Automatisch kreativ

pro et con

Vor- und Nachteile toolgestützter Sprachmigrationen

Anliegen des Vortrags

Automatisch kreativ

- Automatisch versus kreativ ?
- Automatisch und kreativ ?
- Ist toolgestützte Sprachmigration automatisch kreativ ?

Typische Situation

- Programmbestand seit ca. 30 Jahren gewachsen
- Einsatz älterer oder proprietärer Programmiersprachen
- Immer weniger Experten verfügbar
- Dokumentation meist nicht im besten Zustand
- Entwickler der Programme meist nicht mehr zugreifbar

Typische Situation

Programme und Datenbestände:

- Beinhalten ein enormes Wissen über betriebliche Abläufe
- Sind erprobt; Fehler wurden behoben; Mitarbeiter sind an die Programme gewöhnt
- Stellen somit einen beachtlichen Wert dar (Erstellung und Pflege war/ist teuer)

Neue Anforderungen

- Einbindung in Internet-Umgebungen (Browser als Bedienoberfläche, neue Sprachen - Java)
- Einsatz neuer, moderner (objektorientierter) Sprachen plattformübergreifend
- Schnittstellen zu Standardsoftware
- Neue Hardware, die bestmöglich genutzt werden soll
- Ständiger Zwang zu Verbesserung und Beschleunigung betrieblicher Abläufe

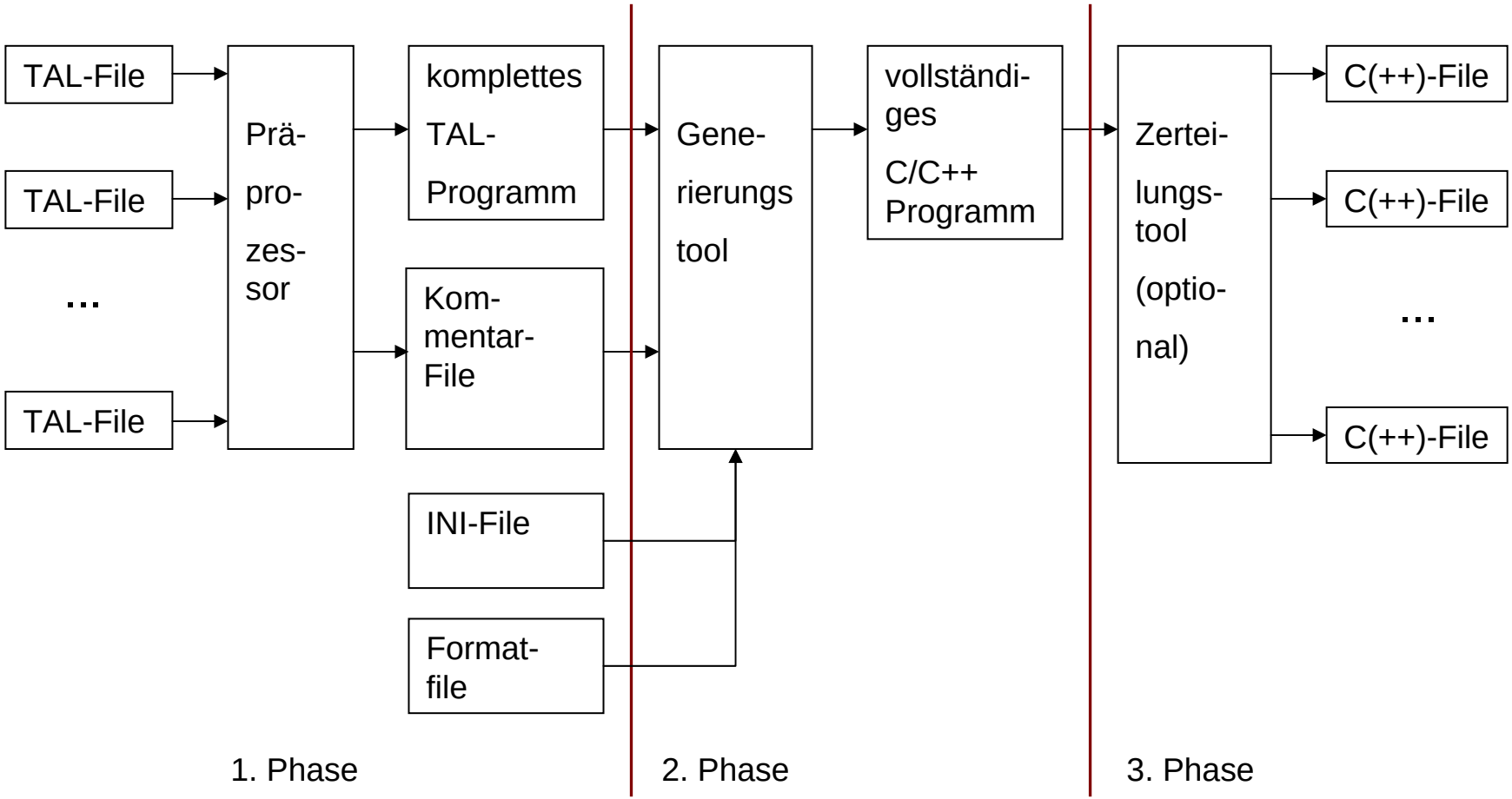
Verschiedene Wege
zur Realisierung der Anforderungen

