

TDL Grammars in C-Sharp

MA work supervised by Emily Bender

Glenn Slayden
University of Washington

DELPH-IN Summit
July 2, 2010

Overview

- thai-language.com
- Matrix grammar of Thai
- C#, CLR, CLI, BCL...
- Project progress
 - GLB calculation
 - DAG representation
- This work is at a very early stage: not presenting results today

Another parser?

“The [DELPH-IN] community can always benefit from another parser.”

-Dan Flickinger

Goals

- Large-scale processing of TDL grammars
- Truly portable binaries between Windows, linux, and Mac
- Robust, commercial grade platform
- Easier to use, more accessible
- Performance: Byte code JIT-compiled to native
- Developer productivity: LINQ
- GUI improvement wishlist
 - Grammar visualization tools
 - Printing
 - TDL editor

Thai-language.com

- Started 1997
- Supervised Thai-English lexicon of 50,000 entries
- Message boards, reading exercises, lessons
- Custom-programmed site has followed the evolution of Windows Server technologies from ASP to .NET 4

The screenshot shows the Thai-language.com website in a Windows Internet Explorer browser window. The page layout includes a navigation menu at the top with links for Welcome, Lessons, Dictionary, Categories, Reference, Forums, Resources, and Store. A search bar is located on the left side. The main content area is divided into several sections:

- Welcome to thai-language.com:** A large section featuring an illustration of a person riding a bicycle with a speech bubble saying "สวัสดีครับ" (Hello). The text welcomes visitors and mentions 13131 audio clips, 47222 dictionary entries, 823 images, and a world-wide community.
- Quick Links:** A list of links including overview of the Thai language, dictionary searches, bulk lookup, new discussions, reference index, lesson index, and site settings.
- Site News:** A section dated June 9, 2010, thanking David for entering new content and expressing concerns for the safety and prosperity of Thailand.
- Mission Statement:** A section stating the mission of the website is to offer English speakers the highest-quality non-commercial Thai language resource on the web.
- Site Technology:** A section mentioning the website is powered by Microsoft and uses Internet Explorer.

The browser's address bar shows the URL http://www.thai-language.com. The status bar at the bottom indicates the site is trusted and protected mode is off.

Matrix Grammar of Thai

- Emily's Ling567 Grammar Engineering, Winter 2009 plus follow-on work
- Phenomena modeled include simple treatments of:
 - aspect auxiliaries
 - numeric-classifiers
 - demonstrative-classifier interaction
 - pro-drop
 - questions
 - adverbs, adjectives
 - copula
 - declarative and interrogative complementizer
 - negation

Current integration between Matrix grammar and thai-language.com

- TDL lexicon emitted
- Testsuite coverage report
 - <http://www.thai-language.com/testsuite-results>
- LKB integration (spotty) uses LKB socket interface
 - <http://www.thai-language.com/id/219435/parse>
 - MRS results bug
 - No UTF8 support; uses numeric lexicon
 - Frequent LKB reboots
- Problems with Thai display in LKB, Emacs, and [incr tsdb()]

ECMA-335: Common Language Infrastructure

- Portable binaries!
- Flagship programming language: C#
- F#, Python, Ruby, VB, PowerShell, ...
 - http://en.wikipedia.org/wiki/List_of_CLI_Languages
- C++/CLI supports mixed mode
- Fully managed: verifiable code, garbage collection, strong guarantees
- Modern concurrency and threading
- 100% Unicode; outstanding legacy encoding support

Common Type System (CTS)

- Traditional OO paradigm:
 - Fields
 - Members
 - private/public
- Single-object, multiple-interface inheritance
- Polymorphism
- Objects freely interoperate between languages and disparate applications

C#

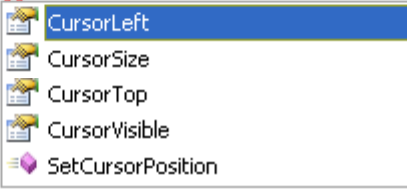
- Vaguely ANSI-C syntax
- Like all CLI languages: strongly-typed
- No pointers, but:
 - supports CLI value types – require no garbage collection: important for performance
 - single-indirection: value types and references can be passed by reference

