2009 年度下期未踏 IT 人材発掘・育成事業　採択案件評価書

1. 担当PM

首藤　一幸 PM(東京工業大学大学院情報理工学研究科　数理・計算科学専攻准教授)

2. 採択者氏名

チーフクリエータ：　松山　朋洋（専修大学　経営学科　経営学部）
コクリエータ：なし

3. プロジェクト管理組織

株式会社オープンテクノロジーズ

4. 委託金支払額

3,000,000 円

5. テーマ名

Emacs における高精度コード補完機能の開発

6. 関連Webサイト

http://cx4a.org/

7. テーマ概要

Emacs の編集インターフェースは非常に洗練されており、プログラマの作業効率に与えている影響は計りしれない。しかしその一方で、コード補完機能においては Visual Studio の IntelliSense や Eclipse の CodeAssist などには遠く及ばない

のも事実である。本プロジェクトでは、主要なプログラミング言語に関してEmacsのコード補完機能を大幅に増強することにより、プログラマの作業効率を全体的に底上することを目的としている。

本プロジェクトの母体となるのは auto-complete.el という拙作のコード補完拡張である。auto-complete.el はポップアップ型のコード補完機能のフレームワークである。現状ではごく簡単なコード補完機能しか提供していないが、プラグインという形でさらに高度なコード補完機能を開発することが可能である。本プロジェクトでは各々の主要なプログラミング言語のためのプラグインを開発することで上記の目的を達成する。本プロジェクトでは C++、Java、 Ruby、 Python などのプログラミング言語に対応する。

現状では、これらのプログラミング言語におけるメソッド補完などの高精度なコード補完は不可能である。本プロジェクトでは、これらのプログラミング言語で高速かつ高精度で行えるようにする。

さらに、以下に挙げる auto-complete.el 自体の拡張および研究も行う。

- 補完インターフェースの研究
- 補完インターフェースの改良
- サマリ機能などの本体機能拡張

また、日本語および英語のドキュメントを作成して、ユーザー(特に新規ユーザー)が簡単に利用できるようにする。

Emacs が開発されてから二十余年、ポップアップ型補完拡張は一度も登場したことがなかった。ましてや動的言語のメソッド補完などは夢のまた夢であった。最近になってようやく、それを実現するための条件が満たされようとしている。私は、本プロジェクトがEmacsの進化に大きく貢献し、今まで見せたことがなかったエディタとして一面を見せはじめるきっかけになると信じている。

8. 採択理由

先端的で極めて利用者の多いテキストエディタ Emacs を対象として、いくつかのプログラミング言語を対象とした高精度コード補完機能を開発・配布するという提案である。これまで提供されてきたコード補完機能は、文脈や構文を考慮せず精度が低い、利用者を長く待たせるといった問題があった。それを解決し、世界中のプログラマの作業効率を高めることを目的とする。

プログラミング、コーディング支援技術の主戦場はEclipse等の高機能IDEに移った。とはいえ、特に先端的な技術者の間で Emacs の人気は根強く、そこでのプログラミング支援技術には大きな需要がある。

オーディションでは松山君の Emacs への並々ならぬ愛情を感じた。ぜひ、Emacs

を世界一のエディタたらしめるために現状欠けている、(まっとうな)コード補完機能を完成させて広めて欲しい。

9. 開発目標

本プロジェクトでは Emacs における C/C++および Ruby の高精度コード補完機能の実現を目標とした。そのためには、従来の Emacs の貧弱なコード補完機能を刷新する必要があった。そこで以前より開発していた auto-complete.el をさらに改善・拡張し、頻度表に基づいた補完候補の並び換え機能や、補完候補のポップアップヘルプ機能を追加し、目標の実現に必要な基盤を整備した。続いて、Ruby の高精度コード補完機能として RSense を開発した。動的型付け言語である Ruby では、コーディング時に高精度コード補完を行うのは極めて困難である。そこで型推論の技術を応用することで、実行時に取るであろう型を静的に解析し、その情報を Emacs から利用することで高精度コード補完を実現した。また、C/C++の高精度コード補完機能として GCCSense を開発した。このソフトウェアは GCC がベースとなっているため、極めて高精度なコード補完が実現できる。