1. Neuentwicklung
2. „Wrapping“ – kapseln und hinzufügen neuer, definierter Schnittstellen
3. Ersatz durch Standardsoftware
4. manuelle Migration
5. Toolgestützte Konvertierung mit Hilfe von Translatoren

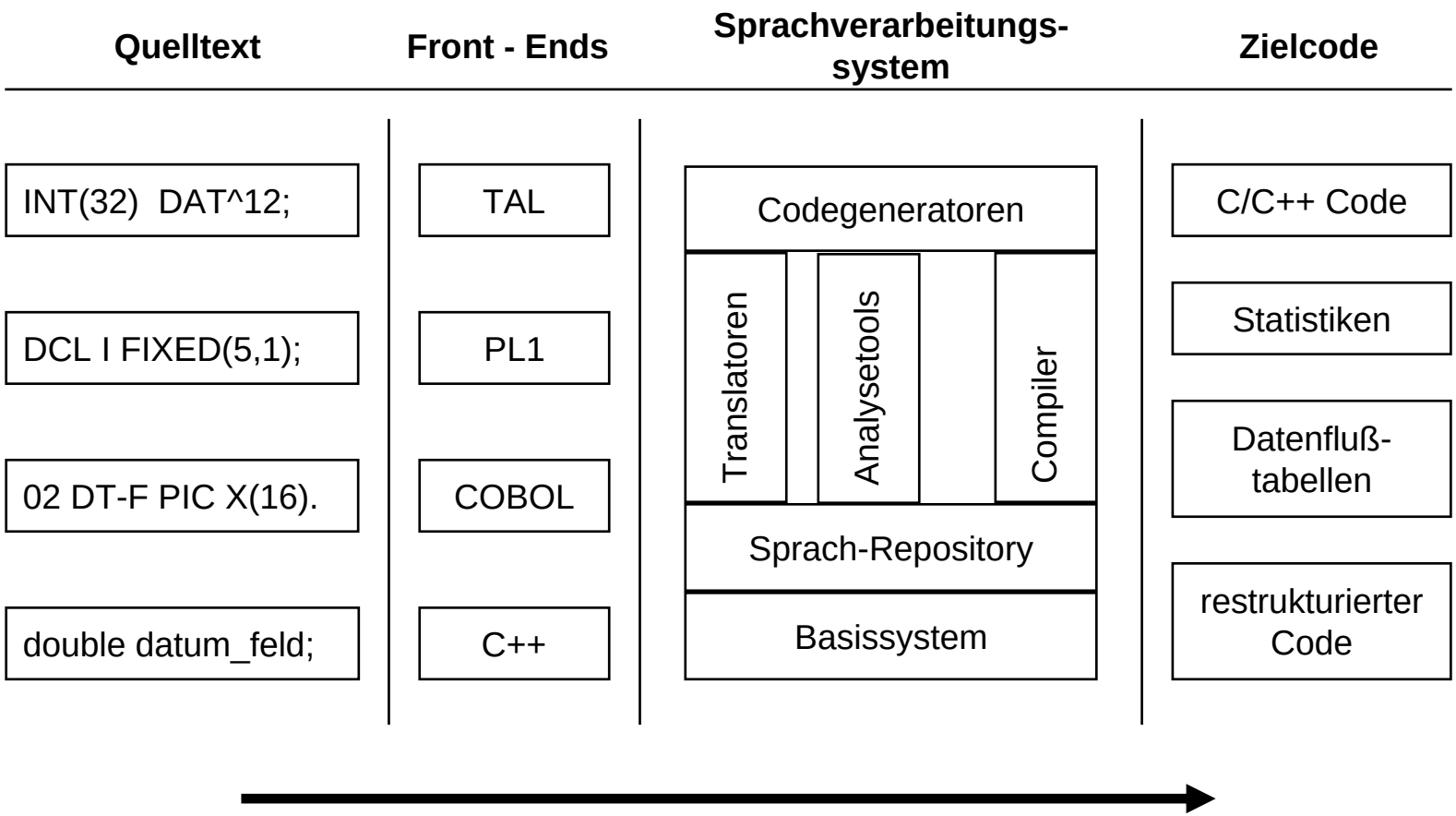
Aufbau eines Translators

- Am Beispiel des TAL → C/C++ Translators
- Automatisch erzeugter Code muß von Programmierern wartbar sein!
- Translator: arbeitet wie ein Compiler, aber muß noch weitere Aufgaben realisieren
  - Kommentare übernehmen
  - Flexible Strukturierung nach Kundenwunsch
  - Erzeugen mehrerer Zieldateien

Schematischer Überblick



Toolset zur Erstellung von Translatoren u.a.



Ausgangssituation einer Migration

Programme und Bibliotheken in der Quellsprache

- Enthalten Kommentare, Direktiven und Programmcode
- Meist über mehrere Files verteilt (**?SOURCE** bei TAL, **COPY** bei Cobol)
- Sind mit dem Compiler der Quellsprache übersetzbar, also als korrekt anzusehen

Vorbereitung

- Quelltext verteilt über verschiedene Maschinen, Volumes und Subvolumes bzw. Verzeichnisse
- Zusammenstellung aller benötigten Quelltextfiles „von Hand“ ist mühsam
- deshalb: Programm **`FindFile`**
  - erkennt Crossreferenzen zwischen verschiedenen Quelltextfiles
  - stellt Liste aller benötigten Files zusammen
  - kopiert diese auf die „Translationsmaschine“

1. Phase: Präprozessorlauf

- Files zu einem vollständigen Programm in einer Datei zusammenziehen
- Kommentare entfernen und in Kommentarfile speichern
- Direktiven auflösen und (z.T.) behandeln (z.B. ?IF, ?SEARCH, ?SOURCE)
- Vorführung 1. Teil: Präprozessor

