3]，**司会**　寺島　元章[5]

司会　Lisp の標準化を模索する試みとして Common Lisp[1] が登場してから 5 年余になります．その言語仕様も定まりつつあり，処理系もいくつか登場しているので，記号処理研究会としてもこの言語のもつさまざまな側面について議論しようということになり，ここに，Common Lisp に深い関心のある方や日本での処理系の作成に携わった方に集まっていただいたというわけです．

それでは，最初に自己紹介を兼ねて「私と Common Lisp との関係」ということを各パネラに述べていただいてから討論に入っていきたいと思います．

井田　私はだいたい 3 つぐらいの立場で Common Lisp と関係していることになります．1 つが本の訳者という立場です．もう 1 つが電子協（JEIDA）の Common Lisp 動向調査委員会の委員長をやっております．それから 3 番目が製作者という立場です．翻訳が大変忙しく，今，自分のものを作るのが遅れておりまして，処理系の製作者としての立場が弱いです．そういう意味から，最初の 2 つの立場で私はこの席に出て来ているというのが自分の気持ちです．

Common Lisp に対する期待が高まっているのは，やはり共通仕様がほしいという自然発生的なニーズが最初にあげられると思います．それにある意味で対応した形で，Guy Steele らが精力的に，共通性を持てる方言を，まず作ろうということで．いくつかのゴールを設定して，もしある程度いけるのなら，これでパソコンからスーパコンピュータのような大きいものまでいける仕様にしてしまおう，というようなイメージがあったと思います．

技術的なテーマとして少しポイントを拾いますと，Common Lisp は多重環境をまず考えていないということを明記しています．この理由はいろいろ考えられると思います．仕様内として多重環境を考えると，束縛のしかたの違いなどでハンディが処理系によってまたは技法によって出てしまう，ということがあり得ると思うんです．コルーチン，マルチプロセス，stack group といったものは，トピックスから外しています．

それから，closure の処理というのが，Common Lisp の，ある意味ではいやらしいところであり，ある意味では洗練されたところになるわけです．環境を閉じ込めるために，変数だけではなくて，たとえばタグの飛び先であるとか function の local な definition であるとか block 名とか，そういったものを閉じ込めるようになっている．インタプリタで作る場合には，それぞれをリストにして保持する，末尾にくっつけるとかいろいろな技法が出て来て，それがいやらしさといわれるところの 1 因になっているのではないかと思います．

binding のことですが，インタプリタでいう場合の shallow とか deep という問題を超えて，従来からコンパイラにあった束縛法と，インタプリティブな環境との擦り合わせっていうのかな，それに対する 1 つの解決を，コンパイラに depend した形でするということで，試みていると言えると思います．

僕は Standard Lisp や Reduce なんかに慣れていたので，けっこうスペック自身にフレッシュに感じるものが正直なところ多かったわけです．パッケージの概念から始まって，いろいろなものが使える機構というのは，これは相当，ある意味で処理系を作る人にとっては，大変だなっていう感じを出すポイントでもあるし，また，効率にもかなり影響するポイントになるところではないかと思います．

データタイプの hierarchy は，一見すると lattice

† 日時　昭和 60 年 6 月 18 日の「記号処理研究会」（場所　津田塾大学）で行われた内容を各パネリストが編集したものである．編集作業では，NTT 電気通信研究所の奥乃博氏，電気通信大学院生の河原崎裕朗君と学部 4 年生の青柳博臣君の協力を得た．

1) 青山学院大学，2) NTT電気通信研究所，3) 京都大学，
4) 日立中央研究所，5) 電気通信大学

に見えるぐらいにきれいな形です．これもマイクロプログラムによるタイプチェックができる機械にとっては楽にできるけれども汎用機の場合には相当良い最適化コンパイラを作らないと処理効率が落ちる一因でもあるかと思います．

オブジェクト指向と Lisp との関係というのは向こうでは，相当問題になっています．なぜかというと AI の多くは Lisp をベースにしているから，その上でオブジェクト指向的な環境を，どうやって標準的なものを考えて載せるかというような話です．まあ現実的には，Flavors 的なものをとるか，Loops 的なものをとるかといったもの，あるいはその中核仕様を核に置いておくとかいったことが議論になっている．具体的なその議論の過程というのは持ち帰りまして，今，何人かの人が解析をしています．

湯浅 僕は，Common Lisp システムを作った人間としてここに招待されたようなんですが，資料を提供するという意味で，だいたい僕らのシステムがどんなものかというのを紹介します．

Kyoto Common Lisp (KCL)[2] といいまして，Common Lisp のフルセットです．非常に移植性が高くて，まず最初に Data General (DG) 社の Eclipse という機械で動きまして，あと VAX だとか SUN だとか Ustation だとかいろんな機械の上で走っています．

最初に作った Eclipse のバージョンは，僕と隣にいる萩谷君と，それから DG 社のプログラマが2人の合計4人で，最初のディストリビューションのバージョンができるまでに6カ月ぐらいを要しました．この時点でフルセットはフルセットなんですが，まだそんなに性能の良いものではなく，たとえばコンパイラが非常にチャチかったり，あるいはメモリ効率も非常に悪いとか，デバッグツールが足りないとかといった欠点がありました．そういうところをあとでぼちぼちと変更していって，今はもう，ずいぶんちがったものになってます．

なぜ Common Lisp システムを作ったかというと，まずわれわれの研究所は人材的にあまり恵まれてませんでして，自分でいい Lisp をデザインするほどの時間的，人材的余裕が，全然無かった．それから，Common Lisp の仕様は，他の Lisp と比べると非常に厳密に書かれていまして，他の Lisp ですと，マニュアルというのは手元に Lisp があって，それを見ながらでないと，とてもわからない（笑）というようなものですが，Common Lisp に関してはマニュアルだけ見てだいたいわかる．もちろんあいまいなところが，たくさんありますけれども，まだ，状況はましである．それから，将来 Common Lisp が普及すれば，仕様が正確に規定されてますので，Lisp のソフトウェアを他の機関と交換できるだろう．Maclisp の系統を引いてますので，全体として見ればいい設計であろう．そして記述能力が高い．これは皆さん意見が一致するところだと思います．