10. 進捗概要

予定していた通り、以下を達成した:

- Emacs 拡張 auto-complete.el への必要な機能の追加。
- Ruby 言語のプログラムを高精度に補完するツール RSense の開発。
- C/C++言語のプログラムを高精度に補完するツール GCCSense の開発。
- これらツールを配布・広報するためのウェブサイトの用意、配布、利用者の獲得、利用者からのフィードバックの獲得。

さらに、次も達成した:

- 第三者からの貢献:RSense、GCCSense を Emacs 以外のエディタで利用するための、他のエディタに対する拡張機能。

■auto-complete.el

auto-complete.el はポップアップ型の自動コード補完を可能にする Emacs 拡張である。従来のコード補完機能はビジュアルにうったえたユーザーインターフェースではないため、補完結果が予測しづらいという問題があった。auto-complete.el はポップアップ補完(図 1)やインライン補完(図 2)といったビジュアルにうったえる現代的なユーザーインターフェースでそのような問題を解決する拡張と言える。

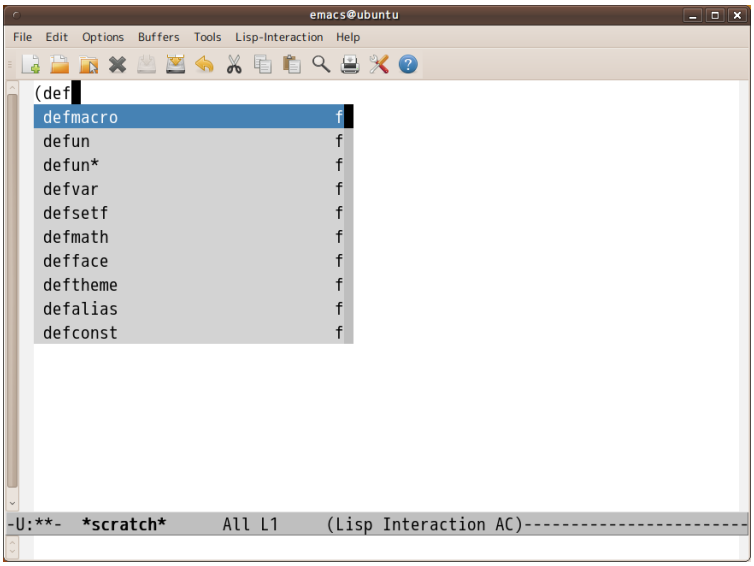

図 1. auto-complete.el によるコード補完（ポップアップ補完）

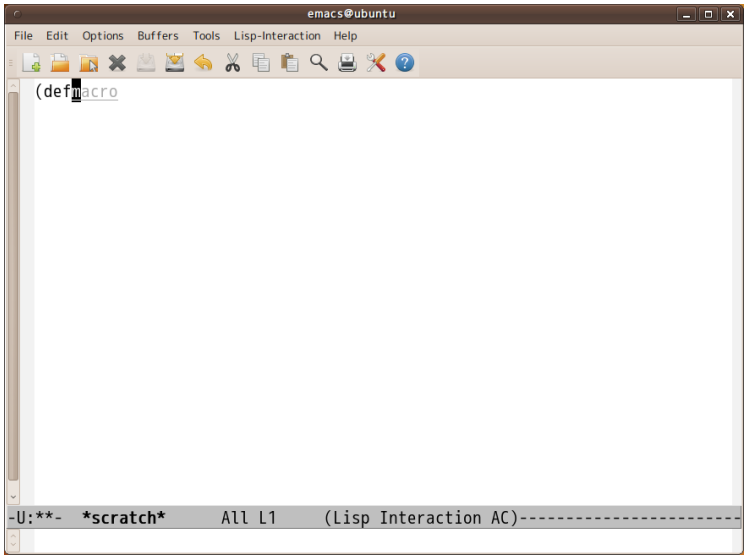

図 2. auto-complete.el によるコード補完（インライン補完）

しかし本プロジェクト以前の auto-complete.el には使いやすさや機能にいくらか問題があった。そこで以下の改善および拡張を行った。

1.曖昧マッチによる補完

曖昧マッチによりタイポに対しても適切な補完ができるようになった。

2.辞書による補完

辞書を利用して補完できるようになった。これにより特定のキーワード(例えば

C++における reinterpret_cast)を補完できるようになった。プログラミング言語用の辞書として現在 12 個の辞書を提供している。ユーザーによる定義も可能。