2. Phase: Generierung

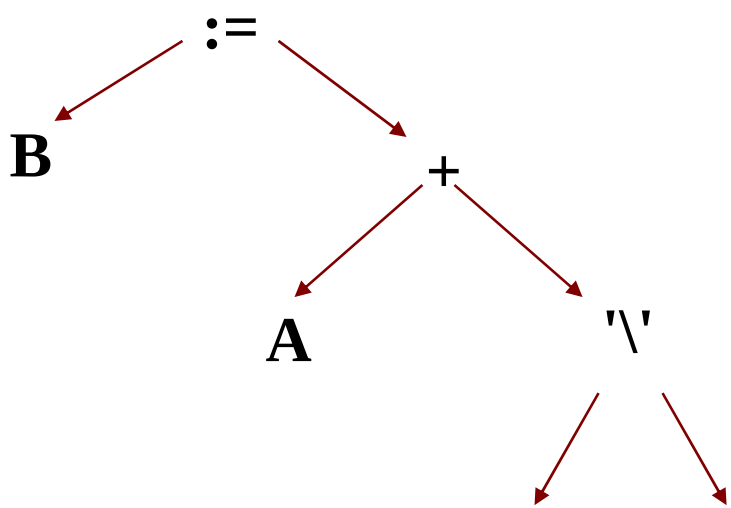
- Besteht aus 3 Teilphasen:
  - READ-Phase
  - TRANSLATE-Phase
  - WRITE-Phase
- Realisiert die eigentliche Konvertierung in den Zielcode

Generierung --- READ-Phase

- Einlesen und Parsen
- Realisiert durch:
  - Scanner (Zerlegung in einzelne TAL-Worte, Zuordnen der Wortart)
  - Parser (Aufbau der internen Programm-repräsentation)
- Interne Darstellung des Programms: attributierte Syntaxbäume

Beispiel: Syntaxbaum

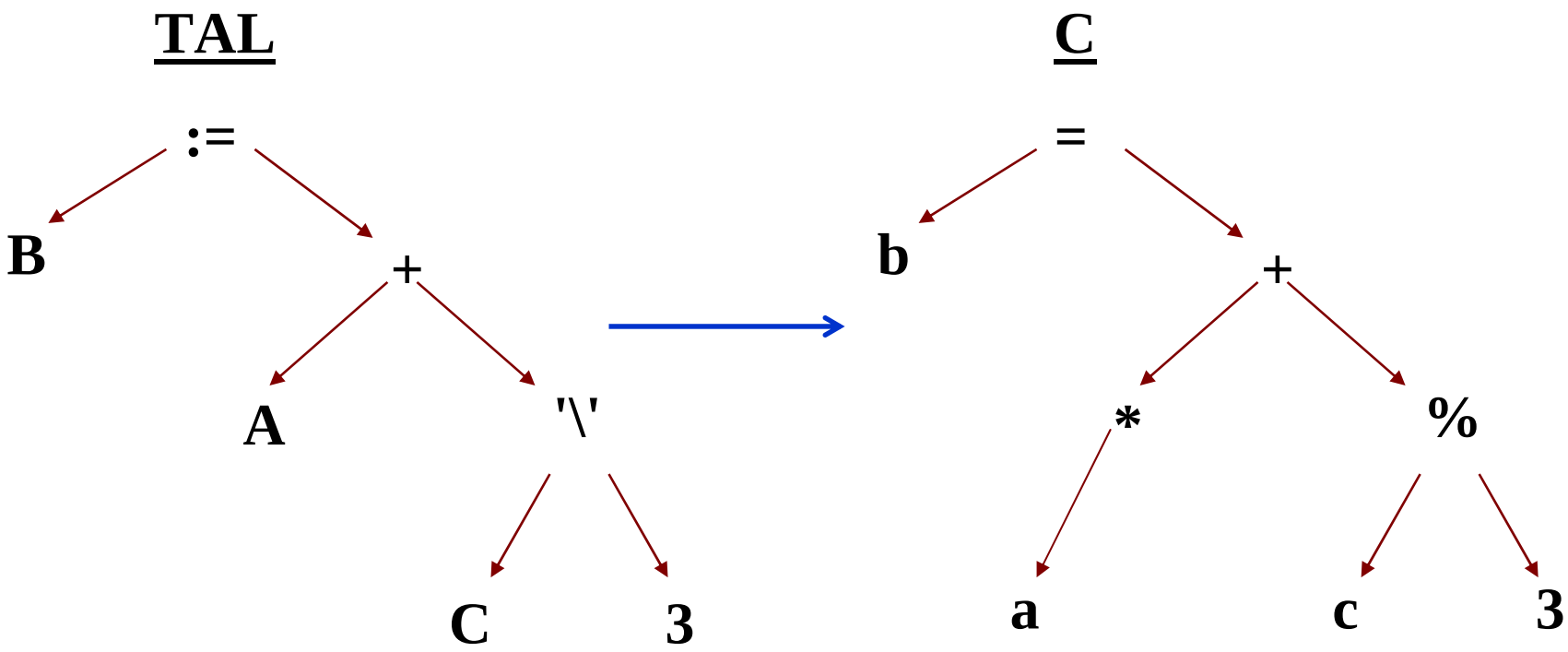

```
INT .A, B, C; ...
B := A + C ‘\‘ 3;
```