最後に，どれくらいのパフォーマンスを持ってるかということなんですが，これは Common Lisp のベンチマークで，スタンフォード大学の Richard Gabriel が集めている資料から抜き出したものです．Common Lisp は，汎用機よりはむしろ専用機にむいた仕様であるということが言われてまして，僕らも実際にシステムを作るときには，「専用機だったら，もっとここは良くなったろうになぁ．」というようなところがいっぱいあったわけなんですけれども，このベンチマークの結果を見ると専用機といえどあまりいい結果は出ていない…．われわれの KCL については，（これは Eclipse 上の KCL のデータなんですけども）Symbolics 3600 とくらべると，だいたい KCL の方が速い．中にはちょっと遅いものもありますが，3600 とちょこちょこぐらいのパフォーマンスです．

専用機に向いている仕様だと言っても，専用機でインプリメントするときに，その向いている性質をうまく使わない限りは，いい結果を得られないわけで，まだ，Common Lisp に向かって Symbolics などは，チューンナップが完全にされてないんじゃないかという気もします．

われわれの話はそれぐらいにいたしまして，実現するときに（主に効率の問題ですが）どういう問題があるかというのを思いつくままにあげてみました．まず，インタプリタが遅い．コンパイラは巨大になって，しかもコンパイル時間は非常にかかる．それから，メモリ管理は非常に効率が悪い．だいたい，この3つがあげられると思います．詳しいことはお手元の資料に書いてありますので，読んでいただければ幸いです．

竹内 私と Common Lisp の関係は，ハエが飛びかっているようなものでありまして（笑），Common Lisp というハエがブーンとうるさく飛び回っているわけですね，周りを．というのは，私たちは，今，TAO というオリジナルな言語をインプリメントして

おります.

こういう独自のものを作っている立場から見ますと，Common Lisp の機能は——アメリカ人に聞くとだいたい Common Lisp と言うには，機能が大きすぎるといって，みんなウェッという顔するんですけれども（笑)——まだ低すぎると思っています（笑).

ハエがブンブン飛びかっていてうるさいなあと思うんですが（笑)，たまにはいいこともあります．私，納豆がきらいなんですが，納豆の上にぱっとハエがとまってくれたので，その納豆を食べなくてすんだということもあります．たとえば，Lispのプログラムをトランスポートするときに，いちばん問題となるのは，I/O なんですね．いつも大苦労しますから，Common Lisp のマニュアルにかかれている I/O の通りに作っていけば大丈夫，マニュアルを書く必要もないというわけです．マニュアルを書かなくてはいけないというのは，インプリメンタにとって頭痛のタネです．この件に関しては，そこを見てくれっていうのは，さっきの納豆にハエがとまって食べなくて済んだという話です.

インタプリタが非常に問題多いって話がありましたけど，私もほとんど同じ意見です．コンパイラとインタプリタの両立性をとるという話がいつのまにかすり替わって，インタプリタが実用にならなくなったのでは，話が矛盾しているというのが，私の感想です.

それから，多重プログラミングに対して，きちんと follow up していないというのが残念ですね．それから，setf というのがあって，MacLisp 以来，アメリカ人は，「いい．いい.」と言ってるんですけども，私たちの TAO の代入機構の方がずっといいと思います.

完全な lexical scoping の closure には問題がありそうです．湯浅さんは，block について，予稿[3] の中で explicit に指定した方がいいと書かれましたが，closure に関しても，close する変数は explicit に指定する方が，いかなる意味においても，プログラミングスタイルとしては正しいと僕は思います．アメリカの連中が，どういうことを考えているのか私にはよくわかりません.

井田 現在オブジェクト指向の仕様のプロポーザルがでていて，12月ごろにはまとまってしまいそうです.

安村 世の中が知識工学とか人工知能とか騒いでいるおかげをもちまして，メーカでもやっと Lisp の研究ができるようになりました．自前で Lisp は作らなければいけないとかねがね思っておりましたから，これは大変よい機会だと思っています.

予稿[3] に添ってお話しますと，従来の Lisp と違っている点としては，lexical scope が採用になったということで，これは非常に良いことです.

それから，generic がかなり徹底しているわけで，これは仕様上，非常にきれいなんですけれども，本当に性能を上げるのにいいかどうか．generic があって十分に速くしようとすると，まず type の宣言を入れなきゃいけない．しかもユーザが間違って，宣言したのとは違うデータを突っ込むとか読み込ませることだってあるわけですから，一応 type check は要るわけですね.

それからパラメタの種類に関しましては，keyword とか optional とか rest とか key 形式とか，いろいろ増えまして，fsubr なんていう変なものがなくなったとか，special の数が少なくなったとか，きれいにはなりました．しかし，たとえば keyword パラメータでの場合，make array という関数で，dimension 以外は，8個フルに設定しようと思うと，keyword をたくさん書かなきゃならないんですね．だからこういうのは，仕様をきれいにするという立場からはいいんですけれども，その裏面が必ずあるということです.

closure に関しましても，言語の仕様からいうと非常にきれいなはなしなんですけど，実際，これを本当の意味での closure として使うユーザがどのくらいいるかというのは，疑問ですね.

それから他の言語との比較の面から見ますと，lexical scope が入ったのはいいんですけど，どうも書いてみると，local 関数が（flet とかなんとかいう……）中途半端な扱いになっている．仕様をきちっとする意味からすると，defun でもって local 関数が書けて然るべきだと思います．そうしないと，scope をもった変数みたいのが書けないんですね．そのため結局 special 変数とかになってしまって，非常にきたならしくなると思います.