Developer Productivity

- “Intellisense” code completion
- Vast Base Class Libraries (BCL)
- Strongly typed “generic” classes (templating)

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace Console1
{
    class Program
    {
        static void Main(string[] args)
        {
            Console.CU
        }
    }
}
```



The image shows a code editor window with a dropdown menu open for the text `Console.CU`. The dropdown menu lists the following options: `CursorLeft`, `CursorSize`, `CursorTop`, `CursorVisible`, and `SetCursorPosition`. The `CursorLeft` option is currently selected and highlighted in blue.

LINQ

- Adds functional-style programming to the imperative paradigm
- Functional operations on sequences:
 - aggregate, partition, set, project, join, restrict, select, generate, ...
- Deferred execution
- Cross-CLI, but most richly exposed in F# and C#
- Optional SQL-like syntax can be used in C#

LINQ example

```
String[] items = { "cat", "pear", "apple", "cat", "banana",  
    "pear", "pear", "apple" };  
  
foreach (var pair in items  
    .GroupBy(k => k)  
    .OrderBy(grp => grp.Key)  
    .Select(grp => new  
        {  
            word=grp.Key,  
            count=grp.Count()  
        }  
    ))  
    Console.WriteLine(pair.word + " " + pair.Count);
```

apple 2

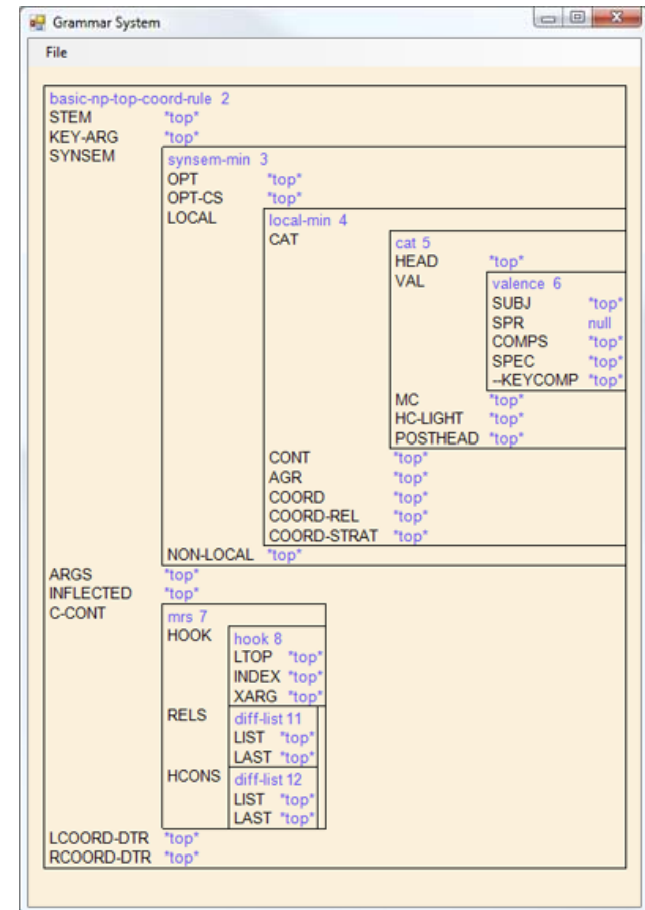
banana 1

cat 2

pear 3

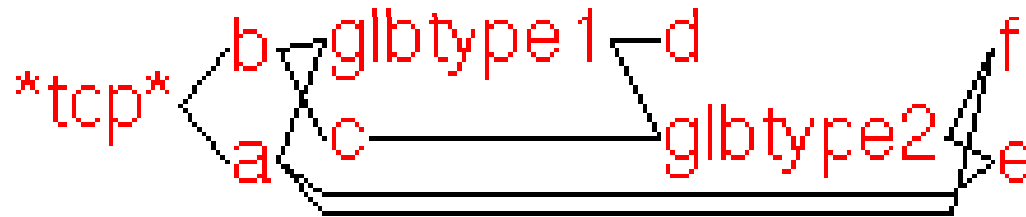
So far

- Robust TDL parser
- Builds type system
- Calculates GLB closure
- Maximal inference for authored types
- DAG representation
 - discussion follows



GLB calculation

