Visualisierung der Constraint-Propagation: VisualCHR

Manual

Matthias Saft

3. November 2000

Inhaltsverzeichnis

1 Einführung **1**

2 Anwendung von VisualCHR **3**

2.1 Starten des Programms . . . 3

2.2 Laden eines JCHR-Programms . . . 6

3 Funktionalitäten von VisualCHR **9**

3.1 Graphische Darstellung und Layout . . . 10

3.2 Auswahl bzw. Eingabe eines Ziels . . . 10

3.3 Schrittweise Auswertung . . . 11

3.4 Visualisierung der Constraints . . . 11

3.5 Visualisierung des Speichers . . . 14

3.6 Ausblenden von Speicher-Knoten . . . 14

3.7 Ausblenden von Constraint-Knoten . . . 16

3.8 Ausblenden von Regel-Knoten . . . 17

3.9 Anzeige der Regeln . . . 17

Literaturverzeichnis **19**

Abbildungsverzeichnis **20**

Kapitel 1

Einführung

Die deklarative Sprache Constraint Handling Rules (CHR) [1] dient der Implementierung von Constraintlösern. Die operationale Semantik von CHR ist die Anwendung einer endlichen Menge von Regeln auf eine endliche Menge von Constraints. Der regelbasierte Ansatz von CHR erlaubt es, die internen Vorgänge eines CHR-Constraintlöser zu betrachten. Somit ist es möglich, detailliert die Constraint-Propagation verfolgen zu können.

Die meisten CHR-Bibliotheken bieten für das Debugging die Möglichkeit, sogenannte *Traces* ausgeben zu lassen. Ein Trace ist prinzipiell eine textuelle Darstellung der Auswertung. Abbildung 1.1 zeigt einen Trace, der die Auswertung der einer Anfrage darstellt. Dieser Trace wurde von der CHR-Bibliothek für SICStus Prolog [2] erzeugt. Wie die Abbildung zeigt, enthalten Traces alle vorgenommenen, aber auch alle versuchten Auswertungsschritte. Ein versuchter Auswertungsschritt ist zum Beispiel die versuchte Anwendung einer entsprechenden Regel auf ein Constraint, für das aber kein passendes Partnerconstraint gefunden werden konnte. Traces stellen aber nicht direkt den aktuellen Inhalt des Constraintspeichers dar. Diese für das Debugging wichtigen Informationen muß man aus dem Trace zusammentragen. Da Traces in der Regel recht lang werden, ist es aber nur schwer möglich, die vorgenommenen Auswertungsschritte nachzuvollziehen, oder den aktuellen Inhalt des Constraintspeichers zu betrachten. Darüber hinaus ist eine textuelle Repräsentation der Informationen nicht wirklich übersichtlich.

Eine graphische Visualisierung der Auswertungsschritte, also beispielsweise des in Abbildung 1.1 dargestellten Traces, würde die Übersichtlichkeit und Verständlichkeit der Vorgänge beim Constraintlösen wesentlich erhöhen. Diese Aufgabe soll das im Folgenden vorgestellte Tool *VisualCHR* übernehmen. VisualCHR ist ein graphischer Tracer, der die Vorgänge bei der Auswertung einer Anfrage, also die Constraint-Propagation, graphisch darstellt. Dazu kann der Benutzer die Anfrage über eine graphische Benutzeroberfläche Schritt für Schritt ausführen. Die im jeweiligen Schritt angewandte Regel, sowie die Veränderungen der Constraints zum jeweiligen Zeitpunkt der Auswertung werden dann graphisch dargestellt. Dabei sollte klar hervorgehen, in welchem Schritt sich welche Veränderungen eingestellt haben.

```
| ?- leq(X, Y), leq(Z, X), leq(Y, Z).
      0   - call    leq(_69,_89)#<c5> ?
      1   1 try     leq:ground @ leq(_69,_89)#<c5>
      1   - insert  leq(_69,_89)#<c5>
      1   - exit    leq(_1640,_1712)#<c5> ?
      0   - call    leq(_126,_1640)#<c6> ?
      1   1 try     leq:ground @ leq(_126,_1640)#<c6>
      1   1 try     leq:transitivity @ leq(_126,_1640)#<c6>, leq(_1640,_1712)#<c5>
      1   1 apply   leq:transitivity @ leq(_126,_1640)#<c6>, leq(_1640,_1712)#<c5> ?
      1   - call    leq(_4493,_1712)#<c7> ?
      2   1 try     leq:ground @ leq(_4493,_1712)#<c7>
      2   - insert  leq(_4493,_1712)#<c7>
      2   - exit    leq(_6587,_6515)#<c7> ?
      1   - insert  leq(_6587,_4565)#<c6>
      1   - exit    leq(_6587,_4565)#<c6> ?
      0   - call    leq(_6515,_6587)#<c8> ?
      1   1 try     leq:ground @ leq(_6515,_6587)#<c8>
      1   1 try     leq:antisymmetry @ leq(_6515,_6587)#<c8>, leq(_6587,_6515)#<c7>
      1   1 apply   leq:antisymmetry @ leq(_6515,_6587)#<c8>, leq(_6587,_6515)#<c7> ?
      1   - remove  leq(_6587,_6515)#<c7>
      1   - remove  leq(_11212,_11283)#<c8>
      1   - wake    leq(_4565,_11212)#<c5> ?
      2   1 try     leq:ground @ leq(_4565,_11212)#<c5>
      2   1 try     leq:antisymmetry @ leq(_4565,_11212)#<c5>, leq(_11212,_4565)#<c6>
      2   1 apply   leq:antisymmetry @ leq(_4565,_11212)#<c5>, leq(_11212,_4565)#<c6> ?
      2   - remove  leq(_11212,_4565)#<c6>
      2   - remove  leq(_14739,_14812)#<c5>
      2   - exit    leq(_14739,_14739)#<c5> ?
      1   - exit    leq(_14739,_14739)#<c8> ?

Y = X,
Z = X ?

yes
```

Abbildung 1.1: Trace der Auswertung von `leq`

Kapitel 2

Anwendung von VisualCHR

2.1 Starten des Programms

Das Programm kann als Applet oder Applikation gestartet werden. Das Applet startet sich automatisch mit dem Laden einer entsprechenden HTML-Seite [5] in einem WWW-Browser. Das Programm muß also nicht auf dem lokalen Rechner installiert werden. Der verwendete WWW-Browser muß Java 1.1-Applets ausführen können, wie zum Beispiel der Netscape Communicator ab Version 4.5 oder Microsoft's Internet Explorer 5.