それから special というのも，結局，従来の計算機との互換上で出たわけですけど，たとえばマニュアル見ますと，**図-1** のような変なものが出てくるんです.

どういうことかといいますと，1行目の “y” というのは declare special，special だって宣言してまして，3行目の “y” は宣言が無いから lexical なんで

```
1 (defun example (x y)
2 (declare (special y))
3 (let ((y 3)(x (* x 2)))
4   (print (+ y (locally (declare (special y)) y)))
5   (let ((y 4)) (declare (special y)) (foo x))))
```

図-1 special 変数と lexical 変数[1]

```
((lamda (a &optional (b 3)
         &rest x &key c (d a))
(list a b c d x))
1 6 :d 8)
=> (1 6 nil 8 (:d 8))
```

図-2 パラメタの対応例[1]

すね．また4行目で special って宣言してあるわけです．このカッコのなかでは後の "y" は special であって，前の "y" は，それが無いから lexical になる．同じ変数なのに使う場所でもって special として使えば special なものも見えてしまうっていうのは非常に気色悪いですね．で，これが従来の Lisp の上からすると，あたりまえなのかもしれませんけど．他の言語を見ていると，ちょっと奇妙な気がします．

それから，パラメタに関しましても，Lisp が，インタプリティブな言語というか，ダイナミックな言語から発生したことから来てると思うんですけど，同じ仮引数が二重に出てきても，後の方が有効になる．それから，パラメタの種別が非常に増えてきて，個々のパラメタは非常にきれいなんですけども，このマニュアルの 65 ページにあった例（図-2）ですと，4番目の d ですね，これは keyword で対応するのと，&rest でもって引っ張ってくるのと両方が，alias というか二重に見えてしまうっていうのもちょっと気持ち悪い気がします．これは，便利なこともあるんですけどね．便利だってことと，きれいだってことは，また別ですから．

最後に数値関数なんですけど，generic をたくさん使いますと，ついつい generic ですべて数値関数を書いて，コンパクトに書きたいというのが願いなんですけど，実際，本当に FORTRAN 並みの精度と性能をあげようとすると，ああいう generic な関数使ったら，たとえば，complex も含めて，ある1つの書き方していいかっていうと，絶対そんなことはない．complex だと，実部と虚部を分けて，その大小関係をも考えて作るという細かいことまでやらなきゃいけないわけで，そうすると，あんまり generic であることが，数値関数にとって，それほどうれしくないわけなんですね．

で，結局今の Common Lisp はわりかし顔つきはちっちゃいからスマートそうに見えるんですけど，いろんな機能をたらふく食べて，図体がでかくなっている．しかもまだなにか食べ足りないんで，マルチタスクだとか Flavor だとかグラフィックスを食べたそうな顔してますね．

そこで，もう少し整理して言いますと，いい点としては，仕様の統一とか，一元化っていう点では lexical scope．これはもう戻れないしこうなるべきだと思いますね．それから closure に関しては，仕様上からはいいけども，使う意味では問題なんで，これはもっと制限された形でもいいかなと思います．それから，multiple values も，実際プログラム書いたうえとかあるいは仕様からしても，これはないと書きにくいプログラムもあるんで，あった方がいいかもしれません．それから generic は関数の数を減らすという意味ではいいが，さっき言いました性能のこととか数値関数を作る意味ではちょっと問題ですね．それから悪い点としては lexical が不徹底であるとか．宣言の考え方，過度の一般化とか，大き過ぎる点，効率に対する配慮がまだ足りないんじゃないかということなどです．結局そういうことをまとめますと，たくさんの人が集まってなにかプログラミング言語を設計しようとすると，こういう矛盾する課題がありまして，これは，一元的に並べてあると，みんなが満たすようなのができるかと思いますけれども，実はそれぞれ違うベクトルを向いていて，作らざるを得ないわけですね．片っぽは標準化とか統一性，それからある面では，機能を大きくしたいとか，あるいは従来との互換とか．で，そうすると，複雑になるとか，規模が大きくなるとか，あるいは性能の効率がおろそかになるとか，お互いに綱を引っ張り合って矛盾し合うんで，どこを一番中心に置いてやるのか，そこが一番問題になるわけです．たとえば，極端なことを言うと効率をまったく無視して，非常にきれいなのを作るか，あるいは標準化だけを中心に作るとか，その辺，今は，妥協の産物になってるんだと思います．

萩谷 技術的なところは湯浅先生の方からやっていただいたので，私は個人的な考えを，非常に簡単に話すだけにします．

今までの話と，似たようなものですが，標準案に，ユニオンとしての標準案と，インタセクションとしての標準案があると思うんです．Common Lisp は，当然ながらユニオンとしての標準案であると思います．ただし，$+\alpha$ と $-\beta$ がもちろんありますが．したが

って，肥大化しているということになります．

それからもう1つ，これも，もうなん回も言われていますが，専用マシンのための言語として設計されたということです．したがって，OS が書けねばならぬ，すべてのアプリケーションが書けねばならぬ，ということになります．そのすべてのアプリケーションというのは，もちろん AI とか数値計算とかウィンドゥシステム，グラフィックス，言語プロセッサ，こういうものが，Lisp で全部書けなければいけない．Lisp しか言語が無いという世界での Lisp なわけですね．

ところが，そのくせ，単なる言語仕様でしか今のところはない．ソフトウェアエンジニアリング的な要素が全然無いわけですね．バージョン管理とかプロジェクト管理とかプログラミング環境とか，そういう要素はもちろん全然規定されてはいない．現在の言語仕様の標準的なレベルからすると，かなり昔のレベルではないかという気はします．

最後に，これは勝手なことを言ってるんですが，「かつて Lisp は自分の言語であり，みんなの言語であった」と，僕は思うわけです．「ところが，しかし，Common Lisp は（USA のと書いてありますけど）みんなで作っているのに，何か他人（ヒト）の言語になってしまった」と，そういう感じがするわけです．デザイナの人たちも，本当に，情熱を持ってやっているのかなぁという一抹の不安をちょっと感じたことがありました．