3.インクリメンタル検索

中間一致による補完候補の検索が可能になった(図 3)。

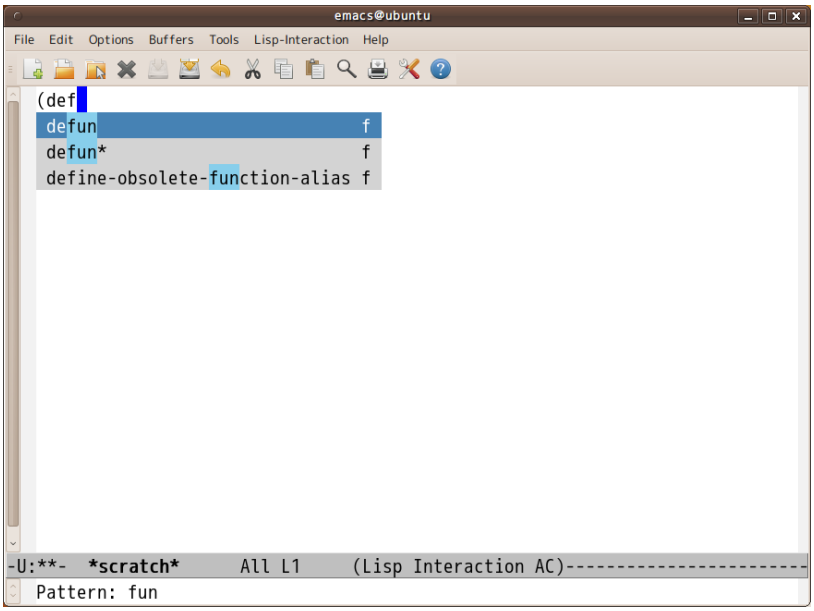

図 3. インクリメンタル検索

4.頻度表に基づく補完候補の並び換え

従来の auto-complete.el では頻繁に入力する補完候補に対しても特別なスコアリングを行っていなかった。そのため時間とともに下がり続けるべき補完コストは常に一定であった。そこでユーザーの補完活動を監視し、補完候補に対して適切にスコアリングすることで、補完コストの低減を可能にした。なお後述のインライン補完とポップアップ補完で若干の挙動の違いがある。

5.ポップアップヘルプ

Eclipse や Visual Studio などの IDE によくある機能として補完候補のポップアップヘルプがある。これがあれば補完候補、例えばメソッドのヘルプをシームレスに表示できる。これはいずれ必須になるので実装した(図 4)。