Die Oberflächenelemente wurden mittels Swing-Komponenten realisiert. Die Klassen dieser Komponenten befinden sich in dem Java-Archiv `swingall.jar`, welches man auf den Internet-Seiten des Programms [5], oder bei [4] herunterladen kann. Dieses Archiv sollte sich im Klassenpfad des Browsers befinden. Wie dies zu bewerkstelligen ist, ist dem Handbuch des Browsers zu entnehmen. Alternativ dazu können die Swing-Klassen auch mit dem Applet geladen werden. Da sie aber eine Größe von etwa 2,5 MB haben, ist bei öfterem Benutzen des Applets diese Alternative nicht sehr ratsam. Die Ladezeiten des Applet können dann bei einer langsamen Verbindung, wie zum Beispiel über ein Modem, durchaus mehr als 10 Minuten betragen.

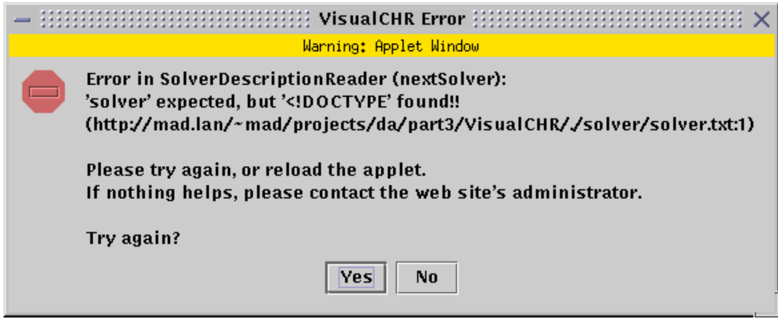


Abbildung 2.1: Fehler beim Laden des Applets

Eine weitere Möglichkeit, das Applet im Browser zu starten, ist die Nutzung

des Java-PlugIns [3]. Mittels dieses PlugIns erreicht man die beste Performanz, da die Java-Implementierungen der gängigen WWW-Browser nicht das Optimum an Performanz erreichen. Außerdem entfällt hier meist das Herunterladen der Swing-Klassen, da diese in der Regel zusammen mit dem PlugIn installiert werden.

Beim Laden des Applets kann unter Umständen ein Fehler auftreten, und zwar bei Verwendung einiger HTTP-Proxies. Für das Applet benötigte Dateien können dann nicht geladen werden. Es erscheint die in Abbildung 2.1 dargestellte Fehlermeldung. Hier kann durch Anklicken von *Yes* erneut versucht werden, diese Dateien vom WWW-Server zu holen. Bleibt der Fehler bestehen, so sollte man den WWW-Browser schließen, ihn neu starten und anschließend das Applet erneut laden. Behebt das erneute Laden des Applets den Fehler nicht, sollte man die Verwendung des HTTP-Proxies im WWW-Browser abschalten.

```
#!/bin/sh

if [ -z $JAVA_HOME ] ; then
    JAVA_HOME=/usr/lib/java
fi

if [ -z $SWING_HOME ] ; then
    SWING_HOME=/usr/lib/swing
fi

if [ -z $JCHR_HOME ] ; then
    JCHR_HOME=/home/proj/chr/jchr
fi

if [ -z $VCHR_HOME ] ; then
    VCHR_HOME=/home/proj/chr/vchr
fi

JAVA_CLASSES=$JAVA_HOME/lib/classes.zip
SWING_CLASSES=$SWING_HOME/swingall.jar
VJCHR_CLASSES=$JCHR_HOME/evaluator.jar:$VCHR_HOME/VisualCHR.jar
SOLVER_CLASSES=.
if [ $@ ] ; then
    SOLVER_CLASSES=$@
fi

CLASSPATH=$JAVA_CLASSES:$SWING_CLASSES:$VJCHR_CLASSES:$SOLVER_CLASSES

PARMS= \
" \
-SKIP_SLEEP 250 \
-COLOR_SCHEME system \
"

$JAVA_HOME/bin/java -classpath $CLASSPATH visualchr.VisualCHR $PARMS
```

Abbildung 2.2: VisualCHR Start-Skript

Soll das Programm als Applikation ausgeführt werden, um eigene JCHR-Löser zu debuggen, muß sich eine Java-Laufzeitumgebung [3] ab der Version

1.1 auf dem ausführenden Rechner befinden. Weiterhin müssen die benötigten Programm-Klassen lokal installiert werden. Diese sind die Swing-Klassen, welche sich im Java-Archiv `swingall.jar` befinden, die Klassen des JCHR-Evaluators im Archiv `evaluator.jar`, sowie das Programm-Archiv `VisualCHR.jar`. Man kann sie auf den Internet-Seiten des Programms [5] herunterladen. Alle diese Archive müssen in den Klassenpfad `CLASSPATH` der Java-Virtual-Machine (VM) eingefügt werden. Wichtig ist, daß sich der Pfad des zu testenden JCHR-Programms ebenfalls im Klassenpfad `CLASSPATH` befinden muß.

Gestartet wird das Programm durch Eingabe der Befehlszeile '`java visualchr.VisualCHR`'. Wurde der Klassenpfad `CLASSPATH` nicht schon vorher erweitert, so kann man ihn auch mittels des Java-Kommandozeilenparameters `-classpath` ändern. Einzelheiten dazu sind der Java-Dokumentation [3] zu entnehmen. Für VisualCHR stehen zwei Kommandozeilenparameter zur Verfügung, `-SKIP_SLEEP n` und `-COLOR_SCHEME {visualchr | system}`. Der Parameter `n` gibt die Anzahl der Millisekunden an, die beim *Skip* zwischen zwei Auswertungsschritten vergehen. Die Zahl sollte so groß gewählt werden, daß noch genug Zeit für das Aktualisieren des Graphs bleibt. Default sind 250 ms. Der Parameter für `-COLOR_SCHEME` gibt an, welches Farbschema die Benutzeroberfläche haben soll, entweder `visualchr` für das VisualCHR eigene Farbschema, oder der Defaultwert `system` für Systemfarben. Das in Abbildung 2.2 dargestellte Start-Skript für UNIX-Systeme, angepaßt auf die lokalen Verhältnisse, erleichtert den Aufruf. Als Kommandozeilenparameter für das Skript kann man den Pfad des zu testenden JCHR-Lösers angeben. Default-Einstellung ist das aktuelle Verzeichnis.