```
a := *top*.
b := *top*.
d := a & b.
c := b.
e := a & c.
f := a & c.
```



LKB result:

12 edges

a-f and a-e are redundant

TGCS result

top:

children: { a b }

a:

children: { glbtype1 }

b:

children: { glbtype1 c }

d:

c:

children: { glbtype2 }

e:

f:

glbtype1:

children: { glbtype2 d }

glbtype2:

children: { f e }

10 edges

none redundant

GLB calculation

- Assign bit codes according to Ait-Kaci (1989):
 - every type gets a unique bit position
 - top == -1
 - code is logical 'or' of its child values (assign from leaf to top)
- glbtypes are given by non-zero logical 'and'
- so far so good

bit twiddling

Looking only at GLBs, we can directly obtain the transitive closure of the graph by carefully manipulating their parents and children. For each GLB:

1. Consider it as a parent: add its children
 - 1a. get a list of draft candidates: all descendants of 'node' The order in which these are added is important because it determines the order of testing children in the next sub-step. The list is needed because children may be eliminated from the list before the point at which they'd otherwise be added.

bit twiddling (cont.)

1b. pick the subset of immediate children from this list.
While the list is not empty, add the oldest item as a child and then remove all of its descendants from the list.

2. Consider it as a child. Add its parents

2a. get a list of draft candidates between 'node' and *top*

2b. pick a minimal set of parents from this list.

Each selection allows us to eliminate others from the list. This effect is maximized by starting with candidates with the fewest additional bits set beyond the required match.

DAG Literature Review (selected)

- Structure-sharing (Pereira 1985)
- Non-destructive (Wroblewski 1987)
- Quasi-destructive (Tomabechi 1991)
 - Over-copying
 - Early copying
- Thread safe (van Lohuizen 2000)
 - parallel unification
 - concurrent unification

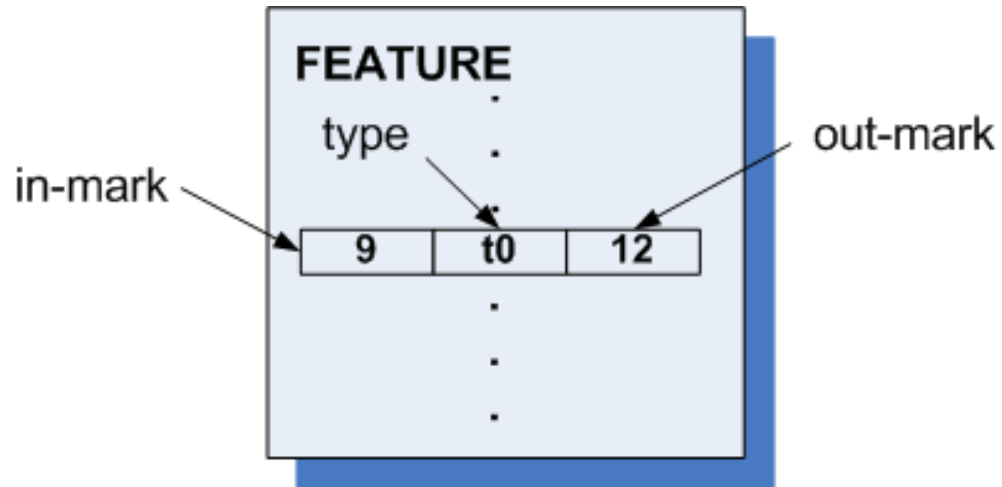