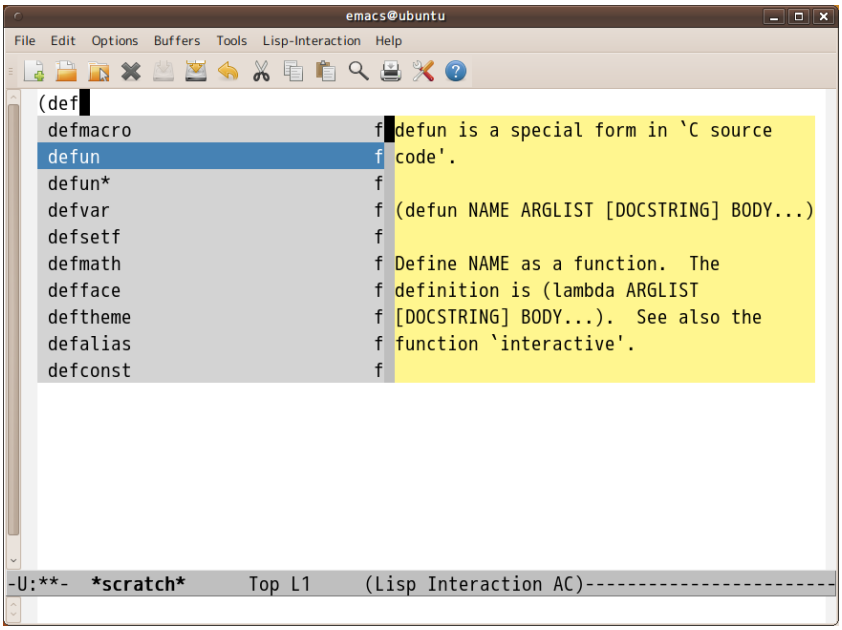

図 4. ポップアップヘルプ

6.補完の遅延

従来は文字の入力と同時に補完が実行されたため、いくらかのもたつきがあり、ストレスの元になっていた。そこで文字入力をトリガーとして、ある程度遅延させてから補完を行うことで、この問題を解決した。

7.補完メニューの遅延

補完の遅延と同様、補完ポップアップメニューの表示も遅延させるようにした。これによりポップアップメニューの表示が極力抑えられ、文字の入力に集中できるようになった。なお auto-complete.el ではこの表示状態をインライン補完と呼び、ポップアップメニューが表示されている表示状態のポップアップ補完と区別している。

本プロジェクト期間中に以上の作業を行った v1.2 と v1.3 をリリースした。また日本語および英語のマニュアルの作成および整備を行い、ユーザーのフィードバックの対応を行った。なお、過去 4 ヶ月間のダウンロード数は 4036 件である。

加えて、auto-complete.el からポップアップ表示部分を popup.el という拡張に分離した。Auto-complete.el の動作自体は変わらないが、従来の Emacs では実現不可能であったポップアップメニュー機能やツールチップ機能を独立させたことの意味は大きいであろう。事実、popup.el を利用した辞書ツールチップ表示機能などが登場しはじめている。

■RSense

Rubyの高精度コード補完機能としてRSenseを開発した。動的型付け言語であるRuby では、コーディング時に高精度コード補完を行うのは極めて困難である。そこで型推論の技術を応用することで、実行時に取るであろう型を静的に解析し、高精度コード補完を可能にした。

RSense は単独のコマンドラインプログラムとして提供されるように設計した。これにより Emacs のみでなく Vim や TextMate などのテキストエディタで利用することが可能になる。コマンドラインから RSense を利用する例を示す。

```
$ rsense code-completion --file=a.rb --location=1:2
completion: > Numeric#> Numeric METHOD
completion: id2name Fixnum#id2name Fixnum METHOD
completion: object_id Object#object_id Object METHOD
completion: frozen? Object#frozen? Object METHOD
completion: zero? Fixnum#zero? Fixnum METHOD
...
```

RSense で利用した型推論アルゴリズムは CPA(Cartesian Product Algorithm)である。CPA は具体型をグラフに飽和させるアルゴリズムをベースとし、関数呼び出しをメモ化で高速化するアルゴリズムである。RSense ではこの CPA に若干の手を加えて、データポリモーフィズムの扱いおよびインクリメンタル解析を可能にした。これにより精度を向上させ、また解析速度も現実的な時間に収まるようになった。