Generierung --- TRANSLATE-Phase

- Aufbau von Syntaxbäumen der Zielsprache (hier C oder C++) aus den Syntaxbäumen der Quellsprache (hier TAL)
- Verwendet „pattern matching“ durch Baumersetzungsregeln
- In dieser Phase sehr viele Design-Entscheidungen und der meiste Aufwand

Konvertierung des Beispielbaums



Generierung --- WRITE-Phase

- Schreibt die Syntaxbäume der Zielsprache (C/C++)
- gewissermaßen Gegenteil des Parsers
- Berücksichtigt Einstellungen zur Formatierung aus dem Einstellungsfile (viele Einflußmöglichkeiten auf das Erscheinungsbild des erzeugten Programms)
- Fügt Kommentare aus Kommentarfile wieder ein
- Ergebnis: ein compilierbares C/C++ File
- Vorführung 2. Teil: Translator

3. Phase: Zerteilung

- Ziel: Wiederherstellung der Modulstruktur analog zur Ausgangssituation (Aufteilung auf mehrere Files)
- Ermöglicht Wartung (Pflege einer mehrfach verwendeten Bibliothek zentral, man erkennt die Struktur wieder, ...)
- Für den Zerteiler werden Informationen in Form von „Spezialkommentaren“ eingefügt. (Wo stand diese Prozedur, Deklaration, ... im Quellcode?)
- Generierung von Zielfiles mit gleichem Namen

Aspekte bei der Umsetzung TAL → C/C++

- „Womit muß man sich dabei so auseinandersetzen?“
- Reihenfolge stellt keine Wertung dar
- Mit diesen Aspekten wird man auch teilweise bei Konvertierung von Hand konfrontiert. (Zumindest, wenn in Auftragsarbeit 1:1 übersetzt wird)!

DEFINES

- Quelltextersetzungen; werden vor der READ-Phase ausgeführt
- Sind im Baum also nicht mehr vorhanden → werden nicht wieder rücktransformiert
- Beispiel:

```
DEFINE Q(X) = X*X#;
A := Q(B) + Q(C);
```

wird in C/C++ zu

```
a = b*b + c*c;
```

Verbesserungen der DEFINE-Konvertierung

1. Ist der DEFINE-Körper (zwischen **=** und #) eine Konstante, so wird ein analoges **#define** erzeugt.
2. Ist der DEFINE-Körper ein kompletter Ausdruck oder eine komplette Anweisung, so kann optional ein Makroaufruf anstelle des expandierten Textes generiert werden.

   Beispiel: **`a = Q(b) + Q(c)`**

   Makro **Q** muß aber von Hand erzeugt werden.