DAG representation

- DELPH-IN TFS grammars have notable features:
 - Fixed type hierarchy
 - A feature can only be introduced in exactly one place in the type hierarchy
 - When unification is successful, all features remain appropriate for their types
- Can we capitalize on these observations, plus intuition about the efficient use of C# value types to improve unification performance

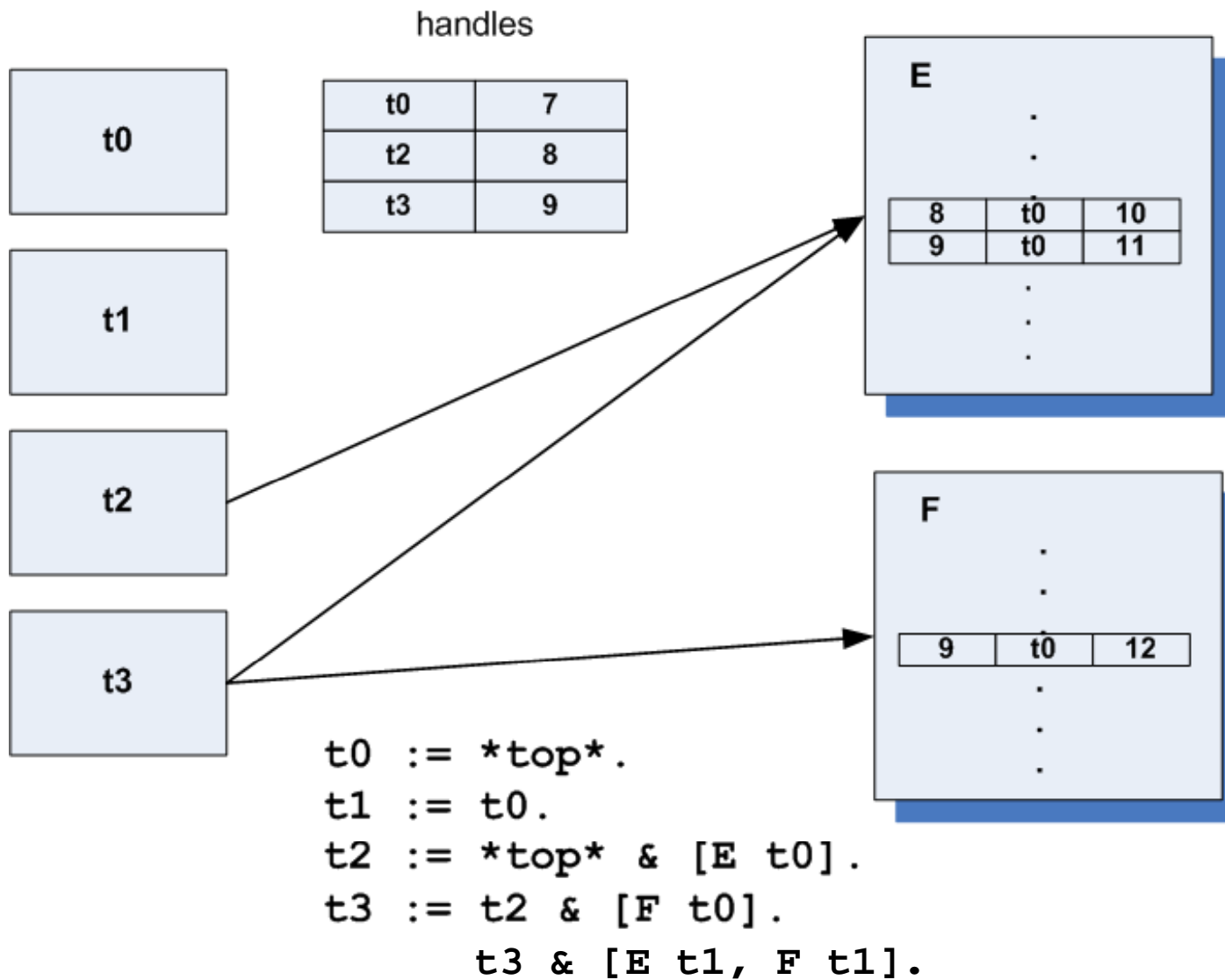
“feature-centric” TFS representation

- Consider two types of TFS edges:
 - Feature-appropriate-for-type
 - These are fixed for the life of the grammar
 - Precomputed and associated with types
 - Value of feature ‘F’ in TFS node of type ‘t0’ constrained by type ‘t1’
 - These edges are manipulated at runtime
 - All edges of this type are stored with feature ‘F’
- Successful unification (simple case): no edges are “created” or “destroyed”

triple description



Example



Intention of this design

- 'generic' list datatypes maintain previously-allocated capacities
- by using generic lists of C# 'value types' rampant DAG copying is expected to invoke minimal (or no) GC activity

Continuing work

- This development environment has very rich concurrency support: consider and seize upon opportunities that present themselves
- Availability of rich test cases (ERG, Thai matrix grammar) facilitates rapid development
- Next step: parsing

References (1/2)

- Hassan Ait-Kaci, Robert Boyer, Patrick Lincoln, Roger Nasr. 1989. "Efficient Implementation of Lattice Operations"
- Ulrich Callmeier. 2001. Efficient Parsing with Large-Scale Unification Grammars. MA Thesis, Universität des Saarlandes - Fachrichtung Informatik.
- Ulrich Callmeier. 2000. PET: a platform for experimentation with efficient HPSG processing techniques. *Natural Language Engineering* 6(1): 99-107.
- Bernd Kiefer, Hans-Ulrich Krieger, John Carroll, and Rob Malouf. 1999. A Bag of Useful Techniques for Efficient and robust Parsing. In *Proceedings of the 37th annual meeting of the Association for Computational Linguistics*. 473-480