図 5 は auto-complete.el を利用して RSense で高精度コード補完を行った様子である。

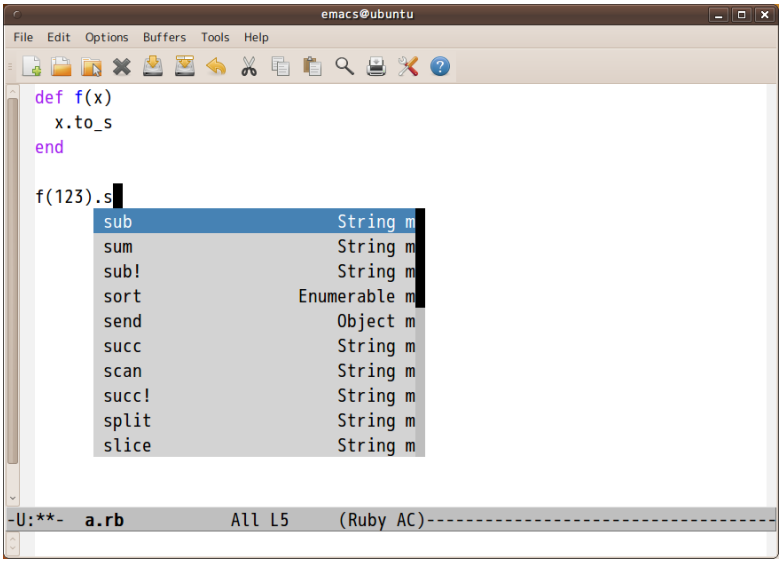

図 5. RSense によるコード補完

また、Ruby のドキュメント検索プログラムと連携することによる補完候補のポップアップヘルプが可能になった（図 6）。

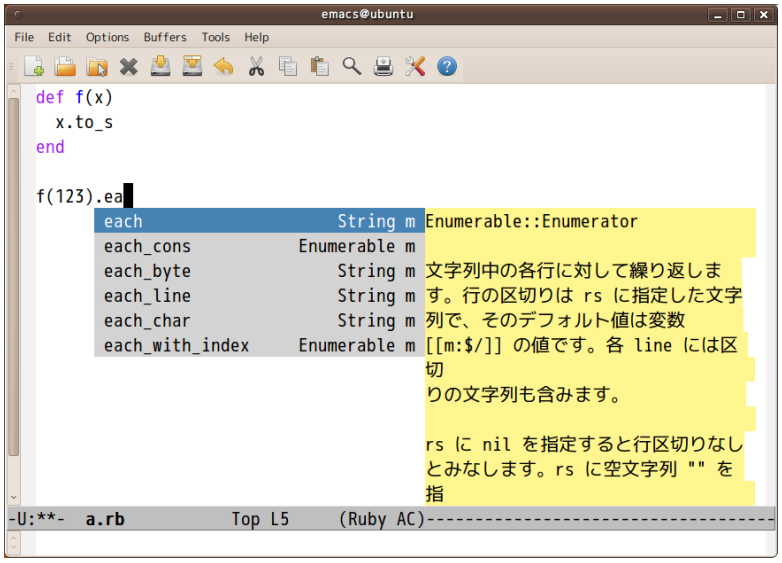

図 6. RSense におけるポップアップヘルプ

さらに追加的に型表示機能や定義元ジャンプ機能を実装した。

前述したように RSense はさまざまなテキストエディタから利用可能である。本プロジェクトでは Emacs の拡張と Vim のプラグインを作成したが、意図せず全くの第三者が他のテキストエディタである TextMate、Redcar、Xyzzy、秀丸のサポートを行ってくれた。