Wurde das Programm erfolgreich gestartet, erscheint der in Abbildung 2.3 dargestellte Startbildschirm. Dabei befindet sich dieser beim Starten als Applet innerhalb der HTML-Seite, und die Menüleiste, in der der Menüpunkt *Close* im Menü *Frame* das Beenden der Applikation erlaubt, ist dann nicht vorhanden. Das Applet beendet sich mit dem Verlassen der HTML-Seite, oder dem Beenden des WWW-Browsers.

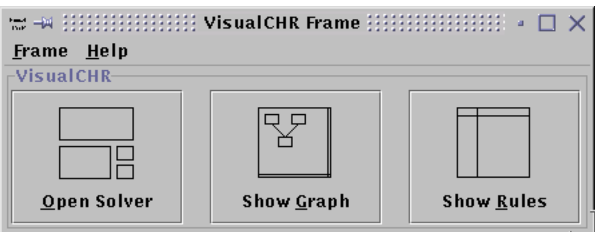


Abbildung 2.3: VisualCHR-Startbildschirm

Alle graphischen Elemente, also Knöpfe, Listen, Legenden und so weiter, enthalten interaktive Erklärungen, sogenannte *ToolTips*. Diese erscheinen etwa eine Sekunde nachdem der Mauszeiger auf ein solches Element bewegt wurde, und verschwinden nach etwa fünf Sekunden wieder. Der Mauszeiger ändert seine Form über Auswahllisten in eine Hand, und über Beschreibungen oder Legenden

wird er als kleines Kreuz dargestellt.

2.2 Laden eines JCHR-Programms

Wurde das Programm erfolgreich gestartet, kann der JCHR-Löser geladen werden. Wie schon erklärt, kann mit einem Applet kein eigener JCHR-Löser geladen werden, sondern nur Beispiel-Löser und -Anfragen. Daher erscheinen in der Applet- und Applikationsversion verschiedene Dialoge zum Öffnen von Lösern.

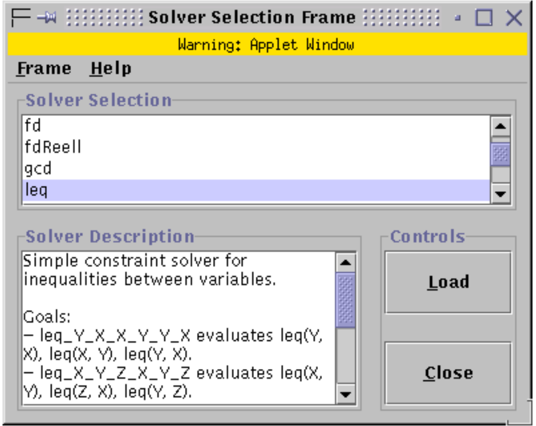


Abbildung 2.4: Laden eines Lösers im Applet

Zum Laden eines Lösers klickt man auf *Open Solver*, und es erscheint ein Dialogfenster *Solver Selection Frame*. Wurde das Applet aufgerufen, so erscheint der in Abbildung 2.4 dargestellte Dialog. Der Knopf ändert sich dann in *Hide Open*. Wird er angeklickt, so schließt sich das Dialogfenster wieder.

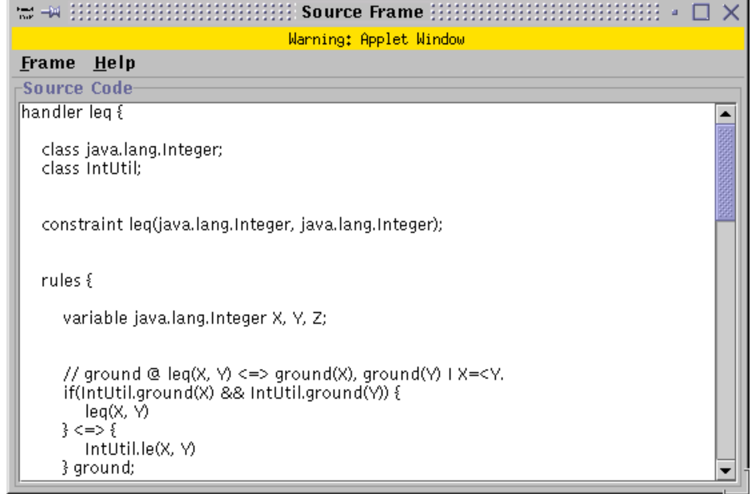


Abbildung 2.5: Anzeige des Quelltextes eines JCHR-Lösers

Aus der Liste *Solver Selection* kann ein zu ladender Beispiel-Löser mittels eines einfachen Klicks ausgewählt werden. Im Bereich *Solver Description* wird eine kurze Beschreibung des ausgewählten Lösers angezeigt. Der ausgewählte Löser kann mittels eines Klicks auf *Load*, oder einem Doppelklick in der Auswahlliste geladen werden. Wurden zuvor Auswertungen mit einem anderen Löser vorgenommen, gehen diese verloren. Der *Load*-Knopf ändert sich in *Show Source*. Wird er angeklickt, oder wiederum in der Auswahlliste auf den selektierten Löser doppelgeklickt, erscheint das in Abbildung 2.5 dargestellte Fenster *Source Frame*. Es stellt den Quelltext des ausgewählten JCHR-Lösers dar. Ein Ändern des Quelltextes ist nicht möglich. Dieses Fenster dient lediglich der Darstellung.

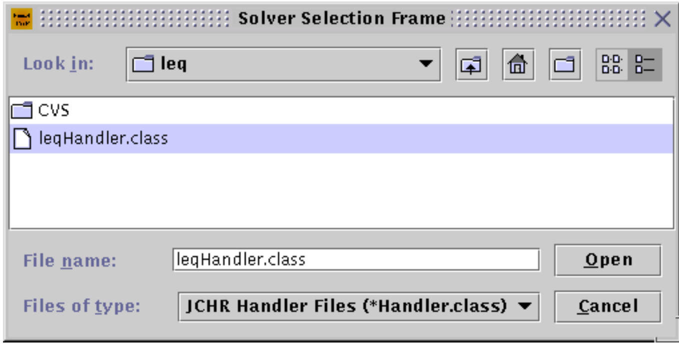


Abbildung 2.6: Laden eines Lösers in der Applikation

Wird das Quelltext-Fenster *Source Frame* angezeigt, so ändert sich der Knopf *Show Source* im *Solver Selection Frame* auf *Hide Source*. Mittels eines Klicks auf diesen Knopf, oder erneutem Doppelklick auf den in der Auswahlliste selektierten Löser, läßt sich das Quelltext-Fenster wieder schließen. Es kann natürlich auch durch auswählen von *Close* aus dem Menü *Frame* geschlossen werden. *Hide Source* wandelt sich beim Schließen des *Source Frames* wieder in *Show Source* zurück.