司会 各パネリストに Common Lisp との係わりをいろいろ話していただきましたが，次は対処法について，たとえば，Common Lisp を積極的に取り入れていくべきか，やっかいな物だが少しは気にしなければならないか，あるいは，完全に無視するのか，そのあたりの意見を述べてもらうことにします．

井田 僕が考えるのは，やはり Common Lisp は必要だという前提があります．教育の現場で考えてみてもスペックの問題，どう教えるかも1つあると思うんです．

それから，Lisp の普及を考えると，多くの人が Lisp を使うのがいいという前提があれば，やはりなんらかの標準仕様を考える必要がある時代が来ていると思います．

で，鍵はやっぱり，日本的な環境で考えると，小さな Lisp が今後どこへ動いていくかっていうのが，1つあると思うんですね．

毎日使えるっていう感じの Lisp があったときに，それがどこで使えるか，どんな仕様になっているか，それがかなり，実際，今後に影響があるんではないかなっていう気がしています．

そういう意味で，言語仕様のサブセッティングの問題とか，商用機との対応は仕様の面での詰めがもう少し，ぜい肉落としとかですね，あるいは少しそういったところを手直しする必要があるだろうというようなことを思っていて，うまく体系が整理されたらですね，Common Lisp の定着が，その頃にはなるかなと思います．

竹内 私たちがどうしてるかといいますと，基本的にどうでもいいところは Common Lisp に合わせています．

なにをどうでもいいと思うかってことなんですけど，関数名とか，グローバルな変数名とか，そういうものをわざわざ違う名前にする必要は全然ありません．

一番問題なのはインタプリタが遅くなるところ，つまりパフォーマンスがどうしても出ない．鉛があるためどうしても1歩1歩足が重くなるというところですね．そういうところは全然気にせずバッサリと切り捨てる方針です．ただし，コンパチパッケージとして，超趣味的検定をやられてもヨロヨロと通るぐらいのものは，サポートしときゃあいいんじゃないかな，というぐらいですね．だいたいそんなものを使うヤツがアホなんで（笑）そんなアホか気違いのために，多大な労力を割くのはまったくバカバカしいから，やめる．非常に健全な発想なんですけど……(笑)．

安村 先程言いましたように，大きすぎるとかいろいろ問題はあるんですけど，さっきも萩谷さんの方から出ましたけども，今まで，こう，非常に local な，たとえばハッカーと言われるような人たちが，Lisp を作って，作った人が使うとか，その周りでしか使わない状態から，かなり，一般的に使うような状態になったと思うんですね．そういう時点で，やっぱり Common Lisp みたいなのがあって，それを使うっていうのは，避けられないことだし．今後は Lisp を作るにしても，Common Lisp コンパチっていうのが，多分，謳い文句になるんじゃないかと思うんですね．それで，方向としては，ソフトの流通とか，互換性とかいう点で Common Lisp ということになると思うんですけど．

ただ，やはり，アメリカで作られたっていうか，よそで勝手に決めて作られたというところがあって，そ

のへんからは，竹内さんの意見と近くなるんですけども……．すべてを本当に真面目にきちっとやる必要が本当にあるかどうか，つまりもうちょっと仕様について細かく議論して是是非非的に臨む必要があるんじゃないかっていう気はします．大局的には Common Lisp の仕様通りにやらなきゃいけない，個別的には，もうちょっといろいろ議論していって，直してもらえるところは直してもらったりする必要があると思います．

萩谷 Common Lisp が，なぜ広まるかっていうのは，これは明らかなことだと思います．なぜかというと，アメリカの Lisp 界っていうか，そういうのは，日本の記号処理研究会よりは大きいですが，(「大きくないよ！.」会場からの声)，大きくないですか？，まあ，というような意見の出るくらいの大きさで，その中の人たちのほとんどが Common Lisp の言語仕様の規定に参加しているわけで，またその中の多くの人たちが実際の処理系を作成しているわけですから，当然ながら Lisp は Common Lisp に流れていく．Lisp を作ったアメリカが，Common Lisp に流れていくわけですから，世界中が流れていかなければならないのは当然なわけでして，要するにきれいごとをごちゃごちゃ言っても，やっぱり Common Lisp の通りに作らないとまずいという，……なんていうか，寂しさを感じているという (笑)，まあ，作った人でないとわからないわけですけど (笑)．

だから，それは，なんて言うか技術的にいろいろな問題点はありますけども，やっぱり根本的にどうしても実用的な言語を目指すときは，やはり無視できない存在だとは思います．

湯浅 Lisp をどういうふうに使うかによって意見が分かれると思うんですが，Common Lisp を使っていていい点は，もちろん，他の Common Lisp で作ったソフトウェアが簡単に載るということです．他人(ヒト)のソフトウェアを使うためのツールみたいな感じで，手元に Common Lisp が1個あるというのは非常に便利なことだと思います．

実際に，どれだけ移植性がいいかと言うと，Spice プロジェクトでいろいろなソフトを作っていますが，それらを KCL に持ってくると，そのまま動いてしまいます．だから，これからどんどん AI 関係のソフトとか，Lisp のアプリケーションが出てくると思うんですが，萩谷君も言ったように，そういう場合にターゲットとなる language specification というのは，Common Lisp になるだろうと……．これは明らかだと思います．そうなると，Common Lisp が無いと世の中に遅れてしまうという，惨めな話になります．

特に日本は AI 関係では，まったくの後進国ですから，どんどんアメリカで作られたいいソフトウェアを導入しなければいけない立場にあると思います．そのためにも，これから役に立つはずです．

いずれにしても僕らは，どっぷりと Common Lisp につかりきってしまって，研究所では Lisp ユーザはみんな Common Lisp (KCL) を使っている状態ですから，なにを言っても仕方がないというわけです(笑)．