- Robert Malouf, John Carroll, and Ann Copestake. 2000. Robert Malouf, John Carroll, and Ann Copestake. 2000. Efficient feature structure operations without compilation. *Natural Language Engineering*, 1(1):1-18.
- Fernando C. N. Pereira. 1985. A structure-sharing representation for unification-based grammar formalisms. In *Proceedings of the 23rd Annual Meeting of the Association for Computational Linguistics*. Chicago, IL, 8-12 July 1985, pages 137-144.
- Hideto Tomabechi. 1991. Quasi-destructive graph unification. In *Proceedings of the 29th Annual Meeting of the Association for Computational Linguistics*, Berkeley, CA.

References (2/2)

- Hideto Tomabechi. 1992. Quasi-destructive graph unifications with structure-sharing. In *Proceedings of the 15th International Conference on Computational Linguistics (COLING-92), Nantes, France*.
- Hideto Tomabechi. 1995. Design of efficient unification for natural language. *Journal of Natural Language Processing*, 2(2):23-58.
- Marcel P. van Lohuizen. 1999. Parallel processing of natural language parsers. In *PARCO '99*.
- Marcel P. van Lohuizen. 2000. Exploiting parallelism in unification-based parsing. In *Proceedings of the Sixth International Workshop on Parsing Technologies (IWPT 2000), Trento, Italy*.
- Marcel P. van Lohuizen. 2000. Memory-efficient and Thread-safe Quasi-Destructive Graph Unification. In *Proceedings of the 38th Meeting of the Association for Computational Linguistics, Hong Kong, China, 2000*.
- Marcel P. van Lohuizen. 2001. A generic approach to parallel chart parsing with an application to LinGO. In *Proceedings of the 39th Meeting of the Association for Computational Linguistics, Toulouse, France*.
- David A. Wroblewski. 1987. Nondestructive graph unification. In *Proceedings of the 6th National Conference on Artificial Intelligence (AAAI-87)*, 582-589. Morgan Kaufmann.