Abbildung 2.7: Fehler beim Laden eines Lösers in der Applikation

Wurde das Programm als Applikation gestartet, erscheint zum Laden eines Lösers der in Abbildung 2.6 dargestellte Dialog. Es ist darauf zu achten, nur vom JCHR-Compiler generierte `*Handler.class` Dateien zu öffnen. Der '`*`' steht in der Regel für den Namen des JCHR-Handlers. Wie schon erklärt muß sich der Pfad des zu ladenden Lösers im Klassenpfad `CLASSPATH` befinden. Andernfalls erscheint eine in Abbildung 2.7 dargestellte Fehlermeldung. Ein Anzeigen des

Quelltextes ist hier nicht vorgesehen, da dieser explizit geladen werden müsste. Die Darstellung macht keinen Sinn, da diese Aufgabe ein beliebiger Texteditor erledigen kann.

Kapitel 3

Funktionalitäten von VisualCHR

Durch Anklicken des Knopfs *Show Graph* im Hauptfenster (Abbildung 2.3) öffnet sich das in Abbildung 3.1 dargestellte Fenster für die graphische Auswertung *Graph Frame*. Der Knopf verändert sich in *Hide Graph*. Wird er erneut angeklickt, so schließt sich das Fenster wieder. Das Menü *Frame* enthält den Punkt *Close*, mit dem das Fenster ebenfalls geschlossen werden kann.

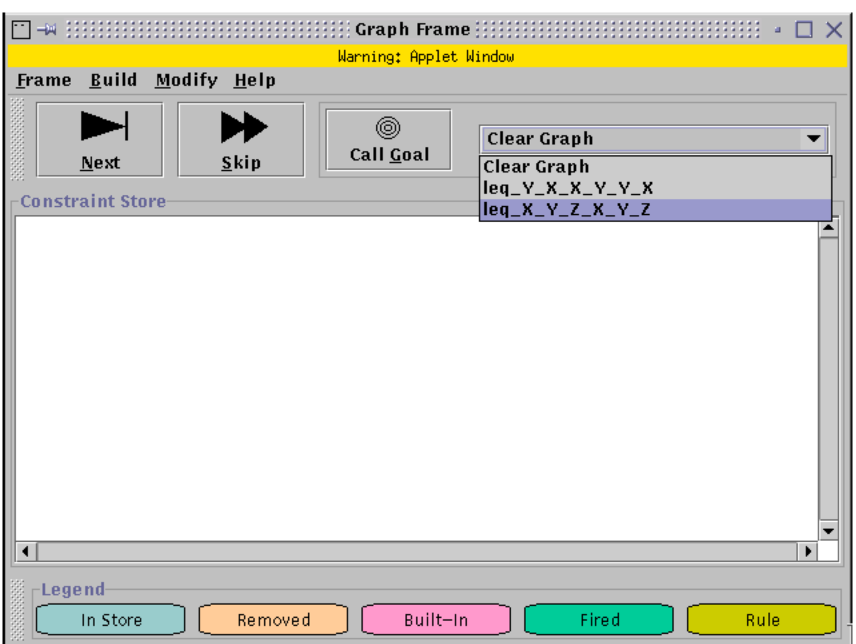


Abbildung 3.1: Graph-Fenster

Die Kontroll- und die Legenden-Leiste lassen sich an der linken, grau unterlegten Fläche aus dem Fenster ziehen. Dies ermöglicht mehr Platz für den Graphen.

3.1 Graphische Darstellung und Layout

Der Constraint-Graph wird nur nach unten erweitert. Hierdurch wird die Reihenfolge der Vorgänge bei der Auswertung des Ziels im Constraintspeicher dargestellt. Die Bedeutung der Farben der Knoten sind in der Legende und in Abbildung 3.2 dargestellt.

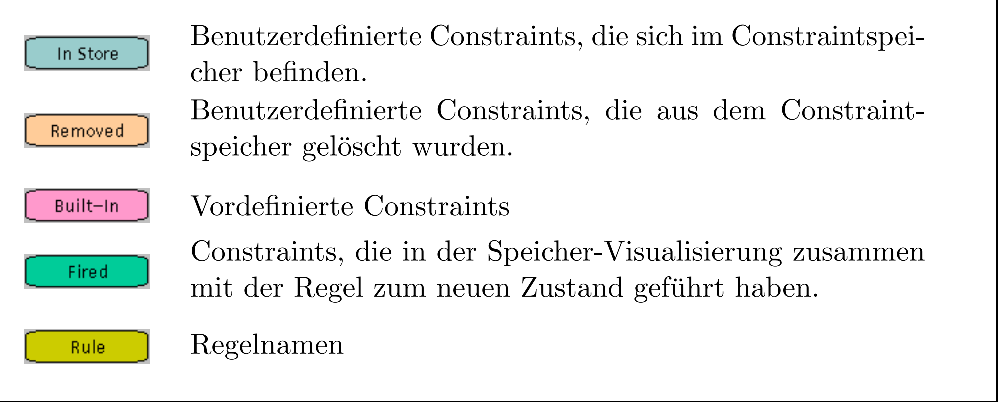


Abbildung 3.2: Bedeutung der Knotenfarben

Das Graphen-Layout ist manuell veränderbar. Die Knoten können bei gedrückter linker Maustaste verschoben werden. Die Kanten passen sich dem automatisch an. Wird die Breite des Fensters verändert, so verschieben sich die Knoten und Kanten wieder zu ihrer ursprünglichen Position zurück. Im Menü *Modify* befindet sich der Punkt *Layout Graph*. Hiermit kann das Layout des Graphen neu berechnet werden.

3.2 Auswahl bzw. Eingabe eines Ziels

In der Kontrolleiste befindet sich ein Auswahlfeld für die Ziele. In der Appletversion ist dieses Feld mit Namen von Beispiel-Zielen für die Auswertung vorbelegt. Das Auswählen des Namen eines Ziels startet die Anfrage. Wie schon erklärt können hier keine anderen als die vordefinierten Ziele ausgewertet werden. In der Applikationsversion ist dieses Feld nur mit *Clear Graph* vorbelegt, was die Auswertung abbricht und graphische Darstellung löscht. In dem Feld können die Namen der im eigenen JCHR-Löser definierten Ziele eingegeben werden. Mit Betätigen der ENTER-Taste oder durch Anklicken von *Call Goal* können diese Anfragen dann gestartet werden. Zusätzlich werden sie in der Liste gespeichert, und müssen somit kein eventuelles zweites mal eingegeben werden. Dabei ist peinlichst genau auf die Schreibweise der im eigenen Löser definierten Ziel-Namen zu achten. Startet die Auswertung nicht, liegt in der Regel ein Schreibfehler vor.