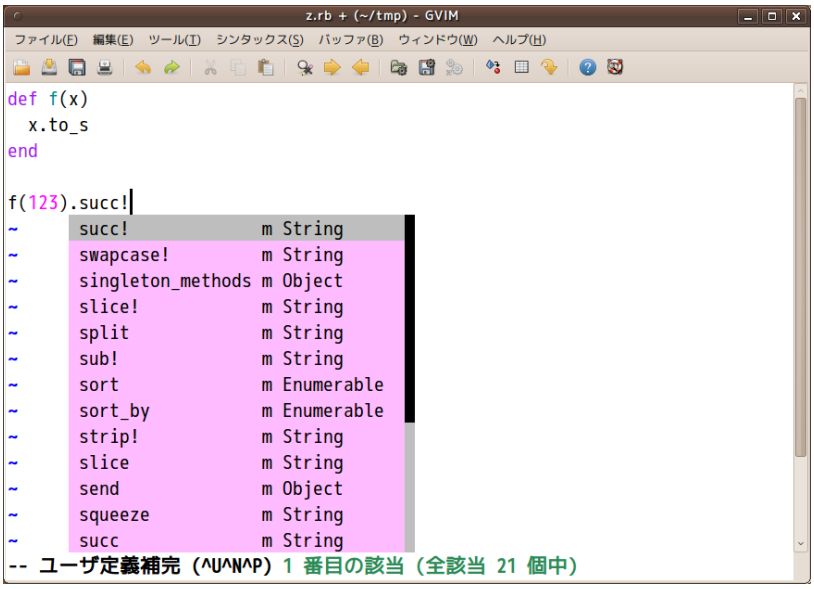

図 7. RSense+Vim

本プロジェクト期間中にv0.1、v0.2、v0.3をリリースした。v0.1リリース時に、「はてなブックマーク」と呼ばれるソーシャルブックマークでホットエントリーになった（かなり多くの注目を浴びた）。また日本語および英語のマニュアルの作成および整備を行い、ユーザーのフィードバックの対応を行った。なお、過去 4 ヶ月間のダウンロード数は 1740 件である。

■GCCSense

C/C++の高精度コード補完機能として GCCSense を開発した。このソフトウェアは GCC がベースとなっているため、極めて高精度なコード補完が実現できる。

このソフトウェアは厳密には以下の 3 つのプログラムから構成される。

・コード補完機能などを実装したカスタム GCC
・GCC の実行を記録および再生する gccrec
・自動的にプリコンパイルヘッダを適用する autopch

カスタム GCC は本質的には GCC であるため、GCC が利用できる環境であればどこでも利用可能である。コマンドラインからカスタム GCC を利用する例を示す。

```
$ g++-code-assist -fsyntax-only -c -code-completion-at=a.cpp:6:7 a.cpp
completion: _M_dataplus std::basic_string<char, std::char_traits<char>, std::allocator<char> >::_Alloc_hider std::basic_string<char, std::char_traits<char>, std::allocator<char> >::_M_dataplus
completion: _M_data _CharT* std::basic_string<_CharT, _Traits, _Alloc>::_M_data() const [with _CharT = char, _Traits = std::char_traits<char>, _Alloc = std::allocator<char>]
completion: _M_data _CharT* std::basic_string<_CharT, _Traits, _Alloc>::_M_data(_CharT*) [with _CharT = char, _Traits = std::char_traits<char>, _Alloc = std::allocator<char>]
completion: _M_rep std::basic_string<_CharT, _Traits, _Alloc>::_Rep* std::basic_string<_CharT, _Traits, _Alloc>::_M_rep() const [with _CharT = char, _Traits = std::char_traits<char>, _Alloc = std::allocator<char>]
completion: _M_ibegin __gnu_cxx::__normal_iterator<typename _Alloc::rebind<_CharT>::other::pointer, std::basic_string<_CharT, _Traits, _Alloc> > std::basic_string<_CharT, _Traits, _Alloc>::_M_ibegin() const [with _CharT = char, _Traits = std::char_traits<char>, _Alloc = std::allocator<char>]
...
```

コード補完といっても実際はコンパイルを行うため、コンパイルオプション（マクロ定義やインクルードパス）に誤りがあると正しくコード補完できないことがある。しかし一般的に、テキストエディタは編集中のファイルの正しいコンパイルオプションなどは知り得ない。つまりこのカスタム GCC だけでは高精度コード補完は実現できない。それを解決するために gccrec という GCC の実行を記録および再生するプログラムを開発した。gccrec をコマンドラインから利用する例を示す。

```
$ # 記録
$ gccrec g++-code-assist -DSOME_IMPORTANT_VAR -I/path/to/include -c a.cpp
$ # 再生
$ gccrec -r -fsyntax-only -code-completion-at=a.cpp:6:7 a.cpp
```

簡単に言えば gccrec はファイル名とコンパイルオプションの対応をデータベースに記録し、必要に応じてコンパイルコマンドを再生するプログラムである。テキストエディタはこの gccrec を利用して正しいコンパイルオプションを取得して高精度コード補完を行う。make などを利用するプロジェクトでは次のように記録することができる。

```
$ make CC='gccrec gcc-code-assist' CXX='gccrec g++-code-assist'
```