司会 今までのことをまとめますと，Common Lisp に流れが移るように感じられます．たとえば，それは教育用 LISP システムの統一化であるとか，ソフトウェアを輸入したときに Common Lisp が無いと困るとかの必要性から，皆がやるのでしかたがないから追随するという，そういう傾向は，やむを得ないと思うんですが．そこで，会場にも大勢の方がいますので，なにか，パネラの方に，これは聞いておきたいっていうことがありましたら，名指しでもけっこうですし，一般的なことでもけっこうですから質問してください．

渡辺隼郎(津田塾大) 私はこんなふうに考えました．

最初に Common Lisp の話，聞いたのは，1982 年のことですが，アメリカ人たちは，negotiation, negotiation って言っていました．Lisp の研究者や作成者たちが集まって negotiation やった，というわけです．negotiation っていうのはどういう感じかというと，たとえば，ベルサイユ条約やヤルタ協定が negotiation です (笑)．すると，それがしばらく続きます．ヒットラーが出てきてこわすまで続きます．それからヤルタ協定，これは未だに続いています．ですから，ある目的のためにやむを得ず従ってるというところであります．ただ，どこかで，気持ちがそれに反するようなところがあって，それが爆発すると崩れるわけです (笑)．私はそういうように理解したいと思っています．

司会 それはどのレベルで出ている話ですか？．

渡辺 私が聞いたのは NIL やなにかの作成者とその辺からです．

梅村恭司 (NTT 通研) 萩谷さんが，先程ユニオン，全部取り込む形の仕様って言ったんですが，逆にアンド，ある程度ということは全然考えられないのでしょうか．もう少し小さくて使えるものはないのでし

ょうか.

萩谷 それは，キーデザイナのひとりの Gabriel の論文「A critique of Common Lisp」[4] の中にも，あのままでは汎用機とかミニコンに載せるにはあまりにも巨大すぎるから，適当なサブセット† を定義すべきだということは書いてあります.

井田 オフィシャルにね.

萩谷 オフィシャルにもあるんですか.

井田 いや彼はオフィシャルにそうした方がいいだろうと言っています．勝手にサブセットを作っちゃうからと.

安村 サブセットを作るんだったらオフィシャルにやらなければ意味がないです．個人的にやったら方言を作っちゃうんで.

井田 僕の資料の Guy Steele からの返事の七番の所がそれに関する所になっているわけです．サブセッティングというのは，なんで Common Lisp の場合出てきているのかというと，いまおっしゃったようなこと，それをすまして言うと技術的な理由により，ですね．技術的な理由によりアンドになるようなものを決めておく必要がある.

実際にオフィシャルなサブセッティングをした方がいいというのは，萩谷さんも言ったように Gabriel もそうだし，それからまた Winston は爆弾をなげかけようとしている人でありまして，「私の決めたサブセッティングの Common Lisp がこれが本当のパソコンの Common Lisp サブセットだ.」って感じで好きに Common Lisp のサブセットを決めてやっているわけですね．オフィシャルには，誰に聞いてもそういうのがあったらいいね，誰か決めてくれとしか言わない．Fahlman からは，「こういうのはパソコンは日本が得意だから日本で決めてアメリカへ返してくれ」という手紙が返って来ました．それが現状で，むこうではちょっと，だいたいそういうよりも，大艦巨砲主義じゃないけどでっかいとこへ，しかし，汎用機じゃなくて，パーソナルな専用機の環境の中にこれから流れていくだろう．それはかなり大きな環境を引きずれるはずだって前提が頭の中にあるから，あまりサブセッティングの方に興味はないっていう人が多いですね.

湯浅 Common Lisp がどれだけ巨大なものかということに関して KCL の資料なんですが，ソース（カーネルはCで書いてあって，あとコンパイラとかライブラリは Lisp で書いてあるんですが），ソースだけで 1 MB あります．システム自体は 1.6 MB の空間を使います．それにあとデータエリアを取りますから，満足に動かそうとすれば 3 MB ぐらいの空間が必要というバケモノです.

奥乃（NTT 通研） 数解研では KCL しか使われてないそうですね．湯浅さんや萩谷さんが以前使われていた Standard Lisp とか Franz Lisp だとか Mac Lisp のほうが，プログラミング環境がよくて，プログラム開発の効率がいいと思うのです．その辺，Common Lisp みたいにインタプリタが遅いと，そういう利点が生かし切れないという気がするのですが，実際に Common Lisp を使われてどうですか.

湯浅 KCL のインタプリタは速いです（笑）．というのは，KCL というのはカーネルをCで書いてまして，古典的な方法で作成しているんですね．Spice などは，まずコンパイラを Lisp で書いてクロス開発するという方法です．その分余計にインタプリタが遅くなる．数解研の環境で言えば，MacLisp なども DEC-2020 の上で動いているんですが，それに比べれば機械のパフォーマンスが違うこともあって，まだ KCL の方が，いくらインタプリタでもいいということです.

司会 想像していたよりも速い結果ですね.

萩谷 なぜ速いかというと，マクロと書いてあるのをほとんど special にするなど，力ずくでインプリメントやっているからです．setf まで special になってますが，get とか aref とかそういうときだけ，マクロ展開を防ぐわけですね．マクロ展開を最小限にくいとめれば速くなるんですが．しかし，Common Lisp の仕様だとどうしてもユーザがなにかやると勝手にマクロができるもんですから，すぐに遅くなります．ですからインタプリタだと，感じとして普通の A-list の作り方をすると 2，3 倍おそくなると思います．従来の value cell に値がつっ込まれているやり方よりは，そのくらい遅くなると思います.

司会 そうするとインタプリタの良し悪しは処理系作成の腕力で決まるという感じですか.

萩谷 いや，プログラムを書くとすぐマクロができてしまうということがあって，マクロがいったんできれば，もう速度がばっと落ちますから.