3.3 Schrittweise Auswertung

Nach dem Starten der Anfrage, werden zuerst die Ziel-Constraints in den Constraintspeicher, und somit auch in den Graphen, eingefügt. *Next* führt, wenn die Auswertung noch nicht abgeschlossen ist, den nächsten Auswertungsschritt aus. Nach Ausführen des letzten Schrittes erscheinen die Knöpfe *Next* und *Skip* in einem dunklen grau, und lassen sich nicht mehr anklicken. *Skip* überspringt die schrittweise Ausführung, und wertet die ausgewählte Anfrage automatisch aus. Dabei werden die durchgeführten Einzelschritte sofort im Graphen dargestellt. Nach dem Anklicken des *Skip* Knopfs verwandelt sich dieser in *Pause*. Wird auf *Pause* geklickt, so hält die automatische Ausführung an, und der Knopf verwandelt sich wieder in *Skip*. Um die ausgewählte Anfrage erneut zu starten, klickt man auf *Call Goal*. Die Funktionalitäten dieser Knöpfe sind zusätzlich im Menü *Build* unter den gleichnamigen Menüpunkten erreichbar.

3.4 Visualisierung der Constraints

Die voreingestellte Visualisierungsart ist die in Abbildung 3.3 dargestellte *Constraint Visualization*. Abbildung 3.3 soll im Folgenden als Beispiel erklärt werden.

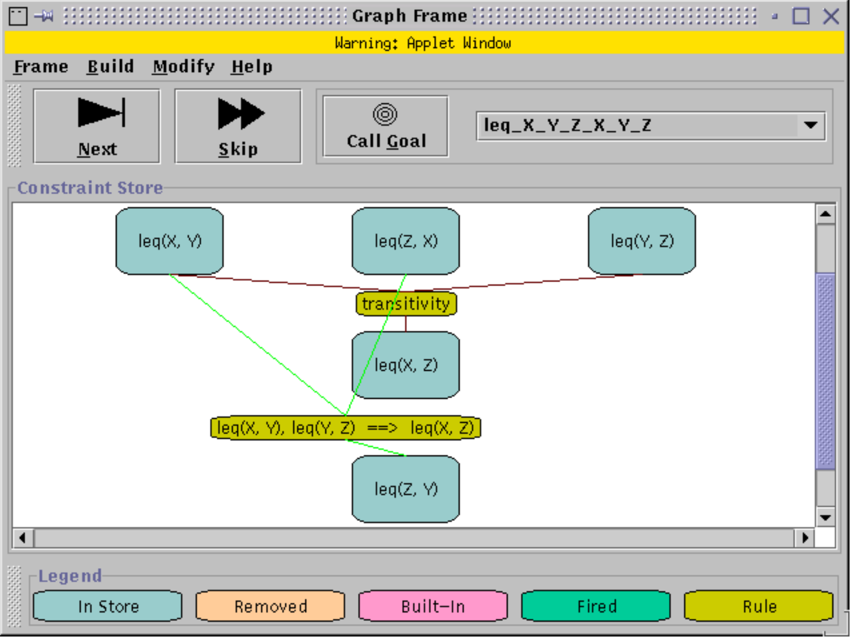


Abbildung 3.3: Visualisierung der Constraints

Jeder „große“ Knoten im Graph steht für ein einzelnes Constraint. Diese Knoten werden im Folgenden *Constraint-Knoten* genannt. Die „kleineren“ Knoten symbolisieren die Regeln, und beinhalten deren Namen. Sie werden

im Folgenden *Regel-Knoten* genannt. Zwischen den Knoten befinden sich die Kanten, wobei sich eine Kante immer nur zwischen einem Constraint- und Regelknoten befinden kann. Klickt man mit der linken Maustaste auf einen Regel-Knoten, so erscheinen die mit ihm verbundenen Kanten hellgrün, und statt des Regelnamens wird die vollständige Regel in einer abstrakten Syntax angezeigt. Ein erneuter Klick auf den Regel-Knoten stellt den vorherigen Zustand wieder her.

Die durch eine Kante oberhalb mit einem Regel-Knoten verbundenen Constraint-Knoten stellen den Kopf der Regel dar, die durch den entsprechenden Regel-Knoten symbolisiert wird. Die unterhalb durch eine Kante mit einem Regel-Knoten verbundenen Constraint-Knoten stellen den Rumpf der entsprechenden Regel dar. Werden Constraints im Kopf einer Regel gelöscht, so erscheinen die Kanten zu den gelöschten Constraints in blauer Farbe. Der Grund für diese gesonderte Farbgebung ist, das mehrere Regeln auf ein Constraint angewendet werden können. Bei gleicher Farbe der Kanten wäre dann bei einer eventuellen Elimination dieses Constraints nicht mehr auf den ersten Blick ersichtlich, welche Regel das Löschen veranlaßt hat. Werden zum Feuern einer Regel vordefinierte *Gleichheitsconstraints* verwendet, wird dies zusätzlich durch graue Kanten angezeigt.