この手法により Firefox、WebKit、Google V8 などの巨大なプロジェクトで高精度コード補完が可能なことを確認した。ただ、これだけではパフォーマンスに問題があることが分かった。C や C++というプログラミング言語の特質上、プリプロセス後のソースコードが非常に巨大になる（例えば数十万行）傾向がある。コード補完対象のソースコードがいくら小さくてもインクルードしているヘッダーファイルが巨大になっては、コンパイル速度に影響がでるのは当然である。実はこのような問題はプリコンパイルヘッダーを利用すれば解決できる。ただし GCC のプリコンパイルヘッダーには重大な制限があり、ヘッダーファイルを一つにまとめるなどの非現実的なソースコードの変更が要求される。そこでそのようなソースコードの変更を自動的に行ってプリコンパイルヘッダーを利用できるようにしてくれる autopch というプログラムを開発した。autopch を利用するにはコンパイルコマンドの前に autopch と書くだけである。

```
$ autopch gcc -DSOME_VAR -I/path/to/include -c a.c
```

プリコンパイルヘッダーを利用することで、コンパイル速度をかなり改善できる。Boost と呼ばれる C++の巨大なライブラリを利用する例では特に顕著であった（4.5 秒から 0.4 秒に改善）。

```
$ # autopch なし
$ time g++ -fsyntax-only -c test.cpp
g++ -fsyntax-only -c test.cpp  4.49s user 0.19s system 95% cpu 4.915 tota
$ # autopch あり初回
$ time autopch g++ -fsyntax-only -c test.cpp
autopch g++ -fsyntax-only -c test.cpp  10.16s user 0.97s system 96% cpu 11.531 total
$ # autopch あり二回目
```

```
$ time autopch g++ -fsyntax-only -c test.cpp
autopch g++ -fsyntax-only -c test.cpp  0.42s user 0.06s system 94%
cpu 0.504 total
$ # autopch あり三回目
time autopch g++ -fsyntax-only -c test.cpp
autopch g++ -fsyntax-only -c test.cpp  0.38s user 0.10s system 90%
cpu 0.524 total
```

以上のプログラムを利用して高速かつ高精度なコード補完を実現した。図 8 は auto-complete.el と GCCSense を利用して高精度コード補完を行った様子である。

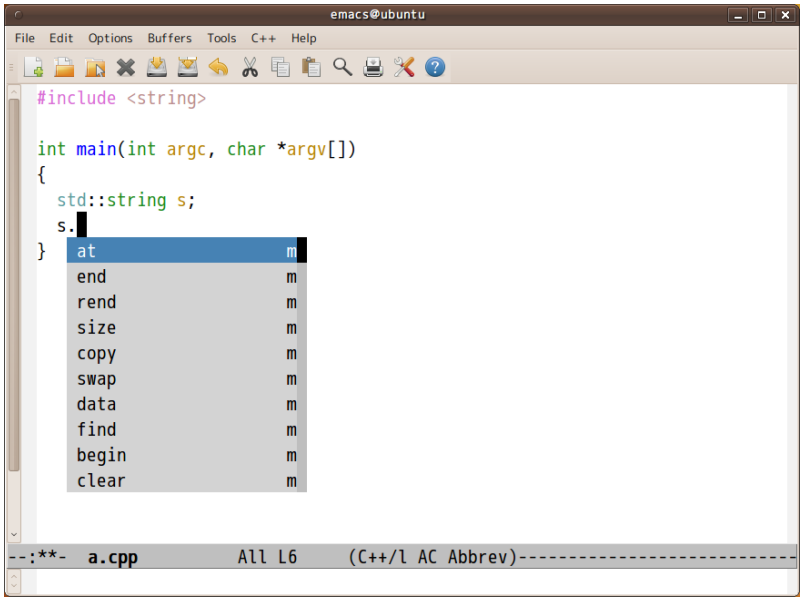

図 8. GCCSense によるコード補完

さらに宣言の表示、宣言元ジャンプ、定義元ジャンプ機能を実装した。

GCCSense はさまざまなテキストエディタから利用可能である。本プロジェクトでは Emacs の拡張と Vim のプラグインを作成したが、意図せず全くの第三者が Redcar と呼ばれるテキストエディタの対応を行ってくれた。