司会 展開しないようにやるとか.

萩谷 あと implicit block でしたっけ，まず関数が implicit block っていうのを必ず設定するので，そう

† 注：現在サブセット仕様については，日本およびフランスで検討が進められている．原案作成のための井田案は 7) にある.

いう意味で，言語仕様そのものがインタプリタを遅くするように，するように(笑)，これでもか，これでもか(笑)というような言語仕様です．それはだから，処理系，いくらがんばっても速くならないっていうところはあると思いますね．

黒川利明(東芝) でもインタプリタががんばれば，けっこううまくいきません？

竹内 まあ，がんばれば，インタプリタの速度は2倍くらい向上すると思うけどね．

萩谷 2倍くらいですか．まあそのくらいですね．

井田 さっきの萩谷さんの setf の話だけれども，当然 defstruct なんかでユーザが作ったのは面倒みるわけでしょう．

萩谷 defstruct は面倒みますが．

湯浅 インタプリタがですか．

奥乃 defstruct で作ったものに setf が動くかどうかということでしょ．

萩谷 それはマクロ展開しないです．

湯浅 かなりインチキなんですけどね．面倒みれるのは special として処理して，だめだったらマクロ展開する．

井田 そういうことになりますよね．

萩谷 当然そうですが．

井田 マニュアルにある fix のものは中に入れちゃってと．

萩谷 (文献5) で紹介された Skef Wholey の歌を参照しながら) この歌に 1db に choke するというのがありますが，こういうのはどう考えても special でいくわけないですよ．

1db は Load Byte で，integer の一部を取り出す関数です．それが (setf (1db place pointer) value) となると，place の pointer で指されたところに value が入らなくてはならない．これを setf でやろうと思うと，なにもかもがすごいことになります．それをがんばった人が Lunar Dave, David Moon です．(といって湯浅に確認を求める)

湯浅 Moon だから Lunar なんです．

萩谷 David Moon がハックして，setf をこれに動くようにしたんですね．そうしたらどういうことになったかというと，setf を作るためには 5 value を返す multiple value 関数を使うことになって，それが次の歌詞にある Gang of Five ということです．

We had a simple SETF,
But it choked on LDB.
So Lunar Dave done fixed it:
Go look at page eighty three.
The Gang of Five they didn't take a poll.

元吉文男 (電総研) Common Lisp だと移植性がいいからというようなことを言われていますが，Fortran でも完全に JIS 規格とか決まっているのに，機械が変わったりすると動かないということがあったんですけれども，Common Lisp でもそれぞれインプリメントされた場所によって，拡張された機能があって，ユーザはそれを知らず知らずのうちに使っている場合が多くて，それをほかの場所にもっていくと，その拡張された機能とか，データタイプなども拡張されていて，それが動かないという場合があるので，Common Lisp だから移植性がいいというと誤解を生じやすいと思います．Common Lisp が広く使われるようになるとこれが問題になるんじゃないかと思うんですが．

安村 その話なんですけれども，たとえば Fortran で JIS 規格とか ISO 規格がなかったときのことを考えて欲しいんですよね．そうするとおそらくはどっかなんか大きな所が作ったやつをね，見様見真似でまねしたりね，あるいは自分でこういうのがいいからって作り出すと思うんですよね．そういうレベルのときに規格があるのと，さらに規格があっても更にマシン依存のところが少し残ったり，処理系依存の部分が残るっていうのは話はやっぱり別だと思うんですよね．そういう意味で Common Lisp. MacLisp のマニュアルなんか今までいい加減だったのが最近 revise されましたが，それでもまだいろいろ書いてないことあるんですけど，それに対してきちっと書かれてて，一応まあ InterLispの人も多少加わったと聞いていますし，あの ZetaLisp とか Lisp Machine Lisp の人も，これから Common Lisp に移行しようというときに，それに乗って処理系がだいたい合わされてきた，そういう意味での Lisp の規格と言うまでいかないですけれども，それに近いものがあってみんな書いて他に移すときの移植性と，それが全然なかったときっていうのはかなり違うので，やはり私はかなり進歩したと思います．実質上はマシン依存のこととかいうのは必ず残りますから，それはまだまだ手直しは避けられないんですがね．でもさっき竹内さんが言われたように関数名とかごくごくつまらないことを今までだったら，一生懸命トランスレータ作ったりとか，パッケージで皮を被せるとかいろんな苦労していたのが，そういう苦

労がだいぶ少なくなるということは言えるんじゃないかと思うんですよね.

稲田信幸(理研) 私は今まで Common Lisp の勉強はしてきてないんですが, Common Lisp の言語仕様をざっと見ますと少し大きすぎるのではないかと思います. OS でもなんでもできるようにと Multics が開発されましたが, 使用してみるとやはり大きすぎるのは良くないということで軽装の Unix が開発され広く使用されています. 移植性の良い応用プログラムのために言語仕様は確かに必要であると思いますが, それにしても少し大きすぎるような気がします. 私は Lisp で書かれた応用プログラムの移植も教育の場ではひとつの教育材料と考えてきましたので, 仕様の異なる Lisp システム間の移植も結構苦にせずに行って参りました. Common Lisp に統一されると逆に教育面でのマイナスも現われるのではないかと危惧する次第です. プラス面とマイナス面を考慮しながら Common Lisp に取り組んでいきたいと思います.

安村 一つの参考になるのは, Abelson and Sussman が「Structure and Interpretation of Computer Programs」[6] という非常にいい本出してて, それは Scheme という Lisp の方言に基づいているんですけれども, ご存知のように Common Lisp の前身になった言語で, あそこで書いてある程度が書ければ教育用には, たぶん足りるんじゃないかと思うんですよね. ただ scheme の場合には, 互換性を考えていない分だけ Common Lisp よりきれいな点がいろいろあるんですね.