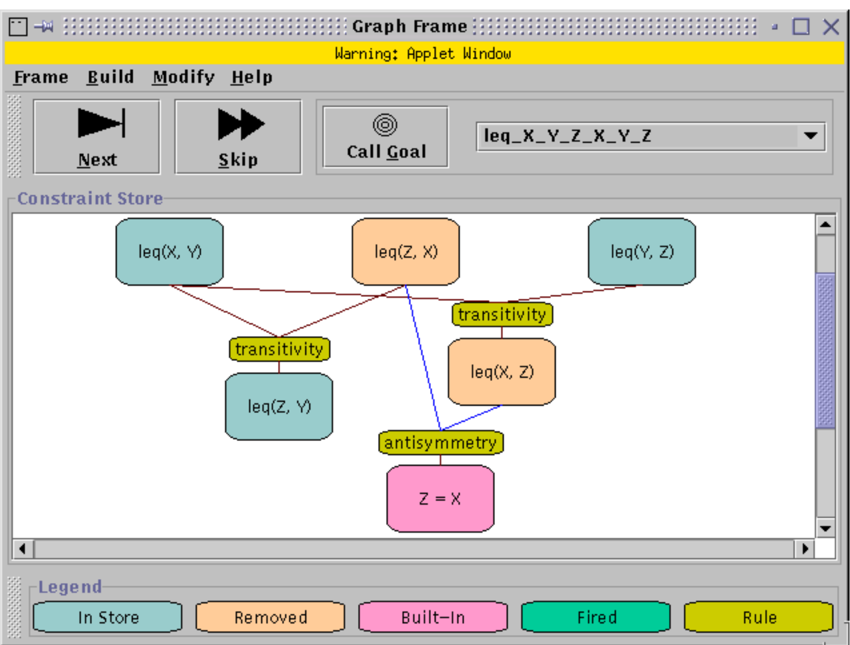


Abbildung 3.4: Weiterer Auswertungsschritt

In der ersten Zeile sind die in den Constraintspeicher eingefügten Ziel-Constraints `leq(X, Y)`, `leq(Z, X)` und `leq(Y, Z)` dargestellt. Sie werden farblich als *In Store* gekennzeichnet. Im ersten Schritt wurde die Regel `transitivity` auf die Constraints `leq(X, Y)` und `leq(Y, Z)` angewendet, und somit das neue Constraint `leq(X, Z)` erzeugt. Die beiden Constraints `leq(X, Y)` und

`leq(Y, Z)` verbleiben im Constraintspeicher. Im zweiten Schritt wurde wiederum die Regel `transitivity` angewendet, diesmal allerdings auf die Constraints `leq(X, Y)` und `leq(Z, X)`, welche wiederum im Speicher verbleiben. Es wird das neue Constraint `leq(Z, Y)` erzeugt. Ein Klick auf den Regelnamen zeigt die Regel `transitivity` selbst an.

Im in Abbildung 3.4 dargestellten dritten Schritt wird die Regel `antisymmetry` auf die Constraints `leq(Z, X)` und `leq(X, Z)` angewendet. Diese beiden benutzerdefinierten Constraints werden durch die Regelanwendung aus dem Constraintspeicher eliminiert. Diese Elimination wird farblich gekennzeichnet, indem die beiden gelöschten Constraints die Farbe *Removed* bekommen. Die Kanten zwischen diesen beiden Constraints und der Regel `antisymmetry` erscheinen in blauer Farbe, da diese Regel für das Löschen dieser Constraints verantwortlich ist.

Es wird das neue (Gleichheits-)Constraint `Z = X` eingefügt. Da es sich hierbei um ein vordefiniertes Constraint handelt, wird es farblich als *Built-In* gekennzeichnet.

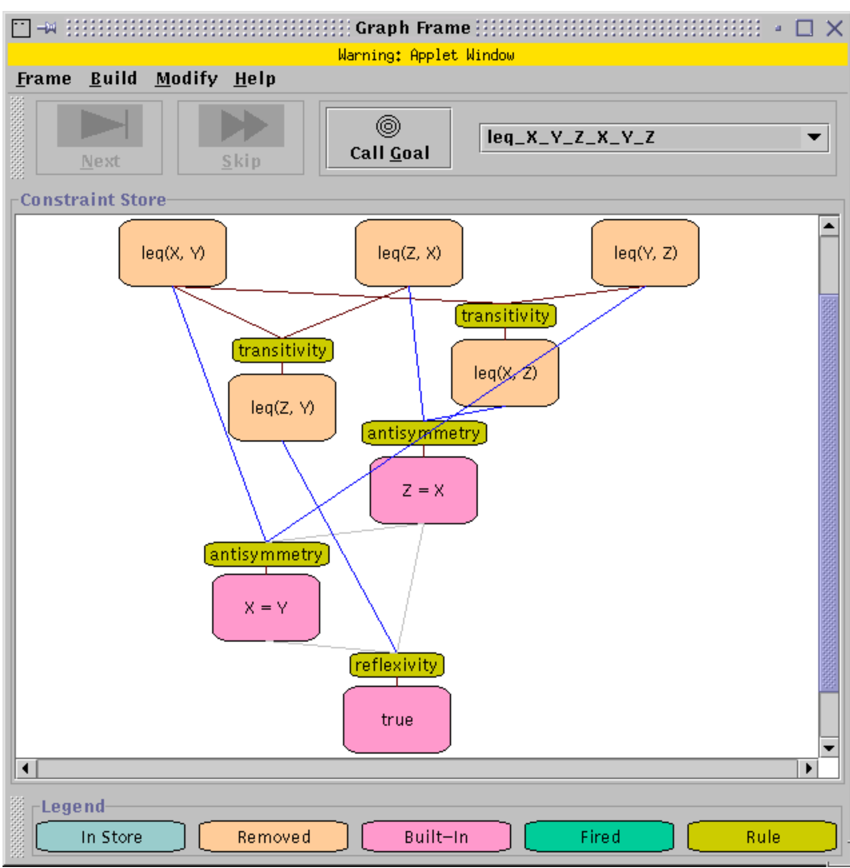


Abbildung 3.5: Letzter Auswertungsschritt

Im nächsten Auswertungsschritt wird die Regel `antisymmetry` auf die beiden Constraints `leq(X, Y)` und `leq(Y, Z)` angewendet. Diese beiden Con-

straints werden durch die Regelanwendung gelöscht, und dafür das vordefinierte Constraint `X = Y` eingefügt. Da zum Feuern der Regel das im letzten Schritt erzeugte Constraint `Z = X` benutzt wurde, wird dies durch die graue Kante dargestellt.

Abbildung 3.5 zeigt den Zustand nach dem letzten Auswertungsschritt. Es wurde die Regel `reflexivity` auf das Constraint `leq(Z, Y)` angewendet. Dieses Constraint wird gelöscht, und dafür das vordefinierte Constraint `true` eingefügt. Die Anwendung dieser Regel ist hier möglich, da mit `Z = X` und `X = Y` auch `Z = Y` gilt. Dies wird durch die beiden grauen Kanten symbolisiert.

Alle benutzerdefinierten Constraints sind jetzt als gelöscht markiert und es verbleiben nur noch die vordefinierten Constraints `Z = X`, `X = Y` und `true`. Es ist jetzt keine Regel mehr anwendbar, und die Auswertung terminiert. Die Lösung ist `Z = X` und `X = Y`.

3.5 Visualisierung des Speichers

Im Menü *Modify* kann in die in Abbildung 3.6 dargestellte *Store Visualization* gewechselt werden. Hierbei wird der gesamte Inhalt des Constraintspeichers zum jeweiligen Zeitpunkt der Auswertung in einem einzelnen Knoten dargestellt. Dabei werden Constraints, die bei einem Auswertungsschritt im Speicher verblieben sind, in den neuen Knoten übernommen. Ebenso verhält es sich mit Variablenbindungen. Diese haben zwar die gleiche Farbe wie vordefinierte Constraints, dürfen aber nicht mit dem vordefinierten Gleichheitsconstraint '`=`' verwechselt werden. Sie werden ab dem Zeitpunkt ihres Auftretens in jeden neuen Knoten eingefügt. Diese Knoten werden im Folgenden *Speicher-Knoten* genannt. Beim Umschalten von Constraint- und Speicher-Visualisierung wird der Graph neu gezeichnet, das heißt, manuelle Layoutänderungen gehen verloren.