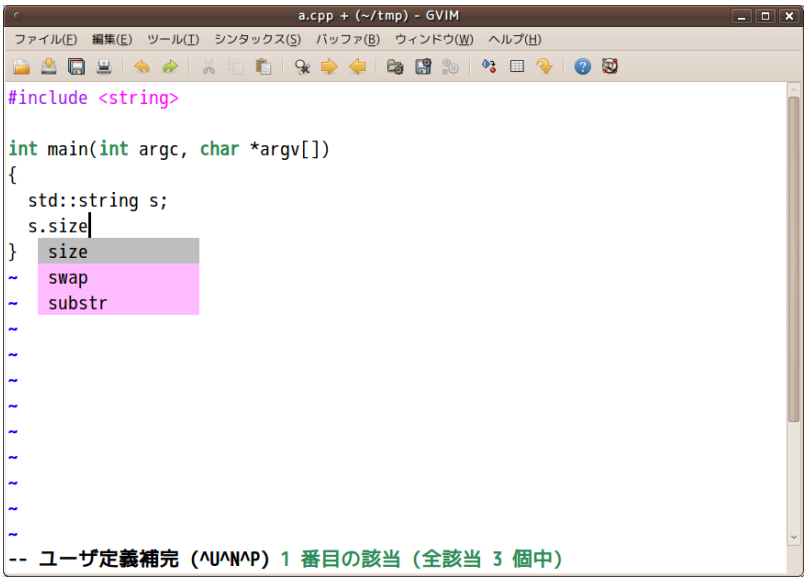

図 9. Vim+GCCSense

本プロジェクト期間中に v0.1 をリリースした。リリース後しばらくして「reddit」と呼ばれるソーシャルブックマークでトップエントリーになった（かなり多くの注目を浴びた）。また日本語および英語のマニュアルの作成および整備を行い、ユーザーのフィードバックの対応を行った。なお、過去 4 ヶ月間のダウンロード数は 701 件である。

12. プロジェクト評価

松山君は、Emacs という、利用者の多い、中でも特に腕の立つプログラマに利用者の多いエディタを対象として、これまでより使い勝手のいいコード補完機能を Ruby、C/C++言語に対して開発した。加えて、用意したウェブサイトを通じて配布、日本語・英語での広報を行い、一定数の利用者を獲得した。成果報告会でのデモンストレーションでは、PM も含めて、会場のほうぼうから「すごい」という感嘆の声があがった。

しかし本プロジェクトの成果はそれだけではない。今日、Eclipse や Emacs（の既定のコード補完機能）では、それらをいくら発展させても、恩恵を受けられるのはそのエディタの利用者に限られる。それに対して松山君は、他エディタの利用者も恩恵を受けられるようなソフトウェア構成法・開発方針を主張し、本プロジェクトにて実践した。その結果、実際に、プロジェクトの成果物であるコード補完機能を Emacs 以外の利用者が利用できるようになった。しかも、他エディタへの対応作業を、松山君ではない第三者が自発的に行ったのである。成果物の一部、Ruby 言語プログラムの補完ツール RSense は、松山君自身の手によって Emacs と Vim から利用可能となった。それに加えて、第三者が自発的に、TextMate、Redcar、Xyzzy、秀丸に対応させた。

松山君は、技術があり、それをもって人々に貢献する、というだけにとどまらず、心の中に、自分の信じるあるべき社会像があり、それを具体化するために自身の技術を使おうとしている。その実践まで至っている技術者は稀である。

13. 今後の課題

当クリエータは現在、Ruby、C/C++に続いてPython言語のコード補完技術に取り組んでいる。それと並行して、当プロジェクトの成果も普及し続けることだろう。普及には困難が伴うだろうが、それはありふれたことである。

課題らしい課題は、クリエータ自身の謙虚さ、慎重さからくる。成果の意義に対して、クリエータが謙虚であるため、成果や意義の宣伝・布教が積極性を欠くことを心配している。