井田 今の教育の方の話では, むこうではやっぱり Common Lisp 準拠のサブセッティングをしたのを, 教えているということは, 僕の知っている範囲でもいくつかの大学でやっていますね. たとえば Stanford では Brooks が自分でこの範囲のものだっていうのを MacLisp の上にセッティングした Common Lisp で授業を進めているわけですね. また一方で, 使える処理系と教材で決まってくるってとこもあるんじゃないでしょうか.

難波憲司(ディジタルコンピュータ) ちょっと個人的な感想でもうしわけないんですけど, 僕もまだ Common Lisp に関して全部勉強してるわけじゃないんですが, 漠然とした感じから言いますと, バックに DoD がいることも考えて, Lisp 界の Ada みたいな感じがするんです. Ada はひところ2年くらい前ですか, 騒がれましたけど, 最近あまり聞かないんですね. ほっといたらまた, どうにかなるんじゃないでしょうか.

奥乃 Common Lisp は Ada in AI って言われています.

萩谷 Ada と Common Lisp の違いはって言うと, Ada はなかなか処理系が出てこないっていうとこがあるんですが, Common Lisp はどんどん出ますね. VAX Lisp も出たし DG からも出てますし, それから Lucid が山のように移植をやっているかこれからやると思います. かなり Ada と Common Lisp では状況がそういう処理系の点で違うと思いますが.

奥乃 Common Lisp が普及するうえでの問題は, Common Lisp のうえでどれだけのアプリケーションが書かれるかということではないでしょうか.

現時点では DARPA のプロジェクトでは, 横で眺めてるんじゃないでしょうか, 共通化という意味では InterLisp にしても, もともとあれは Inter って付いているように BBN と Xerox との間で共通化しようという目的で名付けられたものですし. Standard Lisp っていうのも Reduce に対して Standard にいこうとしたわけですね. そういう共通化の動きっていうのは2回失敗しています. Common Lisp が普及するのはアプリケーション側が乗っていくかどうかっていうことにかなりかかってくるんじゃないかって気がします.

湯浅 Ada と Common Lisp のもう一つ大きな違いは, Ada というのはまったく新しく設計された言語だったのに対して, Common Lisp は MacLisp とか ZetaLisp とかをベースにして, それに Steele の趣味を加えてでき上がったことです(笑). ですから, 実際に MacLisp のソフトはそのままでかなり動きます. ZetaLisp のソフトは ZetaLisp 固有のアセンブラ使ったりとか. グラフィクス使ったり, もちろん flavor 使ったらだめですけど, そういうのがなければ基本的には動く. 今 Lisp のツールを一番豊富に持っているのは Symbolics じゃないかと思うんですが, Symbolics の方では, だんだん Common Lisp に移行してまして, もう彼らが持っているソフトはほとんど Common Lisp に書き換えてるらしいんです. ただ問題なのは, さっきどなたかおっしゃいましたように, Symbolics の Common Lisp というのは, かなり Common Lisp のスーパセットになってまして, いろんな機能を使っているために, それをそのまま, 普通の Common Lisp システムにもってきても動かない

という，そういう問題がすでにあることはあります．

奥乃 InterLisp のコミュニティはどうなっていますか．

井田 コミュニティは分からないけど，InterLisp のコンパイラを書いてる Dr. Masinter という人が PARC にいますね．彼は一生懸命やってましたよ†．

奥乃 InterLisp は大変じゃないでしょうか，名前も変わるし，メンバの一人 Jon L. White は Xerox から Lucid へ移ったから一人いなくなりましたしね．

安村 さっきの奥乃さんの話に関係するんですが，過去に Standard Lisp とか InterLisp という試みがあったんですけれども，うまくいかなかったっていうのは，やっぱりコミュニティが小さかったせいで，小さいうちで使い合っているときには，なかなか統一できない．Expert System だとか Production System だとか，アメリカだと必ずしも Prolog じゃなくて Lisp で書いたりしていますね．そういう人たちが Common Lisp 使うようになると影響力はあると思うんですね．それと，やっぱり，さっき言われた設計者，今までハッカといわれてきた人たちも一応 Common Lisp に乗り換えてみようとか，特にメーカですね，アメリカのメーカで Lisp を提供しているところは Common Lisp に移行しようとしているのが大きいと思います．それが，今までの Standard の試みや，Standard Lisp とか InterLisp の試みと違う点だと思うんですね．

奥乃 今度はちゃんと本が出版されました．

井田 でもワープロで作ってるためにですね，文章の埋め間違いとか，そういうのがけっこうありまして(笑)．翻訳をしていてずいぶん気が付いて誤字も含めて60カ所ぐらいあるんですね．

湯浅 まだ誤植だったらいいんですけど．内容的にでたらめなところが，実はいろいろありまして，MacLisp とか ZetaLisp の知識を仮定しないと全然読めないというようなところもあります．たとえば，lambda 式に quote 付けたものを評価したときに環境はどうなるかという問題があるんですが，どこにも書いてないんです．これを関数だと思うかどうかも書いてないんですよ，実は．で，MacLisp の常識からいえば環境は，なにもない nil の環境で評価するのだと分かります．以前，安村氏に，「このへんのことどっかに書いてあるか」，と聞かれまして，どっかに書いてあるんじゃないかと思って，一生懸命捜したんですけど，どこにも書いてない．もう一つ気が付いたのは，関数の引数の順序です．普通は Lisp は左から右に評価しますが，このことがどこにも書いてないんです(笑)．どこかに，これを臭わせる部分はあるんですが．

井田 その順序に依存するなって書いてあるところがなかったっけ．

湯浅 setf のところに暗に書いてあるんですけどね(笑)．

竹内 萩谷さん，stepper うまく書けましたか？

萩谷 一応書けましたが．eval$_*$ にあたるやつがなくなってたんですが，それは evalhook でがんばるんです．全部 hook にしわよせがきてるんですよ(笑)．