In der Speicher-Visualisierung wird nicht mehr durch die Kanten von und zu den Regel-Knoten angezeigt, welche Constraints zu dem neuen Zustand geführt haben. Dies wird durch eine gesonderte Farbgebung (siehe Abbildung 3.2) erreicht. Constraints, auf die eine Regel angewendet wurde, werden als farblich als *Fired* gekennzeichnet. Kanten werden nur noch zwischen Speicher- und Regel-Knoten dargestellt.

3.6 Ausblenden von Speicher-Knoten

Während der Auswertung werden alle Vorgänge im Constraintspeicher visualisiert. Einige Informationen sind dabei vielleicht gar nicht von Interesse für den Anwender. Damit dieser sich auf die wesentlichen Informationen konzentrieren kann, können die Unwesentlichen ausgeblendet werden. Klickt man mit der rechten Maustaste auf einen Speicher-Knoten, so erscheint ein Popup-Menü. Wird hier *Hide This Store* ausgewählt, so reduziert sich der angeklickte Knoten auf ein kleines Sechseck. Wird *Hide Recursive Backward* bzw. *Hide Recursive*

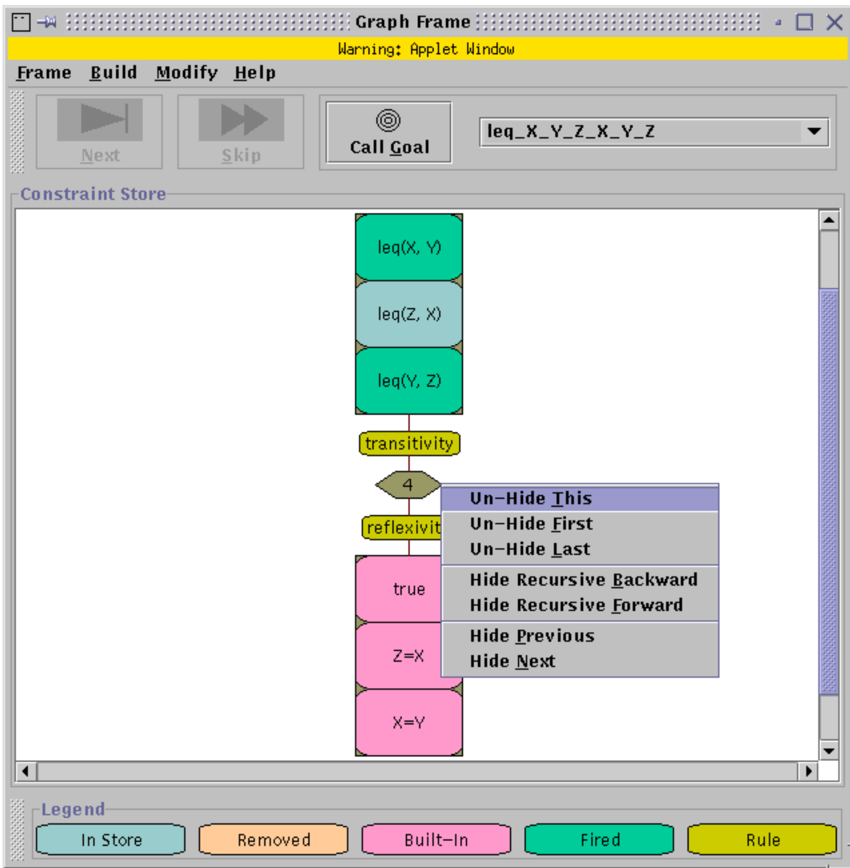


Abbildung 3.6: Visualisierung des Speichers

Forward aus dem Menü ausgewählt, so werden der angeklickte Knoten, sowie alle sich darüber bzw. darunter befindenden Knoten in einem einzelnen reduzierten Knoten versteckt. Die Ziffer in solch einem reduzierten Knoten gibt die Anzahl der in diesem Knoten versteckten Speicher-Knoten an.

Wird mit der rechten Maustaste auf einen reduzierten Knoten geklickt, so erscheint ein anderes Popup-Menü (siehe auch Abbildung 3.6). Mit *Un-Hide This* werden alle im angeklickten Knoten verstecken Speicher-Knoten wiederhergestellt. *Un-Hide First* bzw. *Un-Hide Last* stellen nur den ersten bzw. letzten versteckten Knoten wieder her. Die Punkte *Hide Recursive Backward* und *Hide Recursive Forward* haben die gleiche Wirkung wie die entsprechenden Punkte im Menü der Speicher-Knoten. Mit *Hide Previous* bzw. *Hide Next* lassen sich der vorherige bzw. nächste Knoten im angeklickten Knoten verstecken. Manuelle Layoutänderungen gehen beim Verstecken und Wiederherstellen von Knoten verloren, da sich die Größe der Knoten so stark ändert, daß Überlagerungen im Graphen entstehen würden.

3.7 Ausblenden von Constraint-Knoten

Es lassen sich aber nicht nur Speicher-Knoten verstecken, sondern auch Constraint-Knoten. Zum Verstecken von Constraint-Knoten ruft man im Menü *Modify* den Punkt *Constraint Display Options* auf. Es erscheint das in Abbildung 3.7 dargestellte Dialogfenster. In ihm befinden sich zwei Listen. Sie enthalten keine einzelnen Constraints, sondern *Klassen* von Constraints. Dabei sind zum Beispiel `leq(X, Y)` und `leq(Y, Z)` von der gleichen Klasse `leq`. Die obere Liste enthält alle bisher in der Auswertung aufgetretenen Klassen von benutzerdefinierten Constraints, die untere Liste von vordefinierten Constraints.