Prozedurvariablen

- In C/C++ keine Subprozeduren; Funktionen alle global
- Aufpassen bei Namensgebung (2 Subprozeduren könnten den gleichen Namen haben)
- Subprozedur kann auf Namen der Prozedur zugreifen (Diese Variablen alle globalisieren.)
- Ruft die Prozedur sich selbst, muß der Inhalt dieser Variablen gerettet werden

Beispiel für Prozedurvariablen

TAL-Code:

```
PROC RUN;
BEGIN INT POS := 0;
      INT SUBPROC MAL2;
      BEGIN RETURN POS+POS; END;
      ...
      POS := MAL2;
      ...
END;
```

Beispiel nach der Konvertierung

Code in C/C++:

```
short run_pos;
short run_mal2() { return run_pos+run_pos; }
void run() {
   /* merke akt. Inhalt von run_pos */
   run_pos = 0; ...
   run_pos = run_mal2(); ...
   /* gemerkter Inhalt wieder in run_pos */
}
```

Hardware- und Systemabhängigkeiten

- Zu folgenden systemnahen TAL-Konstrukten keine entsprechenden Befehle in C/C++:
  - CODE-, STACK-, STORE-, USE-, DROP-Statement
  - $CARRY, $OVERFLOW (Standardfunktionen)
  - Adressierung relativ zu L-, S- und G-Register
  - ARMTRAP (Guardian-Funktion)
- Ausgabe einer Warnung
- muß von Hand bereits im TAL-Quelltext geändert werden (Translation ist ein iterativer Prozeß)
- Dazu gibt es einen festen Algorithmus.

SQL Statements

- EXEC SQL ... END EXEC;
- Sehen in TAL und in C/C++ ähnlich aus
- Namen der Hostvariablen werden vom Translator angepaßt
- SQL-INVOKE wird erkannt und korrekt umgesetzt

Kommentareinfügung

- Kommentare entweder dem folgenden oder dem vorherigen Konstrukt (Statement, Deklaration) zugeordnet
- Heuristik – flexibel einstellbar im Formatierungsfile
- Verwechslungen sind möglich
- Übertragung in das Kommentarformat der Zielsprache
- Inhalt bleibt unverändert

Anwendung eines Translators

Vorgehen bei einer toolgestützten Migration:

1. Gründliches Studium der Translatordokumentationen, evtl. Schulung
2. Experimente an kleineren Programmen (Üben)
3. INI- und Formatierungsfile einstellen und anpassen, bis Ausgabe im gewünschten Format
4. Testübersetzung und Eliminieren nichtübertragbarer Anweisungen

Anwendung eines Translators (2)

5. Automatische Übersetzung
6. Test des entstandenen Zielcodes
7. Zerteilung dieses Codes
8. Nachbearbeitung der entstandenen Files (Kommentare zurechtrücken (wo nötig), Formatierung nachbessern, Strukturen, die von DDL kamen wieder von DDL erzeugen lassen, ...)
9. Einzeltests

Vorteile des Verfahrens

- Schneller als andere Migrationsmöglichkeiten
- Weniger Einarbeitungsaufwand in die Quell-Programme
- Erzeugt eine vollständige Liste von Warnungen, die auf kritische Anweisungen hinweisen
- Gibt „nebenbei“ eine Vielzahl von Reengineering-Informationen aus
- Formatierung („look and feel“) einheitlich

Nachteile des Verfahrens

- Vermittelt weniger Einblick in die Programme als z.B. Handkonvertierung
- Keine schrittweise Eingewöhnung in neue Sprache
- Formatierung muß an einigen Stellen etwas nachgebessert werden
- Meist ist eine Laufzeitbibliothek für die erzeugten Programme erforderlich

Automatisch kreativ

pro et con Innovative
Informatikanwendungen GmbH

Annaberger Str. 240
09125 Chemnitz

Tel. +49 371 5 347 353
Fax. +49 371 5 347 345
email: proetcon@tcc-chemnitz.de
http://www.tcc-chemnitz.de/firmen/proetcon