司会 では時間も少なくなりましたので，最後に一言ずつ．

湯浅 仕様を読むとまだまだ仕様といっても不完全な点に気がつきます．今話にあったようなこともありますし，誰にもどういう風に解釈したらいいのか分かっていないというようなところがまだまだあります．そういうのは ARPANet のメイリングリストでどんどんみんな議論をしてるんですけども，ある程度の時期熱心に議論してて，それがいつのまにかプツッと途切れて，どういう方向に納まったのか全然分からない．結局みんな自分の好きなようにインプリメントしているという (笑)，そういうところも実はあるんです．実際にプログラムを書いたときに，そういうことで移植性が損なわれるということは，可能性としては低いんですけども．そういういい加減なところがまだまだあるんで，もうちょっとそこらへん，僕が言ってどうなることでもないんですけど，なんとかして欲しいなと思います．

萩谷 標準案ができたってことは，すごいことだし，さっき Steele の趣味が入ってると言いましたが，なんか，標準案を作るのにそういう趣味を入れるという余裕があるっていうところがうらやましいとこですね (笑)．いいものは取り入れるのはいいことですから，いいところはどんどん取り入れたらいいと思いますが，やはり，これからはわれわれもわれわれというかみなさんも世界に貢献するようなことを自分からやって欲しいなあという気もしますが．

安村 言語の標準化というのはやっぱりわれわれ考えている以上に，重要なことで，ほんのちょっとプログラムを書き換えなきゃ動かないのか，それともそのまま動くのかっていうのは，えらい違いなんですね．

† 注：Xerox Common Lisp は IJCAI-85 で発表され，86 年中には公開予定とのことです．

ここにいる人たちは非常にハッカと言わないまでも非常に手先の器用な人だから，ちょこちょこっとできると思われるでしょうけど，たとえば KCL の Common Lisp をもらって来てね，自分とこで動かそうとしたとき，すぐに，動くか動かないかっていう違いはものすごく大きいと思うんですね．ですから，そういう意味で Common Lisp の出て来たっていうことは，非常に大きい．日本人はともすると，言語の仕様に関して，文句だけ言って，あまり建設的な意見表明がどうも苦手なもんで，たとえば Pascal の標準化とかいってもそれはダメとか言っておしまいで，ここをこう変えればいいって提案がなかなかないんですね．さっきサブセッティングなんか日本が得意だからやってみたらという話がありましたが，井田先生のところでいろいろなさってるみたいですけど，そういうのもどんどんしていただいて，意見をともかくこういう場とか，郵政省の人間メイルだとかですね（笑），利用してもうちょっといろいろ意見交換がされてもいいと思うんですね．そうすることによって Lisp のコミュニティがもっと広がるような気がします．

竹内 このごろ Prolog とか Smalltalk とか，Lisp は少し古いですけど，話題になっておりますけど，全体的な印象からいうと，そういういわゆる新言語に，根が生えているとはとても思えないんです．そういうものを使って本当に世の中に役に立つソフトができること——役に立つと誰かが思う幻想でもいいんだけれど——が，なんによらず一番重要でしょう．Common Lisp が役に立つかどうか，よく分かりません．私は Common Lisp のことを公文式 Lisp と呼んでおりまして，みんなが公文式ドリルをやっていれば教育はいいか（笑），はたして創造的な子供は育つかと（笑）．そういう観点からいうと公文式 Lisp には若干疑問を感じてるということです．

井田 僕はですね，最後に，言うことは，Guy Steele は非常に精力的なまじめな人でだいたいいろんなことを書いても返事を丁寧にしてくれる人だと思います．今ちょっとそれにくわれてやんなっちゃって，もうやりたくないとか言ってましたけど．

いずれにしても，日本からもっとアピールしてもいいんじゃないかと，それを取り上げてもらえるかどうかっていうのは，やっぱりいろんな別の観点っていうのが出てくるかもしれませんけども，それを思うっていうのが一つ．

それから実際に僕のところでまとめてる資料なんかがありまして，object 指向の ARPANet の交信録ですね，このディスカッションをしている過程が全部で 100 枚くらいあるんです．いずれにしてもこういう動きとかいうのは流されるだけじゃなくてある程度やっぱり関与してみて意味が分かったり，良くないとこが分かったり，いい点が分かったりするところがあると思うので，真剣に議論していただく人が増えたらいいなあ，そんな雑感で終わりたいと思います．

司会 私が予稿[3]にも書いたように，Common Lisp が出てきてしばらくたつし，まだ仕様が決まっていないところもいくつかあるようですが，とにかくこれが今後の Lisp の標準になるかどうかは別として，非常に大きな影響を持つことだけは確かでしょう．Lispに関心のある人はこれを理解して，学んで，習得して欲しいと思います．日本でもなんらかのコンタクトを取っていくべきでもあり，完全にすべてが反映されるとは言いませんけど，これに積極的に関与し，またその動向に関心を持っていたらいいんじゃないかと思っています．

それでは今日のフォーラムを終わりたいと思います．

参考文献

1) G. L. Steele Jr.: Common LISP. Digital Press (1984).
（邦訳 後藤英一監訳，井田昌之訳：Common LISP. 共立出版，bit 別冊(1985)）.
2) Yuasa T. and Hagiya M.: Kyoto Common Lisp Report, 帝国印刷出版部 (1985).
3) 第 34 回記号処理研究会資料 32-5〜10，情報処理学会 (1985).
4) Brooks R. A. and Gabriel R. P.: A critique of Common Lisp, Conf. Record of the 1984 ACM Sym. on Lisp and Functional Programming (1984).
5) 湯浅，萩谷：入門（3），bit 6 月号，共立出版 (1985).
6) Abelson H. and Sussman G. J.: Structure and Interpretation of Computer Programs, MIT Press (1985).
7) 井田：A Common Lisp Subset Proposal, 第 36 回記号処理研究会資料 34-4，情報処理学会 (1985).