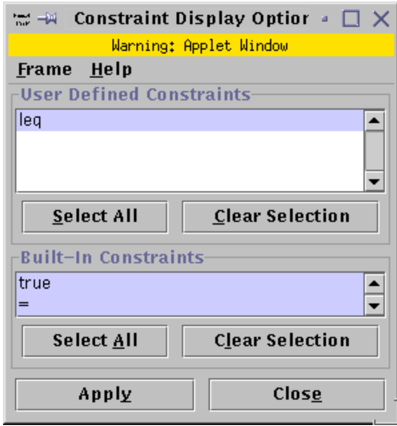


Abbildung 3.7: Constraint-Visualisierungs-Optionen

Es werden nur die Constraints angezeigt, die zu einer der selektierten Klassen gehören. Um alle Klassen zu selektieren, klickt man auf *Select All*. Zum Entfernen aller Selektionen, klickt man auf *Clear Selection*. Mit *Apply* werden die Änderungen im Graphen vorgenommen. Dabei wird das Layout des Graphen neu berechnet, d.h. manuelle Änderungen am Layout gehen verloren. Die Änderungen haben Auswirkung auf beide Darstellungsarten, also auf die Constraint- und Speicher-Visualisierung. Ein Speicher-Knoten wird reduziert dargestellt, wenn alle in ihm enthaltenen Constraint-Knoten ausgeblendet wurden. Der Knopf *Close*, sowie der entsprechende Menüpunkt *Close* im Menü *Frame*, schließen das Dialogfenster.

Alternativ kann zum Verstecken von Constraint-Knoten auch auf die entsprechenden Knoten im Graphen mit der rechten Maustaste geklickt werden. Es erscheint ein Popup-Menü. Der Punkt *Hide This Constraint* blendet nur das angeklickte Constraint aus. Das Layout des Graphen ändert sich hierbei nicht. Der Punkt *Hide All Constraints* versteckt alle Constraints, die der gleichen Klasse wie das angeklickte Constraint angehören. Das Layout des Graphen wird hierbei allerdings wieder neu berechnet. Das Wiederherstellen der ausgeblendeten Constraints ist mittels des oben beschriebenen *Constraint Display Options* Dialogs möglich. Wurden nur einzelne Constraints einer Klasse versteckt, so sind natürlich die Klassen dieser Constraints weiterhin im Dialogfenster selektiert.

Um sie wieder einzublenden, muß dann nur *Apply* angeklickt werden.

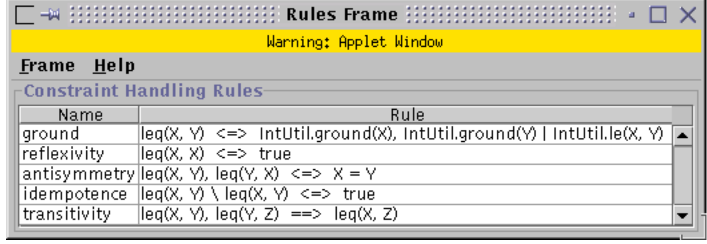
3.8 Ausblenden von Regel-Knoten

Analog dem Ausblenden von Constraint-Knoten funktioniert das Verstecken von Regel-Knoten. Wird im Menü *Modify* den Punkt *Rule Display Options* aufgerufen, erscheint das in Abbildung 3.8 dargestellte Dialogfenster.

Abbildung 3.8: Regel-Visualisierungs-Optionen

Es werden nur die Regeln angezeigt, deren Namen in der Liste *Rule Names* selektiert sind. Die Bedienung des Dialogfensters ist völlig analog der Bedienung des *Constraint Display Options* Dialogs. Klickt man im Graphen mit der rechten Maustaste auf einen Regel-Knoten, so erscheint ein Popup-Menü. Hier können mittels *Hide This Rule* die angeklickte Regel, und durch *Hide All Rules* alle Regeln mit dem gleichen Namen wie die angeklickte Regel ausgeblendet werden. Beim Ausblenden einer Regel werden alle aus der Regelanwendung resultierenden Constraints ebenfalls verborgen. Die ausgeblendeten Regeln können mittels des *Rule Display Options* Dialogs wieder angezeigt werden. Das Verstecken von Regel-Knoten hat, wie bei den Constraint-Knoten auch, Auswirkungen in beiden Darstellungsarten.

3.9 Anzeige der Regeln



Name	Rule
ground	leq(X, Y) <=> IntUtil.ground(X), IntUtil.ground(Y) | IntUtil.le(X, Y)
reflexivity	leq(X, X) <=> true
antisymmetry	leq(X, Y), leq(Y, X) <=> X = Y
idempotence	leq(X, Y) \ leq(X, Y) <=> true
transitivity	leq(X, Y), leq(Y, Z) ==> leq(X, Z)

Abbildung 3.9: Anzeige der Regeln

Durch Anklicken von *Show Rules* im Hauptfenster (Abbildung 2.3) öffnet sich das in Abbildung 3.9 dargestellte Fenster *Rules Frame*. Der Knopf verändert sich in *Hide Rules*. Wird er angeklickt, so schließt sich das Fenster wieder.

Hier werden alle im JCHR-Löser definierten Regeln zusammen mit ihrem Namen angezeigt. Die Syntax der angezeigten Regeln entspricht nicht der JCHR-Syntax, sondern der abstrakten Syntax. Mittels *Close* im Menü *Frame* kann das Fenster wieder geschlossen werden.

Literaturverzeichnis

[1] Thom Frühwirth. Constraint Handling Rules.
Internet: http://www.informatik.uni-muenchen.de/~fruehwir/chr/.

[2] SICS - Swedish Institute of Computer Science, Kista, Sweden.
SICStus Prolog User's Manual Release 3.7.1, October 1998. Internet: http://www.sics.se/isl/sicstus/.

[3] Sun Microsystems. *The Java™ Development Kit.*
Internet: http://java.sun.com/products/jdk/.

[4] Sun Microsystems. *The Java™ Foundation Classes - Swing 1.1.* Internet: http://java.sun.com/products/jfc/.

[5] Visualisation of Constraints: VisualCHR.
Internet: http://www.informatik.uni-muenchen.de/~saft/visualchr/.

Abbildungsverzeichnis

1.1 Trace der Auswertung von `leq` 2

2.1 Fehler beim Laden des Applets 3
2.2 VisualCHR Start-Skript 4
2.3 VisualCHR-Startbildschirm 5
2.4 Laden eines Lösers im Applet 6
2.5 JCHR-Löser Quelltext-Fenster 6
2.6 Laden eines Lösers in der Applikation 7
2.7 Fehler beim Laden eines Lösers in der Applikation 7

3.1 Graph-Fenster 9
3.2 Bedeutung der Knotenfarben 10
3.3 Visualisierung der Constraints 11
3.4 Weiterer Auswertungsschritt 12
3.5 Letzter Auswertungsschritt 13
3.6 Visualisierung des Speichers 15
3.7 Constraint-Visualisierungs-Optionen 16
3.8 Regel-Visualisierungs-Optionen 17
3.9 Anzeige der Regeln 17